The Online Metric Matching Problem for Doubling Metrics

Anupam Gupta* Kevin Lewi†

Abstract

In the online minimum-cost metric matching problem, we are given a metric space with k servers. Requests arrive in an online fashion and each must be assigned to an as-yet-unmatched server immediately upon arrival. An assigned request incurs a cost equal to the distance from the server to which it was matched, and the goal is to minimize the total cost of the matching. In this paper, we study this problem when the metric space is a line, and also when it is doubling. For both of these settings we give $O(\log k)$-competitive randomized algorithms, which significantly reduces the gap between the current $O(\log^2 k)$-competitive randomized algorithms and the constant-competitive lower bounds known for these settings.

The first two algorithms that we present operate by randomly embedding these metric spaces into HSTs, and then running greedy algorithms on the resulting HSTs. Each algorithm has a different method of randomly picking from among the closest available servers in the HST. The first algorithm picks the closest available server in the HST which is also closest on the original metric. We show that the cost incurred by the algorithm on the line is within a constant factor of the cost incurred by the optimal matching on the HST. Our second algorithm picks randomly from among the closest available servers in the HST, where the selection is based upon how the servers are distributed within the tree. This algorithm is $O(1)$-competitive for the types of HSTs obtained from embedding doubling metrics. The third algorithm that we discuss is a very simple randomized algorithm that does not use tree embeddings, yet we are still able to show that it is $O(\log k)$-competitive on the line.

1 Introduction

In the online minimum-cost metric matching problem, the input is a metric space (V, d) with k pre-specified *servers* $S \subseteq V$. The requests $R = r_1, r_2, \ldots, r_k$ (with each $r_i \in V$) arrive online one-by-one; upon arrival each request must be immediately and irrevocably matched to an as-yet-unmatched server. The cost of matching request r to server $\pi_r \in S$ is the distance $d(r, \pi_r)$ in the underlying metric space. The goal is to match requests to servers so as to minimize the total cost of the matching produced, which is simply the sum of the costs of individually matching each request. As usual, we study the problem in the framework of competitive analysis, comparing the cost of our algorithm's matching to the cost of the best offline matching from R to S. (This minimum cost bipartite perfect matching problem can be easily solved offline.)

The online problem was independently introduced in the early 1990's by Kalyanasundaram and Pruhs [KP93], and by Khuller, Mitchell, and Vazirani [KMV94]. Both papers gave deterministic

*Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213.

†Stanford University. Work done when at the Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213.

$2k-1$-competitive algorithms, which is the best possible upper bound on the competitive ratio even when the metric is a star with k leaves with the servers at the leaves. For the star example, any *randomized* algorithm must be $\Omega(H_k)$-competitive, and a randomized greedy algorithm based on the airplane seat problem is indeed $O(H_k)$-competitive, where H_k is the k^{th} harmonic number. Indeed, one can conjecture that $O(\log k)$ is the right answer, but this still remains open. In 2006, Meyerson et al. [MNP06] showed that the randomized greedy algorithm, which assigns to a uniformly random closest server, is $O(\log k)$-competitive when the metric is a α-hierarchically well-separated tree (α-HST) with suitably large separation α between levels, namely with $\alpha = \Omega(\log k)$. This implies an $O(\log^3 k)$-competitive randomized algorithm for general metrics using randomized embeddings into HSTs [FRT03]. Subsequently, Bansal et al. [BBGN07] gave a different algorithm which is $O(\log k)$-competitive on 2-HSTs, resulting in a $O(\log^2 k)$ competitive algorithm on general metrics. It remains an interesting problem to close the gap between $O(\log^2 k)$ and $\Omega(\log k)$ for general metrics.

The gap is even more egregious when we consider natural special classes of metrics such as the line, or "low-dimensional" doubling metrics. For points on the line, the best *deterministic* lower bound is only 9.001 [FHK05] (with the randomized lower bound being even weaker), and no algorithms better than those that apply to general metrics are known for the line or for doubling metrics, both in the deterministic and randomized settings.

1.1 Our Results

In this paper, we make progress on randomized algorithms for restricted classes of metrics. In particular, we show $O(\log k)$-competitive randomized algorithms for the online metric matching problem on the line metric and on doubling metrics.

Our first algorithm *HST-greedy* is a simple algorithm for the line metric. It picks a random distance-preserving binary 2-HST T, and then for each request, it looks at the set of available servers that are closest in the tree and maps the request to the server which is closest along the line. Consequently, it uses both the line and tree distances for the assignment. (One can also consider the request as randomly choosing between the servers closest on its left and right on the line, but the randomness is based on the structure tree, and hence is biased and dependent on previous assignments.)

Theorem 1.1 *The randomized algorithm* HST-greedy *is $O(\log k)$-competitive for metric matching on the line.*

The analysis is somewhat unusual: for a fixed binary HST, it shows that the total cost of the online algorithm *when measured along the line* is bounded by a constant times the optimal matching's total cost—*but now, the optimal matching's cost is measured along this fixed HST*. Theorem 1.1 now follows, since by picking a random HST embedding, the latter quantity is $O(\log k)$ times the optimal matching's cost on the line, in expectation.

We then move to our second algorithm *Random-Subtree*, which generalizes to the broader class of doubling metrics. It is even simpler—for the line metric, it just randomly embeds the line into a 2-HST, and then runs a certain randomized greedy algorithm on this new instance. At first glance, using a 2-HST seems bad, since Meyerson et al. gave examples showing that their randomized greedy algorithm requires a large separation α in the HST. However, we show we can avoid the lower bound by (a) using the fact that the bad examples require large degrees, whereas HSTs

obtained from the line and doubling metrics have small degree, and (b) altering the randomized greedy algorithm slightly (in a way we will soon describe). At a high level, we show that if a metric can be embedded into an α-HST where each vertex has at most Δ children, our randomized algorithm is $O(H_\Delta/\epsilon)$-competitive on such an HST, as long as $\alpha \geq (1+\epsilon)H_\Delta$. (See Theorem 4.1 for a precise statement.) Since all doubling metrics admit such embeddings, this gives us the following result.

Theorem 1.2 *The randomized algorithm* Random-Subtree *is $O(\log k)$-competitive for metric matching on doubling metrics, and hence also for the line.*

To get an idea of how we get this improvement in competitive ratio, we need to understand how our randomized greedy algorithm *Random-Subtree* differs from that of [MNP06]. Instead of picking a *uniformly* random one of the available servers closest to the request in the HST as they do, we imagine the following procedure, starting off at the lowest ancestor of the request that contains an available server. Our algorithm repeatedly moves us to a uniformly random subtree of this node that has an available server until we reach a leaf/server. Note that our process biases towards available servers in subtrees with few available servers, and hence results in such subtrees being empty earlier, which in turn results in fewer choices higher up in the tree for future requests. This property improves upon of the algorithm presented in [MNP06] (which biases towards subtrees with more available servers) and hence is crucially used in our analysis; indeed, our potential function refines the one used in [MNP06] precisely in that it utilizes this very property.

Finally, we consider the following algorithm *Harmonic*: letting s_L and s_R be the closest available left and right servers to the current request r, assign r to s_L with probability

$$\frac{1/d(r,s_L)}{1/d(r,s_L)+1/d(r,s_R)} = \frac{d(r,s_R)}{d(s_L,s_R)},$$

and assign r to s_R with the remaining probability.

Theorem 1.3 *The* Harmonic *algorithm is $O(\log \Delta)$-competitive.*

The rest of the paper follows this outline: we present some notation and preliminaries in Section 2. The *HST-greedy* algorithm is presented and analyzed in Section 3, the *Random-Subtree* algorithm in Section 4, and the *Harmonic* algorithm in Section 5. Useful examples are given in Section A.

1.2 Other Related Work

A survey of initial work on the online metric matching problem can be found in [KP98, Section 2.2], along with several conjectures about the problem. The paper gave a lower bound of 9 for deterministic algorithms via a reduction from the so-called cow-path problem; they conjectured this was tight, but this was disproved in [FHK05]. [KP98] also conjectured that the work function algorithm (see, e.g., [KP95]) obtains a constant competitive ratio on the line for the online metric matching problem. This conjecture was also disproved, this time by Koutsoupias and Nanavati [KN03], who showed an $\Omega(\log k)$ lower bound (and an $O(k)$ upper bound) for the work function algorithm. To the best of our knowledge, there is no algorithm known that is currently conjectured to be $O(1)$, in either the randomized or the deterministic cases.

The online *maximum* matching problem has received much attention in recent years due to its connections with ad-auctions. There, the naïve greedy algorithm is 2-competitive, and the randomized algorithm of Karp, Vazirani and Vazirani is $e/(e-1)$-competitive; both can be shown to be tight.

2 Notation and Preliminaries

An instance of the problem is given by a metric (V, d) with servers at $S \subseteq V$, where $|S| = k$. As mentioned in [MNP06], we can assume that all requests also arrive at vertices in S (with only a constant factor loss in the competitive ratio). Hence, in the rest of the paper, we assume that $S = V$, and hence $|V| = |S| = k$. Moreover, we assume there is only one server at each node, as this is only for ease of exposition and the algorithms easily extend to multiple servers at nodes.

An α-HST (Hierarchically well-Separated Tree) is defined as a rooted tree where all edges at depth i have weight c/α^i for some fixed constant c. Here, the edges at depth 0 are those incident to the root, etc. An HST is Δ-ary if each node has at most Δ children. For the case of the line, we assume that the aspect ratio of the points containing the servers (which, recall, is defined as $\frac{\max_{x,y \in S} d(x,y)}{\min_{x,y \in S} d(x,y)}$) is $O(k^3)$; in Appendix B.1, we show this is without loss of generality. This allows us to embed these points into distributions of dominating *binary* 2-HSTs with expected stretch $O(\log k)$. Furthermore, for HSTs that are constructed from the line, we refer to the *width* of a tree as the maximum line-distance between any two points within the tree. For doubling metrics we cannot make such a general assumption on the aspect ratio; however, by suitably guessing the value of Opt and running the HST construction algorithms only for the top $O(\log k)$ levels, one can still give a reduction to the problem on bounded-degree HSTs with only an $O(\log k)$-expected loss. (Details in Appendix B.1.)

For a node a of a tree, let $T(a)$ represent the subtree rooted at a. Also, define the level of a to be the length of the maximum path from a to a leaf of $T(a)$. When referring to servers to be assigned by requests, we will refer to servers that have not yet been assigned to as available, free, or unassigned. We will use Opt to denote both the optimum matching as well as its cost.

3 An $O(\log k)$ Algorithm for the Line

In this section, we give the details of our first algorithm *HST-greedy*. Recall that this algorithm first picks a random binary 2-HST T that stretches distances in the line by $O(\log k)$ in expectation. It now assigns this request to either the closest free server s_l to its left on the line or the closest free server s_r on its right, whichever is closer in the random HST T. (Since we are dealing with *binary* HSTs, there will be no ties.)

Given a sequence of requests $\sigma = r_1, r_2, \ldots, r_k$ appearing online, we can say that the *HST-greedy* algorithm matches each request r_i to a distinct server $g_\sigma(r_i)$ as follows: for the request r_i, let a_i denote the lowest ancestor of r_i in the binary HST such that the subtree $T(a_i)$ (rooted at a_i) contains a free server. Assign r_i to the free server in $T(a_i)$ that is closest to r_i along the line; this server is called $g_\sigma(r_i)$. If we denote by d_L the metric for the line and by d_T the metric for the HST, the main theorem of this section is the following:

Theorem 3.1 *For any choice of the binary 2-HST T and any request sequence $\sigma = r_1, r_2, \ldots, r_k$, the* HST-greedy *algorithm outputs a matching g_σ such that for any matching f_σ,*

$$\sum_{i=1}^{k} d_L(r_i, g_\sigma(r_i)) \leq O(1) \cdot \sum_{i=1}^{k} d_T(r_i, f_\sigma(r_i)),$$

If we then consider f_σ to be the optimal matching for the line and pick T with expected stretch $O(\log k)$, we get that $E[\sum_{i=1}^k d_T(r_i, f_\sigma(r_i))] \leq O(\log k) \cdot \sum_{i=1}^k d_L(r_i, f_\sigma(r_i)) = O(\log k) \cdot \mathsf{Opt}$, which proves Theorem 1.1.

3.1 Analysis via a "Hybrid" Algorithm

To prove Theorem 3.1, fix the binary 2-HST T, and denote the matching produced by the *HST-greedy* algorithm (the algorithm which we denote by G) on a request sequence σ by g_σ. Now consider a "hybrid" algorithm (denoted by H) that matches the request r_1 to an arbitrary server s_1, and then runs the *HST-greedy* algorithm, G, on the remaining requests in σ. Let the matching produced by H be called h_σ; this matching depends on the choice of s_1.

Lemma 3.2 (Hybrid Lemma) *There is a universal constant λ such that for any fixed binary 2-HST T, any set of servers S on the line, for any request sequence $\sigma = r_1, \ldots, r_k$, and for any choice of assignment $r_1 \to s_1$,*

$$\sum_{i=1}^{k} d_L(r_i, g_\sigma(r_i)) \leq \sum_{i=1}^{k} d_L(r_i, h_\sigma(r_i)) + \lambda \, d_T(r_1, s_1),$$

Note that if g_σ were the *optimal* matching on the line, so that h_σ matches $r_1 \to s_1$ arbitrarily and then finds the optimal matching on the remaining requests, one can easily prove such a claim with the additive term being $2d_L(r_1, s_1)$. We will show that even using the *HST-greedy* algorithm's matching as g_σ satisfies such a property with additive error $\lambda \, d_T(r_1, s_1)$.

Before we prove this lemma, let's use it to prove Theorem 3.1. Given any request sequence σ and matching f_σ, we can define a sequence of hybrid matchings $\{h_\sigma^t\}_{t=0}^k$, where h_σ^t is obtained by matching the first t requests $r_1, \ldots, r_t$ in σ to $f_\sigma(r_1), \ldots, f_\sigma(r_t)$ and the remaining requests $r_{t+1}, \ldots, r_k$ to $g_\sigma(r_{t+1}), \ldots, g_\sigma(r_k)$. Note that h_σ^0 is just the *HST-greedy* matching g_σ, and h_σ^k produces the matching f_σ. Moreover, by ignoring the servers in $\{f_\sigma(r_i) \mid i \leq t\}$ and just considering $r_{t+1}, \ldots, r_k$ as the request sequence, Lemma 3.2 implies

$$\textstyle\sum_{i=t+1}^k d_L(r_i, h_\sigma^t(r_i)) \leq \sum_{i=t+1}^k d_L(r_i, h_\sigma^{t+1}(r_i)) + \lambda \cdot d_T(r_t, f_\sigma(r_t)),$$

since we can regard the assignment $r_t \to f_\sigma(r_t)$ as the arbitrary assignment $r_1 \to s_1$ used in the lemma. Now, by adding $\sum_{i=1}^t d_L(r_i, f_\sigma(r_i))$ to both sides,

$$\textstyle\sum_{i=1}^k d_L(r_i, h_\sigma^t(r_i)) \leq \sum_{i=1}^k d_L(r_i, h_\sigma^{t+1}(r_i)) + \lambda \cdot d_T(r_t, f_\sigma(r_t)).$$

Summing this over all values of t, and using that $h_\sigma^0 = g_\sigma$ and $h_\sigma^k = f_\sigma$, we get

$$\textstyle\sum_{i=1}^k d_L(r_i, g_\sigma(r_i)) \leq \sum_{i=1}^k d_L(r_i, f_\sigma(r_i)) + \lambda \cdot \sum_{i=1}^k d_T(r_t, f_\sigma(r_t)).$$

Finally, since for all pairs of points x, y on the line, $d_L(x, y) \leq d_T(x, y)$, we get $\sum_{i=1}^k d_L(r_i, g_\sigma(r_i)) \leq (\lambda + 1) \cdot \sum_{i=1}^k d_T(r_t, f_\sigma(r_t))$, which proves Theorem 3.1.

