A Nearly Optimal Deterministic Algorithm for Online Transportation Problem

Tsubasa Harada[1]
Institute of Science Tokyo, Japan

Toshiya Itoh
Institute of Science Tokyo, Japan

Abstract

For the online transportation problem with m server sites, it has long been known that the competitive ratio of any deterministic algorithm is at least $2m - 1$. Kalyanasundaram and Pruhs conjectured in 1998 that a deterministic $(2m - 1)$-competitive algorithm exists for this problem, a conjecture that has remained open for over two decades.

In this paper, we propose a new deterministic algorithm for the online transportation problem and show that it achieves a competitive ratio of at most $8m - 5$. This is the first $O(m)$-competitive deterministic algorithm, coming close to the lower bound of $2m - 1$ within a constant factor.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online algorithms, Competitive analysis, Online metric matching, Online weighted matching, Online minimum weight perfect matching, Online transportation problem, Online facility assignment, Greedy algorithm

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.94

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2406.03778v3`

Funding This work was partially supported by Japan Science and Technology Agency (JST) as part of Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE), Grant Number JPMJAP2302.

1 Introduction

1.1 Background

The *online transportation problem* (OTR), also known as the *online facility assignment*, was introduced by Kalyanasundaram and Pruhs [12]. In this problem, k servers are placed at m $(\leq k)$ sites on a metric space and an online algorithm receives (at most) k requests one-by-one in an online manner. The number of servers at one site is considered its capacity. The task of an online algorithm is to assign each request irrevocably and immediately to one of the available servers. The cost of assigning a request to a server is determined by the distance between them. The objective of the problem is to minimize the sum of the costs of assigning all requests. We denote the problem as $\mathrm{OTR}(k, m)$ when there are k servers and m server sites.

The online transportation problem finds application in various scenarios. Here are two examples: In the first example, we regard a server site as a hospital, a hospital's capacity as the number of beds it has, and a request as a patient. This problem can then be viewed as a problem of finding a way to assign patients to hospitals so that patients are transported

[1] Corresponding author

to the hospital as close as possible. In the second example, we regard a server site as a car station, a capacity of the car station as the number of cars it can accommodate, and a request as a user of this car sharing service. The problem can then be viewed as a problem of designing a car sharing service that allows users to use car stations as close as possible.

The *online metric matching* (OMM), or *online weighted matching*, is a special case of OTR in which k servers are placed at distinct k sites, i.e., each server has unit capacity. Let OMM(k) denote OMM with k servers. Kalyanasundaram and Pruhs [10] and Khuller et al. [14] independently showed that for OMM(k), the competitive ratio of any deterministic algorithm is at least $2k-1$. This immediately leads to a lower bound of $2m-1$ on the competitive ratio of any deterministic algorithm for OTR(k,m). Furthermore, they also proposed a $(2k-1)$-competitive algorithm for OMM(k) called *Permutation* in [10]. From this result, the Permutation algorithm could be expected to have a matching upper bound of $2m-1$ on the competitive ratio for OTR(k,m). However, Kalyanasundaram and Pruhs [11] reported without proofs that the competitive ratio of Permutation is $\Theta(k)$ and that the competitive ratio of the natural greedy algorithm is 2^m-1. These results imply the existence of a large gap between the upper bound $O(\min\{k,2^m\})$ and lower bound $\Omega(m)$ on the competitive ratio for OTR(k,m). Since $m \le k$, when k is sufficiently larger than m, e.g., $k=O(2^m)$, this gap becomes even more pronounced. Based on this discussion, they posed the following conjecture.

▶ **Conjecture 1** (Kalyanasundaram and Pruhs [11]). *For* OTR(k,m)*, what is the optimal competitive ratio in terms of m? It seems that there should be a* $(2m-1)$*-competitive algorithm.*

Since the publication of this conjecture, there have been many studies of OMM and OTR. Among them, Nayyar and Raghvendra [17] proved that the competitive ratio of the *Robust-Matching* algorithm [19] is $O(m\log^2 k)$, thereby reducing the upper bound on the competitive ratio for OTR to $O(\min\{m\log^2 k, k, 2^m\})$. However, neither the upper nor the lower bounds on the competitive ratio have been improved since then.

1.2 Our contributions

In this paper, we propose a new deterministic algorithm called *Subtree-Decomposition* and show that it is $(8m-5)$-competitive for OTR with m server sites. This is the first deterministic algorithm achieving $O(m)$-competitiveness within a constant factor of Conjecture 1, significantly reducing the upper bound on the competitive ratio for OTR from $O(\min\{m\log^2 k, k, 2^m\})$ to $8m-5$. Given that the lower bound for OTR is known to be $2m-1$, our algorithm achieves the best competitive ratio in terms of its order with respect to m (and k). Furthermore, our algorithm processes each request in $O(m)$ time, whereas the *Robust-Matching* algorithm, which has a competitive ratio of $O(m\log^2 k)$, requires $O(m^2)$ time per request [19, Theorem 6]. In other words, when k is close to m (e.g. $k=O(m)$), although the upper bounds $8m-5$ and $O(m\log^2 m)$ differ by only a poly-logarithmic factor, our algorithm is computationally more efficient than *Robust-Matching*.

We also develop a generic method to convert an algorithm designed for tree metrics into one for general metric spaces without significantly degrading the competitive ratio. This method can be applied to designing an online algorithm for optimization problems on a metric space, such as OMM.

1.3 Our techniques

Reduction to a special class of OTR instances

First, we will explain how to reduce the task of designing an $O(m)$-competitive algorithm for general instances of OTR to the task of designing an algorithm for more specific instances. To this end, let us begin by defining two special cases of OMM and introducing the concept of *T-strong competitive ratio*, which serves as a stricter performance measure for algorithms compared to the standard competitive ratio.

The first special case of OMM is OMM where each request is placed on the same position as a server (denoted by OMM_S) and the second one is a further special case of OMM_S called *online matching on a power-of-two tree metric* (denoted by OMT^2_S). In this problem, a metric space is induced by a weighted tree $T = (V(T), E(T))$ in which the weight of each edge is a non-negative integer power of two, and the set of server sites coincides with the set $V(T)$ of vertices. Let $\mathrm{OMM}_S(k)$ denote OMM_S with k servers and $\mathrm{OMT}^2_S(T)$ denote OMT^2_S with a metric space T. Note that $|V(T)| = |E(T)| + 1$ is the number of servers in $\mathrm{OMT}^2_S(T)$.

The T-strong competitive ratio is defined for OTR on a tree metric (the "T" is derived from "tree"). While the standard competitive ratio measures both the cost of the algorithm and the cost of the optimal offline algorithm in terms of the sum of the edge weights along the path between requests and servers (commonly referred to as the "path distance"), the T-strong competitive ratio measures the algorithm's cost using the path distance, whereas the cost of the optimal offline algorithm is measured using the weight of the heaviest edge on the path between the requests and servers (referred to as the "max-weight distance"). In other words, we say that an algorithm is T-strongly α-competitive if, for any instance of OTR on a tree metric, the cost incurred by the algorithm, measured by the path distance, is at most α times the optimal offline cost measured by the max-weight distance. Since the max-weight distance is shorter than the path distance, the T-strong competitive ratio is greater than the standard competitive ratio in general.

In this paper, we demonstrate that if a greedy-like algorithm that only uses the positions of a current request and current available server sites (referred to as MPFS [9][2]) is designed to be T-strongly $O(k)$-competitive for OMT^2_S with k servers, it can be transformed into an $O(m)$-competitive algorithm for $\mathrm{OTR}(k, m)$ (Theorem 9). The proof is completed by establishing the following three claims:

(1) If there exists a T-strongly $O(k)$-competitive algorithm for OMT^2_S with k servers, then there exists an $O(k)$-competitive algorithm for $\mathrm{OMM}_S(k)$.

(2) If there exists an $O(k)$-competitive algorithm for $\mathrm{OMM}_S(k)$, then there exists an $O(k)$-competitive algorithm for $\mathrm{OMM}(k)$.

(3) If a certain MPFS algorithm is $O(k)$-competitive for $\mathrm{OMM}(k)$, then that algorithm is $O(m)$-competitive for $\mathrm{OTR}(k, m)$.

