An $O(\log^2 k)$-competitive Algorithm for Metric Bipartite Matching

Nikhil Bansal[1], Niv Buchbinder[2], Anupam Gupta[3], and Joseph (Seffi) Naor[4]

[1] IBM T. J. Watson Research Center, Yorktown Heights, NY 10598.
[2] Computer Science Department, Technion, Haifa, Israel.
[3] Department of Computer Science Carnegie Mellon University.
[4] Microsoft Research, Redmond, WA. On leave from the CS Dept., Technion, Haifa, Israel.

Abstract. We consider the online metric matching problem. In this problem, we are given a graph with edge weights satisfying the triangle inequality, and k vertices that are designated as the right side of the matching. Over time up to k requests arrive at an arbitrary subset of vertices in the graph and each vertex must be matched to a right side vertex immediately upon arrival. A vertex cannot be rematched to another vertex once it is matched. The goal is to minimize the total weight of the matching.
We give a $O(\log^2 k)$ competitive randomized algorithm for the problem. This improves upon the best known guarantee of $O(\log^3 k)$ due to Meyerson, Nanavati and Poplawski [19]. It is well known that no deterministic algorithm can have a competitive less than $2k - 1$, and that no randomized algorithm can have a competitive ratio of less than $\ln k$.

1 Introduction

Matching is one of the most fundamental and well-studied optimization problems and it has played a major role in the development of the theory of algorithms; see, e.g., [22] for the history as well as many details and algorithms. In this paper, we consider an *online version of the matching problem* which was first introduced by Khuller, Mitchell and Vazirani [14], and independently by Kalyanasundaram and Pruhs [9]. In this version, the input consists of an edge-weighted graph with k vertices designated as *right-hand side vertices*, or "servers". At each step, a new request vertex is designated as a *left-side vertex* or "client", appears and must be immediately matched to an available right-hand side vertex. The goal is to minimize the total cost of the matching.

It is easily seen that no online algorithm can be competitive if the edge-weights are allowed to be arbitrary, and hence a natural restriction is to consider the case when the edge-weights correspond to distances in a *metric space* and hence satisfy the triangle inequality. We call this problem the *online metric matching problem* . The problem arises naturally in many settings: for example, consider k fire stations, each of which can handle exactly one fire; when a new fire starts, an available fire station must be assigned to it immediately. Both Khuller, Mitchell and Vazirani [14], and Kalyanasundaram and Pruhs [9] gave $2k-1$ competitive deterministic algorithms for the online metric matching problem (and showed that no better deterministic algorithm is possible even for the star graph). Moreover, Kalyanasundaram and Pruhs also showed that the natural greedy

algorithm that matches a request to its closest available point is $(2^k - 1)$-competitive, and this bound is tight. The online metric matching problem was subsequently studied for special metric spaces, and also in models with resource augmentation; see Section 1.2 for more details and several other lines of related work.

It is not difficult to give a tight $\Theta(\log k)$-competitive *randomized* solution to the online metric matching problem on the star graph, which is also the bad example in the deterministic case (see, e.g.,Section 3.1 for both upper and lower bounds). Hence, a natural question is whether randomization could help obtain an exponential improvement for general metric spaces. In a recent breakthrough, Meyerson, Nanavati and Poplawski [19] give a randomized algorithm with an $O(\log^3 k)$-competitive ratio for general metrics; this is the first algorithm with a performance sublinear in k for general metric spaces. Their approach is to first use results on approximating general metrics by tree metrics [2, 3, 6] to obtain an online metric matching problem instance on a class of trees called $O(\log k)$-HSTs—these are trees where the edge lengths increase by a factor of $O(\log k)$ as one goes from the leaves to the root. Then, they solve the online metric matching problem on these HSTs with a competitive ratio of $O(\log k)$; since they lose $O(\log^2 k)$ in the reduction to the HSTs, the ultimate competitive ratio obtained is $O(\log^3 k)$. In the rest of this paper, we refer to the algorithm from [19] as the *MNP Algorithm*.

One natural approach to improve the competitive ratio is to show that the MNP Algorithm also works on α-HSTs for $\alpha = O(1)$; this would immediately remove one of the logarithmic terms from the competitive ratio. However, [19] sketch an example where running MNP on an α-HST with $\alpha = o(\log k)$ would result in an extremely poor competitive ratio; We discuss this issue further in Section 3.2.

