A Robust and Optimal Online Algorithm for Minimum Metric Bipartite Matching

Sharath Raghvendra*

Dept. of Computer Science, Virginia Tech, Blacksburg, USA
sharathr@vt.edu

Abstract

We study the Online Minimum Metric Bipartite Matching Problem. In this problem, we are given point sets S and R which correspond to the server and request locations; here $|S| = |R| = n$. All these locations are points from some metric space and the cost of matching a server to a request is given by the distance between their locations in this space. In this problem, the request points arrive one at a time. When a request arrives, we must immediately and irrevocably match it to a "free" server. The matching obtained after all the requests are processed is the *online* matching M. The cost of M is the sum of the cost of its edges. The performance of any online algorithm is the worst-case ratio of the cost of its online solution M to the minimum-cost matching.

We present a deterministic online algorithm for this problem. Our algorithm is the first to simultaneously achieve optimal performances in the well-known adversarial and the random arrival models. For the adversarial model, we obtain a competitive ratio of $2n - 1 + o(1)$; it is known that no deterministic algorithm can do better than $2n - 1$. In the random arrival model, our algorithm obtains a competitive ratio of $2\mathrm{H}_n - 1 + o(1)$; where H_n is the nth Harmonic number. We also prove that any online algorithm will have a competitive ratio of at least $2\mathrm{H}_n - 1 - o(1)$ in this model.

We use a new variation of the offline primal-dual method for computing minimum cost matching to compute the online matching. Our primal-dual method is based on a relaxed linear-program. Under metric costs, this specific relaxation helps us relate the cost of the online matching with the offline matching leading to its robust properties.

1998 ACM Subject Classification G.2.2 [Graph Theory] Graph Algorithms

Keywords and phrases Online Algorithms, Metric Bipartite Matching

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2016.18

1 Introduction

In an era of instant gratification, consumers desire speedy access to goods and services. Several new business ventures promise on-demand delivery of such services to consumers. Typically, these ventures have servers in various locations of the city and when a new request arrives, they match one of the available servers to this request. The cost associated with this match is often a metric cost; for instance, it could be the minimum distance traveled by the server to reach the request. A primary objective is to minimize the overall cost of the assignments made. This problem is difficult because all of the request locations are not known in advance.

Each server may have a maximum capacity of how many requests it can serve. A central problem in online algorithms is the *k-server* problem where each of the k servers has a

* The author would like to acknowledge the support of NSF CRII grant CCF-1464276 in conducting this research.

capacity to serve an arbitrary number of requests. In the case where the server capacity is 1, the problem reduces to the well-studied *online minimum metric bipartite matching* problem.

To define this problem, let S be a set of servers and let R be a set of requests. We assume that the locations in $S \cup R$ are points from some metric space. Let $\mathtt{d}(a, b)$ be the distance between any two points in this space. Consider a complete bipartite graph $G(S \cup R, S \times R)$, $|S| = |R| = n$, with the edge set $S \times R$. Every pair of server and request, $(s, r) \in S \times R$ has a distance $\mathtt{d}(s, r)$. We refer to this distance as the *cost* of server s serving request r. A *matching* $M \subseteq S \times R$ is any set of vertex-disjoint edges of the bipartite graph $G(S \cup R, S \times R)$. The cost of any matching M is given by $w(M) = \sum_{(s,r)\in M} \mathtt{d}(s, r)$. A *perfect matching* is a matching where every server in S is serving exactly one request in R, i.e., $|M| = n$. A *minimum-cost perfect matching* is a perfect matching with the minimum cost.

Ideally, we would like to match servers to requests so that the cost of this matching is as small as possible. However, in the *online metric bipartite matching problem*, the requests arrive one at a time and when any request arrives, we have to immediately and irrevocably match it to some unmatched server. The resulting matching is referred to as an *online matching*. Designing an online algorithm which finds a matching with minimum-cost is impossible because, for any partial assignment made by the algorithm, an adversary can easily fill up the remaining request locations in R so that this partial assignment becomes sub-optimal. Therefore, we want our algorithm to compute an online matching which is only near optimal. For any input S, R and any arrival order of requests in R, we say our algorithm is α-*competitive*, for $\alpha > 1$, when the cost of the online matching M is at most α times the minimum cost, i.e.,

$$w(M) \leq \alpha w(M_{\text{OPT}}).$$

Here M_{OPT} is the minimum-cost matching of the locations in S and R.

In the above discussion, note the role of the adversary. In the *adversarial model*, the adversary knows the server locations and the assignments made by the algorithm and generates a sequence to maximize α. In order to account for adversarial input sequence, algorithms which work well in this model may become very cautious in making low-cost assignments. As a result, their performance may be hampered on realistic input sequences.

A less pessimistic model is the *random arrival model*. In this model [13], the adversary chooses the set of request locations R at the start but the arrival order is a permutation chosen uniformly at random from the set of all possible permutations; we refer to this as a *random permutation*. For any input S, R and an arrival order which is a random permutation of R, we say our algorithm is α-*competitive*, for $\alpha > 1$, when the expected cost of the online matching M is at most α times the minimum cost, i.e.,

$$\mathbb{E}[w(M)] \leq \alpha w(M_{\text{OPT}}).$$

In practical situations, one can assume that the requests locations are independent and identically distributed (i.i.d.) random variables from an unknown but fixed distribution $\mathscr{D}$. On many occasions, using historical data, one can learn this distribution $\mathscr{D}$. These KnownIID and UnknownIID models are weaker than the random arrival model. Therefore, the competitive ratio of an algorithm in the random arrival model is an upper bound on its performance in the KnownIID and the UnknownIID models; see [9] for algorithms in these models.

Another popular model of theoretical interest is the *oblivious adversary* model. In this model, the adversary knows the algorithm and decides the request locations and their arrival order. However, the online algorithm is a randomized algorithm and the adversary does not

know the random choices made by the algorithm. This model is weaker than the adversarial model but stronger than the random arrival model.

Below is the summary of relative hardness of all these models

$$\text{Adversarial} \succ \text{Oblivious Adversary} \succ \text{Random Arrival} \succ \textsc{UnknownIID} \succ \textsc{KnownIID}.$$

Existing Work. Solutions for the k-server problem and the online bipartite matching problem use similar mathematical tools and methodologies. Both of these problems have been extensively studied in the adversarial model and the oblivious model. However, we are not aware of any work on these problems under the random arrival model.