Claim (2) was previously proven by Meyerson et al. [16], and Claim (3) was proven by Harada et al. [9]. In this paper, we prove Claim (1). To this end, we begin by noting that it suffices to demonstrate a method for obtaining an $O(k)$-competitive algorithm for $\mathrm{OMM}_S(k)$ with a general k-point metric space by using a T-strongly $O(k)$-competitive algorithm for $\mathrm{OMM}_S(k)$ on a tree metric (where the edge weights are not necessarily powers of two).

Next, we explain how to convert algorithm $\mathcal{A}$, which is T-strongly α-competitive for a tree metric, into algorithm $\mathcal{B}$, which is α-competitive for a general k-point metric space $\mathcal{M}$. $\mathcal{B}$ first treats the given metric space as a weighted complete graph, where the weight of each edge corresponds to the distance between its endpoints. $\mathcal{B}$ then finds a minimum spanning tree (MST) T of this graph and simulates algorithm $\mathcal{A}$ on T.

[2] MPFS is a class of algorithms that generalize the greedy algorithm (see Definition 6 for details).

The key fact to prove that $\mathcal{B}$ is α-competitive for a general metric space $\mathcal{M}$ is that the distance between any two points in the original metric space $\mathcal{M}$ lies between the max-weight distance and the path distance on the MST T. Using this fact, we can verify that $\mathcal{B}$ is α-competitive as follows: First, the $\mathcal{B}$'s cost measured by the distance in $\mathcal{M}$ is at most the $\mathcal{A}$'s cost measured by the path distance on T. Then, by the T-strong competitiveness of $\mathcal{A}$, the $\mathcal{A}$'s cost measured by the path distance is at most α times the optimal offline cost measured by the max-weight distance. Since the optimal offline cost measured by the max-weight distance is not greater than the optimal offline cost measured by the distance in $\mathcal{M}$, it follows that the $\mathcal{B}$'s cost is at most α times the optimal offline cost measured by the distance in $\mathcal{M}$.

Finally, we briefly explain why the distance in the original metric space $\mathcal{M}$ lies between the max-weight distance and the path distance on the MST T. By the triangle inequality, the distance in $\mathcal{M}$ is at most the path distance. To see that the distance in $\mathcal{M}$ is at least the max-weight distance, consider that if the distance in $\mathcal{M}$ between two points u and v were smaller than the max-weight distance on T, one could create a spanning tree with a smaller weight by adding edge (u, v) to T and removing the heaviest edge on the u-v path in T. This would contradict T being an MST.

The above discussion reduces the task of designing an $O(m)$-competitive algorithm for $\mathrm{OTR}(k, m)$ to designing a T-strongly $O(k)$-competitive MPFS algorithm for OMT_S^2 with k servers.

Overview of our algorithm

In the following, we provide an overview of the T-strongly $O(k)$-competitive algorithm for $\mathrm{OMT}_S^2(T)$ with k servers, which we refer to as Subtree-Decomposition (SD). Note that in OMT_S^2, the set of vertices coincides with the set of servers, meaning that $k = |V(T)| = |E(T)| + 1$.

The proposed algorithm is based on a simple depth-first search (DFS) algorithm. The DFS-based algorithm begins by arbitrarily selecting one vertex in T as the root before receiving any requests. When a request arrives at a vertex, the algorithm performs a DFS starting from that vertex, and assigns the request to the first available server it encounters.

Intuitively, one can understand why the T-strong competitive ratio of this DFS-based algorithm is at most $O(k)$ when the given tree T is unweighted (where each edge has unit weight) as follows: The nature of DFS ensures that any edge will be traversed at most twice by the algorithm's assignments. In instances where there exist edges traversed more than twice, the optimal offline cost also increases in proportion to the number of such edges, resulting in a sufficiently small ratio of the algorithm's cost to the optimal offline cost[3]. Thus, the algorithm's cost measured by the path distance is roughly $O(|E(T)|) = O(k)$. On the other hand, for any request sequence where the algorithm incurs a non-zero cost, the optimal offline cost measured by the max-weight distance is at least 1. Therefore, the ratio of the algorithm's cost measured by the path distance to the optimal offline cost measured by the max-weight distance is $O(k)$.

However, naive application of this algorithm to a weighted tree results in inefficiencies. For instance, if a request occurs at a vertex where the edges connecting it to its children have very large weights, while the edge connecting it to its parent has a very small weight,

[3] Note that this statement is imprecise, as discussed and justified in this paper through the analysis via "hybrid algorithm" [7].

the algorithm will prioritize assigning the request to a more costly child. To address this inefficiency, SD performs a stepwise DFS, assigning the request at vertex v to the first available server it encounters by prioritizing the exploration of lighter edges. Specifically, SD proceeds as follows:

(1) Perform a DFS from v, restricted to edges with weights no greater than 1.
(2) If no available server is found in step 1, return to v and perform another DFS, this time restricted to edges with weights no greater than 2.
(3) In subsequent steps, double the threshold for edge weights and perform a DFS from v until an available server is found.

Below, we provide an intuitive explanation for why SD is T-strongly $O(k)$-competitive for $\mathrm{OMT}_S^2(T)$ with k servers. Let 2^n be the weight of the heaviest edge used in the optimal offline assignment. In this case, the optimal offline cost measured by the max-weight distance is at least 2^n. SD is designed to minimize the number of times it traverses heavy edges, and it can be shown that no edge with weight greater than 2^n is traversed in SD's assignment. Furthermore, an edge with weight exactly 2^n is traversed at most twice by the nature of DFS. Similarly, edges with weight 2^{n-1} are traversed up to two times during the $(n-1)$-th DFS and another two times during the final n-th DFS, for a total of four traversals. Extending this reasoning, for each $i = 0, \ldots, n$, the number of traversals for an edge with weight 2^i in SD's assignment is expected to be at most $2(n-i+1) \leq 2^{n-i+1}$. Therefore, the algorithm's cost measured by the path distance is roughly at most $O(2^i \times 2^{n-i+1} \times |E(T)|) = O(2^n|E(T)|)$, and the ratio to the optimal offline cost measured by the max-weight distance is $O(|E(T)|) = O(k)$.

1.4 Related Work

Online Metric Matching on a Line. A line metric is one of the most interesting and well-studied metric spaces for these problems. In particular, OMM on a line has been actively researched [15, 7, 2, 3, 20]. The best upper bound on the competitive ratio for OMM on a line is $O(\log k)$ [20], which is achieved by the Robust-Matching algorithm [19], and the best lower bound on the competitive ratio [18] is $\Omega(\sqrt{\log k})$, where k denotes the number of servers. Note that the lower bound $\Omega(\sqrt{\log k})$ can also be applied to any randomized algorithm for OMM on a line. As can be seen from the above, the best possible competitive ratio for OMM on a line has remained open.

There are also some studies on OTR on a line. Ahmed et al. [1] addressed competitive analysis for OTR on a line under the assumption that the server sites are evenly placed and each site has the same capacity. Under this assumption, they showed (with rough proofs) that the natural greedy algorithm is $4m$-competitive and the Permutation algorithm (called *Optimal-fill* in their paper) is m-competitive for any $m > 2$. On the other hand, Harada and Itoh [8] studied OTR on a line with a general layout of servers. They proposed an $(2\alpha(S)+1)$-competitive algorithm called *PTCP* (Policy Transition at Critical Point), where $\alpha(S)$ is the ratio of the diameter of a set S of m server sites to the maximum distance between two adjacent server sites. They also constructed a layout of servers where PTCP has a constant competitive ratio while Permutation (or Optimal-Fill) has at least an $\Omega(m)$ competitive ratio.

Online Transportation Problem with a Weaker Adversary. For OTR, models with a weaker adversary have also been well analyzed. Here, we will introduce two previous studies. Kalyanasundaram and Pruhs [12] studied OTR under the weakened adversary model where the adversary has only half as many capacities of each server site as the online algorithm and the length of a request sequence is at most $k/2$. They showed that the greedy algorithm is $\Theta(\min(m, \log k))$-competitive and presented an $O(1)$-competitive algorithm under this model. Chung et al. [5] also studied OTR under another weakened adversary where the adversary has one less capacity of each server site against the online algorithm. Under this model, they presented an $O(\log m)$-competitive deterministic algorithm on an α-HST [5] metric where $\alpha = \Omega(\log m)$ and an $O(\log^3 m)$-competitive randomized algorithm on a general metric.