3.2 Proof of the Hybrid Lemma

We now prove Lemma 3.2. Here is the high-level idea: in the hybrid algorithm (which we call H) we assign $r_1 \to s_1$ and then run *HST-greedy*. Now if the (unmodified) *HST-greedy* algorithm G assigns $r_1 \to x_1$, then we can imagine G as having an "excess" free server at s_1 (i.e., this is a server available in G but not in H), whereas H has an "excess" free server at x_1 (which is not available in G). Now, as future requests arrive, the locations of the current excess servers in G and H move around until for some request both algorithms assign to their current excess servers—and from then on, both algorithms behave identically. Our proof shows that (a) the cost difference between the two algorithms is at most the sum of the distances between consecutive positions of the excess servers, and that (b) this quantity is itself at most $O(d_T(r_1, s_1))$.

We first make a few simple claims relating the matchings g_σ and h_σ given any initial set of servers S and request sequence σ. To start off, note that for any requests r_i such that $g_\sigma(r_i) = h_\sigma(r_i)$, we can delete the request r_i from the sequence σ and delete the server $g_\sigma(r_i)$ from S to get another server set and request sequence with the same behavior; hence we will assume for the rest of the section that for each $r_i \in \sigma$, $g_\sigma(r_i) \neq h_\sigma(r_i)$. The next lemma shows that the size of the symmetric difference between the available servers in the execution of the *HST-greedy* algorithm G and those of the hybrid algorithm H stays at exactly two (until it becomes zero). Hence it makes sense to refer to the *unique* excess servers in G and H, which we will call G-cavities and H-cavities.

For convenience, we will say that the request r_t is assigned at time t, and we refer to the situation just before this assignment as being at time t^-, and the situation just after as time t^+; note that $(t-1)^+ = t^-$. Also, define $\mathsf{A}_G(t^+)$ (respectively, $\mathsf{A}_H(t^+)$) to be the set of servers not yet assigned by g_σ (respectively, h_σ) at time t^+, immediately after the assignment of request r_t.

Lemma 3.3 *The following facts hold for* $\mathsf{A}_G(t^+)$ *and* $\mathsf{A}_H(t^+)$.

(1) Either $\mathsf{A}_G(t^+) = \mathsf{A}_H(t^+)$*, or* $|\mathsf{A}_G(t^+) \setminus \mathsf{A}_H(t^+)| = 1 = |\mathsf{A}_G(t^+) \setminus \mathsf{A}_H(t^+)|$.

(2) If $\mathsf{A}_H(t^+) \neq \mathsf{A}_G(t^+)$*, define* g_{t^+} *and so that* $\mathsf{A}_G(t^+) \setminus \mathsf{A}_H(t^+) = \{\mathsf{g}_{t^+}\}$*, and* $\mathsf{A}_H(t^+) \setminus \mathsf{A}_G(t^+) = \{\mathsf{h}_{t^+}\}$*. Then,*

(a) either $\mathsf{g}_{t^+} \neq \mathsf{g}_{t^-}$ *or* $\mathsf{h}_{t^+} \neq \mathsf{h}_{t^-}$.

(b) if $\mathsf{g}_{t^+} \neq \mathsf{g}_{t^-}$ *for* $t > 1$*, it holds that* $h_\sigma(r_t) = \mathsf{g}_{t^+}$ *and* $g_\sigma(r_t) = \mathsf{g}_{t^-}$*, and if* $\mathsf{h}_{t^+} \neq \mathsf{h}_{t^-}$*, we have* $g_\sigma(r_t) = \mathsf{h}_{t^+}$ *and* $h_\sigma(r_t) = \mathsf{h}_{t^-}$.

Proof. Suppose $g_\sigma(r_1) = x_1$; recall that $h_\sigma(r_1) = s_1$. If $x_1 = s_1$, then $\mathsf{A}_G(1^+) = \mathsf{A}_H(1^+)$. Now consider the case where $x_1 \neq s_1$. Hence, $\mathsf{A}_G(1^+) \setminus \mathsf{A}_H(1^+) = \{s_1\}$, whereas $\mathsf{A}_H(1^+) \setminus \mathsf{A}_G(1^+) = \{x_1\}$. Let us call the former a "G-cavity" and the latter an "H-cavity". Now, inductively assume the claim is true at time t^-, just before assigning r_t. If $\mathsf{A}_G(t^-) = \mathsf{A}_H(t^-)$, then the claim is trivially true from then on, so assume there is a unique G-cavity g_{t^-} and H-cavity h_{t^-}. Let $h_\sigma(r_t) = s_t$ and $g_\sigma(r_t) = x_t$. There are some cases to consider:

1. If $x_t = \mathsf{g}_{t^-}$ and $s_t = \mathsf{h}_{t^-}$, then $\mathsf{A}_G(t^+) = \mathsf{A}_H(t^+)$.
2. If $x_t \neq \mathsf{g}_{t^-}$ and $s_t = \mathsf{h}_{t^-}$, then it follows that $\mathsf{A}_H(t^+) \setminus \mathsf{A}_G(t^+) = \{x_t\}$, whereas $\mathsf{A}_G(t^+) \setminus \mathsf{A}_H(t^+) = \{\mathsf{g}_{t^-}\}$—i.e., $\mathsf{g}_{t^+} = \mathsf{g}_{t^-}$ but $\mathsf{h}_{t^+} = x_t$, and so r_t is such that $g_\sigma(r_t) = \mathsf{h}_{t^+}$ and $h_\sigma(r_t) = \mathsf{h}_{t^-}$.

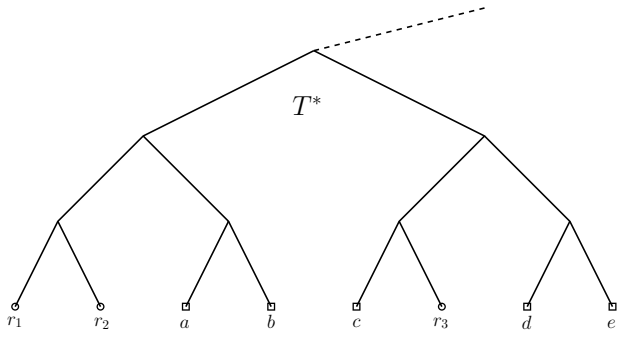


Figure 3.1: In this small example, suppose H is the matching $\{(r_1, c), (r_2, a), (r_3, d)\}$, and G initially matches r_1 to a. Then, the time is currently at 1^+, and $\mathsf{g}_{1^+} = c$, $\mathsf{h}_{1^+} = a$. Suppose G then assigns r_2 to b. Then $\mathsf{g}_{2^+} = \mathsf{g}_{1^+} = c$, and $\mathsf{h}_{2^+} = b$. When r_3 arrives, G may then assign it to c, causing $\mathsf{g}_{3^+} = d$ and $\mathsf{h}_{3^+} = \mathsf{h}_{2^+} = b$.

3. If $x_t = \mathsf{g}_{t^-}$ and $s_t \neq \mathsf{h}_{t^-}$, then it follows that $\mathsf{A}_H(t^+) \setminus \mathsf{A}_G(t^+) = \{\mathsf{h}_{t^-}\}$, whereas $\mathsf{A}_G(t^+) \setminus \mathsf{A}_H(t^+) = \{s_t\}$—i.e., $\mathsf{h}_{t^+} = \mathsf{h}_{t^-}$ but $\mathsf{g}_{t^+} = s_t$, and so r_t is such that $g_\sigma(r_t) = \mathsf{g}_{t^-}$ and $h_\sigma(r_t) = \mathsf{g}_{t^+}$.

4. Finally, we claim that the case $x_t \neq \mathsf{g}_{t^-}$ and $s_t \neq \mathsf{h}_{t^-}$ must imply that $x_t = s_t$. Indeed, let that the lowest ancestor considered by *HST-greedy* when G (respectively, H) makes an assignment for r_t be denoted as a^G (respectively, a^H) If a^G and a^H are not identical, say a^G is lower. Then, $T(a^G)$ contains a server $\mathsf{A}_G(t^-) \setminus \mathsf{A}_H(t^-)$, but since the only server in $\mathsf{A}_G(t^-) \setminus \mathsf{A}_H(t^-)$ is g_{t^-}, we would get $x_t = \mathsf{g}_{t^-}$ and a contradiction; a similar analysis shows that a^H cannot be lower. Hence $a^G = a^H = a$, say. Note that G and H must consider the same set of servers to assign to. Now if both G and H assign r_t to its closest free server within $T(a)$, and neither assignment is with the G-cavity nor the H-cavity, then $x_t = s_t$. Hence, $\mathsf{g}_{t^+} = \mathsf{g}_{t^-}$ and $\mathsf{h}_{t^+} = \mathsf{h}_{t^-}$, and so $\mathsf{A}_G(t^+) \setminus \mathsf{A}_H(t^+) = \{\mathsf{g}_{t^+}\}$ and $\mathsf{A}_H(t^+) \setminus \mathsf{A}_G(t^+) = \{\mathsf{h}_{t^+}\}$.

Also, note that if $\mathsf{A}_G(t^+) \neq \mathsf{A}_H(t^+)$ and $\mathsf{g}_{t^+} \neq \mathsf{g}_{t^-}$, then case 3 is the only possibility, and so $g_\sigma(r_t) = \mathsf{g}_{t^-}$ and $h_\sigma(r_t) = \mathsf{g}_{t^+}$ as stated in the claim. Similarly, if $\mathsf{A}_G(t^+) \neq \mathsf{A}_H(t^+)$ and $\mathsf{h}_{t^+} \neq \mathsf{h}_{t^-}$, then case 2 is the only possibility, and so $g_\sigma(r_t) = \mathsf{h}_{t^+}$ and $h_\sigma(r_t) = \mathsf{h}_{t^-}$. Case 4 does not change the status of the G-cavity or H-cavity, and case 1 results in $\mathsf{A}_G(t^+) = \mathsf{A}_H(t^+)$, so no other cases are possible.

Another way to view the above lemma is to consider the symmetric difference $g_\sigma \Delta h_\sigma$ of the two matchings and to claim that this is a single path or cycle. We start off with two edges $(r_1, s_1), (r_1, x_1)$; each subsequent time we place down two edges adjacent to r_t, and these extend the path (in cases 2 and 3) until we close a cycle (as in case 1, when both the G-cavity and H-cavity disappear), at which time the process stops. ■

As stated in the above lemma, we will use g_{t^+} to represent the unique G-cavity at time t^+ and h_{t^+} the unique H-cavity at time t^+. By redefining k appropriately, we will assume that $\mathsf{A}_G(t^+) \neq \mathsf{A}_H(t^+)$ for all times $t < k$, with $\mathsf{A}_G(k^+) = \mathsf{A}_H(k^+)$, and hence g_{t^+} and h_{t^+} are defined for all times $t \in \{1, 2, \ldots, k-1\}$. We can thus ignore all times $t > k$ since the two algorithms from then on behave identically. Finally, note that if $u \leq t$, then $\mathsf{A}_G(t^+) \subseteq \mathsf{A}_G(u^+)$. Now, to prove

the "hybrid lemma"', Lemma 3.2, we show that the difference in costs can be measured merely in terms of the distances (according to the line) traveled by the cavities.

Lemma 3.4 (Accounting Lemma)

$$\sum_{t=1}^{k} d_L(r_t, g_\sigma(r_t)) - \sum_{t=1}^{k} d_L(r_t, h_\sigma(r_t)) \leq 2\sum_{t=2}^{k-1} d_L(\mathsf{g}_{t^-}, \mathsf{g}_{t^+}) + 2\sum_{t=2}^{k-1} d_L(\mathsf{h}_{t^-}, \mathsf{h}_{t^+}) + 2\, d_T(r_1, s_1),$$

i.e., the difference in costs between G and H is at most twice the line distance traveled by the G-cavities and H-cavities, plus twice the tree distance from $r_1 \to s_1$.

Proof. First, we consider the cases where $t > 1$. By the triangle inequality, we get that for any t,

$$d_L(r_t, g_\sigma(r_t)) - d_L(r_t, h_\sigma(r_t)) \leq d_L(g_\sigma(r_t), h_\sigma(r_t)).$$

If $g_\sigma(r_t) = h_\sigma(r_t)$, then $d_L(r_t, g_\sigma(r_t)) - d_L(r_t, h_\sigma(r_t)) = 0$, so we assume that $g_\sigma(r_t) \neq h_\sigma(r_t)$. By Lemma 3.3, the three possible cases to consider are when $\mathsf{A}_G(t^+) = \mathsf{A}_H(t^+)$, $\mathsf{A}_G(t^+) \neq \mathsf{A}_H(t^+)$ and $\mathsf{g}_{t^+} = \mathsf{g}_{t^-}$, and $\mathsf{A}_G(t^+) \neq \mathsf{A}_H(t^+)$ and $\mathsf{h}_{t^+} = \mathsf{h}_{t^-}$.

There are three cases. First, if $\mathsf{A}_G(t^+) \neq \mathsf{A}_H(t^+)$. Then, if $\mathsf{g}_{t^+} \neq \mathsf{g}_{t^-}$, then by Lemma 3.3, $d_L(g_\sigma(r_t), h_\sigma(r_t)) = d_L(\mathsf{g}_{t^-}, \mathsf{g}_{t^+})$. Second: if $\mathsf{h}_{t^+} \neq \mathsf{h}_{t^-}$, then Lemma 3.3 again gives us that $d_L(g_\sigma(r_t), h_\sigma(r_t)) = d_L(\mathsf{h}_{t^-}, \mathsf{h}_{t^+})$. Third, if $\mathsf{A}_G(t^+) = \mathsf{A}_H(t^+)$, then this must be time $t = k$ (i.e., the last request), with $h_\sigma(r_t) = \mathsf{h}_{t^-}$ and $g_\sigma(r_t) = \mathsf{g}_{t^-}$. In this case $d_L(g_\sigma(r_k), h_\sigma(r_k)) = d_L(\mathsf{g}_{k^-}, \mathsf{h}_{k^-})$—which by the triangle inequality applied to all the previous differences, is at most $\sum_{t=2}^{k-1} \big(d_L(\mathsf{g}_{t^-}, \mathsf{g}_{t^+}) + d_L(\mathsf{h}_{t^-}, \mathsf{h}_{t^+})\big) + d_L(r_1, \mathsf{g}_{1^+}) + d_L(r_1, \mathsf{h}_{1^+}) = \sum_{t=2}^{k-1} \big(d_L(\mathsf{g}_{t^-}, \mathsf{g}_{t^+}) + d_L(\mathsf{h}_{t^-}, \mathsf{h}_{t^+})\big) + d_L(r_1, h_\sigma(r_1)) + d_L(r_1, g_\sigma(r_1))$.