The k-server problem is central to the theory of online algorithms. The problem was first posted by Manasse *et al.* [14]. In the adversarial model, the best-known deterministic algorithm for this problem is the $2k-1$-competitive work function algorithm [12]. In this problem, we assume there are k servers, each of which can serve arbitrary many of the n arriving requests. It is known that no deterministic algorithm can achieve a competitive ratio better than k and is conjectured that in fact there is a k-competitive algorithm for this problem. This conjecture is popularly called the *k-server conjecture*.

For the online metric bipartite matching problem, in the adversarial model, there is a $2n-1$-competitive deterministic algorithm by Khuller *et al.* [10] and Kalyanasundaram and Pruhs [8]. They also show that no online algorithm can achieve a better competitive ratio in this model.

For the oblivious adversary, there are $O(\text{poly}\log n)$-competitive algorithms for both the k-server problem and the online metric bipartite matching problem. Bansal *et al.* [5] achieve an $O(\log^2 n)$-competitive algorithm for the metric bipartite matching problem. For the k-server problem, Bansal *et al.* [4] presented a $O(\text{poly}\log n \log k)$-competitive algorithm.

All of these algorithms use a standard approach. They first embed the metric space into a tree metric that leads to a $O(\log n)$ distortion in costs. Then, they design a $\log n$-competitive algorithm for this tree metric. As a consequence, these results obtain a $\log^2 n$-competitive algorithm ($\text{poly}\log n$-competitive for the k-server). The bottleneck in improving existing work is the $O(\log n)$-distortion associated with the tree metric. An open question is whether one can design an $O(\log n)$-competitive algorithm for these problems.

Also note that for bounded doubling dimension metric, there is an a $O(d \log n)$-competitive algorithm in the oblivious model [6]; here d is the doubling dimension of the metric space. In the adversarial model, the question of finding a deterministic $O(1)$-competitive online algorithm for the line metric remains an important open question; see [3, 11] for results on this special case. We would like to note the existence of several fast primal-dual algorithms to compute approximate (offline) matching in metric and geometric settings [7, 2, 16, 17, 1].

Our Results. In this paper, we give a robust deterministic online algorithm for the metric bipartite matching problem. Our algorithm achieves an optimal performance of $2n-1$ in the adversarial model. This same algorithm has an exponentially better performance in the random arrival model where we obtain optimal $2\mathrm{H}_n - 1 + o(1)$-competitive ratio. Here H_n is the nth harmonic number (approximately $\ln n$). We also prove that no algorithm can achieve a competitive ratio better than $2\mathrm{H}_n - 1 - o(1)$.

To our knowledge, this is the first online algorithm which achieves optimal performances in two different models simultaneously. Our algorithm's robustness across different models of adversaries is also crucial in practical settings where there is limited information about the model of adversary.

Unlike previous work, our approach does not use a tree metric, thereby allowing us to achieve a $O(\log n)$-competitive algorithm in the random arrival model.

Our Techniques. Many deterministic online algorithms use the best minimum-cost offline solution to construct the online solution. This includes the work-function algorithm for the k-server problem and the deterministic algorithms for the online minimum metric bipartite matching problem. Our approach is similar in style, except we use, for some $t > 1$, a t-approximate minimum-cost solution to guide our online solution. This approximate matching is derived from a relaxed linear program where the constraint for every non-matching edge is relaxed by a multiplicative factor of t. We will fix the value of t for the entire execution of the algorithm. When a new request arrives, our algorithm updates the offline matching by computing an augmenting path P. For the online matching, the algorithm simply matches the two free end points of P.

We observe that for larger values of t, the algorithm picks an augmenting path P of a smaller length leading to lower cost of the online matching. On the other hand, larger values of t causes the offline matching to be a weaker approximation which leads to a weaker bound for the online matching. Therefore, it may seem that the best trade-off between these opposing observations is achieved at some finite value of t.

However, surprisingly, we show that the performance of our online algorithm improves as $t \to \infty$. We show that the competitive ratio of our algorithm is $(2 + \frac{2}{t-1})n - (1 + \frac{2}{t-1})$ in the adversarial model and $(2 + \frac{2}{t-1})\mathrm{H}_n - (1 + \frac{2}{t-1})$ in the random arrival model.

2 Preliminaries

In this section, we present preliminary notations required to describe our algorithm.

Given a matching M^* on this bipartite graph, an *alternating path* (or cycle) is a simple path (resp. cycle) whose edges alternate between those in M^* and those not in M^*. We refer to any vertex that is not matched in M^* as a *free vertex*. An *augmenting path* P is an alternating path between two free vertices. We can *augment* M^* by one edge along P if we remove the edges of $P \cap M^*$ from M^* and add the edges of $P \setminus M^*$ to M^*. After augmenting, the new matching is given by $M^* \leftarrow M^* \oplus P$, where $\oplus$ is the symmetric difference operator. For a parameter $t \geq 1$, we define the *t-net-cost* of an augmenting path P as follows:

$$\phi_t(P) = t \left(\sum_{(s,r) \in P \setminus M^*} \mathtt{d}(s,r) \right) - \sum_{(s,r) \in P \cap M^*} \mathtt{d}(s,r)\,.$$

When $t = 1$, we can interpret the t-net-cost of a path as the increase in the cost of the matching due to augmenting it along P, i.e., for $t = 1$, $\phi_1(P) = w(M \oplus P) - w(M)$. The well-known Hungarian method iteratively augments along an augmenting path with the minimum 1-net-cost to compute the optimal matching. For $t > 1$, the t-net-cost $\phi_t(P)$ can be very different from $w(M \oplus P) - w(M)$. As noted earlier, larger values of t yields smaller length augmenting paths. In Figure 1, there are two augmenting paths. Let the augmenting path from r to s' be P' and the augmenting path from r to s be P. When $t = 1$, the $\phi_1(P) = \phi_1(P') = 1$. However, $\phi_2(P') = 3$ and $\phi_2(P) = 4$. As t increases, the difference between the t-net-costs of these paths is magnified.

We derive an alternate interpretation of the t-net-cost in Section 3 which will be crucial in providing guarantees for our online solution. The definition of t-net-cost easily extends to alternating paths and cycles as well.

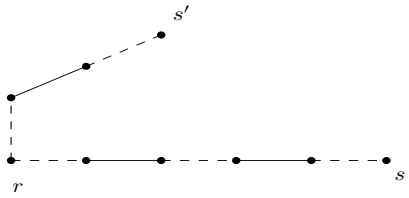

Figure 1 All solid edges are edges in the matching M^* and the dashed edges are not in M^*. The cost of every edge is 1.