Randomized Algorithms for Online Metric Matching. There are also many studies on randomized algorithms for OMM. For a general metric space, Meyerson et al. [16] showed the lower bound $\Omega(\log k)$ and the upper bound $O(\log^3 k)$ on the competitive ratio for OMM with k servers. The upper bound was improved to be $O(\log^2 k)$ by Bansal et al. [4]. As can be seen from the above, there still has been a gap between the upper bound $O(\log^2 k)$ and the lower bound $\Omega(\log k)$ for OMM. For doubling metrics, Gupta and Lewi [7] showed an $O(\log k)$-competitive randomized algorithm.

Randomized algorithms for OTR(k, m) have not been studied for a long time, but recently Kalyanasundaram et al. [13] proposed an $O(\log^2 m)$-competitive algorithm.

Stochastic Online Metric Matching. Raghvendra [19] considered OMM under the random arrival model in which the adversary chooses the set of request locations at the start but the arrival order is a permutation chosen uniformly at random from the set of all possible permutations. He proposed an algorithm called *Robust-Matching* and showed that it is $(2H_k - 1)$-competitive and best possible for OMM(k) under the random arrival model, where H_k denotes the k-th harmonic number. Furthermore, Robust-Matching also has the best possible competitive ratio of $2k - 1$ under the normal adversarial model.

Gupta et al. [6] considered OMM under online i.i.d. arrivals. In this model, requests are drawn independently from a known probability distribution over a metric space. They proposed an algorithm called *FAIR-BIAS* and showed that it is $O((\log \log \log k)^2)$-competitive for a general metric space and 9-competitive for a tree metric under this model.

2 Preliminaries

2.1 Definition of Problems

In this section, we define the online metric matching, the online transportation and their variants.

Online Metric Matching

To begin with, we define the online metric matching (denoted by OMM). An instance of OMM is represented by a quadruple $I = (X, d, S, \sigma)$, where X is a set of points, $d : X \times X \to \mathbb{R}_{\geq 0}$ is a distance function on X, S is a finite subset of X, and $\sigma = r_1 \dots r_{|S|} \in X^{|S|}$. An element of S is called a *server* and the t-th entry r_t of σ is called the *request at time t* or *t-th request*.

X, d and S are given to an online algorithm in advance, while requests are given one-by-one from r_1 to $r_{|S|}$. At each step, an online algorithm $\mathcal{A}$ for OMM maintains "assignment" that is initialized to $\emptyset$. When a request r_t is revealed, $\mathcal{A}$ must assign r_t to one of the available servers irrevocably. If r_t is assigned to the server s_t, then the pair (r_t, s_t) is added to the current assignment and the cost $d(r_t, s_t)$ is incurred for this pair. The cost of the assignment is the sum of the costs of all the pairs contained in it. The goal of an online algorithm is to minimize the cost of the final assignment.

We use the following notation on OMM.

(1) $\mathrm{OMM}(k)$ denotes OMM with k servers.

(2) OMM_S denotes OMM where requests arrive at the position of some server, i.e., $X = S$. We simply write an instance I of OMM_S as $I = (S, d, \sigma)$ instead of (S, d, S, σ).

(3) $\mathrm{OMM}_S(k)$ denotes OMM_S with k servers.

Two Metrics on Edge-Weighted Trees

Before presenting online metric matching on a power-of-two tree metric, we first introduce the definitions and notations for the two distance functions used for edge-weighted trees. For an edge-weighted tree T, we use d_T to denote the path distance on $V(T)$, i.e., for any $u, v \in V(T)$, $d_T(u, v) := \sum_{e \in E(P_T(u,v))} w_T(e)$, where $P_T(u, v)$ denotes the unique simple path in T from u to v. In this paper, we define another distance $d_T^{\max}$ on T as follows:

$$d_T^{\max}(u, v) := \max_{e \in E(P_T(u,v))} w_T(e).$$

We call $d_T^{\max}$ the *max-weight distance* on T. It is easy to verify that $d_T^{\max}$ defines a metric on $V(T)$ when each edge has a positive weight. By the above definition, we can observe the following relationship between the path distance and the max-weight distance.

▶ Remark 2. For any edge-weighted tree T and any $u, v \in V(T)$, we have

$$d_T^{\max}(u, v) \le d_T(u, v) \le |E(T)| \cdot d_T^{\max}(u, v),$$

where d_T denotes the path distance on T and $d_T^{\max}$ denotes the max-weight distance on T.

Online Matching on a Power-of-two Tree Metric

Then, we introduce a notion of power-of-two weighted tree and define a special case of OMM_S called online matching on a power-of-two tree metric (denoted by OMT_S^2).

▶ **Definition 3** (Power-of-two Weighted Tree). *Let $T = (V(T), E(T))$ be a tree and $w_T : E(T) \to \mathbb{R}_{\ge 0}$ be a weight function of T. We say that T is a power-of-two weighted tree if for any $e \in E(T)$, there exists a non-negative integer i such that $w_T(e) = 2^i$.*

▶ **Definition 4** (Power-of-two Tree Metric). *We say that a metric space (X, d) is a power-of-two tree metric if there exists a power-of-two weighted tree T such that $V(T) = X$ and $d = d_T$. In this case, we also say that (X, d) is induced by T.*

The online matching on a power-of-two tree metric is a variant of the online metric matching problem where a metric space (X, d) is induced by a power-of-two weighted tree T and a set S of servers is $V(T)$. In other words, an instance of OMT_S^2 is an instance $I = (X, d, S, \sigma)$ of OMM such that $X = S = V(T)$ and $d = d_T$.

For a power-of-two weighted tree T, $\mathrm{OMT}_S^2(T)$ denotes OMT_S^2 where the metric space is induced by T. We simply write an instance I of $\mathrm{OMT}_S^2(T)$ as $I = (T, \sigma)$ instead of $(V(T), d_T, V(T), \sigma)$ for OMM.

Online Transportation Problem

Finally, we define the online transportation problem (denoted by OTR). In this problem, k servers are clustered in specific m $(\le k)$ locations called *server sites*. The number of servers at a server site is called the *capacity* of that site. Assume that each site has a capacity of at least 1. An instance of OTR is represented by a quintuple (X, d, S, c, σ). Here, (X, d) represents the metric space, S represents the set of server sites, $c : S \to \mathbb{N}$ represents the capacity of each site, and $\sigma = r_1 \dots r_k \in X^k$ represents the request sequence[4]. OTR is the same as OMM except that an online algorithm can assign up to $c(s)$ requests to one server site s. In other words, OMM can be viewed as a special case of OTR where $k = m$ and each site has unit capacity. We use OTR(k, m) to denote OTR with k servers and m $(\leq k)$ server sites.

2.2 Notation and Terminology

Let $I = (X, d, S, c, \sigma)$ be any instance of OTR(k, m), r_t be the t-th request of I and $\mathcal{A}$ be any online algorithm for OTR. We use $s_t(I, \mathcal{A})$ to denote the server to which $\mathcal{A}$ assigns r_t when processing I. Let $\mathcal{A}(I)$ be the total cost incurred when $\mathcal{A}$ processes I, i.e.,

$$\mathcal{A}(I) = \sum_{t=1}^{k} d(r_t, s_t(I, \mathcal{A})).$$

Opt denotes the optimal offline algorithm, i.e., Opt knows the entire σ in advance and assigns r_t to $s_t(I, \mathrm{Opt})$ to minimize the total cost $\mathrm{Opt}(I)$. At any step of the execution of an online algorithm, a server site is called *free* if the number of requests assigned to it is less than its capacity, and *full* otherwise. Let $F_t(\mathcal{A}, I)$ be the set of all free server sites just after $\mathcal{A}$ assigns r_t to a server. We say that $\mathcal{A}$ is α-competitive if $\mathcal{A}(I) \leq \alpha \cdot \mathrm{Opt}(I)$ for any instance I of OTR. The competitive ratio $\mathcal{R}(\mathcal{A})$ of $\mathcal{A}$ is defined to be the infimum of α such that $\mathcal{A}$ is α-competitive, i.e., $\mathcal{R}(\mathcal{A}) = \inf\{\alpha : \mathcal{A} \text{ is } \alpha\text{-competitive}\}$. In this paper, we consider only algorithms that, when a request r arrives at the position of a free server site, assign r to that site[5].

For an instance $I = (X, d, S, \sigma)$ of OMM(k), we simply say that a server is *free* when no request is assigned to it, and *full* otherwise. In other respects, we use the same notation and terminology as in OTR.