1.1 Our results

Given that the MNP algorithm cannot be directly improved, we devise a new algorithm that proves our main technical result:

Theorem 1 (Upper Bound for 2-HSTs). *There is an $O(\log k)$-competitive algorithm for the online metric matching problem problem on 2-HSTs.*

Using standard results on approximating general metric spaces by HSTs (see Section 2), the above theorem immediately implies the following result on arbitrary metric spaces.

Corollary 1 (Upper Bound for Arbitrary Metrics). *There is an $O(\log^2 k)$-competitive algorithm for the online metric matching problem problem on arbitrary metric spaces.*

Our Techniques. Let us briefly discuss our techniques, as the proof of Theorem 1 requires a few conceptual steps. The first step shows that the natural greedy offline algorithm that repeatedly matches the closest (request,server) pair is optimal for HSTs. (This is not immediate, since the greedy algorithm is known to be bad even for a set of points on the line [21].) This analysis of the greedy algorithm allows us to lower bound the optimal cost of an instance.

The next step shows that we can imagine working in a more flexible model, where (some) server reassignments are permitted. In particular, if client c is previously assigned to server s (incurring a cost of $d(c, s)$) and a client c' arrives, we are allowed

to assign c' to s, and to reassign c to some s' incurring an additional cost of $d(c, s') + d(c', s)$), as long as the new server (i.e., s') for c is no closer to it than the old one (i.e., s). Given an online algorithm that works in this (restricted) reassignment model, we show how to get an online algorithm in the (no reassignment) original model with no greater cost. Finally, we give an online algorithm in this restricted-reassignment model which terminates with the greedy assignment, and where the total expected cost incurred during all the reassignments is at most $O(\log k)$ times the greedy cost, giving us the theorem.

1.2 Related Work

Apart from the initial work of Khuller, Mitchell and Vazirani [14], and Kalyanasundaram and Pruhs [9] on the online metric matching problem, there are several bodies of work related to our paper.

Special Metrics. The online metric matching problem has been studied for the case of special metrics as well. For example, the case of a line metric arises naturally at ski shops that have skis of various heights. As the skiers arrive one-by-one, the goal is to match skiers to skis such that the total mismatch between the length of the desired skis and the actual skis is minimized. The line case also generalizes the well-known "cow-path" problem [1]. Koutsoupias and Nanavati [15] showed that the Generalized Work Function algorithm of [10] is not constant competitive for the line metric. Fuchs et al. [7] showed a lower bound of 9.001 for the line metric, which implies that it is strictly harder than the cowpath problem. Kalyanasundaram and Pruhs [11] also considered the problem for general metrics when the adversary is allowed only half as many servers as the online servers. Recently, Chung, Pruhs and Uthaisombut [5] considered the case when the online algorithm is allowed *one extra server* at each point where the adversary has a server: somewhat surprisingly, they gave a deterministic algorithm that achieves a competitive ratio of polylog(k). Many variations of the problem have also been investigated: e.g., the minimum online bottleneck matching, the maximization version of the problem, and the case where the points are uniformly distributed on a disk in the Euclidean plane; for these, we refer the reader to the survey by Kalyanasundaram and Pruhs [10] and the references therein.

The k-Server Problem. The online metric matching problem is similar to the well-studied *k-server problem* introduced by Manasse, McGeoch and Sleator [17]: indeed, it can be viewed as a restricted case of the k-server problem where the location of the servers is fixed and not allowed to change. The methods used to give lower and upper bounds for the online matching problem, particularly on the uniform metric, are closely related to the results for k-server. Moreover, techniques used for the k-server problems, most notably work-function algorithms, have also been studied for the online metric matching problem [16, 15]. Throughout this paper, we will adopt the k-server view of the problem: given an underlying metric space and k "servers" with fixed locations, find the minimum cost matching between the servers and the requests arriving online.

Offline Heuristics. The *metric case* of the matching problem has also been studied offline in the context of finding fast and simple heuristics. Reingold and Tarjan [21] showed that greedily matching the closest (request, server) pair, and recursing on the

remaining instance gave an $O(k^{\log \frac{3}{2}})$-approximation to the min-cost matching, and that this bound was tight. A more sophisticated algorithm achieving an approximation bound of $O(\log k)$ was given by Plaisted [20]. Finally, Goemans and Williamson [8] constructed a primal-dual algorithm that achieves an $2 - 2/n$-approximation for the minimum-weight perfect matching problem.