Feasibility of a Matching. For every vertex v of the graph $G(S \cup R, S \times R)$, let its dual weight be $y(v)$. For a parameter $t \geq 1$, we define a *t-feasible* matching to be a matching M^* and a set of dual weights $y(\cdot)$ on the vertex set such that for every edge between request $r \in R$ and server $s \in S$, we have

$$y(s) + y(r) \leq t\mathtt{d}(s, r), \tag{1}$$

$$y(s) + y(r) = \mathtt{d}(s, r) \quad \text{for } (s, r) \in M^*. \tag{2}$$

Using t-feasibility and t-net-cost we describe our algorithm next.

2.1 Algorithm

In this section, we present our algorithm without fixing the parameter t. Eventually, to obtain the bounds on the competitive ratio, we set $t = n^2 + 1$.

For every vertex $v \in S \cup R$, our algorithm will maintain a dual weight $y(v)$. At the start of the algorithm, the dual weight of every vertex is set to 0. Recollect that, in the online setting, requests from the set R arrive one at a time. For those requests $r' \in R$ which have not yet arrived, their dual weight remains 0, i.e., $y(r') = 0$. The algorithm also maintains two matchings M and M^*; both these matchings are initialized to $\emptyset$ at the start. M and M^* match all the request seen so far to servers in S. The matching M^* together with the dual weights $y(\cdot)$ is a t-feasible matching; we refer to this as the *offline matching*. The matching M, on the other hand, is the *online matching*.

Algorithm. Given a new request r, our algorithm computes the minimum t-net-cost augmenting path P with respect to matching M^*. P starts at r and ends at some free vertex s. The algorithm updates M^* by augmenting it along P, i.e., $M^* \leftarrow M^* \oplus P$. For the online matching M, the algorithm will match the server s to r, i.e., $M \leftarrow M \cup \{(s, r)\}$.

At any given stage in the algorithm, let S_F be the set of free servers in S with respect to the offline matching M^*. It follows from the description of our algorithm that S_F is also the set of free servers with respect to the online matching M.

Our algorithm maintains the following invariants:

(I1) M^* and dual weights $y(\cdot)$ form a t-feasible matching, and,

(I2) for every vertex $s \in S$, $y(s) \leq 0$ and if $s \in S_F$, $y(s) = 0$.

Next, we present an $O(n^2)$ time algorithm to compute the minimum t-net-cost augmenting path P and update the matchings M, M^* and the dual weights. After describing the algorithm, we prove the invariants (I1) and (I2).

Algorithm Details. To compute a minimum t-net-cost augmenting path P with respect to the offline matching M^*, we construct a *weighted residual graph* G_{M^*} with $S \cup R$ as the vertex set and E_{M^*} as the set of edges as follows. Let $\overrightarrow{sr}$ represent an edge directed from s to r. For every edge $(s, r) \in M^*$, we have an edge $\overrightarrow{sr}$ in E_{M^*}. For every edge $(s, r) \in (S \times R) \setminus M^*$, there is an edge $\overrightarrow{rs}$ in E_{M^*}. Every edge $\overrightarrow{ab} \in E_{M^*}$ is assigned a cost as follows:

- If $(a, b) \in M^*$, we set the cost of the edge to be the slack $s(a, b) = \mathtt{d}(a, b) - y(a) - y(b)$. From t-feasibility (condition (2)) of M^*, we know the slack of every edge in the matching is $s(a, b) = 0$.
- If $(a, b) \notin M^*$, we set the cost of the edge (a, b) to be the slack $s(a, b) = t\mathtt{d}(a, b) - y(a) - y(b)$. From t-feasibility (condition (1)) of M^*, we know $s(a, b) \geq 0$.

By construction, every edge in E^*_M has a non-negative edge cost. Also, notice that the set of nodes in $G(S \cup R, S \times R)$ (henceforth referred to as G) and G_{M^*} are identical. G_{M^*} and G have the same set of edges except that the edges of G_{M^*} have directions. For any directed path $\overrightarrow{P}$ in G_{M^*}, we can define its *associated path* P by replacing every edge $\overrightarrow{ab} \in \overrightarrow{P}$ with the corresponding undirected edge (a, b) from G. For any directed path $\overrightarrow{P}$ in G_{M^*}, its associated path P is an alternating path in G. More so, if the two end vertices of $\overrightarrow{P}$ are free vertices, then the associated path P will be an augmenting path.

To compute the minimum t-net-cost augmenting path P, we simply execute Dijkstra's algorithm and find the minimum-cost path from r to every other node in G_{M^*}. For any node v, let d_v be the cost of the shortest path from r to v. Among all free servers of S, we pick $s \in S_F$ with the lowest minimum-cost path from r in G_{M^*}, i.e., $s = \arg\min_{s' \in S_F} d_{s'}$. Let this lowest minimum-cost path be $\overrightarrow{P}$. Clearly $\overrightarrow{P}$ is a directed path from r to s. Let P be the associated augmenting path of $\overrightarrow{P}$ in G. In Lemma 1 and Corollary 2, we show that P is the minimum t-net-cost augmenting path starting at r.

Before augmenting the matching M^* along P, we update the dual weights of all nodes of $S \cup R$. Let $d_s = d$ be the cost of the directed path $\overrightarrow{P}$. We update the dual weight of every node v as follows:

(a) If $d_v \geq d$, then $y(v)$ remains unchanged.
(b) If $d_v < d$, and $v \in R$, then we increase the dual weight $y(v) \leftarrow y(v) + d - d_v$
(c) If $d_v < d$, and $v \in S$, then we decrease the dual weight $y(v) \leftarrow y(v) - d + d_v$.

In Lemma 4, we show that the updated dual weights and the matching M^* are t-feasible. We also show, in Lemma 4, that after the dual updates, every edge in $P \setminus M^*$ will satisfy (1) with equality.

At this point, we update matching M^* by augmenting M^* along P, i.e., $M^* \leftarrow M^* \oplus P$. We also update the dual weight of every vertex $r' \in R \cap P$ as follows: $y(r') \leftarrow y(r') - (t-1)\mathtt{d}(s', r')$; here s' is the match of r' in the updated M^*. Lemma 5 will show that the matching after the augmentation and the updated dual weights remain t-feasible.