Now, adding over all times $t \in \{2, \ldots, k\}$, we get that $\sum_{t=2}^{k} d_L(r_t, g_\sigma(r_t)) - d_L(r_t, h_\sigma(r_t))$ is at most

$$\sum_{t=2}^{k} d_L(g_\sigma(r_t), h_\sigma(r_t)) \leq \sum_{t=2}^{k-1} 2\big(d_L(\mathsf{g}_{t^-}, \mathsf{g}_{t^+}) + d_L(\mathsf{h}_{t^-}, \mathsf{h}_{t^+})\big) + d_L(r_1, h_\sigma(r_1)) + d_L(r_1, g_\sigma(r_1)).$$

Finally, adding in $d_L(r_1, g_\sigma(r_1)) - d_L(r_1, h_\sigma(r_1))$ to both sides, and noting that

$$2d_L(r_1, g_\sigma(r_1)) \leq 2d_T(r_1, g_\sigma(r_1)) \leq 2d_T(r_1, h_\sigma(r_1)),$$

we get

$$\sum_{t=1}^{k} d_L(r_t, g_\sigma(r_t)) - d_L(r_t, h_\sigma(r_t)) \leq \sum_{t=2}^{k-1} 2\big(d_L(\mathsf{g}_{t^-}, \mathsf{g}_{t^+}) + d_L(\mathsf{h}_{t^-}, \mathsf{h}_{t^+})\big) + 2\, d_T(r_1, h_\sigma(r_1)),$$

which completes the proof. ∎

3.2.1 Distance Traveled by the Cavities

Given this "accounting lemma" Lemma 3.4, we now bound the total distance traveled by the cavities. If a^* is the least common ancestor of r_1 and s_1 in T, the following lemma shows us that the cavities must stay within $T(a^*)$, the subtree rooted at a^*.

Lemma 3.5 *Set a^* to be the least common ancestor of r_1 and s_1 in T, and let $T(a^*)$ be the subtree rooted at a^*. For all times $t < k$, both g_{t^+} and h_{t^+} lie within $T(a^*)$.*

Proof. To begin, $\mathsf{g}_{1^+} = s_1$, and hence in $T(a^*)$. Also, $\mathsf{h}_{1^+} = x_1$ is chosen by *HST-greedy*, and must lie in the lowest subtree containing both r_1 and a free server; hence this is a (not necessarily proper) subtree of $T(a^*)$. To get a contradiction, suppose there is a $t < k$ such that g_{t^-} and h_{t^-} lie within $T(a^*)$, but the claim is false at time t^+. Since r_t assigns to a unique server, it cannot move both g_{t^+} and h_{t^+} out of $T(a^*)$ at the same time. Suppose $\mathsf{g}_{t^+} \in T(a^*)$ and $\mathsf{h}_{t^+} \notin T(a^*)$. Then, $\mathsf{h}_{t^+} \neq \mathsf{h}_{t^-}$ but $\mathsf{g}_{t^+} = \mathsf{g}_{t^-}$, and so by Lemma 3.3, $g_\sigma(r_t) = \mathsf{h}_{t^+}$ and $h_\sigma(r_t) = \mathsf{h}_{t^-}$. However, g_{t^+} is in $\mathsf{A}_G(t^-)$ and lies within $T(a^*)$, and so G would have preferred to assign r_t to g_{t^+} rather than h_{t^+}, a contradiction. The same holds true if $\mathsf{g}_{t^+} \notin T(a^*)$ and $\mathsf{h}_{t^+} \in T(a^*)$. ∎

Henceforth, let us denote the subtree $T(a^*)$ as T^*. The rest of the proof shows that each of the cavities travels a total distance of $O(2^{\text{level}(a^*)}) = O(d_T(r_1, s_1))$ which completes the proof. At a high level, the proof proceeds thus: consider the various locations of the G-cavity over time, and view this as the cavity moving over time. We show that whenever this motion changes direction, the level of the least common ancestor of these locations strictly increases, and hence the sum of the distances traveled behaves like a geometric sum. First, consider some useful definitions to help formalize this idea:

Definition 3.6 (Direction) *We define $\mathsf{dir}_\mathsf{g}(t^+)$, the* direction *of the G-cavity g_{t^+}, to be either L or R as follows: initially, we say that $\mathsf{dir}_\mathsf{g}(1^+) := \mathsf{L}$. For $t > 1$, if g_{t^+} is located to the left (respectively, right) of g_{t^-} on the line, then $\mathsf{dir}_\mathsf{g}(t^+) := \mathsf{L}$ (respectively, R), and if $\mathsf{g}_{t^+} = \mathsf{g}_{t^-}$ then $\mathsf{dir}_\mathsf{g}(t^+) := \mathsf{dir}_\mathsf{g}(t^-)$. An analogous definition follows for $\mathsf{dir}_\mathsf{h}(t^+)$, the direction of the H-cavity h_{t^+}.*

Definition 3.7 *Let $a_\mathsf{g}(t^+)$ be the least common ancestor of $\mathsf{g}_{1^+}, \ldots, \mathsf{g}_{t^+}$, and define $T_\mathsf{g}(t^+) = T(a_\mathsf{g}(t^+))$. Similarly, if $a_\mathsf{h}(t^+)$ is the least common ancestor of $\mathsf{h}_{1^+}, \ldots, \mathsf{h}_{t^+}$, then $T_\mathsf{h}(t^+) = T(a_\mathsf{h}(t^+))$.*

We know that for all t, $T_\mathsf{g}(t^+), T_\mathsf{h}(t^+) \subseteq T^*$ (see Lemma 3.5 for a justification). Moreover, there is the obvious monotonicity property that if $u \leq t$, then $T_\mathsf{g}(u^+) \subseteq T_\mathsf{g}(t^+)$. We now need more information on the servers that lie between g_{t^-} and g_{t^+} and between h_{t^-} and h_{t^+}. Note that any of the properties proved below for G also hold analogously for H by symmetry.

Lemma 3.8 *For $t > 1$, there are no servers in $\mathsf{A}_G(t^+)$ that lie between g_{t^-} and g_{t^+}. Likewise, there are no servers in $\mathsf{A}_H(t^+)$ that lie between between h_{t^-} and h_{t^+}.*

Proof. We only deal with $\mathsf{A}_G(t^+)$, since a proof for $\mathsf{A}_H(t^+)$ follows similarly. If $\mathsf{g}_{t^+} = \mathsf{g}_{t^-}$, then the claim follows trivially, so assume that this is not the case. Consider the request r_t, and note that by Lemma 3.3, $h_\sigma(r_t) = \mathsf{g}_{t^+}$ and $g_\sigma(r_t) = \mathsf{g}_{t^-}$. There are certainly no servers in $\mathsf{A}_G(t^+)$ that lie between r_t and g_{t^-} since G chose the closest server in the direction of g_{t^-} from r_t. Also, we know that there are no servers in $\mathsf{A}_H(t^+)$ that lie between r_t and g_{t^+}, since H chose the closest server in the direction of g_{t^+} from r_t. The only server in $\mathsf{A}_G(t^+)$ that might be between r_t and g_{t^+}, then, is g_{t^-}. However, even in this case, there are still no servers in $\mathsf{A}_G(t^+)$ that are between g_{t^-} and g_{t^+}. ∎

It also follows that the relation described by Lemma 3.8 is transitive, so one can show that for $1 \leq i < j \leq k$, there are no servers in $\mathsf{A}_G(j^+)$ that lie between g_{i^+} and g_{j^+} (and, no servers in $\mathsf{A}_H(j^+)$ that lie between h_{i^+} and h_{j^+}). We will use this fact in our next proof, which also considers the "direction" in which g_{t^+} or h_{t^+} is moving (denoted $\mathsf{dir}_\mathsf{g}(t^+)$ and $\mathsf{dir}_\mathsf{h}(t^+)$).

Lemma 3.9 *If $\mathsf{dir}_\mathsf{g}(t^+) = \mathsf{L}$, then there are no servers in $\mathsf{A}_G(t^+)$ to the right of g_{t^+} that are within $T_\mathsf{g}(t^+)$, and if $\mathsf{dir}_\mathsf{g}(t^+) = \mathsf{R}$, then there are no servers in $\mathsf{A}_G(t^+)$ to the left of g_{t^+} that are within $T_\mathsf{g}(t^+)$. Similarly, if $\mathsf{dir}_\mathsf{h}(t^+) = \mathsf{L}$, then there are no servers in $\mathsf{A}_H(t^+)$ to the right of h_{t^+} that are within $T_\mathsf{h}(t^+)$, and if $\mathsf{dir}_\mathsf{h}(t^+) = \mathsf{R}$, then there are no servers in $\mathsf{A}_H(t^+)$ to the left of h_{t^+} that are within $T_\mathsf{h}(t^+)$.*

Proof. We will show the case for when $\mathsf{dir}_\mathsf{g}(t^+) = \mathsf{L}$, since the proof for when $\mathsf{dir}_\mathsf{g}(t^+) = \mathsf{R}$ is essentially the same. Assume for the sake of contradiction that there exists some server $s \in \mathsf{A}_G(t^+)$ to the right of g_{t^+} within $T_\mathsf{g}(t^+)$. First, under these assumptions, we claim that for $i < t$, every g_{i^+} must lie to the left of s and to the right of g_{t^+}. To show the former claim holds, suppose some g_{v^+} was such that s was to the left of it—then, s lies between g_{v^+} and g_{t^+}, yet $s \in \mathsf{A}_G(t^+)$, a contradiction. For the latter claim, note that g_{t^-} must be to the right of g_{t^+}, since $\mathsf{dir}_\mathsf{g}(t^+) = \mathsf{L}$. If there were some maximal v such that g_{v^+} lies to the left of g_{t^+}, then $\mathsf{g}_{t^+} \notin \mathsf{A}_G(t^-)$, a contradiction.

Let the left and right child subtrees of $T_\mathsf{g}(t^+)$ be called T_1 and T_2, respectively. One can show that g_{t^+} must lie within T_1 and s lies within T_2 using the previous fact and the definition of $T_\mathsf{g}(t^+)$. Now, let $u \in [2, t]$ be the largest integer such that g_{u^-} is in T_2. Note that such a u must exist, since if every G-cavity was within T_1, then $T_\mathsf{g}(t^+)$ would be equal to T_1, which cannot be the case.

Now, by Lemma 3.3, r_{u+1} is such that $g_\sigma(r_{u+1}) = \mathsf{g}_{u^-}$ and $h_\sigma(r_{u+1}) = \mathsf{g}_{u^+}$. Note that by the maximality assumption of u and the fact that g_{t^+} is in T_1, g_{u^+} cannot lie in T_2. If r_{u+1} lies within T_2 or to the right of s, then H would have assigned r_{u+1} to s rather than g_{u^+}, a contradiction. Otherwise, if r_{u+1} lies within T_1 or to the left of T_1, then G would have assigned r_{u+1} to g_{u^+} rather than g_{u^-}, which is another contradiction. Since both cases lead to the desired contradiction, we can conclude that no such $s \in \mathsf{A}_G(t^+)$ can exist, which proves the claim. An analogous proof can be made for $\mathsf{A}_H(t^+)$. ∎

The following lemma can then be established from the previous result:

Lemma 3.10 *Suppose $\mathsf{dir}_\mathsf{g}(t^-) \neq \mathsf{dir}_\mathsf{g}(t^+)$. Then the level of the tree $T_\mathsf{g}(t^+)$ is strictly greater than the level of $T_\mathsf{g}(t^-)$. Similarly, if $\mathsf{dir}_\mathsf{h}(t^-) \neq \mathsf{dir}_\mathsf{h}(t^+)$, then the level of the tree $T_\mathsf{h}(t^+)$ is strictly greater than the level of $T_\mathsf{h}(t^-)$.*

Proof. If $\mathsf{dir}_\mathsf{g}(t^-) = \mathsf{L}$, then there are no servers in $\mathsf{A}_G(t^-)$ within $T_\mathsf{g}(t^-)$ to the right of g_{t^-}. But since g_{t^+} is to the right of g_{t^-}, and g_{t^+} was a server in $\mathsf{A}_G(t^-)$, it must be the case that $\mathsf{g}_{t^+} \notin T_\mathsf{g}(t^-)$. Thus, $T_\mathsf{g}(t^-)$ does not include g_{t^+}, yet $T_\mathsf{g}(t^+)$ must include g_{t^+}, and so the level of $T_\mathsf{g}(t^+)$ is greater than the level of $T_\mathsf{g}(t^-)$. A similar proof can be used to show that the same result holds for $\mathsf{dir}_\mathsf{g}(t^-) = \mathsf{R}$, and an analogous result also holds for $T_\mathsf{h}(t^+)$. ∎

Finally, we can show that the distance traveled by the cavities is at most on the order of the tree-distance of the initial assignment $r_1 \to s_1$.

Lemma 3.11 *For the binary α-HST, the total distance traveled by either the G-cavities or H-cavities is at most $O(d_T(r_1, s_1))$.*

Proof. As t increases, $T_{\mathsf{g}}(t^+)$ may change, and define ρ_t such that $T(\rho_t) = T_{\mathsf{g}}(t^+)$; note that $\text{level}(\rho_t)$ is non-decreasing. Moreover, as long as the scope stays fixed at some subtree $T(\rho_t)$, the G-cavity g_{t^+} cannot change direction, and hence the total distance it travels is at most the width of $T(\rho_t)$, which is $O(\alpha^{\text{level}(\rho_t)})$. Finally, each of the ρ_i's is a descendent of a^*, the root of T^*. Hence the total distance traveled by the G-cavity is at most $1 + \alpha + \alpha^2 + \ldots + \alpha^{\text{level}(a^*)}$, which is $O(\alpha^{\text{level}(a^*)}) = O(d_T(r_1, s_1))$. A similar argument holds for the distances traveled by the H-cavities. ∎

Plugging Lemma 3.11 into the "accounting lemma" (Lemma 3.4), we get

$$\sum_{t=1}^{k} d_L(r_t, g_\sigma(r_t)) - \sum_{t=1}^{k} d_L(r_t, h_\sigma(r_t)) \leq O(d_T(r_1, s_1)) + 2\, d_T(r_1, s_1).$$

Note that $d_T(r_1, s_1) \geq d_L(r_1, s_1)$, and hence the expression on the right is at most $\lambda\, d_T(r_1, s_1)$ for some $\lambda = O(1)$. This completes the proof of Lemma 3.2 (the "hybrid lemma") and hence the proofs for Theorem 3.1 and Theorem 1.1. Although in this section we only covered Δ-ary HSTs with $\Delta = 2$, a generalization of *HST-greedy* can be shown to be $O(\Delta^2)$ times the tree-cost of the optimal matching on any Δ-ary HST (as opposed to the $O(1)$ factor we show in our analysis), and hence still $O(\log k)$-competitive on the line if Δ is regarded as a small constant.