The Online Bipartite Matching Problem. A different (and equally natural) version of the bipartite matching problem was proposed by Karp, Vazirani and Vazirani [13]. In their version of the problem, the right side of the bipartite graph is given in advance and the vertices on the left side (along with their incident edges) are revealed sequentially. Upon arrival, each vertex on the left-hand side must be matched to a right-hand side vertex (if possible); moreover, these decisions are irrevocable. Karp et al. considered the *unweighted* case of the problem, where the goal is to match as many vertices as possible, and gave an optimal $(1 - 1/e)$-competitive randomized algorithm. Note that in this version of the problem triangle inequality does not hold. A generalization to the online b-matching was considered in [12], and several extensions have recently been considered in the context of allocating ad-auctions in electronic markets [18, 4].

2 Preliminaries

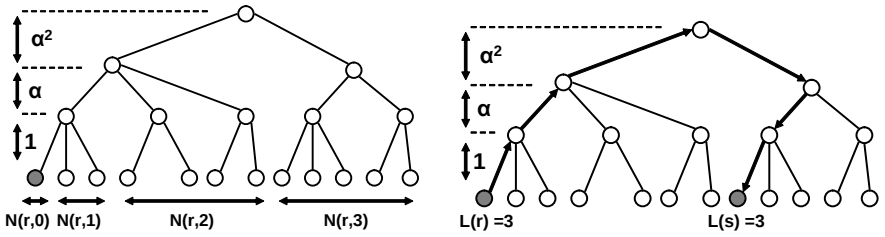
The *metric matching problem* is formally defined as follows. We are given a metric space (V, d). In addition, we are given a set of *requests* $R \subseteq V$ where $|R| = k$, and a set of *servers* $S \subseteq V$ with $|S| = k$; the sets R and S do not have to be disjoint. The objective is to find a minimum cost bipartite matching between the requests in R and the servers in S. In the *online* version of the problem, we know in advance the metric space and the server set S; however, the set R of requests arrive one-by-one in an online fashion. Upon arrival, a request must be immediately and irrevocably matched to some unmatched server in S; changing the assignment of a previously-matched request or server is not allowed. Let $R = \{r_1, r_2, \ldots, r_k\}$ be the set of requests according to their arrival order and let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of servers.

Hierarchically Well-Separated Trees (HSTs). Our results, as well as the previous results of [19], use a special type of tree metrics called *HSTs*. (See Figure 1 for an illustration.)

Definition 1 (HSTs). *Given a parameter $\alpha \geq 1$, an α-Hierarchically Well-Separated Tree (α-HST) is a rooted tree $T = (V, E)$ along with a length function d on the edges which satisfies the following properties:*

1. *For each node v, all the children of v are at the same distance from v.*
2. *For any node v, if $p(v)$ is the parent of v and $c(v)$ is any child of v, then $d(p(v), v) = \alpha \cdot d(v, c(v))$.*
3. *Each leaf has the same distance to its parent.*

We view an α-HST as a leveled tree, where all the leaves are at the same level, and the edge-lengths increase geometrically by a factor of α as we go up the tree from the leaves to the root.



(a) (b)

Fig. 1. Figure (a) shows an α-HST. For a request r it illustrates the set of nodes $N(r, \cdot)$. If request r is matched to server s as in Figure (b), then $L(r) = L(s) = 3$.

Let $\mathcal{H}$ be an α-HST with n leaves. For each leaf node i (that can be either request node or a server node), let $T(i, \ell)$ be the set of leaf nodes that are in the sub-tree of height ℓ that contains node i. Let $N(i, \ell)$ be the leaf nodes that are in the sub-tree $T(i, \ell)$ and not in sub-tree $T(i, \ell - 1)$. The distance of node i from each node in $N(i, \ell)$ is exactly $2\frac{(\alpha^\ell - 1)}{\alpha - 1}$. It is easy to verify that for each node i, the sets $N(i, \ell)$ induce a partition of the servers with respect to the node i. Given a matching in which request r is matched with server s we define $L(r)$ to be the level ℓ for which $s \in N(r, \ell)$. Similarly, let $L(s)$ be the level for which $r \in N(s, \ell)$. If server s is not matched yet we define $L(s) = \infty$. (The definitions are illustrated in Figure 1.)

Reducing from General Metrics to HSTs. The results of [6] imply that given any metric (V, d) on n points, there is a probability distribution on α-HST's with the following properties: **(a)** For each HST T in the support of the distribution, the leaves of T correspond to the nodes of V, and the distance in the tree $d_T(u, v) \geq d(u, v)$ for all $u, v \in V$, and **(b)** the expected distance $E[d_T(u, v)]$ between two nodes $u, v \in V$ is at most $O(\alpha \log n)d(u, v)$, where the expectation is taken over the random choice of the HST T. Furthermore, one can sample from this distribution in polynomial time.