In processing a new request, note that we update the dual weights twice. First, we update them right before augmenting the matching along P as describe in a–c. Then, immediately after augmenting M^* along the path P, we update the dual weight again. In Lemma 4 and Lemma 5, we will show that both these updates do not violate the t-feasibility property of M^*. Therefore, (I1) holds. The proofs of Lemma 1, Lemma 3, Lemma 4, Lemma 5 and Proof of (I2) are variants of the proofs for the standard primal-dual based Hungarian method. For the sake of completion, we present these proofs. An expert may choose to skip these proofs.

The following lemma will show that Dijkstra's algorithm will compute the minimum t-net-cost path between r and any free server s.

▶ **Lemma 1.** *For any free server $s \in S_F$, let $\overrightarrow{P}$ be a directed path from the new request r to s in G_{M^*} with a cost d'. Then, the associated path P of $\overrightarrow{P}$ is an augmenting path whose t-net-cost is d'.*

Proof. The cost of $\overrightarrow{P}$ is:

$$d' = \sum_{\overrightarrow{ab} \in \overrightarrow{P}} s(a,b) \tag{3}$$

For the associated path P, from the definition of t-net-cost and the feasibility of dual weights, we have

$$\begin{aligned}\phi_t(P) &= t(\sum_{(s',r') \in P \setminus M^*} \mathtt{d}(s',r')) - \sum_{(s',r') \in P \cap M^*} \mathtt{d}(s',r') \\ &= t(\sum_{(s',r') \in P \setminus M^*} \mathtt{d}(s',r')) - \sum_{(s',r') \in P \cap M^*} (y(s') + y(r')) \end{aligned} \tag{4}$$

Since the first and the last vertex of the associated path P, i.e., s and r, are unmatched in M^*, both the first and the last edge of P is not in the matching M^*. Therefore, we can write (4) as:

$$\phi_t(P) = \sum_{(s',r') \in P \setminus M^*} (t(\mathtt{d}(s',r')) - y(s') - y(r')) + y(s) + y(r) \tag{5}$$

Note that $y(s)$ is 0 by invariant (I2) and by construction $y(r)$ is 0. For every edge $(s',r') \in P \cap M^*$, $\mathtt{d}(s',r') - y(s') - y(r') = 0$. Combining this, we can rewrite equation 5 as

$$\begin{aligned}\phi_t(P) &= \sum_{(s',r') \in P \setminus M^*} (t(\mathtt{d}(s',r')) - y(s') - y(r')) + \sum_{(s',r') \in P \cap M^*} (\mathtt{d}(s',r') - y(s') - y(r')) \\ &= \sum_{\overrightarrow{ab} \in \overrightarrow{P}} s(a,b) = d' \end{aligned}$$

◀

As an immediate corollary to Lemma 1, we conclude that the path chosen by our algorithm is the minimum t-net-cost path.

▶ **Corollary 2.** *In processing the new request r, the augmenting path P chosen by our algorithm has the smallest t-net-cost among all augmenting paths that begin at r.*

In order to show that M^* and dual weights $y(\cdot)$ form a t-feasible, we need to show that all edges in the complete bipartite graph $G(S \cup R, S \times R)$ satisfy t-feasibility conditions (1) and (2). This includes the edges incident on requests that have not yet arrived. The following simple lemma will show that any such edge will satisfy t-feasibility conditions at all times.

▶ **Lemma 3.** *At any stage of the algorithm, consider any edge $(r',s') \in R \times S$ where r' is a request that has not yet arrived. We claim that the edge (r',s') satisfies the t-feasibility condition.*

Proof. Since r' has not yet arrived, r' is a free node. Therefore, the edge (r',s') is not in the current matching M^*. By construction, every request that has not yet arrived has a dual weight of 0. Therefore, $y(r') = 0$. From invariant (I2), $y(s') \leq 0$. Since $t \geq 1$ and $\mathtt{d}(s',r') \geq 0$, we have

$$y(r') + y(s') \leq 0 \leq t\mathtt{d}(s',r'),$$

showing that (r',s') satisfies (1). ◀

From this point onwards, to show that M^* is t-feasible, we will focus only on the edges incident on requests that have already arrived. Lemma 3, allows us to ignore all other edges incident on requests which have not yet arrived.

The following lemma shows that the updated dual weights before augmenting along P satisfy the desired properties:

▶ **Lemma 4.** *Let $y(\cdot)$ be the dual weights assigned to $S \cup R$ before executing Dijkstra's algorithm and let $y'(\cdot)$ be the updated dual weights computed before augmenting M^* along P. The matching M^* and dual weights $y'(\cdot)$ are t-feasible. Furthermore, suppose P is the augmenting path chosen by the algorithm. Then, every edge of P has 0 slack with respect to the updated dual weights $y'(\cdot)$, i.e., for every $(s', r') \in M^* \cap P$, $\mathtt{d}(s', r') - y(s') - y(r') = 0$ and for every $(s', r') \in P \setminus M^*$, $t\mathtt{d}(s', r') - y(s') - y(r') = 0$.*

Proof. First, for any edge $(s', r') \in M^*$, we show that the updated dual weights does not violate feasibility condition (2). For any edge $(s', r') \in M^*$ there is a directed edge $\overrightarrow{s'r'}$ in G_{M^*}. By construction, all edges incident on r' except for the edge $\overrightarrow{s'r'}$ is directed away from r' (edges that are not in the matching are directed away from the requests in G_{M^*}). Therefore any path in G_{M^*} from the new request r to r' must contain the edge $\overrightarrow{s'r'}$; note that since $(s', r') \in M^*$, it has 0 slack and therefore the cost of this edge in G_{M^*} is $s(s', r') = 0$. Therefore, the shortest path from r to r' has the same cost as the shortest path from r to s', i.e., $d_{s'} = d_{r'}$. If $d_{s'} \geq d$, the dual weights of s' and r' are not updated (update condition (a)), and therefore the edge continues to satisfy (2). On the other hand, if $(d_{s'} = d_{r'}) < d$, the dual weight $y(r')$ increases by $d - d_{r'}$ and the dual weight of $y(s')$ decreases by $d - d_{s'}$. Since $d_{s'} = d_{r'}$, we have

$$y'(s') + y'(r') = y(s') - d + d_{s'} + y(r') + d - d_{r'} = y(s') + y(r') = \mathtt{d}(s', r').$$

Therefore, every edge $(s', r') \in M^*$ continues to satisfy the t-feasibility condition (2).