4 The Random-Subtree Algorithm

We now turn to showing that a different randomized algorithm gives an $O(\log k)$ competitive ratio for the line; the proof generalizes to doubling metrics too. To start off, we use the fact that *binary* 2-HSTs approximate the line metric with $O(\log k)$ expected stretch. It is not difficult to show that the (deterministic) greedy algorithm on a binary 2-HST is $O(\log k)$-competitive compared to the optimal solution on the tree, which implies an $O(\log^2 k)$-competitive ratio in all. In this section, we show that randomization helps: a certain randomized greedy algorithm is $O(1)$-competitive on the binary 2-HST, giving us a different $O(\log k)$-competitive algorithm for the line. In fact, the proof extends to HSTs obtained from doubling metrics, and hence proves Theorem 1.2.

4.1 The Algorithm

Let us define the algorithm *Random-Subtree* for online metric matching on an arbitrary HST as follows: when a request r comes in, consider its lowest ancestor node a whose subtree $T(a)$ also contains a free server. Now we choose a random free server in the subtree rooted at a as follows: from among those of a's children whose subtrees contain a free server under them, we choose such a child of a uniformly at random, and repeat this process until we reach a leaf/server s—we then map r to server s. Observe that ours is a different randomized greedy algorithm from that in [MNP06], which would have chosen a server uniformly at random from among all the servers under a. Our main theorem is the following.

Theorem 4.1 *The algorithm* Random-Subtree *is* $2(1 + 1/\epsilon)H_\Delta$*-competitive on* Δ*-ary* α*-HSTs, as long as* $\alpha \geq \max((1 + \epsilon)H_\Delta, 2)$.

Since the line embeds into binary 2-HSTs with expected stretch $O(\log k)$, we get an $O(\log k)$-competitive randomized algorithm for the line. Moreover, in Appendix B.2, we show that an

algorithm for Δ-ary α-HSTs satisfying the property above (with $\Delta = O(1)$) implies an algorithm for doubling metrics with an additional loss of $O(\log k)$; this proves Theorem 1.2.

The proof of the theorem goes thus: we first just consider the edges incident to the root (which we call root-edges) of an Δ-ary α-HST, and count the number of times these edges are used. Specifically, we show that for any sequence of requests, the number of requests that use the root-edges in our algorithm is at most H_Δ times the minimum number of requests that must use these root-edges. This "root-edges lemma" is the technical heart of our analysis; getting H_Δ instead of H_k (obtained in [MNP06]) requires defining the right potential function, and carefully accounting for the gain we get from using the *Random-Subtree* algorithm rather than the randomized greedy algorithm of [MNP06].

Having proved the root-edges lemma, notice that for any fixed vertex v in an HST, the subtree rooted at v is another HST on which we can apply the root-edges lemma to bound the cost incurred on the edges incident to v. Consequently, applying this for every internal vertex in the HST and summing up the results shows that the total cost remains at most $O(H_\Delta) \cdot \mathsf{Opt}$, as long as the parameter α for the HST is larger than H_Δ.

4.2 The Root-Edges Lemma

Consider a Δ-ary α-HST T with a set of requests $R \cup R'$ such that the requests in R originate at the leaves of T, and those in R' originate at the root. We assume that the number of servers in T is at least $|R \cup R'|$. Let $T_1, T_2, \ldots, T_\Delta$ denote the Δ child subtrees of T. We can assume that T has exactly Δ child subtrees without loss of generality because we can create extra subtrees with no available servers as needed without changing the analysis or behavior of the algorithm. We will use $R(T_i)$ to denote the set of requests that originate in subtree T_i. Let n_i be the number of servers in T_i, and let $M^* = \sum_{i=1}^{\Delta} \max(|R \cap T_i| - n_i, 0)$.

Fact 4.2 *In any assignment of requests in $R \cup R'$ to servers, at least $M^* + |R'|$ requests use root-edges.*

Proof. The number of requests that originate in a subtree T_i is $|R \cap R(T_i)|$, so $|R \cap R(T_i)| - n_i$ represents the number of requests that originate in T_i and must assign to servers outside of T_i, and hence, must use a root-edge. If this quantity is negative, we then use 0 as the lower bound for T_i. Thus, the sum of this quantity over all subtrees is therefore a lower bound on the number of requests that use root-edges. ∎

Now, let the random variable M count the number of requests in $R \cup R'$ that use a root-edge when assigned by the algorithm *Random-Subtree*.

Lemma 4.3 (Root-Edges Lemma)

$$E[M] \leq H_\Delta \cdot (M^* + |R'|).$$

Proof. Let the k requests $R \cup R'$ be labeled $r_1, r_2, \ldots, r_k$, where r_1 is the earliest request and r_k is the latest request. The request r_t is assigned at time t, and we refer to the situation just before this assignment as being at time t^-, and the situation just after as time t^+. Note that t^- for $t = 1$ (denoted as 1^-) represents the time before any request assignments have been made, and t^+ for $t = k$ (denoted as k^+) represents the time after all request assignments have been made. Let

$R_t = \{r_t, r_{t+1}, \ldots, r_k\}$, the set of requests at time t^- that have yet to arrive. At time t^-, let $n_{i,t}$ be the number of available servers in tree T_i. A subtree T_i is said to be *open* at time t^- if $n_{i,t} > 0$ (there are available servers at time t^- in T_i). Let η_t be the number of open subtrees of T at time t^-.

Define the first $\min(n_{i,t}, |R_t \cap R(T_i)|)$ requests of T_i to be the *local requests of T_i at time t^-* (these are the ones in $R(T_i)$ that have the lowest numbered indices), and the remaining requests in T_i to be the *global requests of T_i at time t^-*.[1] Let $\mathcal{L}_{i,t}$ and $\mathcal{G}_{i,t}$ be the set of local and global requests in T_i at time t^-, and let $\mathcal{L}_t := \cup_i \mathcal{L}_{i,t}$ and $\mathcal{G}_t := \cup_i \mathcal{G}_{i,t}$. For convenience, we say that a request r_j *becomes global* at time t if r_j is local at time t^-, but r_j is global at time t^+. Let requests in $\mathcal{R}_t := R_t \cap R'$ be called *root requests of T at time t^-*.

As a sanity check, note that at the beginning (at time 1^-), the set of pending requests $R_1 = R \cup R'$, the number of pending requests in subtree T_i is $n_{i,1} = n_i$, the number of global requests in T_i is $|\mathcal{G}_{i,1}| = \max(|R \cap R(T_i)| - n_i, 0)$ (so the total number of global requests at time 1^- is M^*), and the number of root requests is $|\mathcal{R}_1| = |R'|$.

Recall that global requests of T_i must assign to servers outside of T_i: while an optimal offline algorithm can identify where to assign these global requests, an online algorithm may assign a global request from T_i to some subtree T_j that only has as many servers as future requests, which causes some local request in T_j to become global. Hence we want to upper-bound the number of future requests in R_{t+1} that become global due to our assignment for r_t. We associate with each request in R_t a "cost" at time t^- which represents this upper bound. Later, we will use the cost function to define the potential function. The cost function at time t^- is $F_t : R_t \to \mathbb{Z}_{\geq 0}$; we say it is *well-formed* if it satisfies two properties:

- $F_t(r_j) = 0$ if and only if $r_j \in \mathcal{L}_t$ (i.e., it is a local request at time t^-), and
- for all global and root requests $r_j \in \mathcal{G}_t \cup \mathcal{R}_t$, $F_t(r_j)$ is an upper bound on the random variable η_j, the number of open subtrees at time j^-.

Constructing the Well-Formed Cost Functions. We set $F_1(r_j) = \Delta$ (the degree of the tree) for all $r_j \in \mathcal{G}_1 \cup \mathcal{R}_1$ (global and root requests at time 1^-), and $F_1(r_j) = 0$ for all $r_j \in \mathcal{L}_1$ (local requests at time 1^-). It is immediate that the map F_1 is well-formed.

Now at each time t^+, we will define the next function F_{t+1} using F_t. For this, first consider time t^-, and suppose that the map F_t is well-formed. The easy case first: If $r_t \in \mathcal{L}_t$, then define $F_{t+1}(r) = F_t(r)$ for all $r \in R_t$. In this case if a request in R_t is a local/global/root request at time t^-, it remains a local/global/root request at time t^+, so F_{t+1} is still well-formed.

On the other hand, suppose $r_t \in \mathcal{G}_t \cup \mathcal{R}_t$, i.e., it is a global or root request. Recall there are η_t open subtrees at time t^-. Each open subtree T_i contains $|R_t \cap R(T_i)|$ requests and $n_{i,t}$ free servers, so if $|R_t \cap R(T_i)| \geq n_{i,t}$ then assigning r_t to a server in this subtree would cause some request r_j in $R_t \cap R(T_i)$ to become global at time t (because $n_{i,t+1}$ would become $n_{i,t} - 1$). In this case, define $a_t(T_i) := j$, the index of the request r_j that turns global in subtree T_i. Else, if no request in $R_t \cap R(T_i)$ would become global, set $a_t(T_i) := k + i$ (which cannot be the index of any request,

[1] The idea behind calling requests local/global is this: assuming no servers in T_i are used up by requests from other subtrees, the local servers will be assigned within T_i by our algorithm, whereas the global ones will be assigned to other subtrees (and hence use a root-edge). Of course, as servers within T_i are used by requests in other subtrees, some local requests become global.

since there are only k requests). Let $A_t = \{a_t(T_i) \mid T_i \text{ open at time } t^-\}$; note that $|A_t| = \eta_t$. Now denote the elements of A_t by $\{p_j\}_{j=1}^{\eta_t}$ such that $p_1 < p_2 < \cdots < p_{\eta_t}$. Note that each p_j corresponds

(Another sanity check: we claim that the last entry $p_{\eta_t} > k$; indeed, if r_t is a global or root request, there must be some open subtree T_i which has more available servers than requests.) Now, let T_i be the subtree that r_t assigns to, chosen by picking out of the open subtrees uniformly at random. We now define the map F_{t+1} at time t^+. There are two cases to consider:

- If $a_t(T_i) > k$ (i.e., none of the requests in $R(T_i) \cap R_{t+1}$ become global due to assigning r_t), then we set $F_{t+1}(r) = F_t(r)$ for all requests $r \in R_{t+1}$.
- If $a_t(T_i) \leq k$, then say $a_t(T_i) = p_{\eta_t - q + 1}$ in the ordering given above (i.e., $a_t(T_i)$ was the q^{th} largest value in A_t). Now assign $F_{t+1}(r) = F_t(r)$ for all $r \in R_{t+1} \setminus \{r_{a_t(T_i)}\}$, and $F_{t+1}(r_{a_t(T_i)}) = q - 1$.

Lemma 4.4 *The map F_{t+1} is well-formed.*

Proof. By induction, the map F_t was well-formed. In the first case when r_t is local, the claim follows since the sets of local/global/root requests in R_{t+1} remain unchanged and $\eta_{t+1} \leq \eta_t$.

Suppose now that r_t is a global or root request that is assigned into T_i. If none of the requests in $R(T_i)$ become global due to this change, the well-formedness of F_{t+1} follows again. So, let's consider the case where the request $r_j \in R(T_i)$ becomes global because of r_t. We previously defined that $j = a_t(T_i)$, and that j is the q^{th} largest of the sequence of A_t. Moreover, since r_j is mapped by F_{t+1} to an integer $q \leq k$, it suffices to show that at most $q - 1$ subtrees will be open at time j^-. Indeed, we claim that for any subtree T_h with $a_t(T_h) < a_t(T_i) = j$, there will be no servers available in T_h at time j^-. To see this, note that since $a_t(T_h) < k$, there must be some request that becomes global if r_t assigns in T_h. Thus, the number of requests in T_h that had indices smaller than j (and hence arrive before r_j) was at least $n_{h,t}$, the number of available servers in T_h at time t^-. Hence, these requests alone would cause T_h to be closed. Moreover, for subtree T_i, the fact that r_j becomes global at time t means that T_i will also be closed at time j^-. Hence, the only open subtrees at time j^- would be the subtrees T_ℓ which were open at time t^-, which were such that $a_t(T_\ell) > j$. There are at most $q - 1$ of such subtrees T_ℓ, since j is the q^{th} largest of the sequence. This shows that F_{t+1} is well-formed. ∎

Note that maps F_t and F_{t+1} are either the same or differ on at most one request r_j that becomes global at time t, in which case $F_{t+1}(r_j)$ becomes positive. Moreover, $F_{t'}(r_j) = F_{t+1}(r_j)$ for all times $t' \in [t+1, j]$.

The Potential Function Analysis. We are now in a position to define the potential function,

$$\Phi_t = \textstyle\sum_{r \in R_t} H_{F_t(r)}, \tag{4.1}$$

where we consider $H_0 = 0$. Also, define ρ_t to be the number of requests that our algorithm has already matched outside of their subtrees at time t^-. The proof of the root-edges lemma will then follow immediately from the following claim proved using induction.

Lemma 4.5 *For all $t \in [1, k+1]$, $E[\Phi_t + \rho_t] \leq H_\Delta \cdot (M^* + |R'|)$.*

Proof. We prove this by induction on time t^-. The base case is for 1^-. Then, $\rho_1 = 0$, the number of global/root requests is $M^* + |R'|$, and since each such request r has $F_1(r) = \Delta$, we get that $\Phi_1 = H_\Delta \cdot (M^* + |R'|)$.

Inductively assume the claim is true at time t^-. Thus, $E[\Phi_t + \rho_t] \leq H_\Delta \cdot (M^* + |R'|)$. We want to show the same at time t^+, right after r_t has been assigned. We claim that $E[\Phi_{t+1} + \rho_{t+1}] \leq \Phi_t + \rho_t$, which will complete the proof. There are two cases to consider:

- Suppose r_t is a local request. Its subtree contains an unassigned server, so $\rho_{t+1} = \rho_t$. Moreover, $F_t(r_t) = 0$ since F_t is well-formed, so $\Phi_{t+1} = \Phi_t$.
- Suppose r_t is a global request and gets assigned to subtree T_i. In this case, $\rho_{t+1} = \rho_t + 1$. Now consider $E[\Phi_t - \Phi_{t+1}]$, which is

  $$H_{F_t(r_t)} - \tfrac{1}{F_t(r_t)} \textstyle\sum_{j=0}^{F_t(r_t)-1} H_j \geq H_{F_t(r_t)} - \tfrac{1}{\eta_t} \sum_{j=0}^{\eta_t - 1} H_j = 1$$

  since $F_t(r_t) \geq \eta_t$ by the well-formedness of map F_t.