Using this result, a β-competitive randomized algorithm for our problem on an α-HST directly implies an $O(\alpha\beta \log n)$-competitive randomized algorithm on the original metric. Note that a-priori, the number of nodes n in the metric space can be much larger than k. However, we can still replace $\log n$ by $\log k$ following an idea of Meyerson et al. [19]: we construct the HST only for the submetric induced on the k server nodes. Now, whenever a request arrives at some point p, we pretend that it has arrived at the server $s(p)$ closest to it, and handle it accordingly. Using the triangle inequality and the fact that $d(p, s(p))$ summed over all requests is a lower bound on the optimum solution, can change the competitive ratio by at most a constant factor.

3 Previous algorithms

In this section we describe several basic results and arguments, which will crucially be used in the rest of the paper. We also describe the randomized greedy algorithm which

is the basis of the previous $O(\log^3 k)$ result of [19]. We show that the analysis is in fact tight, and hence a different algorithm is necessary to obtain a better result.

3.1 The Uniform Metric

We begin by describing the $\Theta(\log k)$ lower and upper bounds for the uniform metric. Recall that the uniform metric consists of a set of points such that any two points are at unit distance from each other. For the lower bound, consider the uniform metric on $k+1$ points labeled $0, 1, \ldots, k$, and suppose that points $1, 2, \ldots, k$ (i.e., all except 0) contain one server each. The adversary places the first request at point 0. At each subsequent step for the next $k-1$ steps, the adversary requests a point that has not been requested thus far and is most likely to have a matched server. Note that just before the i^{th} request is made (for $i \geq 2$), there are $(k-i+2)$ unrequested points each containing a server, and one of these servers has already been matched. Hence the probability that the server at the requested point is already matched is at least $1/(k-i+2)$. Summing over all the requests, the expected cost incurred by any online algorithm is at least $1 + \sum_{i=2}^{k} 1/(k-i+2) = H_k = \Omega(\log k)$. The offline algorithm on the other hand only incurs a cost of 1.

We now show a matching upper bound. For a uniform metric (on, say $n \geq k$ points), each request is either collocated with a server, or else it is at unit distance from it. Consider the following algorithm inspired by the above lower bound: when a request arrives, if there a collocated server still available, we match the request to it; otherwise we choose an available server at unit distance uniformly at random and match the request to it. Consider an instance where u of the arriving requests are not collocated with a server (i.e., lie outside the set S). Clearly, the optimum offline algorithm has cost u; moreover, the online algorithm also pays cost for each such non-collocated request. Now consider the requests that arrive at points collocated with a server. Just before the i^{th} such request arrives, suppose there have already been u_i non-collocated requests. The crucial observation is that all the previous $i-1$ server locations where collocated requests arrived have already been matched (either to some collocated request or one of the u_i requests). Morever, for each of the remaining $k-i+1$ server locations, each of them is unavailable with probability exactly $u_i/(k-i+1)$, which is at most $u/(k-i+1)$. Hence, the i^{th} request has an expected cost of at most $u/(k-i+1)$; summing up over the non-collocated and the collocated requests, the total expected cost incurred is

$$u + \sum_{i=1}^{k-u} \frac{u}{k-i+1} \leq u + uH_k = O(u \log k).$$

3.2 The Randomized Greedy Algorithm

Meyerson et al. [19] considered the following simple randomized greedy algorithm: whenever a request arrives, match it to the nearest unmatched server. If there are several unmatched servers that qualify, choose one of these unmatched servers uniformly at random. In general metrics this approach can yield a competitive ratio as bad as $2^k - 1$ [9]. However, this algorithm performs quite well on α-HST metrics with large enough value of α. Specifically, Meyerson et al. analysed this algorithm on an α-HST with

$\alpha \geq 2\ln k + 1$, and proved it is $O(\log k)$-competitive. This immediately implied an $O(\alpha \log^2 k) = O(\log^3 k)$ competitive algorithm for general metrics.