For any edge $(s', r') \in (S \times R) \setminus M^*$, there is an edge $\overrightarrow{r's'}$ in G_{M^*}. Since shortest path costs satisfy triangle inequality, we have $d_{s'} \leq d_{r'} + s(r', s')$, or

$$d_{s'} - d_{r'} \leq s(r', s').$$

After the dual weights are updated, we have

$$y'(r') + y'(s') = y(r') + d - d_{r'} + y(s') - d + d_{s'} \leq y(s') + y(r') + s(s', r') \leq t\mathtt{d}(s', r').$$

Therefore, every edge (s', r') remains feasible after the dual updates.

For every edge in $P \cap M^*$, we have already shown that the edge satisfies t-feasibility condition (2) and therefore has a slack of 0. Next, we show that the slack on every edge $(s', r') \in P \setminus M$ is also 0. Since $(s', r') \notin M^*$, there is a directed edge $\overrightarrow{r's'}$ in G_{M^*}. From the optimal substructure property of shortest paths, we have $d_{s'} = d_{r'} + s(r', s')$,. Therefore, we have

$$y'(r') + y'(s') = y(r') + d - d_{r'} + y(s') - d + d_{s'} = y(s') + y(r') + s(s', r') = t\mathtt{d}(s', r'),$$

implying that the slack on every such edge after the dual weights are updated is 0. ◀

At this point, the algorithm augments M^* along P and updates the dual weights again. The following lemma shows that the augmentation process and the updated dual weights continue to satisfy t-feasibility conditions.

▶ **Lemma 5.** *After augmentation, the updated dual weights together with the updated matching M^* form a t-feasible matching.*

Proof. For the sake of this proof, let us refer to the matching before augmenting along P as M^* and the matching after augmentation as M', i.e., $M' \leftarrow M^* \oplus P$. Also, let $y(\cdot)$ represent the dual weight right before augmenting M^* along P and let $y'(\cdot)$ represent the updated dual weight after augmentation.

Augmenting along P removed edges from $P \cap M^*$ and adds edges of $P \setminus M^*$ to matching. Then, the algorithm updates dual weights of every vertex in $R \cap P$. Every edge in $(s', r') \in M^* \cap M'$ is vertex-disjoint from the path P. Therefore, for every such edge, the dual weights of s' and r' remains unchanged and the edge continues to satisfy (2).

For any edge $(s', r') \in P \setminus M^*$, we know (s', r') is in the updated matching M'. From Lemma 4, every such edge has a 0 slack and therefore,

$$y(s') + y(r') = t\mathtt{d}(s', r').$$

The updated dual weights for r' is $y'(r') \leftarrow y(r') - (t-1)\mathtt{d}(s', r')$, whereas the dual weight for s' remains unchanged. Therefore, the new dual weight will satisfy

$$y'(s') + y'(r') + (t-1)\mathtt{d}(s', r') = t\mathtt{d}(s', r'),$$

or

$$y'(s') + y'(r') = \mathtt{d}(s', r')$$

satisfying feasibility condition (2) with respect to matching M'.

For every other edge $(s', r') \in (S \times R) \setminus M'$, if r' is not on P, then the dual weights of s' and r' do not change and therefore the edge continues to be feasible. On the other hand, if r' is on the path P, suppose s'' is the match of r' in M'. the dual update will be as follows: $y'(r') \leftarrow y(r') - (t-1)\mathtt{d}(s'', r')$. Therefore,

$$y'(s') + y'(r') = y(s') + y(r') - (t-1)\mathtt{d}(s'', r') \leq \mathtt{d}(s', r').$$

The last inequality follows from the fact that $t \geq 1$, $\mathtt{d}(s'', r') \geq 0$, and the dual weights before augmentation satisfied (1). ◀

Proof of (I2). Initially, for every vertex $s' \in S$, its dual weight $y(s') = 0$. At the end of any iteration, we claim that the dual weight of every vertex in S_F remains 0. Recollect that our algorithm selects the free vertex $v \in S_F$ with the smallest d_v value. Therefore, for every other free vertex $v' \in S_F \setminus \{v\}$, $d_{v'} \geq d_v$ and therefore from (a), the dual weight of v' is not updated and remains 0. After augmentation, only the dual weights of points on P get updated. Since every vertex of P is matched after augmentation, there is no vertex of S_F on P. Therefore, the dual weight of every vertex of S_F remains 0.

For every vertex $v \in S$, initially $y(v) = 0$.In the update procedure for dual weights, if the vertex v belongs to S, its dual weight only reduces (see condition (c) for updating dual weight). The update procedure after augmentation, on the other hand, does not change the dual weight of any vertex in S. Therefore, the dual weight of v at any time during the algorithm is $y(v) \leq 0$. ◀

Efficiency. To process a new request, the algorithm executes Dijkstra's algorithm in $O(n^2)$ time and updates the matching and the dual weights by simply processing every node individually in $O(n)$ time.

▶ **Theorem 6.** *When a new request arrives, our algorithm processes and assigns an unmatched server to this request in $O(n^2)$ time.*

For $t = 1$, our algorithm is identical to the algorithm of Khuller *et al.* [10]. For $t > 1$, we are able to relate the cost of the online matching produced by our algorithm to the t-net-cost of the paths produced by the algorithm (Lemma 7(ii)). This relation is crucial to achieve desired performance bounds of our algorithm.

3 Performance of the Algorithm

To simplify the analysis of the algorithm, we introduce a few notations. We index the requests in the order of their arrival, i.e., let r_i be the ith request to arrive. To process request r_i, let P_i be the augmenting path computed by our algorithm. Let s_i be the free server at the other end of the augmenting path P_i. Let M_i^* be the offline matching after the ith request has been processed. Note that M_0^* is an empty matching and $M_n^* = M^*$ is the final matching after all the n requests have been processed. The online matching M_i is the online matching after i requests have been processed. M_i consists of edges $\bigcup_{j=1}^{i}(s_j, r_j)$.

For any path P, let $\ell(P) = \sum_{(s,r)\in P} \mathtt{d}(s,r)$ be its *length*. Next, we prove a useful relation between the t-net-cost of augmenting paths produced by our algorithm and the cost of the online matching. We utilize the metric property of costs to establish this relation.

▶ **Lemma 7.** *Let $t \geq 1$. Let $P_1, \ldots, P_n$ be the augmenting paths computed by our algorithm in that order. Then, the t-net-cost of these paths relate to the cost of the online matching as follows:*

(i) $\phi_t(P_i) \leq t\mathtt{d}(s_i, r_i) \leq t\ell(P_i)$.

(ii) $\sum_{i=1}^{n} \phi_t(P_i) \geq ((t-1)/2)w(M) + ((t+1)/2)w(M^*)$.