2.3 Technical Lemmas from Prior Work

In this section, we present two prior results that play crucial roles in achieving our objective of designing an $O(m)$-competitive deterministic algorithm for OTR(k, m). The following theorem claims that if there exists an $O(k)$-competitive algorithm for OMM$_S(k)$, then there also exists an $O(k)$-competitive algorithm for OMM(k).

▶ **Theorem 5** (Meyerson et al. [16]). *If there exists an α-competitive algorithm $\mathcal{A}$ for* OMM$_S$*, then there exists a $(2\alpha + 1)$-competitive algorithm $\mathcal{B}$ for* OMM.

The algorithm $\mathcal{B}$ is designed as follows: When a request arrives, it is first moved to the nearest server site (not necessarily available), and then assigned to a server according to $\mathcal{A}$.

Next, we introduce a class of algorithms for OTR called MPFS (Most Preferred Free Servers) and remark that to design an $O(m)$-competitive algorithm for OTR(k, m), it suffices to design an $O(k)$-competitive MPFS algorithm for OMM$_S(k)$.

[4] For OTR, the number of requests is generally set to be at most k. As can be seen from [9, Lemma 2.1], however, the assumption that the number of requests is exactly k has no effect on the competitive analysis.

[5] Algorithms that do not satisfy this condition are known to be not α-competitive for any $\alpha > 0$.

▶ **Definition 6** (MPFS Algorithm [9]). *Let $\mathcal{A}$ be a deterministic online algorithm for* OTR*. We say that $\mathcal{A}$ is an MPFS (most preferred free servers) algorithm if it deals with a request sequence $\sigma = r_1 \dots r_k$ as follows:*

(1) *For each $1 \leq t \leq k$, the priority (with no ties) of all server sites for r_t is determined by only the positions of r_t and all server sites S,*

(2) *$\mathcal{A}$ assigns r_t to a server with the highest priority among all free server sites $F_{t-1}(\mathcal{A}, I)$.*

Let $\mathcal{MPFS}$ be the class of MPFS algorithms for OTR. By its definition, given a new request r and a set $F \subseteq S$ of current free server sites, an MPFS algorithm $\mathcal{A}$ uniquely determines a server s to which r is assigned. We use $s^{\mathcal{A}}(r, F)$ to denote such s, i.e., a server to which $\mathcal{A}$ assigns r when F is a set of free server sites. For any MPFS algorithm, it is immediate that the following remark holds.

▶ Remark 7. Let $\mathcal{A} \in \mathcal{MPFS}$ and $s = s^{\mathcal{A}}(r, F)$. If $s \in F' \subseteq F$, then $s^{\mathcal{A}}(r, F') = s$.

Moreover, the following strong theorem [9] is known for MPFS algorithms.

▶ **Theorem 8** (Harada et al. [9, Corollary 3.10]). *Let $\mathcal{A} \in \mathcal{MPFS}$ and suppose that $\mathcal{A}$ is $\alpha(k)$-competitive for* OMM(k)*, where $\alpha(k)$ is a non-decreasing function of k. Then, for any $m \leq k$, $\mathcal{A}$ is $\alpha(m)$-competitive for* OTR(k, m).

Proof Sketch. For simplicity, consider an instance $I = (X, d, S, c, \sigma)$ where $c(s) \equiv \ell > 1$, i.e., the number of requests and servers is $k = m\ell$. By Hall's theorem, we can "partition" the request sequence $\sigma = r_1 \dots r_k$ into ℓ request sequences $\sigma_i = r_1^i \dots r_m^i$ for $i = 1, \dots, \ell$, such that both $\mathcal{A}$ and Opt assign requests in σ_i to distinct m server sites.

By the definition of the MPFS algorithm, when executing the instance $I_i = (X, d, S, \sigma_i)$ for OMM (i.e., when all server sites have unit capacity), each request is assigned to the same server site as in the original execution of I. Thus, we have $\mathcal{A}(I) = \sum_i \mathcal{A}(I_i)$. Similarly, by optimality, we also have $\mathrm{Opt}(I) = \sum_i \mathrm{Opt}(I_i)$. Therefore, $\max_i \mathcal{A}(I_i)/\mathrm{Opt}(I_i)$ is an upper bound on the competitive ratio of $\mathcal{A}$. This implies that the competitive ratio of $\mathcal{A}$ for OMM dominates the competitive ratio for OTR. ◀

By Theorems 5 and 8, if there exists an $\alpha(k)$-competitive MPFS algorithm for $\mathrm{OMM}_S(k)$, then we easily obtain a $(2\alpha(m) + 1)$-competitive algorithm for OTR(k, m). Note that if $\mathcal{A}$ is an MPFS algorithm, then the algorithm $\mathcal{B}$ shown in Theorem 5 is also an MPFS algorithm.

3 T-strong competitive ratio for Tree Metrics

In this section, we introduce a new concept of T-strong competitive ratio for OMT_S^2 to represent the performance of an algorithm and show that T-strong competitiveness for OMT_S^2 is closely related to standard competitiveness for OMM_S. The most important result in this section is the following theorem. Thanks to this theorem, our goal of designing an $O(m)$-competitive algorithm for OTR(k, m) is reduced to designing a T-strongly $O(k)$-competitive MPFS algorithm for OMT_S^2 with k servers.

▶ **Theorem 9.** *If there exists a T-strongly $\alpha(k)$-competitive MPFS algorithm for OMT_S^2 with k servers, then there exists a $(4\alpha(m) + 1)$-competitive algorithm $\mathcal{B}$ for* OTR(k, m)*, where $\alpha(\cdot)$ is a non-decreasing function.*

We begin with defining T-strong competitiveness. Hereafter, we use $\mathrm{Opt}_T^{\max}$ to denote the optimal offline algorithm where the cost of assigning a request to a server is measured by the max-weight distance on T.

▶ **Definition 10** (T-strong competitive ratio). *Let $I = (T, \sigma)$ be any instance of* OMT_S^2 *and* $\mathrm{Opt}_T^{\max}(I)$ *denote the minimum cost of assigning all requests to servers, where the cost is measured by the max-weight distance* $d_T^{\max}$ *on T. We say that an algorithm $\mathcal{A}$ is T-strongly α-competitive if, for any instance I of* OMT_S^2*, it follows that* $\mathcal{A}(I) \leq \alpha \cdot \mathrm{Opt}_T^{\max}(I)$*, where the cost $\mathcal{A}(I)$ is measured by the path distance d_T on T. In addition, the T-strong competitive ratio of $\mathcal{A}$ is defined to be the infimum of α such that $\mathcal{A}$ is T-strongly α-competitive.*

▶ Remark 11. In the rest of this paper, the cost of an algorithm is generally measured by the path distance, unless we specifically use the notations $\mathrm{Opt}_T^{\max}$ or $d_T^{\max}$.

▶ Remark 12. By Remark 2, we have $\mathrm{Opt}_T^{\max}(I) \leq \mathrm{Opt}(I)$. Therefore, if an algorithm $\mathcal{A}$ is T-strongly α-competitive for $\mathrm{OMT}_S^2(T)$, then $\mathcal{A}$ is also α-competitive for $\mathrm{OMT}_S^2(T)$.

The following theorem presents a method to convert a T-strongly α-competitive algorithm for OMT_S^2 into a 2α-competitive algorithm for OMM_S with a general metric.

▶ **Theorem 13.** *If there exists a T-strongly α-competitive algorithm $\mathcal{A}$ for* OMT_S^2*, then there exists a 2α-competitive algorithm $\mathcal{B}$ for* OMM_S*.*

The proof is given in Theorem 13 of the full version. Here, we only describe the definition of $\mathcal{B}$. $\mathcal{B}$ first constructs a power-of-two weighted tree T by using (S, d) and then simulates $\mathcal{A}$ with input instance $I' = (T, \sigma)$. Now we describe how to construct T by using (S, d). First, we represent the k-point metric space (S, d) as an edge-weighted k-vertex complete graph $K_{S,d}$ in which each edge (u, v) has a weight $d(u, v)$. Let T' be a minimum spanning tree of the $K_{S,d}$. Next, we obtain a power-of-two weighted tree T by adjusting the edge weights of the T' as follows: for each edge $e \in E(T')$, if $2^{i-1} < w_{T'}(e) \leq 2^i$, then $w_T(e) := 2^i$, i.e., $w_T(e) = 2^{\lceil \log_2 w_{T'}(e) \rceil}$.