An appealing approach to improve the competitive factor is to try to remove the requirement that $\alpha \geq 2\ln k + 1$. Meyerson et al. showed that $\alpha = \Omega(\log k)$ is necessary for the randomized greedy algorithm to work. Specifically, it is possible to prove the following Lemma:

Lemma 1 ([19]). *For any constant ℓ, there exists an ℓ level α-HST and an input instance with optimal cost $O(\alpha^{\ell-1})$, such that the MNP Algorithm incurs a cost of $\Omega(\sum_{i=0}^{\ell-1}(\log k)^{i+1}\alpha^{\ell-i-1})$.*

We remark that it is easily checked that if $\alpha = o(\log k)$, then the online cost is substantially larger than the offline cost. A close inspection of the lower bound example reveals that the lower levels have a disproportionately higher contribution to the online cost as compared with the offline algorithm. The main problem is that the MNP algorithm incurs too much cost in the lower levels until it realizes that there are no available servers in a subtree and that it needs to find a server outside the subtree. The lower bound example motivates a different approach that is used by our new constructed algorithm.

4 An $O(\log k)$ Algorithm for 2-HST's

In this section we present a simple online algorithm which is $O(\log k)$-competitive on an α-HST, for any constant value of α. For simplicity we set $\alpha = 2$. Our algorithm has three conceptual steps. First, in Section 4.1 we describe a simple offline algorithm that computes an optimal matching on an HST. In Section 4.2 we define a restricted reassignment online model which is easier to handle. We then prove that we can obtain an online algorithm with *no reassignments* from any online algorithm in the restricted reassignment model without compromising the competitive ratio. Finally, in Section 4.3, we design a simple online algorithm in the restricted reassignment model and prove that it is $O(\log k)$-competitive.

4.1 An Offline Algorithm

In this section we design a simple **offline** algorithm that computes an optimal solution on an HST metric. This algorithm is essentially the greedy approach that matches the closest request-server pair, and then recurses on the remaining instance. Reingold and Tarjan [21] proved that this approach leads to a very poor $\theta(k^{\log \frac{3}{2}})$-approximation in general metrics. However, we show here that this greedy approach leads to an optimal solution in the special case of HST metrics.

1. Let R and S be the current sets of unmatched requests and unmatched servers, respectively. Initially, R and S contain all requests and servers.
2. Iterate on the levels from level $\ell = 0$ and up until the highest level.
3. Iterate on the requests $r \in R$ in **any order**.
4. For each request r, if $N(r, \ell) \cap S \neq \emptyset$ match r to **any server** in $N(r, \ell) \cap S$ and remove r and s from R and S, respectively. Otherwise, continue to the next request in R.

We refer to this algorithm as the *Generic Algorithm*, since it considers the requests in arbitrary order, and the server it chooses for each request r from the set $N(r, \ell) \cap S$ is also arbitrary. Thus, the algorithm is ***flexible*** with respect to these choices, meaning that the output of the algorithm is not unique. This property of the algorithm will be very important in the sequel. We say that a matching M is ***feasible*** with respect to the Generic Algorithm if there exists a run of the algorithm that can generate M. The next Lemma proves that the algorithm outputs an optimal matching on any HST metric.

Lemma 2. *The Generic Algorithm generates an optimal matching on an HST metric.*

Proof. Consider a subtree T_i rooted at a node i at height ℓ. Let $R(i)$ and $S(i)$ be the number of requests and the number of servers in T_i respectively. Clearly, any solution must match at least $E(i) = \max(0, R(i) - S(i))$ requests belonging to T_i to servers that lie outside T_i. These requests must incur a cost of $2\alpha^{\ell} \cdot E(i)$ when going up from level ℓ to level $\ell + 1$ (node i's parent) and then coming down from level $\ell + 1$ back to level ℓ. Thus, the optimum cost OPT(T) on the whole HST is at least $\sum_{i \in T} 2\alpha^{\ell(i)} E(i)$, where the summation is over all nodes i of T, and $\ell(i)$ denotes the level of i.

Now consider the behavior of our Generic Algorithm. When it first considers level ℓ, in each subtree T_i rooted at level ℓ, no server in T_i can be occupied by a request outside T_i. Moreover, at Step (3), the unsatisfied requests in T_i are matched within T_i as much as possible. Thus, exactly $E(i)$ requests remain unmatched in T_i after level ℓ is processed, and hence, by the same reasoning as above, the matching produced has cost exactly $\sum_{i \in T} 2\alpha^{\ell(i)} E(i)$.

4.2 A Restricted Reassignment Online Model

In this section we define a different online model and prove that it suffices to design a competitive online algorithm for this modified model. We refer to the new model as the *restricted reassignment online model*; as the name suggests, this model allows some reassignment of previously arrived requests. Specifically, in the new model we are allowed to reassign a previously matched request r_p with the following restriction: if, currently, r_p is matched to a server belonging to $N(r_p, \ell)$ for some value ℓ, then the algorithm is allowed to reassign r_p only to a server belonging to $N(r_p, \ell')$, where $\ell' \geq \ell$. The online algorithm in the restricted reassignment model pays the cost of **all** reassignments performed, and not just the cost of the final matching computed.