Proof of (i). From triangle inequality, the length $\ell(P_i)$ of path P_i is at most the distance $\mathtt{d}(s_i, r_i)$ between its end-points. Therefore,

$$\mathtt{d}(s_i, r_i) \leq \ell(P_i). \tag{6}$$

With respect to matching M_{i-1}^*, the edge (s_i, r_i) is an augmenting path of length 1. The t-net-cost of this path is $t\mathtt{d}(a_i, b_i)$. Since, P_i is the minimum net-cost path with respect to M_{i-1}^*, $\phi_t(P_i) \leq t\mathtt{d}(a_i, b_i)$. This, combined with (6) implies (i). ◀

Proof of (ii). Since the matchings M_i^* and M_{i-1}^* differ only in the edges of the augmenting path P_i, we have

$$\begin{aligned} w(M_i^*) - w(M_{i-1}^*) &= \sum_{(s,r)\in P_i \setminus M_{i-1}^*} \mathtt{d}(s,r) - \sum_{(s,r)\in P_i \cap M_{i-1}^*} \mathtt{d}(s,r) \qquad (7)\\ &= \phi_t(P_i) - \left((t-1) \sum_{(s,r)\in P_i \setminus M_{i-1}^*} \mathtt{d}(s,r) \right) \\ &= \phi_t(P_i) - \left(\frac{t-1}{2} \sum_{(s,r)\in P_i \setminus M_{i-1}^*} \mathtt{d}(s,r) + \frac{t-1}{2} \sum_{(s,r)\in P_i \setminus M_{i-1}^*} \mathtt{d}(s,r) \right) \end{aligned}$$

The second equality follows from the definition of $\phi_t(\cdot)$.

We add and subtract $(\frac{t-1}{2})\sum_{(s,r)\in P_i\cap M^*_{i-1}} \mathtt{d}(s,r)$ to the RHS and get the following

$$\begin{aligned} w(M^*_i)-w(M^*_{i-1}) &= \phi_t(P_i)-\frac{t-1}{2}\left(\sum_{(s,r)\in P_i\setminus M^*_{i-1}}\mathtt{d}(s,r)+\sum_{(s,r)\in P_i\cap M^*_{i-1}}\mathtt{d}(s,r)\right)\\ &\quad -\frac{t-1}{2}\left(\sum_{(s,r)\in P_i\setminus M^*_{i-1}}\mathtt{d}(s,r)-\sum_{(s,r)\in P_i\cap M^*_{i-1}}\mathtt{d}(s,r)\right)\\ &= \phi_t(P_i)-\frac{t-1}{2}(\sum_{(s,r)\in P_i}\mathtt{d}(s,r))-\frac{t-1}{2}(w(M^*_i)-w(M^*_{i-1})) \end{aligned}$$

The last equality follows from (7). Rearranging terms and setting $\sum_{(s,r)\in P_i}\mathtt{d}(s,r)=\ell(P_i)$, we get,

$$\begin{aligned} \frac{t+1}{2}(w(M^*_i)-w(M^*_{i-1})) &= \phi_t(P_i)-\frac{t-1}{2}\ell(P_i)\\ \frac{t+1}{2}\sum_{i=1}^{n}(w(M^*_i)-w(M^*_{i-1})) &= \sum_{i=1}^{n}\phi_t(P_i)-\frac{t-1}{2}\sum_{i=1}^{n}\ell(P_i)\\ \frac{t+1}{2}w(M^*) &\le \sum_{i=1}^{n}\phi_t(P_i)-\frac{t-1}{2}w(M) \end{aligned}$$

In the second to last equation, the summation on the LHS telescopes canceling all terms except $w(M^*_n)-w(M^*_0)$. Since $M^*_n=M^*$ and M^*_0 is an empty matching, we get $w(M^*_n)-w(M^*_0)=w(M^*)$. From triangle inequality, we know that $\ell(P_i)\ge\mathtt{d}(s_i,r_i)$. From this, we immediately get the last inequality. Rearranging the terms, we immediately get (ii). ◀

Next, equipped with the properties from Lemma 7, we will analyze the performance of our algorithm in the adversarial and the random arrival models.

To analyze the performance in the online models, let M^i_{OPT} be the minimum-cost matching of the first i requests to the set of servers. Also, we will denote M^n_{OPT} as M_{OPT}. It is easy to see that $w(M^i_{\text{OPT}})\le w(M_{\text{OPT}})$. This is because M_{OPT} contains a matching of the first i request to servers S where as M^i_{OPT} is the smallest possible such matching. The cost of M^i_{OPT}, therefore, should be less than the cost of M_{OPT}.

$$w(M^i_{\text{OPT}})\le w(M_{\text{OPT}}).$$

Performance in Adversarial Model. We show that the performance of our algorithm in the adversarial model is optimal.

▶ **Lemma 8.** *The competitive ratio of our algorithm in the adversarial model is* $2n-1+o(1)$.

Proof. Consider the graph $\tilde{G}$ with vertex set $S\cup R$ and the edges of the symmetric difference of M_{OPT} and M^*_{i-1}, i.e., $\tilde{G}(S\cup R, M_{\text{OPT}}\oplus M^*_{i-1})$. Since M_{OPT} is a perfect matching, this graph $\tilde{G}$ contains $n-i+1$ vertex-disjoint augmenting paths with respect to M^*_{i-1}. There is one augmenting path for each of the $n-i+1$ requests that have not yet arrived. Let $\{r'_1,r'_2,\ldots,r'_{n-i+1}\}$ be the requests that have not yet arrived and let the $n-i+1$ augmenting paths be $\{P'_1,P'_2,\ldots,P'_{n-i+1}\}$, where P'_j is an augmenting path that has r'_j as one of its end-vertex.