Hence, in both cases, conditioned on all assignments made before time t^-, the value $E[\Phi_{t+1} + \rho_{t+1}] \leq \Phi_t + \rho_t$, where the expectation is taken over the random choices of r_t. This completes the induction, and the proof of the lemma. ∎

Since $\rho_{k+1} = M$ and $\Phi_{k+1} = 0$, using Lemma 4.5 with $t = k + 1$ finishes the proof of the root-edges lemma. ∎

4.3 Bounding the Total Cost

In the previous section, we proved the root-edges lemma, which compared the number of times edges incident to the root of the HST were used by the algorithm to that of the optimal matching, for any set of requests. We now apply this lemma at each subtree of the original HST to get the following claim.

Lemma 4.6 *Consider a Δ-ary α-HST T, any set R of requests at the leaves of T, and requests R' at the root of T, such that $|R \cup R'|$ is at most the number of servers in T. If $\mathsf{Alg}(R \cup R', T)$ denotes the cost of* Random-Subtree *on requests $R \cup R'$ on tree T, and $\mathsf{Opt}(R \cup R', T)$ the cost of the optimal solution, we have*

$$E[\mathsf{Alg}(R \cup R', T)] \leq c \cdot H_\Delta \cdot \mathsf{Opt}(R \cup R', T)$$

for $c = 2(1 + 1/\epsilon)$ as long as $\alpha \geq \max\{2, (1 + \epsilon)H_\Delta\}$.

Proof. We prove this by induction on the depth of the HST. The base case is when the HST is a star—by scaling, we can assume it has unit edge lengths. In this case we can directly apply Lemma 4.3 (the root-edges lemma). Recall that we defined M to be the number of requests in $R \cup R'$ that use root edges in our algorithm's matching and M^* to be the number of requests in R that use root-edges in the optimal matching. Now our algorithm incurs a cost of at most $2M$, whereas $\mathsf{Opt}(R \cup R', T) = |R'| + 2M^* \geq |R'| + M^*$. By the root-edges lemma, we get

$$E[\mathsf{Alg}(R \cup R', T)] \leq E[2M] \leq 2H_\Delta \cdot \mathsf{Opt}(R \cup R', T) \leq cH_\Delta \cdot \mathsf{Opt}(R \cup R', T).$$

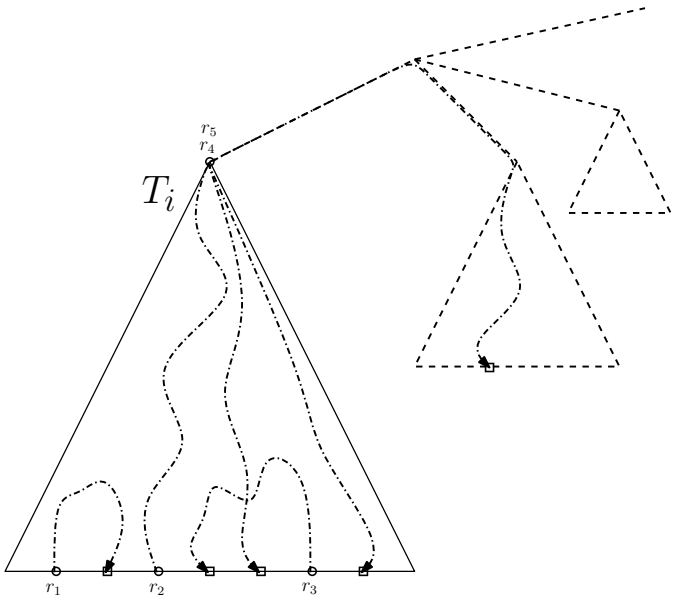


Figure 4.2: If the dotted lines represent the assignments that Opt makes within T_i, then $\Lambda_i^* = \{r_1, r_3\}$, $\Gamma_i^* = \{r_2\}$, and the set R' of T_i is $\{r_4, r_5\}$.

For the inductive step, we prove the claim for an α-HST T assuming it inductively holds for all the α-HSTs T_i that are the subtrees of the root. Let the length of the root-edges be 1, the length of their children edges be $1/\alpha$, etc. If we define the length of any root-leaf path in T to be $(1+\beta)$, then we get $\beta \leq \frac{1}{\alpha-1}$. Let n_i be the number of servers in subtree T_i of T.

Consider the optimal matching Opt, and define the following quantities:

- Let Γ_i^* be the requests originating in T_i that Opt matches *outside* T_i (call these the Opt-*global* clients), and let $M_i^* = |\Gamma_i^*|$.
- Let Λ_i^* be the requests originating in T_i that Opt satisfies with servers in T_i (these are the Opt-*local* clients).
- Let $R_i = \Lambda_i^* \cup \Gamma_i^*$ be all the requests originating in T_i, and note that $R = \cup_i R_i$.

Fact 4.7 (Optimal Cost)

$$\mathsf{Opt}(R \cup R', T) = \sum_{i=1}^{\Delta} \mathsf{Opt}(\Lambda_i^*, T_i) + \sum_{i=1}^{\Delta} M_i^* \cdot 2(1+\beta) + |R'| \cdot (1+\beta).$$

Proof. We can partition the set $R \cup R'$ into the three types of subsets Λ_i^*, Γ_i^*, and R'. For each request r in Λ_i^* we note that the optimal cost of assigning r is determined by $\mathsf{Opt}(\Lambda_i^*, T_i)$, since r's server must be in T_i. For every request in Γ_i^*, we must assign from some subtree T_i to T_j by using a root-edge. Thus, we simply pay twice the length from the root to a leaf for each request in Γ_i^*, which can be expressed as $2(1+\beta)M_i^*$ for each subtree T_i. Finally, the requests that begin at the

root (of which there are $|R'|$ many) will pay the length of the root to a leaf, which is exactly $1 + \beta$. ■

Now, let M_i be the set of requests originating outside T_i (but possibly at the root of T_i) that the algorithm satisfies by assigning into T_i. Look at $R_i \cup M_i$—these are all the requests that the subtree T_i encounters, and let X_i be the first n_i of these requests, those which can be satisfied within the subtree T_i. Define $\widehat{R_i} = X_i \cap R_i$ and $\widehat{M_i} = X_i \cap M_i$. (Note that the sets M_i, X_i, $\widehat{R_i}$, and $\widehat{M_i}$ are all random variables.)

Fact 4.8

$$E[\mathsf{Alg}(R \cup R', T)] = \sum_{i=1}^{\Delta} E[\mathsf{Alg}(\widehat{R_i} \cup \widehat{M_i}, T_i)] + \sum_{i=1}^{\Delta} E[|M_i|] \cdot (2 + \beta).$$

Proof. Suppose r is some request in T_i that Alg assigned to some server in $T_j \neq T_i$. We account for this assignment's cost by breaking the path from r to s into two parts. The initial part, accounted for by the latter term of the equation, includes the edges used from r to the root along with both edges incident to the root. The path from r to the root is of length $\beta + 1$, and the additional root-edge is of length 1, giving us $\beta + 2$. Since there are $|M_i|$ such requests for each subtree T_i, we see that the second term covers all of the initial parts of the paths of each global request.

The reason for the above convention is that now that we have covered all outgoing requests, we can imagine all global incoming requests as having originated at the root of the tree, since their inital parts have already been accounted for. Therefore, this quantity can be described as $\mathsf{Alg}(\widehat{R_i} \cup \widehat{M_i}, T_i)$ for each subtree T_i. ■

By our inductive assumption we know that for any $\widehat{R_i}$ and $\widehat{M_i}$ defined for a tree T_i,

$$E[\mathsf{Alg}(\widehat{R_i} \cup \widehat{M_i}, T_i)] \leq c\, H_\Delta\ \mathsf{Opt}(\widehat{R_i} \cup \widehat{M_i}, T_i). \tag{4.2}$$

Fact 4.9

$$\mathsf{Opt}(\widehat{R_i} \cup \widehat{M_i}, T_i) \leq \mathsf{Opt}(\Lambda_i^*, T_i) + M_i^* \cdot 2\beta + |M_i| \cdot \beta.$$

Proof. To bound Opt's cost on $\widehat{R_i} \cup \widehat{M_i}$, we imagine the requests in $\widehat{R_i} \cap \Lambda_i^*$ being sent according to where $\mathsf{Opt}(\Lambda_i^*, T_i)$ sent them, and the remaining requests being assigned arbitrarily to the remaining servers. The former cost is upper bounded by $\mathsf{Opt}(\Lambda_i^*, T_i)$. For the latter term, there are $|\widehat{R_i} \cap \Gamma_i^*| \leq |\Gamma_i^*| = M_i^*$ requests which incur a cost of at most 2β (since they go from some leaf to another within the fixed subtree T_i), and the remaining requests—at most M_i of them—incur a cost of at most β (since they go from the root of T_i to a leaf). ■

Using Facts 4.8 and 4.9 with equation (4.2), we get

$$E[\mathsf{Alg}(R \cup R', T)] \leq c\, H_\Delta \sum_{i=1}^{\Delta} (\mathsf{Opt}(\Lambda_i^*, T_i) + M_i^* \cdot 2\beta + E[|M_i|] \cdot \beta) + \sum_{i=1}^{\Delta} E[|M_i|] \cdot (2 + \beta).$$

Using Fact 4.7, we can rewrite the above expression as

$$E[\mathsf{Alg}(R \cup R', T)] \leq c\, H_\Delta \left(\mathsf{Opt}(R \cup R', T) - \sum_{i=1}^{\Delta} M_i^* \cdot 2(1+\beta) - |R'| \cdot (1+\beta) \right) + c\, H_\Delta \sum_{i=1}^{\Delta} (M_i^* \cdot 2\beta + E[|M_i|] \cdot \beta) + \sum_{i=1}^{\Delta} E[|M_i|] \cdot (2+\beta).$$

To prove Lemma 4.6, it now suffices to show that all the terms apart from $c\, H_\Delta\, \mathsf{Opt}(R \cup R', T)$ sum to something non-positive. In other words, we would like to show that

$$\sum_{i=1}^{\Delta} c\, H_\Delta \left(M_i^* \cdot 2\beta + E[|M_i|] \cdot \beta \right) + E[|M_i|] \cdot (2+\beta) \leq \sum_{i=1}^{\Delta} c\, H_\Delta\, (2M_i^* + |R'|)(1+\beta).$$

Now we are almost done. We use the root-edges lemma to claim that for each i,

$$E[|M_i|] \leq H_\Delta \cdot \sum_{i=1}^{\Delta} \left(M_i^* + |R'| \right).$$

Using this, abbreviating $M^* = \sum_{i=1}^{\Delta} M_i^*$ and $r' = |R'|$, and cancelling H_Δ throughout, it suffices to show that

$$c\, M^*\, 2\beta + (M^* + r')(cH_\Delta\beta + (2+\beta)) \leq c\, (2M^* + r')(1+\beta).$$

Or equivalently, it suffices to choose c such that

$$c \geq \frac{(M^* + r')(2+\beta)}{(2M^* + r')(1+\beta) - 2\beta\, M^* - (M^* + r')H_\Delta\beta}$$

as long as the expression in the denominator is positive. But the expression on the right is

$$\frac{M^*(2/\beta + 1) + r'(2/\beta + 1)}{M^*(2/\beta - H_\Delta) + r'(1/\beta + 1 - H_\Delta)} \leq \max\left(\frac{2/\beta + 1}{2/\beta - H_\Delta}, \frac{2/\beta + 1}{1/\beta + 1 - H_\Delta} \right).$$

Consequently, if the larger of $\frac{2/\beta+1}{2/\beta - H_\Delta}$ and $\frac{2/\beta+1}{1/\beta+1-H_\Delta}$ is bounded above by c, we will be done.

Now, since $\alpha \geq 2$, then $1/\beta \geq \alpha - 1 \geq 1$, and $(2/\beta - H_\Delta) \geq (1/\beta + 1 - H_\Delta)$. Thus for our setting of the parameter α, we get

$$\frac{2/\beta + 1}{2/\beta - H_\Delta} \leq \frac{2/\beta + 1}{1/\beta + 1 - H_\Delta},$$

and we can focus showing the latter is at most c. This just requires some algebra, and is shown in the next lemma.

Fact 4.10 *For $1/\beta \geq \alpha - 1$, $\alpha \geq (1+\epsilon)H_\Delta$ and $c = 2(1 + \frac{1}{\epsilon})$, we have that*

$$\frac{2/\beta + 1}{1/\beta + 1 - H_\Delta} \leq c.$$

Proof. Note that

$$c\, H_\Delta\, \epsilon = 2(1+\epsilon)H_\Delta.$$

Now we substitute $x = (1+\epsilon)H_\Delta$ to get

$$c(x - H_\Delta) \geq 2x - 1 \implies c \geq \frac{2x-1}{x - H_\Delta}.$$

Observe that $f(x) = \frac{2x-1}{x-H_\Delta}$ is a decreasing function for $x > H_\Delta$. Hence $x \leq \alpha \leq 1/\beta + 1 \implies f(x) \geq f(\alpha) \geq f(1/\beta + 1)$. But since $c \geq f(x)$, this completes the calculation. ■

This completes the proof of the bound on the expected cost of *Random-Subtree*. ■

The lemma above directly proves Theorem 4.1. As an aside, note that 2-HSTs that have large degree, or binary HST's that have $\alpha \approx 1$ (say $\alpha = 1 + 1/\log k$), can both simulate star metrics, on which we have an $\Omega(\log k)$ lower bound—hence we do need some relationship between α and Δ. It is an interesting open problem to see if we can obtain constant-competitive algorithms for Δ-ary α-HSTs where $\alpha \leq H_\Delta$.

5 The Harmonic Algorithm for the Line

To prove Theorem 1.3, we first give a lemma which analyzes the expected difference in cost between running *Harmonic* on all the requests, and running the optimal algorithm for the first step and *Harmonic* thenceforth; using this bound in a hybrid argument proves Theorem 1.3. For a request sequence $\sigma = r_1, \ldots, r_k$, let g_σ be the matching obtained by assigning $r_1, \ldots, r_k$ using *Harmonic*. Let $N(r_t)$ be the set of available neighboring servers to r_t—those which are closest to r_t on the left or right and available at time t^-. Define h_σ to be a matching obtained by first matching r_1 to an $s_1 \in N(r_1)$, and then using *Harmonic* to assign $r_2, \ldots, r_k$. We will use G to represent the algorithm that produces g_σ and H for h_σ.