We claim that any online algorithm in the restricted reassignment model can be transformed to an online algorithm in the original model (where no reassignments are allowed) with no additional cost. This is done by a very simple method. First, we can assume without loss of generality that the algorithm in the new model is lazy and does not reassign requests unnecessarily. That is, it only reassigns a request if a currently occupied server by it must be used to match another request. Consider a move of the algorithm, where r is matched to s_1 that was previously matched to r_1. Request r_1 is then matched to server s_2 which was previously matched to r_2, and so on, until request r_t which was previously matched to s_t is reassigned to a vacant server s_{t+1}. The change in the matching is viewed in the following:

$$\underbrace{s_1 \leftarrow r_1, s_2 \leftarrow r_2, \ldots, s_t \leftarrow r_t}_{\text{The original matching}} \Rightarrow \underbrace{r \rightarrow s_1, r_1 \rightarrow s_2, \ldots, r_{t-1} \rightarrow r_t, r_t \rightarrow s_{t+1}}_{\text{The matching after the reassignment process}}$$

The cost of reassigning the requests in the new model is $d(r, s_1) + \sum_{i=1}^{t} d(r_i, s_{i+1})$. An algorithm in an online model with no reassignments would simulate this move by simply matching r directly to s_{t+1}, paying a cost of $d(r, s_{t+1})$. The following lemma shows that the total cost of this algorithm with no reassignments is no more than the cost incurred in the restricted reassignment model.

Lemma 3. *In any iteration of the algorithm:* $d(r, s_{t+1}) \leq d(r, s_1) + \sum_{i=1}^{t} d(r_i, s_{i+1})$

Proof. The claim follows directly from the restrictions on the reassignments in the new model. Assume that r_i was matched to a server s_i in level ℓ. Thus, r_i and s_i both belong to the tree $T(r_i, \ell)$. By the restriction on the reassignments, r_i cannot be reassigned to a server in $T(r_i, \ell-1)$. Thus, the path from r_i to server s_{i+1} passes through the root of the tree $T(r_i, \ell)$. Therefore, the path from s_i (that is, in the sub-tree $T(r_i, \ell)$) to s_{i+1} is at most $d(r_i, s_{i+1})$. Thus, we get that for any $i \in \{1, 2, \ldots, t\}$, $d(s_i, s_{i+1}) \leq d(r_i, s_{i+1})$. Using the triangle inequality we get that:

$$d(r, s_{t+1}) \leq d(r, s_1) + \sum_{i=1}^{t} d(s_i, s_{i+1}) \leq d(r, s_1) + \sum_{i=1}^{t-1} d(r_i, s_{i+1})$$

4.3 An $O(\log k)$-Competitive Algorithm in the Restricted Reassignment Model

In this section we present an $O(\log k)$-competitive algorithm for the online metric matching in the restricted reassignment model. The algorithm is as follows:

Initially, set $L(s) = \infty$ for all servers in S.
When request r arrives, set $L(r) = 0$:

1. Find the lowest level $\ell \geq L(r)$ in which there exists a server $s \in N(r, \ell) \cap S$ such that $L(s) > \ell$.
2. Choose uniformly at random a server s among the servers in $N(r, \ell) \cap S$ for which $L(s) > \ell$.
3. Match r to s (and set $L(r) = L(s) = \ell$).
4. If s was previously matched to another request r', then reassign r' using the same procedure (return to Step (1) with r').

Note that Step (1) above ensures that for each request r, its level $L(r)$ can only increase during the execution of the algorithm. Thus, the algorithm satisfies the requirement of the restricted reassignment model. Suppose the arrival of a new request r causes the reassignment of requests $r_1, r_2, \ldots, r_p$. Then, the number of reassignments is at most the height of the HST, since for all $i < p$, $L(r_i) < L(r_{i+1})$. Also, by the condition in Step (1) above, the level $L(s)$ of each server s can only decrease during the execution of the algorithm. The next lemma shows that the final solution produced by the algorithm is optimal (without taking into account the cost of reassignments).

Lemma 4. *The final matching produced by the online algorithm is optimal.*

Proof. We show that the solution produced by the online algorithm is feasible with respect to the offline Generic Algorithm of Section 4.1. Thus, by Lemma 2 the solution is optimal.