In particular, there is an augmenting path in $\tilde{G}$ with the i^{th} request r_i as one of its end vertex. Let P be this augmenting path.

$$\phi_t(P) = t \sum_{(s,r)\in M_{\text{OPT}}\cap P} \mathtt{d}(s,r) - \sum_{(s,r)\in M^*_{i-1}\cap P} \mathtt{d}(s,r) \leq t \sum_{(s,r)\in M_{\text{OPT}}\cap P} \mathtt{d}(s,r) \leq tw(M_{\text{OPT}})$$

While processing the ith request, our algorithm produces the minimum t-net-cost augmenting path. Therefore, the augmenting path P_i generated by our algorithm will have

$$\phi_t(P_i) \leq \phi_t(P) \leq tw(M_{\text{OPT}}).$$

If we sum over all the n augmenting paths generated by our algorithm, we get

$$\sum_{i=1}^{n} \phi_t(P_i) \leq ntw(M_{\text{OPT}}).$$

Using property (ii) in Lemma 5 in conjunction with the previous inequality, we get the following

$$\begin{aligned}
ntw(M_{\text{OPT}}) &\geq ((t-1)/2)w(M) + ((t+1)/2))w(M^*) \\
2ntw(M_{\text{OPT}}) - (t+1)w(M^*) &\geq (t-1)w(M) \\
2ntw(M_{\text{OPT}}) - (t+1)w(M_{\text{OPT}}) &\geq (t-1)w(M) \\
w(M) &\leq w(M_{\text{OPT}})(2nt-(t+1))/(t-1) \\
w(M)/w(M_{\text{OPT}}) &\leq (2+2/(t-1))n - (1+2/(t-1))
\end{aligned}$$

The last inequality upper bounds the competitive ratio of the algorithm. If we set $t = n^2+1$, the upper bound can be simplified to $2n-1+1/n$ which is $2n-1+o(1)$. ◀

As shown in [8], no deterministic algorithm can achieve a competitive ratio better than $2n-1$. Therefore, our algorithm is optimal.

Performance in Random Arrival Model. In the random arrival model, we show that the performance ratio of our algorithm is $2\text{H}_n - 1 + o(1)$.

▶ **Lemma 9.** *In the random arrival model, the competitive ratio of our algorithm is* $2H_n - 1 + o(1)$.

Proof. Consider the graph $\tilde{G}$ with vertex set $S\cup R$ and the edges of the symmetric difference of M_{OPT} and M^*_{i-1}, i.e., $\tilde{G}(S\cup R, M_{\text{OPT}} \oplus M^*_{i-1})$. Since M_{OPT} is a perfect matching, this graph contains $n-i+1$ vertex-disjoint augmenting paths with respect to M^*_{i-1}. There is one augmenting path for each of the $n-i+1$ requests that have not yet arrived. Let $\{r'_1, r'_2, \ldots, r'_{n-i+1}\}$ be the requests that have not yet arrived and let the $n-i+1$ augmenting paths in $\tilde{G}$ be $\{P'_1, P'_2, \ldots, P'_{n-i+1}\}$, where P'_j is an augmenting path that has r'_j as one of its end vertex.

$$\begin{aligned}
\sum_{j=1}^{n-i+1} \phi_t(P'_j) &= \sum_{j=1}^{n-i+1} \left(t \sum_{(s,r)\in P'_j \setminus M^*_{i-1}} \mathtt{d}(s,r) - \sum_{(s,r)\in P'_j \cap M^*_{i-1}} \mathtt{d}(s,r) \right) \\
&= \sum_{j=1}^{n-i+1} \left(t \sum_{(s,r)\in P'_j \cap M_{\text{OPT}}} \mathtt{d}(s,r) - \sum_{(s,r)\in P'_j \cap M^*_{i-1}} \mathtt{d}(s,r) \right) \\
&\leq \sum_{j=1}^{n-i+1} \left(t \sum_{(s,r)\in P'_j \cap M_{\text{OPT}}} \mathtt{d}(s,r) \right) \leq tw(M_{\text{OPT}}).
\end{aligned}$$

The second equation follows from the fact that these paths are formed by the symmetric difference of M^*_{i-1} and M_{OPT} and therefore $P'_j \setminus M^*_{i-1} = P'_j \cap M_{\text{OPT}}$. The last inequality follows from the fact that all the augmenting paths are vertex disjoint.

Let j be such that the i^{th} request r_i is r'_j. The algorithm computes the minimum t-net-cost augmenting path P_i from r_i with respect to the matching M^*_{i-1}. Therefore, the t-net-cost of P_i should be less than the t-net-cost of P'_j (note that r_i is one of the end-vertex of P'_j).

$$\phi_t(P_i) \leq \phi_t(P'_j).$$

In the random arrival model, the input request sequence is a random permutation. Therefore, the i^{th} request can be any one of the remaining $n-i+1$ requests with the same probability and we have,

$$\mathbb{E}[\phi_t(P_i)] \leq \frac{1}{n-i+1} \sum_{j=1}^{n-i+1} \phi_t(P'_j) \leq \frac{1}{n-i+1} t w(M_{\text{OPT}}).$$

From linearity of expectation,

$$\mathbb{E}[\sum_{i=1}^{n} \phi_t(P_i)] \leq \sum_{i=1}^{n} \frac{1}{n-i+1} t w(M_{\text{OPT}}) = t\mathrm{H}_n w(M_{\text{OPT}}).$$

From Lemma 7 (ii) and the obvious fact that $w(M^*) \geq w(M_{\text{OPT}})$ we have,

$$\begin{aligned}
\mathbb{E}[\sum_{i=1}^{n} \phi_t(P_i)] &\geq \frac{t-1}{2}\mathbb{E}[w(M)] + \frac{t+1}{2}\mathbb{E}[w(M^*)] \\
t\mathrm{H}_n w(M_{\text{OPT}}) &\geq \frac{t-1}{2}\mathbb{E}[w(M)] + \frac{t+1}{2} w(M_{\text{OPT}}) \\
(t-1)\mathbb{E}[w(M)] &\leq 2t\mathrm{H}_n w(M_{\text{OPT}}) - (t+1) w(M_{\text{OPT}}) \\
\frac{\mathbb{E}[w(M)]}{w(M_{\text{OPT}})} &\leq \frac{2t\mathrm{H}_n - (t+1)}{t-1} \\
\frac{\mathbb{E}[w(M)]}{w(M_{\text{OPT}})} &\leq (2 + \frac{2}{t-1})\mathrm{H}_n - (1 + \frac{2}{t-1})
\end{aligned}$$

By setting $t = nH_n + 1$, we can bound this competitive ratio by $2\mathrm{H}_n - 1 + o(1)$. ◀

▶ **Theorem 10.** *There is an algorithm for the online minimum metric bipartite matching problem that has a competitive ratio of $2n-1+o(1)$ in the adversarial model. This algorithm also has a competitive ratio of $2H_n - 1 + o(1)$ in the random arrival model.*

The performance of our algorithm is optimal in the random arrival model. A lower bound construction for the oblivious adversary model is described in [15]. We adapt this construction in the random arrival model. Obtaining a tight lower bound of $2\mathrm{H}_n - 1 - o(1)$ requires some technical calculations which we present next.