By Theorems 13, 5 and 8, we obtain Theorem 9. Thus, in the rest of this paper, we will focus on designing an MPFS algorithm for OMT_S^2.

4 New Algorithm: Subtree-Decomposition

In this section, we propose a new MPFS algorithm for OMT_S^2 called Subtree-Decomposition (SD). In the subsequent sections, we use $\mathcal{A}^*$ to denote the SD algorithm unless otherwise specified.

4.1 Notation for Trees

To begin with, we describe the notation for trees used in this paper. Let $T = (V(T), E(T))$ be a power-of-two weighted tree and d_T be the path distance on T. Suppose that T is rooted at $\rho \in V(T)$. For T and any subtree U of T, we use the following notation.

- For $v \in V(T)$, $\mathrm{par}(v)$ denotes the parent of v.
- $\rho(U)$ denotes the closest vertex in $V(U)$ to the root ρ. We simply refer to $\rho(U)$ as the *root* of U.
- $w_U^{\max}$ denotes the maximum weight in U, i.e., $w_U^{\max} := \max_{(u,v)\in E(U)} d_T(u, v)$.
- $E_{\max}(U)$ denotes the set of heaviest edges in U, i.e.,

$$E_{\max}(U) := \{(u, v) \in E(U) : d_T(u, v) = w_U^{\max}\}.$$

The notations $v \in V(T)$ and $e \in E(T)$ are sometimes abbreviated to $v \in T$ and $e \in T$ respectively when context makes it clear.

4.2 Definition of Subtree-Decomposition

Then, we define the SD algorithm denoted by $\mathcal{A}^*$. Before presenting the formal definition, we describe the intuitive behavior of SD.

Intuitive behavior. Let $T = (V(T), E(T))$ be a given power-of-two weighted tree rooted at an arbitrary vertex $\rho \in V(T)$. For each vertex v, let $W_i(v) \subseteq V(T)$ be the set of vertices reachable from vertex v only through edges with weights at most 2^i. When a new request arrives at vertex v, for $i = 1, 2, \ldots$, SD performs a DFS starting from v to search for a free server in $W_i(v)$. SD then assigns the request to the first free server located by a DFS.

Decomposition of a Tree. Next, to describe the algorithm more precisely and simplify the inductive analysis, we redefine the above algorithm recursively. To this end, we decompose a given power-of-two weighted tree $T = (V(T), E(T))$ into subtrees (hence the algorithm's name) as follows:

Let $\rho^{(1)} := \rho$ and we define subtrees $T^{(1)}$ and $T^{(2)}$ as follows: We arbitrarily choose $\rho^{(2)}$, one of the children of the root ρ $(= \rho^{(1)})$, and consider a graph $T \setminus (\rho^{(1)}, \rho^{(2)})$ obtained by removing edge $(\rho^{(1)}, \rho^{(2)})$ from T. Note that $T \setminus (\rho^{(1)}, \rho^{(2)})$ consists of two subtrees. Let $T^{(1)}$ be the subtree that contains $\rho^{(1)}$ and $T^{(2)}$ be the other subtree that contains $\rho^{(2)}$.

Next, let $\rho_0 := \rho$ and we define subtrees $T_0, T_1, \ldots$ as follows: Let T_0 be the maximal subtree of T that contains the root ρ $(= \rho_0)$ and does not contain any heaviest edge in $E_{\max}(T)$. We use $\rho_1, \ldots, \rho_l$ to denote the vertices v such that $v \notin T_0$ and $\mathrm{par}(v) \in T_0$. Note that $(\rho_i, \mathrm{par}(\rho_i)) \in E_{\max}(T)$ by the definition of T_0. For $i = 1, \ldots, l$, let T_i be the subtree of T rooted at ρ_i. By the definition of $\{T_i\}_{i=0}^{l}$, $\{V(T_i)\}_{i=0}^{l}$ is a partition of $V(T)$. Then, for any vertex v, there uniquely exists a subtree T_i such that $v \in T_i$.

Recursive Definition. Now we are ready to describe the formal and recursive definition of SD. For the base case where $|V(T)| = 1$, SD assigns each request to the unique server. For $j = 1, 2$, let $\mathcal{A}^*_{(j)}$ be the SD algorithm for $T^{(j)}$ rooted at $\rho^{(j)}$ and for $i = 0, \ldots, l$, let $\mathcal{A}^*_i$ be the SD algorithm for T_i rooted at ρ_i. $\mathcal{A}^*$ has two phases: When all servers in T_0 are full, Phase 1 terminates and Phase 2 follows.

Let r be a new request and $F \subseteq V(T)$ be a set of free servers. $\mathcal{A}^*$ assigns r by using $\mathcal{A}^*_{(j)}$ for $j = 1, 2$ and $\mathcal{A}^*_i$ for $i = 1, \ldots, l$.

Phase 1: There is at least one free server in $\boldsymbol{T_0}$. Assume that $r \in T_i$ for some $i = 0, \ldots, l$. If there exists a free server in T_i, then $\mathcal{A}^*$ assigns r to a server in T_i according to $\mathcal{A}^*_i$. Otherwise, $\mathcal{A}^*$ assigns r to a server in T_0 according to $\mathcal{A}^*_0$ by regarding $\mathrm{par}(\rho_i)$ as a new request and $F \cap V(T_0)$ as a set of free servers.

Phase 2: There is no free server in $\boldsymbol{T_0}$. Assume that $r \in T^{(j)}$ for some $j = 1, 2$ and $r \in T_i$ for some $i = 0, \ldots, l$. If there exists a free server in T_i, then $\mathcal{A}^*$ assigns r to a server in T_i according to $\mathcal{A}^*_i$. Otherwise, if there exists a free server in $T^{(j)}$, then $\mathcal{A}^*$ assigns r to a server in $T^{(j)}$ according to $\mathcal{A}^*_{(j)}$. Otherwise, $\mathcal{A}^*$ assigns r to a server in $T^{(3-j)}$ according to $\mathcal{A}^*_{(3-j)}$ by regarding $\rho^{(3-j)}$ as a new request and $F \cap V(T^{(3-j)})$ as a set of free servers.

We establish that SD is an MPFS algorithm and that the processing time of SD per request is $O(m)$ through the following propositions. The proof of Proposition 14 is provided in Proposition 15 of the full version.

▶ **Proposition 14.** *Subtree-Decomposition is an MPFS algorithm.*

▶ **Proposition 15.** *Subtree-Decomposition processes each request in $O(m) = O(|V(T)|)$ time.*

Proof. The proof is by induction on $|V(T)|$. For the base case $|V(T)| = 1$, each request is obviously processed in $O(1)$ time. For the inductive step, suppose that the processing time of $\mathcal{A}_i^*$ per request is $O(|V(T_i)|)$ for $i = 0, \dots, l$ and that of $\mathcal{A}_{(j)}^*$ is $O(|V(T^{(j)})|)$ for $j = 1, 2$. If SD is in Phase 1 and a request r is in T_0, then the processing time is $O(|V(T_0)|)$ by the induction hypothesis. If SD is in Phase 1 and a request r is in T_i for some $i = 1, \dots, l$, then the processing time is $O(|V(T_0)|) + O(|V(T_i)|) = O(|V(T)|)$ by the induction hypothesis. Otherwise, i.e., SD is in Phase 2, the processing time is $O(|V(T^{(1)})|) + O(|V(T^{(2)})|) = O(|V(T)|)$ by the induction hypothesis. Thus, the proposition holds. ◀

5 Hybrid Algorithm

In this section, we introduce the notion of hybrid algorithms and their properties. This idea was initiated by Gupta and Lewi [7] and very useful in analyzing the competitive ratio of an MPFS algorithm. Note that all definitions and results in this section are applicable to any (not necessarily SD) MPFS algorithm. To begin with, we define hybrid algorithms.