Consider the final matching produced upon termination of the online algorithm. Let $R_i \subseteq R$ be the set of requests such that $L(r) = i$. We show how to obtain the final online solution using the Generic Algorithm. In the first round we match all the servers in R_0 to the servers that are used by the online algorithm. In the second round we match the requests in R_1 to the servers used by the online algorithm, and so on. It suffices to show that this corresponds to a feasible run of the Generic Algorithm. To this end, it is enough to show that for any i, after having matched the subset R_i, we cannot match any request $r \in R \setminus \left(\bigcup_{j=0}^{i} R_i\right)$ to servers in level i with respect to r.

Assuming that this is not true, then after having matched the requests in R_i, there still exists an unmatched request r that can be matched with a server $s \in N(r, i)$. Consider the online iteration in which request r had $L(r) \leq i$ and was matched (or reassigned) to a server in a level higher than i. Such an iteration must exist as $L(r)$ starts from zero and the request r is eventually matched to a level higher than i. Moreover, in this iteration, $L(s) > i$, since upon termination of the algorithm $L(s) > i$ (none of the requests from the subset $\bigcup_{j=1}^{i} R_j$ is matched to s), and the level $L(s)$ of each server can only decrease during the execution of the algorithm. Therefore, at this iteration, request r could have been matched with a server in level i (with respect to r). This contradicts the fact that in that iteration request r chose to be matched with a server with a level strictly larger than i.

Lemma 5. *The expected cost of the reassignments of a request r which is matched upon termination of the online algorithm to a server s (at distance $d(r, s)$) is $O(\log k)d(r, s)$.*

Proof. Consider a request r which is matched upon termination of the online algorithm to a server s at level $L(r) = L(s)$. During the execution of the algorithm, $L(r)$ is monotonically non-decreasing. Consider a level ℓ, $0 \leq \ell \leq L(r)$. We prove that the expected number of times r is matched (or reassigned) to servers in $N(r, \ell)$ is $O(\log N(r, \ell)) = O(\log k)$. The intuition is the following. During the execution of the algorithm, request r can cause a reassignment of request r' only if r' is matched to a server s' which is strictly closer to r (i.e. $s' \in N(\ell, r)$, $s' \in N(\ell', r')$ and $\ell < \ell'$). In this case we say that r is *stronger* than r'. Request r' then chooses a new server which is at least at the same distance from r' as s' and is not occupied by any request which is at least as strong as r'. The set of servers satisfying the latter condition is the feasible set for r' and its size is monotonically (strictly) decreasing over the execution of the algorithm. The main observation is that r' always chooses uniformly at random a new server from the set of feasible servers. Thus, the probability that the next request which is stronger than r' will cause another reassignment of r' is at most the inverse of the size of the set of feasible servers for r'. This gives us the harmonic number as an upper bound on the expected number of reassignments, similarly to the uniform metric case.

Formally, fix any sequence of requests. Assume that at some time during the online algorithm request r is matched to a server in $N(r, \ell)$ (if there is no such iteration then we are done). Next, define the set $W \subseteq N(r, \ell)$ of feasible matchings for request r to be the set of servers in $N(r, \ell)$ for which $L(s) > \ell$. The size of W can only decrease throughout the execution of the algorithm. Next, consider the arrival order of requests which are at least as strong as r and are matched to servers in W (until either request r is matched to a higher level or until the end of the execution). If some request having

the same strength as r arrives the probability that it causes a reassignment of r is zero. Otherwise, since in each reassignment, the request r chooses uniformly at random a server among the remaining servers in W, the probability that such a request causes r to be reassigned is at most $\frac{1}{|W|}$, where $|W|$ is the current size of the set of feasible servers for r. In either case the size of W decreases by 1 after each such arrival. Since initially $|W| \leq |N(r,\ell)|$ (similarly to the uniform metric), it follows that the expected number of reassignments until r is matched to a higher level, or is matched to its final server, is $O(H_{N(r,\ell)}) = O(\log |N(r,\ell)|) = O(\log k)$.

Since the metric is a 2-HST, the cost of each reassignment of request r in level i costs $2(2^i - 1)$. Therefore, the total expected cost of reassigning request r is at most: $O(\log k) \sum_{i=0}^{L(r)} 2(2^i - 1) = O(\log k) 2^{L(r)} = O(\log k) d(r,s)$, where the last inequality follows since $d(r,s) = 2(2^{L(r)} - 1)$.