Lemma 5.1

$$E[\sum_{i=1}^{k} d(r_i, g_\sigma(r_i)) - d(r_i, h_\sigma(r_i)] \leq O(\log \Delta) \cdot d(r_1, s_1).$$

In other words, the expected cost of G for any request sequence is at most the expected cost of H on the same request sequence *plus* $O(\log \Delta)\, d(r_1, s_1)$—the difference is proportional to the length of this forced initial assignment. This allows us to immediately prove Theorem 1.3—let us present this proof before we commence the proof of Lemma 5.1. We first show the following lemma.

Lemma 5.2 *There exists an optimal matching f_σ such that $f_\sigma(r_1) \in N(r_1)$.*

Proof. Assume to get a contradiction that no such optimal matching exists. Then, let f be an arbitrary optimal matching; $f(r_1)$ is either to the left or right of r_1. We handle the case where $f(r_1)$ is to the right, as the other case can be shown via a symmetric argument. Let s_R be the closest available server to the right of r_1: by our contradictoin assumption, $f(r_1)$ is to the right of s_R. If s_R is never taken by a future request, then clearly f cannot be optimal, as its total cost could be decreased by assigning $r_1 \to s_R$. Thus, let r_t be the request such that $f(r_t) = s_R$.

Either r_t lies to the left, colocated with, or to the right of s_R. If r_t lies to the left of or is colocated with s_R, then we can construct a matching f' where $f'(r_1) = s_R$, $f'(r_t) = f(r_1)$, and $f'(r_i) = f(r_i)$ for all other requests. Note that the cost of f' and f are the same, which contradicts the assumption that no such optimal matching exists. If r_t lies to the right of s_R, then f' has a cost which is less than that of f—another contradiction, since f has optimal cost. Since we have reached a contradiction in all cases, the lemma follows. ∎

Now, given any request sequence σ and an optimal matching f_σ for this sequence such that $f_\sigma(r_1) \in N(r_1)$, we can define a sequence of hybrid matchings $\{h_\sigma^t\}_{t=0}^k$, where h_σ^t is obtained by matching the first t requests $r_1, \ldots, r_t$ in σ to $f_\sigma(r_1), \ldots, f_\sigma(r_t)$ and the remaining requests $r_{t+1}, \ldots, r_k$ to $g_\sigma(r_{t+1}), \ldots, g_\sigma(r_k)$. Note that h_σ^0 is just the *Harmonic* matching g_σ, and h_σ^k produces the optimal matching f_σ. Moreover, by ignoring the servers in $\{f_\sigma(r_i) \mid i \leq t\}$ and just considering $r_{t+1}, \ldots, r_k$ as the request sequence, Lemma 5.1 implies

$$E[\textstyle\sum_{i=t+1}^k d(r_i, h_\sigma^t(r_i))] \leq E[\sum_{i=t+1}^k d(r_i, h_\sigma^{t+1}(r_i))] + O(\log \Delta) \cdot d(r_t, f_\sigma(r_t)),$$

since we can regard the assignment $r_t \to f_\sigma(r_t) \in N(r_t)$ as the assignment $r_1 \to s_1$ used in Lemma 5.1. Now, by adding $\sum_{i=1}^t d(r_i, f_\sigma(r_i))$ to both sides,

$$E[\textstyle\sum_{i=1}^k d(r_i, h_\sigma^t(r_i))] \leq E[\sum_{i=1}^k d(r_i, h_\sigma^{t+1}(r_i))] + O(\log \Delta) \cdot d(r_t, f_\sigma(r_t)).$$

Summing this over all values of t, and using that $h_\sigma^0 = g_\sigma$ and $h_\sigma^k = f_\sigma$, we get

$$E[\textstyle\sum_{i=1}^k d(r_i, g_\sigma(r_i))] \leq E[\sum_{i=1}^k d(r_i, f_\sigma(r_i))] + O(\log \Delta) \cdot \sum_{i=1}^k d(r_i, f_\sigma(r_i)).$$

The left side is the expected cost of *Harmonic*, and the right side is the cost of the optimal matching, which proves Theorem 1.3.

5.1 A Coupling Argument

We now prove Lemma 5.1. Let $\mathsf{A}_G(t)$ be the set of free servers at time t^+ when running algorithm G, and $\mathsf{A}_H(t)$ be similarly defined for algorithm H. Note that if at time t^+, $\mathsf{A}_G(t) = \mathsf{A}_H(t)$, then the expected difference in costs between algorithms G and H on requests $r_{t+1}, \ldots, r_k$ is 0. Thus, we can without loss of generality only consider the time instants where $\mathsf{A}_G(t) \neq \mathsf{A}_H(t)$, as this will not change the expected difference in costs between the two algorithms. Under this assumption, we have that $\mathsf{A}_G(t) \neq \mathsf{A}_H(t)$ for all t that we consider.

Let g_1 be the only element of $\mathsf{A}_G(1) \setminus \mathsf{A}_H(1)$ and h_1 the only element of $\mathsf{A}_H(1) \setminus \mathsf{A}_G(1)$. Assume that g_1 is to the left of h_1, as the other case is covered by symmetry. We now give a coupling π between the executions of G and H from the two different starting configurations (which is equivalently a coupling between the evolutions of sets $\mathsf{A}_G(t)$ and $\mathsf{A}_H(t)$). Recall that a valid coupling should satisfy the property that the marginals should give us a faithful execution of *Harmonic* on G and H respectively. In fact we define the coupling π only on pairs of states satisfying $|\mathsf{A}_G(t) \setminus \mathsf{A}_H(t)| = 1 = |\mathsf{A}_H(t) \setminus \mathsf{A}_G(t)|$; since we start off with such a pair of initial states, and since π will ensure that the pair of states resulting from each step of this coupling will continue to have this property (see Invariant 1 below), this will suffice for our purposes.

Since we only consider pairs of states such that $|\mathsf{A}_G(t) \setminus \mathsf{A}_H(t)| = 1 = |\mathsf{A}_H(t) \setminus \mathsf{A}_G(t)|$ for all t, let us define g_t and h_t to be the unique server in the $\mathsf{A}_G(t) \setminus \mathsf{A}_H(t)$ and $\mathsf{A}_H(t) \setminus \mathsf{A}_G(t)$ respectively,

and $\delta_t = d(g_t, h_t)$ to be the distance between these two servers. In fact, our coupling will also ensure the property that there are no available servers between g_t and h_t (see Invariant 2 below).

We now define some notation that will ease the presentation of the remainder of the proof. We say that $r \to_G s_1$ and $r \to_H s_2$ if r assigns to s_1 in G and s_2 in H. Also, let $\Delta c(r) = d(r, s_1) - d(r, s_2)$. Furthermore, define the joint distribution π on the assignments made by r_t in G and H. We will use $\Pr_\pi[E]$ to denote the probability of an event E occurring under distribution π, and $\Pr_G$ and $\Pr_H$ the probabilities of the events occurring under algorithm G and H. Let $N_G(r_t)$ be the available neighboring servers to r_t in G, and similarly for $N_H(r_t)$. There are four cases to consider: either request r_t appears to the left of both g_t and h_t with $g_t \in N_G(r_t)$, r_t appears between g_t and h_t, r_t appears to the right of both g_t and h_t with $h_t \in N_H(r_t)$, or $N_G(r_t) = N_H(r_t)$. We will call these cases Case 1, 2, 3, and 4, respectively. Also, let $\mathcal{E}_n$ be the event that there exists a time t such that $\delta_t \geq n$. Then, let $Q_{i,n} = \Pr_\pi[\mathcal{E}_n \mid \mathcal{E}_i]$.

We can rewrite the expression $E[d(r_t, g_\sigma(r_t)) - d(r_t, h_\sigma(r_t))]$ as equivalent to $\sum_{s \in N_G(r_t)} d(r_t, s) \cdot \Pr_G[r_t \to_G s] - \sum_{s \in N_H(r_t)} d(r_t, s) \cdot \Pr_H[r_t \to_H s]$. Also, let $\mathsf{DiffCost}(r_t)$ be the random variable that takes on value $d(r_t, s_G) - d(r_t, s_H)$ with probability $\Pr_\pi[r_t \to_G s_G, r_t \to_H s_H]$ for $s_G \in N_G(r_t)$ and $s_H \in N_H(r_t)$, so that $E[\mathsf{DiffCost}(r_t)] = \sum_{s_G \in N_G(r_t), s_H \in N_H(r_t)} (d(r_t, s_G) - d(r_t, s_H)) \cdot \Pr_\pi[r_t \to_G s_G, r_t \to_H s_H]$.

For each case we will prove the following invariants, given the inductive assumption that they all hold prior to the arrival of request r_t:

Invariant 1. Either $\mathsf{A}_G(t+1) = \mathsf{A}_H(t+1)$, or $|\mathsf{A}_G(t+1) \backslash \mathsf{A}_H(t+1)| = 1 = |\mathsf{A}_H(t+1) \backslash \mathsf{A}_G(t+1)|$.

Invariant 2. If $\mathsf{A}_G(t+1) \neq \mathsf{A}_H(t+1)$, then there are no available servers at time t^+ that lie between g_{t+1} and h_{t+1}.

For the base case of $t = 1$, Invariant 1 has already been established, and Invariant 2 holds by the definition of the neighborhood of r_1 since $g_2, h_2 \in N(r_1)$. By inductively assuming that Invariants 1 and 2 hold, Cases 1,2,3, and 4 are the only cases encounterable by a request r_t. To see this, note that either none of the two servers in $N_G(r_t) \cup N_H(r_t)$ is g_t or h_t (which results in Case 4), or one, or both of the servers in $N_G(r_t) \cup N_H(r_t)$ is g_t and/or h_t (which results in Cases 1, 2, or 3). We will also show that the following claims hold for each of the four cases.

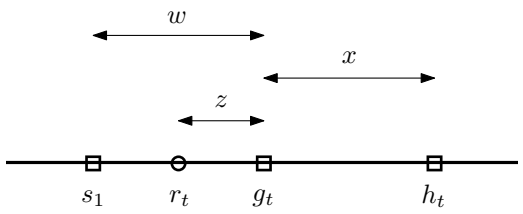
Claim 3. If $\delta_{t+1} \neq 0$, then $\Delta c(r_t) \leq \delta_{t+1} - \delta_t$. If $\delta_{t+1} = 0$, then $\Delta c(r_t) \leq \delta_t$.

Claim 4. $E[d(r_t, g_\sigma(r_t)) - d(r_t, h_\sigma(r_t))] = E[\mathsf{DiffCost}(r_t)]$.

Claim 5. $Q_{i,n} = i/n$.

5.1.1 Case 1

The first case is when r_t to the left of both g_t and h_t, and $g_t \in N_G(r_t)$. Let s_1 be the other server (not g_t) in $N_G(r_t)$, and set $w = d(s_1, g_t)$, $z = d(r_t, g_t)$, and $x = \delta_t$.



Set $p = \frac{w-z}{w}$ and $q = \frac{w-z}{w+x}$. We define the distribution π for this case, and δ_{t+1} and $\Delta c(r_t)$ are also noted for each event:

Event	Assignments	$\Pr_\pi$	$\delta_{t+1} - \delta_t$	$\Delta c(r_t)$
1	$r_t \to_G s_1, r_t \to_H s_1$	$1-p$	0	0
2	$r_t \to_G g_t, r_t \to_H s_1$	$p-q$	w	$2z-w$
3	$r_t \to_G g_t, r_t \to_H h_t$	q	$-x$	$-x$

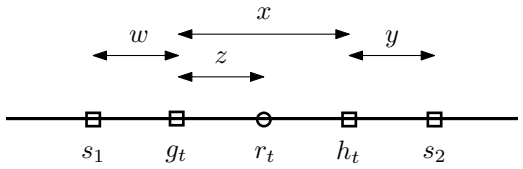
In Event 1, $g_{t+1} = g_t$ and $h_{t+1} = h_t$, and in Event 2, $g_{t+1} = s_1$ and $h_{t+1} = h_t$, so Invariants 1 and 2 are maintained for these events. For Event 3, $\mathsf{A}_G(t+1) = \mathsf{A}_H(t+1)$, so Invariants 1 and 2 are also maintained. One can also see that $\delta_{t+1} - \delta_t \geq \Delta c(r_t)$ for all three events as well, which proves Claim 3.

For Claim 4, note that $\sum_{s \in N_G(r_t)} d(r_t, s) \cdot \Pr_G[r_t \to_G s] - \sum_{s \in N_H(r_t)} d(r_t, s) \cdot \Pr_H[r_t \to_H s] = 2z(w-z)/w - 2(x+z)(w-z)/(w+x) = (w-z)/w \cdot (2z-w) + (w-z) - (2x+2z)(w-z)/(w+x) = (-x)(w-z)/(w+x) + (w-z/w - (w-z)/(w+x))(2z-w) = (p-q)(2z-w) + q(-x) = \sum_{s_G \in N_G(r_t), s_H \in N_H(r_t)} (d(r_t, s_G) - d(r_t, s_H)) \cdot \Pr_\pi[r_t \to_G s_G, r_t \to_H s_H]$, so Claim 4 also holds.

To prove Claim 5, we will proceed by induction on $n - i$. When $n - i = 0$, then clearly $Q_{i,n} = 1$. Suppose $\delta_t = x$, and inductively assume that $Q_{j,n} = j/n$ for all $j > x$. We have that $Q_x = (p-q)Q_{x+w} + (1-p)Q_x$. This gives us $Q_x = \frac{p-q}{p} Q_{x+w}$, and so $Q_x \leq (1 - q/p)(x+w)/n = x/n$.

5.1.2 Case 2

The second case is when r_t appears between g_t and h_t. Let $s_1 \neq h_t$ be the other server in $N_H(r_t)$, and $s_2 \neq g_t$ the other server in $N_G(r_t)$, and set $w = d(s_1, g_t)$, $z = d(r, g_t)$, $x = \delta_t$, and $y = d(h_t, s_2)$.



Set $p = \frac{x+y-z}{x+y}$ and $q = \frac{w+z}{w+x}$. We define the distribution π for this case, and δ_{t+1} and $\Delta c(r_t)$ are also noted for each event:

Event	Assignments	$\Pr_\pi$	$\delta_{t+1} - \delta_t$	$\Delta c(r_t)$
1	$r_t \to_G s_2, r_t \to_H h$	$1-p$	y	y
2	$r_t \to_G g, r_t \to_H h$	$p+q-1$	$-x$	$2z-x$
3	$r_t \to_G g, r_t \to_H s_1$	$1-q$	w	$-w$

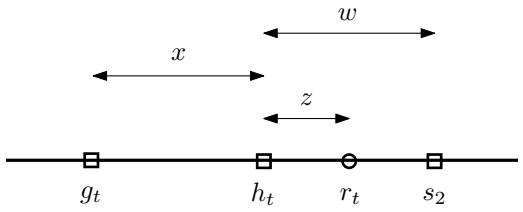
In Event 1, $g_{t+1} = g_t$ and $h_{t+1} = s_2$, and in Event 3, $g_{t+1} = s_1$ and $h_{t+1} = h_t$, so Invariants 1 and 2 are maintained for these events. For Event 2, $\mathsf{A}_G(t+1) = \mathsf{A}_H(t+1)$, so Invariants 1 and 2 are also maintained. One can also see that $\delta_{t+1} - \delta_t \geq \Delta c(r_t)$ for all three events as well, which proves Claim 3.