▶ **Definition 16** (Hybrid Algorithm). *Let $\mathcal{A} \in \mathcal{MPFS}$. For a positive integer t_{d} and a server a_{d}, consider an algorithm $\mathcal{H} = (\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}})$ that assigns requests for OMM_S as follows:*
(1) *The first $t_{\mathsf{d}} - 1$ requests are assigned according to $\mathcal{A}$,*
(2) *if a_{d} is free just before the t_{d}-th request is revealed, then the t_{d}-th request is assigned to a_{d} and the subsequent requests are assigned according to $\mathcal{A}$, and*
(3) *if a_{d} is full just before the t_{d}-th request is revealed, then the t_{d}-th and subsequent requests are assigned according to $\mathcal{A}$.*
We call $\mathcal{H} = (\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}})$ a hybrid algorithm *of $\mathcal{A}$ with* decoupling time *t_{d} and* decoupling server *a_{d}.*

One can observe that $\mathcal{A}$ and $\mathcal{H}$ output different assignments when decoupling server a_{d} is free at decoupling time t_{d} and $\mathcal{A}$ assigns the t_{d}-th request to a server other than a_{d}, i.e., $a_{\mathsf{d}} \in F_{t_{\mathsf{d}}}(\mathcal{A}, I)$. The purpose of considering the hybrid algorithm is to reduce the evaluation of the competitive ratio of a certain MPFS algorithm $\mathcal{A}$ to evaluating the difference in cost between $\mathcal{A}$ and its hybrid algorithm $\mathcal{H}$. Therefore, we are only interested in cases where the original MPFS algorithm $\mathcal{A}$ and its hybrid algorithm $\mathcal{H}$ output different assignments. Motivated by this insight, we give the following definition of hybrid instances.

▶ **Definition 17** (Hybrid Instance). *Let $I = (S, d, \sigma)$ be any instance of OMM_S and $\mathcal{H} = (\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}})$ be a hybrid algorithm of $\mathcal{A} \in \mathcal{MPFS}$. We say that I is* valid *with respect to $\mathcal{H}$ if $\mathcal{A}$ and $\mathcal{H}$ output different assignments when processing I, i.e., $a_{\mathsf{d}} \in F_{t_{\mathsf{d}}}(\mathcal{A}, I)$ and* invalid *otherwise. We refer to a pair $H = (\mathcal{H}, I)$ as a* hybrid instance *of $\mathcal{A}$ if $\mathcal{H}$ is a hybrid algorithm of $\mathcal{A}$ and I is valid with respect to $\mathcal{H}$.*

The following lemma shows that by evaluating the cost difference between an MPFS algorithm $\mathcal{A}$ and its hybrid algorithm $\mathcal{H}$, we can determine the competitive ratio of $\mathcal{A}$. The proof of Lemma 18 is given in Lemma 19 of the full version.

▶ **Lemma 18.** *The following claims hold for any $\mathcal{A} \in \mathcal{MPFS}$.*
(1) *Suppose that $\mathcal{A}(I) - \mathcal{H}(I) \leq \alpha \cdot d(r_{t_{\mathsf{d}}}, a_{\mathsf{d}})$ for any hybrid instance $H = (\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (S, d, \sigma))$ of $\mathrm{OMM}_S(k)$. Then, $\mathcal{A}$ is $(\alpha + 1)$-competitive for $\mathrm{OMM}_S(k)$.*
(2) *Furthermore, if $\mathcal{A}(I) - \mathcal{H}(I) \leq \alpha \cdot d_T^{\max}(r_{t_{\mathsf{d}}}, a_{\mathsf{d}})$ for any hybrid instance $H = (\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (T, \sigma))$ of $\mathrm{OMT}_S^2(T)$, then $\mathcal{A}$ is T-strongly $(\alpha + |E(T)|)$-competitive for $\mathrm{OMT}_S^2(T)$.*

Next, we introduce an important concept called *cavities* [7], which is defined for a hybrid instance $H = (\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (S, d, \sigma))$ of OMM_S. First, consider the moment when $\mathcal{A}$ assigns the t_{d}-th request to $s_{t_{\mathsf{d}}}(I, \mathcal{A})$ and $\mathcal{H}$ assigns it to a_{d}. We regard $\mathcal{A}$ as having an "extra" free server at a_{d} (i.e., a server that is free for $\mathcal{A}$ but full for $\mathcal{H}$). Similarly, we consider $\mathcal{H}$ to have an "extra" free server at $s_{t_{\mathsf{d}}}(I, \mathcal{A})$. The locations of these extra free servers may move as future requests arrive. Eventually, when both $\mathcal{A}$ and $\mathcal{H}$ assign a request to their respective extra free servers, the sets of free servers of $\mathcal{A}$ and $\mathcal{H}$ align, and from that point onward, $\mathcal{A}$ and $\mathcal{H}$ assign a subsequent request to the same server. These extra free servers are referred to as *cavities.*

The following lemma [8] shows that for any hybrid instance $(\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (S, d, \sigma))$, the cavity of $\mathcal{A}$ (i.e., a server that is free for $\mathcal{A}$ but full for $\mathcal{H}$) and the cavity of $\mathcal{H}$ (a server that are free for $\mathcal{H}$ but full for $\mathcal{A}$) are uniquely determined at each time, if they exist. This holds by the property of MPFS algorithms.

▶ **Lemma 19** (Harada and Itoh [8, Lemma 3]). *Let $H = (\mathcal{H}, I)$ be a hybrid instance of $\mathcal{A} \in \mathcal{MPFS}$, where $\mathcal{H} = (\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}})$. Then, there uniquely exist a positive integer t^H_{c} $(\geq t_{\mathsf{d}})$ and sequences of servers $\{h^H_t\}_{t=t_{\mathsf{d}}}^{t^H_{\mathsf{c}}}$ and $\{a^H_t\}_{t=t_{\mathsf{d}}}^{t^H_{\mathsf{c}}}$ such that*
(1) *$F_t(\mathcal{H}, I) \setminus F_t(\mathcal{A}, I) = \{h^H_t\}$ for each $t_{\mathsf{d}} \leq t \leq t^H_{\mathsf{c}}$,*
(2) *$F_t(\mathcal{A}, I) \setminus F_t(\mathcal{H}, I) = \{a^H_t\}$ for each $t_{\mathsf{d}} \leq t \leq t^H_{\mathsf{c}}$ and*
(3) *$F_t(\mathcal{A}, I) = F_t(\mathcal{H}, I)$ for each $t^H_{\mathsf{c}} + 1 \leq t$.*

We call t^H_{c} the *coupling time* of H and refer to h^H_t (resp. a^H_t) as *$\mathcal{H}$-cavity* (resp. *$\mathcal{A}$-cavity*) of H at time t. $\mathcal{H}$-cavities and $\mathcal{A}$-cavities are collectively called *cavities.* In particular, $h^H_{t_{\mathsf{d}}}$ (resp. $a^H_{t_{\mathsf{d}}}$) is called the *first $\mathcal{H}$-cavity* (resp. *the first $\mathcal{A}$-cavity*) of H.

For simplicity, t^H_{c}, h^H_t and a^H_t are abbreviated as t_{c}, h_t and a_t respectively when H is clear from the context. We use $\mathrm{Cav}_{\mathcal{H}}(H)$ and $\mathrm{Cav}_{\mathcal{A}}(H)$ to denote the sets of $\mathcal{H}$-cavities and $\mathcal{A}$-cavities of H respectively, i.e., $\mathrm{Cav}_{\mathcal{H}}(H) := \{h_t\}_{t=t_{\mathsf{d}}}^{t_{\mathsf{c}}}$, $\mathrm{Cav}_{\mathcal{A}}(H) := \{a_t\}_{t=t_{\mathsf{d}}}^{t_{\mathsf{c}}}$. In addition, let $\mathrm{Cav}(H) := \mathrm{Cav}_{\mathcal{H}}(H) \cup \mathrm{Cav}_{\mathcal{A}}(H)$. By the definition of cavities, it is easy to see that the first $\mathcal{A}$-cavity is the decoupling server of $\mathcal{H}$ and the first $\mathcal{H}$-cavity is $s_{t_{\mathsf{d}}}(I, \mathcal{A})$.

We summarize the properties of cavities and assignments in the following proposition. Intuitively, this proposition asserts the following:

- At each time, either the $\mathcal{H}$-cavity or the $\mathcal{A}$-cavity moves, or neither moves.
- If at time t, either one of the $\mathcal{H}$-cavity or the $\mathcal{A}$-cavity moves, then we can determine the servers to which $\mathcal{H}$ and $\mathcal{A}$ assigned requests at time t.