Theorem 2. *The online algorithm is $O(\log k)$ competitive on HST's.*

Proof. By linearity of expectation and Lemma 5 the total expected cost of the algorithm is $O(\log k)$ times the final cost of the solution of the online algorithm. By Lemma 4 the final solution is optimal and thus the total expected cost of the algorithm is actually $O(\log k)$ times the optimum. Finally, by Lemma 3 we can translate this algorithm easily to the model with no reassignments without increasing the cost.

5 Conclusions

In this paper, we designed an algorithm for the online metric matching problem which is $O(\log k)$-competitive on 2-HST's, and thus $O(\log^2 k)$-competitive for general metrics. The main open question is to design an algorithm with competitive ratio $O(\log k)$, or to improve the known lower bound. The analysis of such an algorithm cannot proceed along the same lines as we pursue, since approximating general metrics by an HST incurs a loss of $\Omega(\log k)$ in the worst case, and moreover, there is an $\Omega(\log k)$ lower bound for the online metric matching problem even on HSTs.

Interestingly, the claims in Sections 4.2 and 4.3 can be extended easily to general metrics, which implies that our algorithm is $O(\log k)$-competitive for those metrics on which the greedy approach of Section 4.1 generates a constant factor approximation. However, since the lower bound of Reingold and Tarjan [21] shows the existence of metrics for which the greedy approach produces an $\Omega(k^{\log \frac{3}{2}})$-approximate solution, applying our algorithm directly on such metrics might be as bad as $\Omega(k^{\log \frac{3}{2}} \log k)$-competitive. Nonetheless, a possible direction to obtain an $O(\log k)$-competitive algorithm for general metrics might be to combine our techniques in Sections 4.2 and 4.3 with a different offline heuristic, that results in an $O(1)$-approximation. It is somewhat strange to look for an $O(1)$-approximation for a problem that is in polynomially solvable. However, such an algorithm that takes advantage of the metric properties of the graph might be a key ingredient. A possible starting point can be the "hyper-greedy" heuristic of Supowit, Plaisted, and Reingold [23] that achieves an $O(\log k)$-approximation to the offline metric matching problem, or the $2 - 2/n$ primal dual approximation algorithm of Goemans and Williamson [8].

References

1. R. A. Baeza-Yates, J. C. Culberson, and G. J. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.
2. Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *IEEE Symposium on Foundations of Computer Science*, pages 184–193, 1996.
3. Y. Bartal. On approximating arbitrary metrics by tree metrics. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1998.
4. N. Buchbinder, K. Jain, and J. Naor. The adauctions problem and extensions. To Appear in ESA 2007.
5. C. Chung, K. Pruhs, and P. Uthaisombut. The online transportation problem: On the exponential boost of one extra server. Under submission.
6. J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC '03:*, pages 448–455, 2003.
7. B. Fuchs, W. Hochstattler, and W. Kern. Online matching on a line. *Theoretical Computer Science*, pages 251–264, 2005.
8. M.X. Goemans, D.P. Williamson. A General Approximation Technique for Constrained Forest Problems. *SIAM Journal on Computing*, 24, 296–317, 1995.
9. B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993.
10. 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.
11. B. Kalyanasundaram and K. Pruhs. The online transportation problem. *SIAM J. Discrete Math.*, 13(3):370–383, 2000.
12. B. Kalyanasundaram and K. Pruhs. An optimal deterministic algorithm for online b - matching. *Theoretical Computer Science*, 233(1–2):319–325, 2000.
13. R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for online bipartite matching. In *In Proceedings of the 22nd Annual STOC*, pages 352–358, 1990.
14. 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.
15. E. Koutsoupias and A. Nanavati. The online matching problem on a line. In *WAOA03*, pages 179–191, 2003.
16. E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.
17. M. S. Manasse, L. McGeoch, and D. D. Sleator. Competitive algorithms for online problems. In *STOC: ACM Symposium on Theory of Computing*, pages 322–333, 1988.
18. A. Mehta, A. Saberi, U. V. Vazirani, and V. V. Vazirani. Adwords and generalized on-line matching. In *IEEE Symposium on Foundations of Computer Science*, pages 264–273, 2005.
19. A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *SODA '06:*, pages 954–959, 2006.
20. D. A. Plaisted. Heuristic matching for graphs satisfying the triangle inequality. *J. Algorithms*, 5(2):163–179, 1984.
21. E. M. Reingold and R. E. Tarjan. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10(4):676–681, 1981.
22. A. Schrijver. *Combinatorial optimization. Polyhedra and efficiency.*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003.
23. K. J. Supowit, E. M. Reingold, and D. A. Plaisted. The travelling salesman problem and minimum matching in the unit square. *SIAM J. Comput.*, 12(1):144–156, 1983.