Lower Bound in the Random Arrival Model. Consider a undirected weighted star graph with a vertex v connected by an edge to every other vertex $v_1, \ldots v_n$ of this graph. Note that this graph has $n+1$ vertices and n edges. The weight of each of these n edges is 1. Consider the shortest path metric on this graph. For any pair (v, v_i), the shortest path distance is 1. Every other pair, (v_i, v_j) will have a shortest path distance of 2; this is because the only path between them goes via v and has cost 2. Given this graph, we place our servers at nodes $S = \{v_1, \ldots, v_n\}$. The adversary chooses request locations at nodes $R = \{v, v_1, \ldots v_n\} \setminus \{v_t\}$,

where t is chosen uniformly at random from integers between 1 and n. Let σ be a random permutation of requests in R.

First, note that the minimum-cost matching of S and R is of cost 1 – the request at location v is matched to server at location v_t and every other server at location v_i is matched to the corresponding request at location v_i.

Consider any online algorithm for this input instance. Let us fix v to be the jth request in σ. All requests that arrived before v will be matched with zero cost to servers at the same location. It is easy to see that if the algorithm pursues any other matching scheme, it will only have a larger final cost. The cost of matching v is 1 since every server is at a distance 1 from v. Therefore, after processing the jth request, the total cost of the matching is 1.

Consider j'th request for any $j' > j$. Request $r_{j'}$ has to be in one of the remaining $n - j' + 1$ locations from the set $\{v_1, \ldots, v_n\}$ that have not yet seen a request. Since t is chosen uniformly at random, the next request $r_{j'}$ can be any one of these $n - j' + 1$ locations with the same probability. Next, observe that, the servers in exactly one of these $n - j' + 1$ locations is already matched. If the next request is at this location, it will incur a cost of 2. This can happen with a probability of $1/(n - j' + 1)$. Therefore, the expected cost of serving the j'th request is at least $2/(n - j' + 1)$. Given that v was the jth request, the expected cost incurred will be at least

$$1 + \sum_{j'=j+1}^{n} (2/(n - j' + 1)) \quad \geq \quad 1 + 2\mathrm{H}_{n-j} = 1 + 2\ln(n - j) + 2\epsilon_{n-j} + 2\gamma, \tag{8}$$

where γ is the Euler–Mascheroni constant and $\epsilon_k \approx 1/2k$. Since σ is a random permutation, j can be any particular index between 1 and n with probability $1/n$. When $j = n$, the cost incurred by the algorithm is exactly 1. Equation 8 is meaningful only for $1 \leq j \leq n - 1$. Therefore, the expected cost incurred by any algorithm will be at least

$$\begin{aligned}
&\frac{1}{n}(1 + \sum_{j=1}^{n-1}(2\ln(n - j) + 2\epsilon_{n-j} + 2\gamma + 1)) \\
&= \frac{1}{n}(1 + 2\ln((n-1)!) + n - 1 + 2\sum_{j=1}^{n}\epsilon_{n-j} + 2(n-1)\gamma) \\
&= \frac{1}{n}(n + 2\ln((n-1)!) + 2\sum_{j=1}^{n-1}\epsilon_{n-j} + 2(n-1)\gamma) \\
&= 1 + \frac{2}{n}((n-1)\ln(n-1) - (n-1) + (n-1)\gamma + O(\log n)) \\
&\geq 1 + 2\ln n - 2 + \gamma + \epsilon_n - o(1) \\
&\geq 2\mathrm{H}_n - 2 + 1 - o(1) \\
&\geq 2\mathrm{H}_n - 1 - o(1)
\end{aligned}$$

The third equality follows from Sterling's approximation which gives us $\ln((n-1)!) = (n-1)\ln(n-1) - (n-1) + O(\log n)$. Also, $\sum_{j=1}^{n-1}\epsilon_{n-j} = O(\log n)$.

Therefore, in the random arrival model, any online matching generated by the algorithm will have an expected cost of $2\mathrm{H}_n - 1 - o(1)$ for this input.

▶ **Theorem 11.** *In the random arrival model, any online algorithm for the minimum metric bipartite matching problem will have a competitive ratio of at least $2H_n - 1 - o(1)$, where H_n is the nth Harmonic number.*

4 Conclusion

In this paper, we design a robust deterministic online algorithm for the minimum metric bipartite matching problem. Our algorithm achieves an optimal competitive ratio in both the adversarial and the random arrival models. We need such robust solutions for practically motivated real-time matching problems. We conclude with a few open questions:

(a) Can we extend our approach to the k-server problem and achieve better quality solutions?

(b) Can we improve the performance of our algorithm in special metrics such as the line metric or the Euclidean metric in $2d$?

(c) Can we extend our algorithm to the oblivious model and obtain a $O(\log n)$-competitive algorithm?

References

1 Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31–June 03, 2014*, pages 555–564, 2014.

2 Pankaj K. Agarwal and Kasturi R. Varadarajan. A near-linear constant-factor approximation for euclidean bipartite matching? In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 247–252, 2004.

3 Antonios Antoniadis, Neal Barcelo, Michael Nugent, Kirk Pruhs, and Michele Scquizzato. A o(n)-competitive deterministic algorithm for online matching on a line. In *Approximation and Online Algorithms – 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014*, pages 11–22, 2014.

4 N. Bansal, N. Buchbinder, A Madry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *Proceedings of the IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 267–276, Oct 2011.

5 Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Naor. An $o(\log^2 k)$-competitive algorithm for metric bipartite matching. In *Algorithms – ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 522–533, 2007.

6 A. Gupta and K. Lewi. The online metric matching problem for doubling metrics. In *Automata, Languages, and Programming*, volume 7391 of *LNCS*, pages 424–435. Springer, 2012.

7 Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 39–42, 2007.

8 Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993.

9 Chinmay Karande, Aranyak Mehta, and Pushkar Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC'11, pages 587–596, New York, NY, USA, 2011. ACM.

10 Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994.

11 Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In Roberto Solis-Oba and Klaus Jansen, editors, *Approximation and Online Algorithms: First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003. Revised Papers*, pages 179–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

12 Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, September 1995.

13 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: An approach based on strongly factor-revealing lps. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, STOC'11, pages 597–606, 2011.

14 Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, May 1990.

15 A. Meyerson, A. Nanavati, and L. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm*, pages 954–959, 2006.

16 R. Sharathkumar and Pankaj K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 306–317, 2012.

17 R. Sharathkumar and Pankaj K. Agarwal. A near-linear time ϵ-approximation algorithm for geometric bipartite matching. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC'12, pages 385–394. ACM, 2012. doi:10.1145/2213977.2214014.