▶ **Proposition 20** (Harada and Itoh [8, Proposition 1]). *Let $H = (\mathcal{H}, I)$ be a hybrid instance of $\mathcal{A} \in \mathcal{MPFS}$, where $\mathcal{H} = (\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}})$. Then, the following properties hold:*
(1) *$h_{t-1} = h_t$ or $a_{t-1} = a_t$ for each $t_{\mathsf{d}} + 1 \leq t \leq t_{\mathsf{c}}$,*
(2) *If $h_{t-1} \neq h_t$, then r_t is assigned to h_t by $\mathcal{A}$ and to h_{t-1} by $\mathcal{H}$ for each $t_{\mathsf{d}} + 1 \leq t \leq t_{\mathsf{c}}$,*
(3) *If $a_{t-1} \neq a_t$, then r_t is assigned to a_{t-1} by $\mathcal{A}$ and to a_t by $\mathcal{H}$ for each $t_{\mathsf{d}} + 1 \leq t \leq t_{\mathsf{c}}$,*
(4) *If $h_{t-1} = h_t$ and $a_{t-1} = a_t$, then $\mathcal{A}$ and $\mathcal{H}$ assign r_t to the same server and*
(5) *$r_{t_{\mathsf{c}}+1}$ is assigned to $a_{t_{\mathsf{c}}}$ by $\mathcal{A}$ and to $h_{t_{\mathsf{c}}}$ by $\mathcal{H}$.*

As mentioned before, we aim to upper bound the difference in cost between an MPFS algorithm $\mathcal{A}$ and its hybrid algorithm $\mathcal{H}$. To this end, the hybrid cycle defined in the following definition plays a crucial role since the length of the hybrid cycle provides the upper bound on the difference in cost between $\mathcal{A}$ and $\mathcal{H}$ (Lemma 22). For simplicity, we use the following notation for a distance function $d : X \times X \to \mathbb{R}_{\geq 0}$ and $x_1, \ldots, x_n \in X$: $\mathring{d}(x_1, \ldots, x_n) := d(x_1, x_n) + \sum_{i=1}^{n-1} d(x_i, x_{i+1})$.

▶ **Definition 21** (Hybrid Cycle). *Let $H = (\mathcal{H}, I)$ be a hybrid instance of $\mathcal{A} \in \mathcal{MPFS}$ for OMM_S, where $\mathcal{H} = (\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}})$ and $I = (S, d, \sigma)$. We refer to the cycle $h_{t_{\mathsf{d}}} \to \cdots \to h_{t_{\mathsf{c}}} \to a_{t_{\mathsf{c}}} \to \cdots \to a_{t_{\mathsf{d}}} \to h_{t_{\mathsf{d}}}$ as the* hybrid cycle *of H and define the* length *of the hybrid cycle to be*

$$\mathring{d}(H) := \mathring{d}(h_{t_{\mathsf{d}}}, \ldots, h_{t_{\mathsf{c}}}, a_{t_{\mathsf{c}}}, \ldots, a_{t_{\mathsf{d}}})$$
$$= d(h_{t_{\mathsf{d}}}, a_{t_{\mathsf{d}}}) + d(h_{t_{\mathsf{c}}}, a_{t_{\mathsf{c}}}) + \sum_{t=t_{\mathsf{d}}+1}^{t_{\mathsf{c}}} \left(d(h_{t-1}, h_t) + d(a_{t-1}, a_t)\right).$$

Applying Gupta and Lewi's method [7] to MPFS algorithms yields the following lemma and its corollary. The corollary implies that we can analyze the competitive ratio of MPFS algorithms by evaluating the length of the hybrid cycle.

▶ **Lemma 22.** *Let $\mathcal{A} \in \mathcal{MPFS}$ and $H = (\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (S, d, \sigma))$ be any hybrid instance of $\mathrm{OMM}_S(k)$. Then, the difference in cost between $\mathcal{A}$ and $\mathcal{H}$ is at most the length of the hybrid cycle, i.e., $\mathcal{A}(I) - \mathcal{H}(I) \leq \mathring{d}(H)$.*

The proof of Lemma 22 can be found in Lemma 23 of the full version. By Lemmas 18 and 22, we immediately have the following corollary.

▶ **Corollary 23.** *The following claims hold for any $\mathcal{A} \in \mathcal{MPFS}$.*

(1) *If $\mathring{d}(H) \leq \alpha \cdot d(r_{t_{\mathsf{d}}}, a_{\mathsf{d}})$ for any hybrid instance $H = (\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (S, d, \sigma))$ of $\mathrm{OMM}_S(k)$, then $\mathcal{A}$ is $(\alpha + 1)$-competitive for $\mathrm{OMM}_S(k)$.*

(2) *If $\mathring{d}_T(H) \leq \alpha \cdot d_T^{\max}(r_{t_{\mathsf{d}}}, a_{\mathsf{d}})$ for any hybrid instance $H = (\mathcal{H}, I) = ((\mathcal{A}, t_{\mathsf{d}}, a_{\mathsf{d}}), (T, \sigma))$ of $\mathrm{OMT}_S^2(T)$, then $\mathcal{A}$ is T-strongly $(\alpha + |E(T)|)$-competitive for $\mathrm{OMT}_S^2(T)$.*

By this corollary, we aim to obtain an inequality of the form $\mathring{d}(H) \leq \alpha \cdot d(r_{t_{\mathsf{d}}}, a_{\mathsf{d}})$, and focus on hybrid instances that have a certain first $\mathcal{H}$-cavity and $\mathcal{A}$-cavity.

6 Competitive analysis of Subtree-Decomposition

In this section, we derive the upper bound on the competitive ratio of the SD algorithm $\mathcal{A}^*$. To this end, the following lemma is important.

▶ **Lemma 24.** *Let $H = (\mathcal{H}, I) = ((\mathcal{A}^*, t_{\mathsf{d}}, a_{\mathsf{d}}), (T, \sigma))$ be any hybrid instance for $\mathrm{OMT}_S^2(T)$ and let T_H be the minimal subtree of T that contains all cavities of H. Then, we have*

$$\mathring{d}_T(H) \leq 2|E(T_H)| w_{T_H}^{\max}.$$

Note that Lemma 24 presents a simplified form of Lemma 35 of the full version, and the proof can be found in Appendix J of the full version.

In what follows, we introduce two propositions and show that $\mathcal{A}^*$ is T-strongly $3|E(T)|$-competitive for $\mathrm{OMT}_S^2(T)$. The proofs can be found in Propositions 36 and 37 of the full version.

▶ **Proposition 25.** *Consider $\mathcal{A}^*$ for $\mathrm{OMT}_S^2(T)$. Let s, s' be any vertex of T such that $s \neq s'$. For any $r \in V(T)$, if $\mathcal{A}^*$'s priority of s for r is higher than that of s', then $d_T(r, s') \geq d_T^{\max}(r, s') \geq d_T^{\max}(s, s')$.*

▶ **Proposition 26.** *Let $H = ((\mathcal{A}^*, t_{\mathsf{d}}, a_{\mathsf{d}}), (T, \sigma))$ be a hybrid instance for $\mathrm{OMT}_S^2(T)$. Then, $d_T^{\max}(h_{t_{\mathsf{d}}}, a_{\mathsf{d}}) = w_{T_H}^{\max}$.*

▶ **Theorem 27.** *Subtree-Decomposition is T-strongly $(3k - 3)$-competitive for OMT_S^2 with k servers.*

Proof. Let $H = ((\mathcal{A}^*, t_{\mathsf{d}}, a_{\mathsf{d}}), (T, \sigma))$ be any hybrid instance for $\mathrm{OMT}_S^2(T)$. By Lemma 24 and Proposition 26, we have $\mathring{d}_T(H) \le 2|E(T_H)|w_{T_H}^{\max} = 2|E(T_H)|d_T^{\max}(h_{t_{\mathsf{d}}}, a_{\mathsf{d}})$. Since $\mathcal{A}^*$ assigns $r_{t_{\mathsf{d}}}$ to not a_{d} but $h_{t_{\mathsf{d}}}$, the $\mathcal{A}^*$'s priority of $h_{t_{\mathsf{d}}}$ for $r_{t_{\mathsf{d}}}$ is higher than that of a_{d}. Therefore, by Proposition 25, we have $d_T^{\max}(h_{t_{\mathsf{d}}}, a_{\mathsf{d}}) \le d_T^{\max}(r_{t_{\mathsf{d}}}, a_{\mathsf{d}})$ and then

$$\mathring{d}_T(H) \le 2|E(T_H)|d_T^{\max}(r_{t_{\mathsf{d}}}, a_{\mathsf{d}}) \le 2|E(T)|d_T^{\max}(r_{t_{\mathsf{d}}}, a_{\mathsf{d}}).$$

By (2) of Corollary 23, it follows that $\mathcal{A}^*$ is T-strongly $3|E(T)|$-competitive for $\mathrm{OMT}_S^2(T)$. By substituting $|E(T)| = k - 1$, we can see that $\mathcal{A}^*$ is T-strongly $(3k-3)$-competitive for OMT_S^2 with k servers. ◀

By applying Theorem 9, we obtain an $O(m)$-competitive algorithm for $\mathrm{OTR}(k, m)$.