For Claim 4, note that $\sum_{s \in N_G(r_t)} d(r_t, s) \cdot \Pr_G[r_t \to_G s] - \sum_{s \in N_H(r_t)} d(r_t, s) \cdot \Pr_H[r_t \to_H s] = 2z(x+y-z)/(x+y) - 2(x-z)(w+z)/(w+x) = \frac{2z(x+y-z)}{x+y} - \frac{2(x-z)(w+z)}{w+x} = (1-p)y + (p+q-1)(2z-x) + (1-q)(-w) = \sum_{s_G \in N_G(r_t), s_H \in N_H(r_t)} (d(r_t, s_G) - d(r_t, s_H)) \cdot \Pr_\pi[r_t \to_G s_G, r_t \to_H s_H]$, so Claim 4 also holds.

To prove Claim 5, we will proceed by induction on $n - i$. When $n - i = 0$, then clearly $Q_{i,n} = 1$. Suppose $\delta_t = x$, and inductively assume that $Q_{j,n} = j/n$ for all $j > x$. $Q_x = (1-p)Q_{x+y} + (1-q)Q_{x+w} = \frac{z}{x+y}(x+y)/n + \frac{x-z}{x+w}(x+w)/n = x/n$.

5.1.3 Case 3

The third case is when r_t to the right of both g_t and h_t, and $h_t \in N_H(r_t)$. Let s_2 be the other server (not g_t) in $N_G(r_t)$, and set $w = d(s_2, h_t)$, $z = d(r_t, h_t)$, and $x = \delta_t$.



Set $p = \frac{w-z}{w}$ and $q = \frac{w-z}{x+w}$. We define the distribution π for this case, and δ_{t+1} and $\Delta c(r_t)$ are also noted for each event:

Event	Assignments	$\Pr_\pi$	$\delta_{t+1} - \delta_t$	$\Delta c(r_t)$
1	$r_t \to_G s_2, r_t \to_H s_2$	$1-p$	0	0
2	$r_t \to_G s_2, r_t \to_H h$	$p-q$	w	$w-2z$
3	$r_t \to_G g, r_t \to_H h$	q	$-x$	x

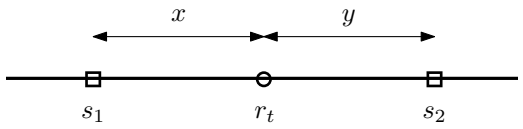
In Event 1, $g_{t+1} = g_t$ and $h_{t+1} = h_t$, and in Event 2, $g_{t+1} = g_t$ and $h_{t+1} = s_2$. so Claimss 1 and 2 are maintained for these events. For Event 3, $\mathsf{A}_G(t+1) = \mathsf{A}_H(t+1)$, so Invariants 1 and 2 are also maintained. One can also see that $\delta_{t+1} - \delta_t \geq \Delta c(r_t)$ for all three events as well, which proves Claim 3.

For Claim 4, note that $\sum_{s \in N_G(r_t)} d(r_t, s) \cdot \Pr_G[r_t \to_G s] - \sum_{s \in N_H(r_t)} d(r_t, s) \cdot \Pr_H[r_t \to_H s] = 2(x+z)(w-z)/(w+x) - 2z(w-z)/w = (2x+2z)(w-z)/(w+x) - (w-z)/w \cdot (2z-w) - (w-z) = (p-q)(w-2z) + qx = \sum_{s_G \in N_G(r_t), s_H \in N_H(r_t)} (d(r_t, s_G) - d(r_t, s_H)) \cdot \Pr_\pi[r_t \to_G s_G, r_t \to_H s_H]$, so Claim 4 also holds.

To prove Claim 5, we will proceed by induction on $n - i$. When $n - i = 0$, then clearly $Q_{i,n} = 1$. Suppose $\delta_t = x$, and inductively assume that $Q_{j,n} = j/n$ for all $j > x$. We have that $Q_x = (p-q)Q_{x+w} + (1-p)Q_x$. This gives us $Q_x = \frac{p-q}{p} Q_{x+w}$, and so $Q_x \leq (1-q/p)(x+w)/n = x/n$.

5.1.4 Case 4

The fourth case is when $N_G(r_t) = N_H(r_t)$. Let $s_1 \in N_G(r_t)$ be the server on the left and s_2 the server on the right in $N_G(r_t)$. Let $x = d(s_1, r_t)$ and $y = d(r_t, s_2)$.



Set $p = \frac{y}{x+y}$ and $q = \frac{x}{x+y}$. We define the distribution π for this case, and δ_{t+1} and $\Delta c(r_t)$ are also noted for each event:

Event	Assignments	$\Pr_\pi$	$\delta_{t+1} - \delta_t$	$\Delta c(r_t)$
1	$r_t \to_G s_1, r_t \to_H s_1$	p	0	0
2	$r_t \to_G s_2, r_t \to_H s_2$	q	0	0

In both events, $g_{t+1} = g_t$ and $h_{t+1} = h_t$, so Invariants 1 and 2 are also maintained. One can also see that $\delta_{t+1} - \delta_t \geq \Delta c(r_t)$ for both events as well, which proves Claim 3.

For Claim 4, note that $\sum_{s \in N_G(r_t)} d(r_t, s) \cdot \Pr_G[r_t \to_G s] - \sum_{s \in N_H(r_t)} d(r_t, s) \cdot \Pr_H[r_t \to_H s] = 0 = \sum_{s_G \in N_G(r_t), s_H \in N_H(r_t)} (d(r_t, s_G) - d(r_t, s_H)) \cdot \Pr_\pi[r_t \to_G s_G, r_t \to_H s_H]$, so Claim 4 also holds.

Since $\delta_{t+1} = \delta_t$ with probability 1 under π, then $Q_{i,n} = i/n$ still holds with the arrival of r_t, and so Claim 5 is trivially true.

5.2 Analysis of Cases

Using Claim 4, we have that $E[d(r_t, g_\sigma(r_t)) - d(r_t, h_\sigma(r_t))] = E[\mathsf{DiffCost}(r_t)]$. So, in order to complete the proof for Lemma 5.1, it remains to show that $E[\sum_{t=1}^{k} \mathsf{DiffCost}(r_t)] \leq O(\log \Delta) \cdot d(r_1, s_1)$.

Now, let A_i be the event that $2^{i-1} \cdot \delta_1 \leq \max_j \delta_j < 2^i \cdot \delta_1$. Notice that $\Pr[A_i]$ is at most $Q_{\delta_1, j}$ for some $j \in [2^{i-1} \cdot \delta_1, 2^i \cdot \delta_1]$, which is in turn at most $\frac{\delta_1}{2^{i-1} \cdot \delta_1} = 1/2^{i-1}$ by Claim 5. Set $X_m = \sum_{t=1}^{m} \mathsf{DiffCost}(r_t)$. Then we have that

$$E[X_k] = \sum_{i=1}^{\log \Delta} E[X_k \mid A_i] \cdot \Pr[A_i],$$

since $\max_j \delta_j$ cannot exceed Δ.

To bound $E[X_k \mid A_i]$, we use Claim 3. Let q be the time such that $\delta_q = 0$ and $\delta_{q-1} > 0$, and if no such time exists, set $q = k + 1$. Then, for $m < q$, $X_m = (\delta_m - \delta_{m-1}) + (\delta_{m-1} - \delta_{m-2}) + \cdots + (\delta_2 - \delta_1) = \delta_m - \delta_1 \leq \delta_m$. For $m = q$, Claim 3 gives us that $X_m - X_{m-1} \leq \delta_{m-1}$, so that $X_m \leq 2 \cdot \delta_{m-1}$. Then, for all $m > q$, $X_m = X_{m-1}$ since $\mathsf{A}_G(r_m) = \mathsf{A}_H(r_m)$. Thus, $X_k \leq 2 \max_j \delta_j \leq 2^{i+1} \cdot \delta_1$.

Hence, we have that $\sum_{i=1}^{\log \Delta} E[X_k \mid A_i] \cdot \Pr[A_i] \leq \sum_{i=1}^{\log \Delta} (2^{i+1} \cdot \delta_1)/2^{i-1} = 4\delta_1 \cdot \log \Delta$. Finally, we compare $E[\delta_1]$ with $d(r_1, s_1)$. Recall that h_1 is the other neighbor (not s_1) that is closest to r_1. Then $E[\delta_1] = \frac{d(r_1, h_1)}{d(h_1, s_1)} \cdot 0 + \frac{d(r_1, s_1)}{d(h_1, s_1)} \cdot d(h_1, s_1) = d(r_1, s_1)$, and so Lemma 5.1 follows.

6 Concluding Remarks

In this paper we gave three different $O(\log k)$-competitive algorithms for online metric matching on the line, one of them also applicable to doubling metrics. These algorithms do no better than $O(\log k)$ even on the line, and it would be intriguing to devise new algorithms that close the gap between this logarithmic upper bound and the $O(1)$ lower bound for special metrics. Improving the deterministic upper bound for the line and the randomized upper bound for general metrics both continue to be fascinating open problems.

Acknowledgments. We thank Ravi Krishnaswamy, Kirk Pruhs, and Matthias Englert for useful conversations.

References

[BBGN07] N. Bansal, N. Buchbinder, A. Gupta, and J. S. Naor. An $o(\log^2 k)$-competitive algorithm for metric bipartite matching. In *Proceedings of the 15th annual European Symposium on Algorithms*, pages 522–533, 2007.

[FHK05] B. Fuchs, W. Hochstattler, and W. Kern. Online matching on a line. *Theoretical Computer Science*, 332:251–264, 2005.

[FRT03] J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 448–455, 2003.

[GKL03] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low–distortion embeddings. In *FOCS '03*, pages 534–543, 2003.

[KMV94] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994.

[KN03] E. Koutsoupias and A. Nanavati. The online matching problem on a line. In *WAOA03*, pages 179–191, 2003.

[KP93] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993.

[KP95] E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42:971–983, September 1995.

[KP98] B. Kalyanasundaram and K. Pruhs. Online network optimization problems, 1998. Online Algorithms: The State of the Art , eds. A. Fiat and G. Woeginger, Lecture Notes in Computer Science 1442, Springer-Verlag.

[KP06] Rohit Khandekar and Vinayaka Pandit. Online sorting buffers on line. In *STACS 2006*, volume 3884 of *Lecture Notes in Comput. Sci.*, pages 584–595. Springer, Berlin, 2006.

[MNP06] A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 954–959, 2006.

[Tal04] Kunal Talwar. Bypassing the embedding: Algorithms for low-dimensional metrics. In *STOC '04*, pages 281–290, 2004.

A Some Illustrative Examples

In this section, we give a few well-known but useful bad examples for some natural algorithms for the line, and a (perhaps less known) observation that a preprocessing step of bounding the aspect ratio might offer hope for greedy algorithms previously considered useless. We also present a tight instance for the *HST-greedy* and *Random-Subtree* algorithms on which they both incur an $\Omega(\log k)$ times the optimal cost in expectation.

A.1 A Bad Example for Greedy and a Tight Example for Algorithms Similar to *Harmonic*

Consider k servers at the points $\{-(1+\epsilon), 1, 2, 2^2, \ldots, 2^{k-2}\}$ on the integer line. The requests appear at the points $0, 1, 2, 2^2, \ldots, 2^{k-2}$. The deterministic greedy algorithm, which assigns to the closest available server, will always assign each request (except the last) to the closest server on its right, incurring a cost of $2^{k-1} + (1+\epsilon)$ on k requests, whereas the optimal cost is $1+\epsilon$. This shows a competitive ratio lower bound of $2^{\Omega(k)}$.

Now consider a randomized algorithm on the line with the following property: *if the closest open servers to the left and right are equidistant from the new request, assign to each with probability* $1/2$. (Note that the randomized algorithm discussed in Section 5 has this property.) The probability that the algorithm uses the t^{th} request to assign to the server at location $-(1+\epsilon)$ is about 2^{-t}, in which case the algorithm pays approximately 2^t for the matching. Thus the expected cost on k requests is $\Theta(k)$, and giving a competitive ratio lower bound of $\Omega(k)$.

While these lower bounds sound terrible, let us observe that it is easy to mitigate their badness somewhat. Using standard guess-and-double ideas and massaging the line metric (outlined in Section B.1), we can get another line metric instance where the aspect ratio is bounded by $O(k^2)$. For such low aspect-ratio instances, the above examples show a lower bound of $\Omega(k)$ and $\Omega(\log k)$ respectively. Indeed, the $\Omega(\log k)$ lower bound for this class of randomized algorithms matches the upper bound of $O(\log k)$ proved for *Harmonic* in Section 5.

A.2 A Tight Example for HST-greedy and Random-Subtree

Consider the simple setting where the k servers are at $1, 3, 4, 5, \ldots, k+1$. The k requests are at $2, 3, 4, \ldots, k+1$. Consider a random embedding of the line into a binary 2-HST. Let W be the random variable representing the width of the largest subtree of the HST that contains the point 2 but *not* the point 1. Let $\mathcal{C}$ be the class of algorithms that serve a new request by assigning it to some server within the subtree rooted at the least common ancestor of the new request and its closest free server. Note that *HST-greedy* and Random-Subtree are both members of $\mathcal{C}$. Then, any algorithm of $\mathcal{C}$ will use the request that arrives at some point $x \geq W$ to assign to the server at point 1, thus incurring a cost of at least W for the matching.

A standard calculation shows that the expected stretch of the edge $(1, 2)$ in the HST will be $\Theta(\log k)$; indeed, no sublogarithmic stretch is possible even for embedding the line into 2-HSTs, and the stretch is worst on the edges of the graph. The structure of the binary 2-HST now implies that the size of the largest subtree that contains the point 2 but not the point 1 has expected size $\Theta(\log k)$. Hence, $E[W] = \Theta(\log k)$. Since the optimal algorithm has cost 1, the expected competitive ratio of *HST-greedy* and Random-Subtree (and any algorithm that is inherently greedy and

uses an HST to break ties) is $\Omega(\log k)$ for this setting.

A.3 An Example for the MNP algorithm

We now show an example where *Random-Subtree* gets an asymptotic improvement over the algorithm presented in [MNP06], which we will from now on refer to as the MNP algorithm. In particular, the instance is a Δ-ary tree where the MNP algorithm has competitive ratio $\Omega(\Delta)$, whereas *Random-Subtree* has competitive ratio $O(\log \Delta)$. Consider the one-level tree with Δ leaves $\ell_1, \cdots, \ell_\Delta$ ordered from left to right, all of which share the same parent, and each of the Δ edges have a cost of 1. For convenience, we will also denote e_i to be the edge incident to the leaf ℓ_i. The setting of servers is as follows: ℓ_1 has one server, and for $2 \leq i \leq \Delta$, ℓ_i has $2^{\Delta-i}$ servers. Hence, the number of servers $k = 1 + \sum_{i=2}^{\Delta} 2^{\Delta-i} = 2^{\Delta-1}$, so it follows that $\Delta = \Theta(\log k)$. The first two requests appear at ℓ_1, and for each $i \in [2, \Delta - 1]$, the next $2^{\Delta-i}$ requests appear at ℓ_i, again from left to right.

First, note that two requests will appear at ℓ_1 where there is only one server, and no requests will appear at ℓ_Δ where there is one server. For all other leaves ℓ_i, the number of requests that appear at ℓ_i is the same as the number of servers located at ℓ_i. Also, an optimal matching can match r_1 to the only server at ℓ_Δ for which there is no colocated request, and then the remaining requests $r_2, \cdots, r_k$ can be matched to their colocated servers. Thus, the optimal matching has a cost of 2.

Note that for $i \in [2, \Delta - 1]$, the first $2^{\Delta-i} - 1$ requests of ℓ_i will always assign to their colocated servers within ℓ_i. Since the random choices of *Random-Subtree* do not depend on the number of servers occupying a leaf, these $2^{\Delta-i} - 1$ request-server pairs can be removed from the example without changing the distribution of matchings across the remaining requests nor the cost of the matching. After the removal of these request-server pairs, each leaf seats exactly one server, and so the modified instance is equivalent to the uniform metric with Δ requests and Δ servers, with each point at distance 2 away from one another. Setting X_i to be the indicator random variable with $X_i = 1$ if the ith request cannot assign to its colocated server and $X_i = 0$ otherwise, we get that $X_1 = 0$, $X_2 = 1$, $\Pr[X_i = 1] = 1/(\Delta - i + 2)$ for $3 \leq i \leq \Delta - 1$, and $X_\Delta = 0$. Note that $2\sum_{i=1}^{\Delta} \Pr[X_i = 1]$ represents the total cost of the matching, and so $2\sum_{i=1}^{\Delta} \Pr[X_i = 1] = 2\sum_{i=1}^{\Delta} 1/(\Delta - i + 2) \leq 2H_\Delta$. Thus, *Random-Subtree* has expected competitive ratio $O(\log \Delta)$.

We now analyze the performance of the MNP algorithm. Since the MNP algorithm assigns to a random server rather than a random leaf to break ties for a new request, we cannot simply remove the colocated request-server pairs as in the analysis for *Random-Subtree*. Let A_i represent the event that the MNP algorithm uses edge e_i in assigning a request to a server.

Lemma A.1 *For all $i \in [2, \Delta]$ and $j \in [1, i-1]$, $\Pr[A_i \mid A_j] \geq 2^{j-i}$.*

Proof. Given that A_1 occurred, there must exist a request r_x which originates from ℓ_j and assigns into another leaf. Note that if r_x has been requested, originating from ℓ_j, then all leaves $\ell_1, \cdots, \ell_{j-1}$ have no more available servers. Hence, the total number of available servers is at most $2^{\Delta-j}$. The probability that r_x assigns into ℓ_i is $2^{\Delta-i}/2^{\Delta-j} \geq 2^{j-i}$. This means that A_i occurs, so we deduce that $\Pr[A_i \mid A_i] \geq 2^{j-i}$. ∎

Lemma A.2 *For all $i \in [1, \Delta]$, $\Pr[A_i] \geq 1/2$.*

Proof. Note that $\Pr[A_1] = 1$. Inductively assume that for some i, $\Pr[A_j] \geq 1/2$ for all $j \in [1, i-1]$. Then, $\Pr[A_i] \geq \sum_{j=1}^{i-1} \Pr[A_i \mid A_j] \cdot \Pr[A_j]$. We use the fact that $\Pr[A_1] = 1$, breaking up the summation to get that

$$\Pr[A_i] \geq \frac{1}{2}\Pr[A_i \mid A_1] + \frac{1}{2}\sum_{j=1}^{i-1} \Pr[A_i \mid A_j].$$

Now, $\Pr[A_i \mid A_1] \geq 2^{1-i}$ and $\Pr[A_i \mid A_j] \geq 2^{j-i}$, so we conclude that $\Pr[A_i] \geq 2^{-i} + 2^{-i-1}(2^i - 2) = 2^{-i} + 1/2 - 2^{-i} = 1/2$, thus completing the induction step. ∎

Since the MNP algorithm uses each of the Δ edges with probability $1/2$, the MNP algorithm must pay at least $\Delta/2$ on expectation. Thus, the expected competitive ratio is $\Omega(\Delta)$.

B Discharging Some Assumptions

In this section, we show how to reduce the online metric matching to the special case where (a) we are given a constant factor approximation to the cost of the optimal matching, and (b) where the aspect ratio of the metric we consider is at most $O(k^3)$.

B.1 Bounding the Aspect Ratio

First, let us assume that we know of some value Z such that $Z \in [\mathsf{Opt}, 10\mathsf{Opt}]$, i.e., a constant factor approximation to the optimal value for eventual request sequence σ. In this case, we show how to reduce the general problem to metrics where the aspect ratio is $O(k^3)$. Note that the constant 10 is not important, since any constant strictly greater than 1 would suffice at the expense of increasing the constant in the big-Oh.

Suppose the input metric is (V, d). Right off the bat, we can assume that all requests come at server locations: as shown in [MNP06], this changes the competitive ratio by at most a constant factor, and we can then ignore all of the non-server vertices. We will henceforth assume that $|V| = k$. Let f^*_σ be the optimal matching for sequence σ.

In the case that the metric is not given by a graph, it will still be useful to view the metric as a edge-weighted complete graph $G = (V, K_{|V|})$ where the length/weight of edge (u, v) is $d(u, v)$. We perform the following operations on the graph:

- Delete all edges of length more than Z so that the graph may now be disconnected.
- Take all edges of length less than $Z/20k^2$ and make their length equal to $Z/20k^2$.

Call this new graph $G' = (V', E')$, and let d' be the shortest-path metric in this graph. Note that the aspect ratio of this metric is $20k^3$.

The optimal cost of any request sequence may be higher on this new graph/metric compared to the cost on the original metric; certainly it is no lower since all distances in G' are higher than in G. However, we claim it has not increased by too much. Indeed, consider the same matching f^*_σ: the cost in G' of any request-server pair could have increased from (almost) zero to $Z/20k^2 \times (k-1) \leq Z/20k$. This is because each pair is connected by a path of length at most $k-1$. Moreover, we

are using the property that no $(r, f^*(r))$ in the optimal solution could have used an edge that was deleted. But there are only k such pairs, so the additive increase is only at most $Z/20$. However, we know that $Z \in [\mathsf{Opt}, 10\,\mathsf{Opt}]$, so the cost of the matching f^*_σ in the new metric is at most $\mathsf{Opt} + Z/20 \leq 1.5\,\mathsf{Opt}$. And the cost of the optimal matching on G' can be no more than the cost of this particular matching.

Finally, suppose we have a C-competitive algorithm $\mathcal{A}$ for metrics with aspect-ratio bounded by $20k^3$. Given an arbitrary instance (V, d) and an estimate Z for Opt as above, we can construct (V, d') and run $\mathcal{A}$ on it. We know that for an input sequence σ, we will obtain a matching whose cost (when measured according to d') is at most $1.5\,C$ times the optimal cost on d. Finally, the cost according to d is only less, which gives us a $1.5\,C$-competitive algorithm for general metrics.

B.1.1 The "Guess-and-Double" Step

Now we need to discharge the assumption that our algorithm has an estimate of Opt. Indeed, suppose we have an online algorithm $\mathcal{A}_{guess}$ that takes in a metric space (V, d) and an estimate Z and guarantees C-competitiveness for all sequences σ such that the optimal cost $\mathsf{Opt}(\sigma)$ of satisfying σ satisfies $Z \in [\mathsf{Opt}(\sigma), 10\mathsf{Opt}(\sigma)]$. Then we claim we can get an algorithm $\mathcal{A}$ that achieves $O(C)$ competitiveness on all sequences without such an estimate Z.

We now describe the behavior of algorithm $\mathcal{A}$ on a metric (V, d), when given an online sequence of requests σ. Let $R(t)$ be the sequence of requests $r_1, r_2, \ldots, r_t$. Let us assume that the minimum distance in the metric is 1; this can be ensured by a suitable scaling. For each $i \in \mathbb{Z}_{\geq 0}$, let τ_i be the maximum index such that the optimal solution on the request sequence $R(\tau_i)$ has cost less than 10^i, so that $R(\tau_i + 1)$ is the first prefix on which the cost is at least 10^i. Note that $R(\tau_0)$ must have cost zero, since it has cost less than $10^0 = 1$, and we assumed the least distance was 1. Finally, having seen the sequence $R(t)$, we know if $t - 1$ was τ_i for some i or not.

Suppose that we have been feeding the online requests to $\mathcal{A}_{guess}$ with some guess $Z = 10^i$, so that we are at time t and indeed $Z \in (\mathsf{Opt}(R(t)), 10\mathsf{Opt}(R(t))]$. Now when we get r_{t+1}, if it is still the case that $Z \in (\mathsf{Opt}(R(t+1)), 10\mathsf{Opt}(R(t+1))]$, then we should continue to feed this request to $\mathcal{A}_{guess}(Z)$. If, on the other hand, $\mathsf{Opt}(R(t+1))$ becomes at least $Z = 10^i$—in other words, if $t = \tau_i$—then we want to undo all of the assignments we have done so far (and hence pay for sending the requests back from their servers to their original locations), find a new parameter $Z' = 10^j$ with $j > i$ such that $Z' \in (\mathsf{Opt}(R(t+1)), 10\mathsf{Opt}(R(t+1))]$, re-feed the entire sequence of requests $R(t+1)$ seen so far to $\mathcal{A}_{guess}(Z')$, and continue from there.

Note that the total cost incurred while we were feeding the requests to $\mathcal{A}_{guess}(Z = 10^i)$ would be at most $C \cdot \mathsf{Opt}(R(\tau_i)) < C \cdot 10^i$, by the C-competitiveness of the algorithm $\mathcal{A}_{guess}(Z)$. When we undo these assignments (which indeed we cannot), we would have to pay this amount all over again. So, assuming we could undo these assignments and assuming the final optimal cost $\mathsf{Opt}(\sigma) \in [10^\ell, 10^{\ell+1})$, the total cost incurred is at most $C \cdot (2 \cdot (1 + 10 + \cdots + 10^\ell) + \mathsf{Opt}(\sigma)) \leq 4C \cdot \mathsf{Opt}(\sigma)$.

Finally, we don't need to actually undo and redo these assignments: we can merely set up a bijection between actual assignments of requests, and their ideal locations where we would have mapped them had we undone/redone the assigments. When we want to map a future request r' to a location taken by some request r that is ideally mapped elsewhere, we can instead map r' to that ideal location instead. The triangle inequality ensures that the cost incurred is no more than

this undo/redo model, and hence we have a $4C$-competitive algorithm which works without any guesses on Opt.

To summarize, it suffices to derive a C-competitive algorithm for points on the line where the aspect ratio is bounded by $O(k^3)$, and we are also given an approximate value for Opt of the input sequence. Indeed, given such an algorithm, we can use the above reductions to obtain an $O(C)$-competitive algorithm for all point sets on the line, and requires no guesses on Opt. However, a word of caution: these reductions do not maintain properties such as doubling dimension, and hence cannot be used indiscriminately.

B.2 Embedding Lines and Doubling Metrics into HSTs

It is a standard fact that a set $S \subseteq \mathbb{R}$ of points on the line with aspect ratio $\mathrm{poly}(k)$ can be embedded into dominating *binary* 2-HSTs with expected $O(\log k)$. (This has been previously used, e.g., in the paper of Khandekar and Pandit [KP06].)

For the case of doubling metrics, directly changing the distances of the metric to ensure a bounded aspect-ratio can alter the doubling dimension in undesirable ways. For this case we use the observation from the previous section that it suffices to give algorithms for the problem which take in an estimate $Z \in (\mathsf{Opt}, 10\mathsf{Opt}]$. Hence, we can again delete all the edges of the graph representing the metric which are longer than Z to ensure that the maximum distance between points in the metric is at most kZ.

Now we run the [FRT03]-based embeddings from [GKL03, Section 6] or [Tal04, Section 3], which embed doubling metrics into distributions of HSTs; however we stop the process when the diameter of the clusters becomes Z/k^2, say, and co-locate all servers still sharing a cluster at the same leaf of the HST. This does not result in a *dominating* HST, since distances in the HST may be smaller than in the original metric. However, the distance of any request-server pair is smaller only by at most $2Z/k^2$. So if we find a competitive matching in the HST and translate it back to the original metric, the cost of the solution may increase by at most $O(Z/k) = o(\mathsf{Opt})$.

Finally, we need to pin down the relationship between the degree and α-parameter of α-HSTs obtained from doubling metrics in the above process. Since we want an α-HST, we want the diameters of the clusters at each level shrink by a factor of α at each level of the recursion. Hence

- the expected stretch will be $O(\alpha\, \mathsf{dim_D}\, \log k)$.
- the number of children, Δ of each internal node is at most the size of the $1/(c_1\,\alpha)$-net of each cluster (for some constant $c_1 > 1$). Since the doubling dimension is $\mathsf{dim_D}$, the size of an δ-net is at most $(c_2/\delta)^{\mathsf{dim_D}}$. Hence, plugging in $\delta = 1/(c_1\alpha)$, we get $\Delta \leq (c_1 c_2\, \alpha)^{\mathsf{dim_D}}$.

Hence, $2H_\Delta = 2\mathsf{dim_D} \log(c_1 c_2\, \alpha)$. If we set $\alpha = c_3 \mathsf{dim_D} \log \mathsf{dim_D}$ for a large constant c_3, we satisfy $\alpha \geq 2H_\Delta$—this can then be plugged into Theorem 4.1 for the setting $\epsilon = 1$. This gives us the following theorem, which suffices to get $O(\log k)$-competitive algorithms for online metric matchings on doubling metrics.

Lemma B.1 *Consider a doubling metric (V, d) with doubling dimension $\mathsf{dim_D}$, and consider a parameter Z such that the diameter of (V, d) is at most kZ. There exists an $\alpha = \alpha(\mathsf{dim_D})$ (V, d) can be embedded into a distribution over α-HSTs such that*

- *each node in the α-HSTs have at most Δ children, for some Δ satisfying $\alpha \geq 2H_\Delta$,*
- *the expected stretch is at most $O(\alpha \cdot \mathsf{dim}_\mathsf{D} \cdot \log k)$, which is $O(\log k)$ when $\mathsf{dim}_\mathsf{D} = O(1)$, and*
- *for each HST in the distribution and for any set of k vertex pairs (a_i, b_i), the cost of $\sum_{i=1}^{k} d_T(a_i, b_i)$ in the HST is at most $O(Z/k)$ smaller than $\sum_{i=1}^{k} d(a_i, b_i)$ in the original metric.*