▶ **Corollary 28.** *For* $\mathrm{OTR}(k, m)$*, there exists a deterministic* $(12m-11)$*-competitive algorithm.*

In fact, with a more careful analysis, we can demonstrate that there exists a $(4k-3)$-competitive algorithm for $\mathrm{OMM}_S(k)$ and an $(8m-5)$-competitive algorithm for $\mathrm{OTR}(k, m)$.

▶ **Theorem 29.** *There exists a deterministic* $(4k-3)$*-competitive algorithm for* $\mathrm{OMM}_S(k)$.

Proof. Fix any instance $I = (S, d, \sigma)$ of $\mathrm{OMM}_S(k)$ and Let $\mathcal{B}^*$ be the algorithm obtained by transforming SD using the method in Theorem 13. Recall that $\mathcal{B}^*$ first constructs a power-of-two weighted tree T by using the given metric space (S, d) and then simulates SD with an instance $I' = (T, \sigma)$. We use the following claim which establishes the relationships between the two types of metrics on T and the original metric. The proof is provided in Claim 14 of the full version.

▷ Claim 30. Let (S, d, σ) be any instance for $\mathrm{OMM}_S(k)$ and T be a power-of-two weighted tree constructed by $\mathcal{B}^*$. Then, for any two different points $u, v \in S$, $d_T^{\max}(u, v) < 2d(u, v) \le 2d_T(u, v)$.

Let $H = ((\mathcal{B}^*, t_{\mathsf{d}}, a_{\mathsf{d}}), (S, d, \sigma))$ be a hybrid instance of $\mathcal{B}^*$ for $\mathrm{OMM}_S(k)$ and $H' = ((\mathcal{A}^*, t_{\mathsf{d}}, a_{\mathsf{d}}), (T, \sigma))$ be a hybrid instance of $\mathcal{A}^*$ for $\mathrm{OMT}_S^2(T)$. Then, we have

$$\begin{aligned}\mathring{d}(H) &\le \mathring{d}_T(H') \le 2E_w(T_{H'}) = 2|E(T_{H'})|w_{T_{H'}}^{\max} \\ &\le 2|E(T)|d_T^{\max}(r_{t_{\mathsf{d}}}, a_{\mathsf{d}}) \le 4|E(T)|d(r_{t_{\mathsf{d}}}, a_{\mathsf{d}}) = (4k-4)d(r_{t_{\mathsf{d}}}, a_{\mathsf{d}}),\end{aligned}$$

where the first and fourth inequality is due to Claim 30, the second inequality is due to Lemma 24 and the third inequality is due to Proposition 26 and $E(T_{H'}) \subseteq E(T)$. By (1) of Corollary 23, this implies that $\mathcal{B}^*$ is $(4k-3)$-competitive. ◀

By applying Theorems 5 and 8, we obtain an $(8m-5)$-competitive algorithm for $\mathrm{OTR}(k, m)$.

▶ **Theorem 31.** *There exists a deterministic* $(8m-5)$*-competitive algorithm for* $\mathrm{OTR}(k, m)$.

7 Conclusion

In this paper, we addressed the online transportation problem $\mathrm{OTR}(k, m)$ with k servers located at m server sites.

We first defined the T-strong competitive ratio for a tree metric and demonstrate a generic method for converting an algorithm designed for tree metrics into one for general metric spaces. We believe that this technique can be applied to the open problem of designing an $O(\log k)$-competitive randomized algorithm for $\mathrm{OMM}(k)$ and so on.

We then introduced a new algorithm called Subtree-Decomposition and proved that it is T-strongly $(3k-3)$-competitive for $\mathrm{OMT}_S^2(T)$ with k servers (in Theorem 27). We also demonstrated the existence of $(4k-3)$-competitive algorithms for $\mathrm{OMM}_S(k)$ (in Theorem 29) and $(8m-5)$-competitive algorithms for $\mathrm{OTR}(k,m)$ (in Theorem 31). For any deterministic algorithm of $\mathrm{OTR}(k,m)$, this upper bound on the competitive ratio is tight up to a constant factor with respect to its dependence on the number of server sites. We conjecture that there exists an MPFS algorithm for $\mathrm{OTR}(k,m)$ that achieves the matching upper bound of $2m-1$.

References

1 Abu Reyan Ahmed, Md. Saidur Rahman, and Stephen G. Kobourov. Online facility assignment. *Theor. Comput. Sci.*, 806:455–467, 2020. doi:10.1016/J.TCS.2019.08.011.

2 Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. A $o(n)$-competitive deterministic algorithm for online matching on a line. In *Approximation and Online Algorithms - 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers*, volume 8952 of *Lecture Notes in Computer Science*, pages 11–22. Springer, 2014. doi:10.1007/978-3-319-18263-6_2.

3 Antonios Antoniadis, Carsten Fischer, and Andreas Tönnis. A collection of lower bounds for online matching on the line. In *LATIN 2018: Theoretical Informatics: 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings 13*, pages 52–65. Springer, 2018. doi:10.1007/978-3-319-77404-6_5.

4 Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Naor. An $O(\log^2 k)$-competitive algorithm for metric bipartite matching. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 522–533. Springer, 2007. doi:10.1007/978-3-540-75520-3_47.

5 Christine Chung, Kirk Pruhs, and Patchrawat Uthaisombut. The online transportation problem: On the exponential boost of one extra server. In *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, volume 4957 of *Lecture Notes in Computer Science*, pages 228–239. Springer, 2008. doi:10.1007/978-3-540-78773-0_20.

6 Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.67.

7 Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, volume 7391 of *Lecture Notes in Computer Science*, pages 424–435. Springer, 2012. doi:10.1007/978-3-642-31594-7_36.

8 Tsubasa Harada and Toshiya Itoh. Online facility assignment for general layout of servers on a line. In *Combinatorial Optimization and Applications - 16th International Conference, COCOA 2023, Hawaii, HI, USA, December 15-17, 2023, Proceedings, Part II*, volume 14462 of *Lecture Notes in Computer Science*, pages 310–322. Springer, 2023. doi:10.1007/978-3-031-49614-1_23.

9 Tsubasa Harada, Toshiya Itoh, and Shuichi Miyazaki. Capacity-insensitive algorithms for online facility assignment problems on a line. *Discret. Math. Algorithms Appl.*, 16(5):2350057:1–2350057:39, 2024. doi:10.1142/S179383092350057X.

10 Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993. doi:10.1006/JAGM.1993.1026.

11 Bala Kalyanasundaram and Kirk Pruhs. On-line network optimization problems. *Developments from a June 1996 seminar on Online algorithms: the state of the art*, pages 268–280, 1998.

12 Bala Kalyanasundaram and Kirk Pruhs. The online transportation problem. *SIAM J. Discret. Math.*, 13(3):370–383, 2000. doi:10.1137/S0895480198342310.

13 Bala Kalyanasundaram, Kirk Pruhs, and Clifford Stein. A randomized algorithm for online metric b-matching. *Oper. Res. Lett.*, 51(6):591–594, 2023. doi:10.1016/J.ORL.2023.09.002.

14 Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994. doi:10.1016/0304-3975(94)90042-6.

15 Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *Approximation and Online Algorithms, First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003, Revised Papers*, volume 2909 of *Lecture Notes in Computer Science*, pages 179–191. Springer, 2003. doi:10.1007/978-3-540-24592-6_14.

16 Adam Meyerson, Akash Nanavati, and Laura J. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 954–959. ACM Press, 2006. URL: http://dl.acm.org/citation.cfm?id=1109557.1109662.

17 Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 505–515. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.53.

18 Enoch Peserico and Michele Scquizzato. Matching on the line admits no o($\sqrt{\text{log n}}$)-competitive algorithm. *ACM Trans. Algorithms*, 19(3):28:1–28:4, 2023. doi:10.1145/3594873.

19 Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, volume 60 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.APPROX-RANDOM.2016.18.

20 Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, volume 99 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.SOCG.2018.67.