Connecticut College
Digital Commons @ Connecticut College

Computer Science Honors Papers | Computer Science Department

2016

The Online Minimum Matching Problem on the Line

Maximillian Bender
Connecticut College, maxbender@gmail.com

Follow this and additional works at: http://digitalcommons.conncoll.edu/comscihp

 Part of the Systems Architecture Commons

Recommended Citation
Bender, Maximillian, "The Online Minimum Matching Problem on the Line" (2016). *Computer Science Honors Papers*. 5.
http://digitalcommons.conncoll.edu/comscihp/5

This Honors Paper is brought to you for free and open access by the Computer Science Department at Digital Commons @ Connecticut College. It has been accepted for inclusion in Computer Science Honors Papers by an authorized administrator of Digital Commons @ Connecticut College. For more information, please contact bpancier@conncoll.edu.
The views expressed in this paper are solely those of the author.

The Online Minimum Matching Problem On The Line

Maximillian CW Bender

Christine Chung

Computer Science
Connecticut College

1 Introduction

In this problem problem we are a priori given a set of servers in a metric space. One at a time an equal number of requests will arrive in locations unknown until arrival. As each request arrives, it must be irrevocably matched to a server, under the restriction that server can only be matched to one request. The two different goals we discuss for the problem are 1) the *minimum weight* objective, which minimizes the average (or equivalently total) distance between any request and its paired server and 2) the *bottleneck* objective, which minimizes the maximal distance.

When the metric is the line, then this problem is equivalent to the following shoe store problem: here, the owner has a set of shoes of various sizes and will sell them to to customers who will arrive sequentially, where each customer finishes their transaction before the next customer arrives. If the owner tries to minimize the average difference in shoe size between a shoe and its buyer, then the owner is using the minimum weight objective. If the owner instead tries to minimize the maximal difference, then the owner is using the bottleneck objective.

This problem has been studied most for the general metric, however significant work remains to be done even in the special case of the line metric. Despite the problem having a very 'simple' description, there is still a large gap between the effectiveness of the best known algorithms and the established lower bound for the effectiveness of any algorithm on this problem. In fact, Koutsoupias and Nanavati in [8] consider the line-metric to be the most interesting, citing reasons such as the offline-version being a trivial problem, the relation to the well-studied cow-path problem, and the applications of this problem in web-markets.

1.1 Formal Definitions

Formally, for any given instance I of the problem we will denote the set of n servers by $S \subset \mathbb{R}$ and the sequence of requests by $R = (r_1, ..., r_n)$. Since I is defined by its servers and requests, we will consider I to be equivalent to the the tuple (S, R). We will define a completed matching between the set of servers and the requests to be a bijective function $f : R \to S$. Hence for the *bottleneck objective* the goal is to construct a function f such that the *cost* of f defined by $c(f) = \max_{i \in \{1,...,n\}} |f(r_i) - r_i|$ is minimized, whereas for the minimum weight objective the the goal is to construct f such that the cost $c(f) = \sum_{i=1}^{n} |f(r_i) - r_i|$ is minimized.

Competitive analysis is conventionally used to determine the effectiveness of algorithms for online problems. Define $\mathcal{A}$ to be the set of all algorithms for the problem and define $\mathcal{I}$ to be the set of all possible instances. Because the method of analysis for deterministic and randomized algorithms is fundamentally different, we define $\mathcal{A}_{\mathcal{D}} \subset \mathcal{A}$ to be the set of deterministic algorithms. The *competitive ratio* ρ for a deterministic algorithm $A \in \mathcal{A}_{\mathcal{D}}$ on an online problem is given by

$$\rho = \max_{I \in \mathcal{I}} \left(\frac{\mathrm{A}(I)}{\textsc{Opt}(I)} \right),$$

where $A(I)$ and $\textsc{Opt}(I)$ denote the cost of the matching function f the algorithm outputs and the cost of the optimal offline solution's matching function respectively. Since we will be looking at the competitive ratio of randomized algorithms, the competitive ratio will be

evaluated in expectation; specifically, for any randomized algorithm $A \in \mathcal{A}$ the competitive ratio is defined to be

$$\rho = \max_{I \in \mathcal{I}} \left(E\left[\frac{A(I)}{\textsc{Opt}(I)} \right] \right).$$

For notational convenience, when we say that an algorithm is $h(n)$-*competitive*, then we mean that the competitive ratio ρ for that algorithm is no better than $h(n)$ on instances with n servers.

To establish lower bounds for the competitive ratios of these problems, we use Yao's minimax principle. Yao's principle implies that for any probability distributions p over $\mathcal{A}_{\mathcal{D}}$ and q over $\mathcal{I}$, where A is some algorithm chosen according to p and I is some instance chosen according to q, then

$$\max_{X \in \mathcal{I}} E\left[A\left(X\right)\right] \geq \min_{B \in \mathcal{A}_{\mathcal{D}}} E\left[B\left(I\right)\right].$$

Hence by defining c_a^p as the smallest competitive ratio of the algorithm a under the distribution p, Yao's principle implies that

$$\rho(A) \geq \min_{B \in \mathcal{A}_{\mathcal{D}}} c_B^p,$$

where $\rho(A)$ denotes the competitive ratio of A.

1.2 Previous Results

The natural greedy algorithm, which sends each incoming request to its closest available server, has been shown to be no better than $(2^n - 1)$-competitive on the line metric for both the bottleneck and minimum weight objectives [6]. For the minimum weight objective, Kalyanasundaram and Pruhs in [6] and Khuller, Mitchell, and Vazirani in [7] gave a $(2n - 1)$-competitive deterministic algorithm, Permutation. For the general metric, Meyerson, Nanavati, and Poplawski [9] gave a $O(\log^3 n)$-competitive randomized algorithm, which was improved upon by Bansal, Buchbinder, Gupta, and Naor [2] to give a $O(\log^2 n)$-competitive randomized algorithm, but there still remains a gap to the established lower bound of $\Omega(\log n)$. In the case where the metric is specifically the line, the Work Function Algorithm was shown to have competitive ratio $O(n)$ in [8], but this upper bound was improved by Antoniadis, Barcelo, Nugent, Pruhs, and Scquizzato [1] who gave a $o(n)$-competitive deterministic algorithm on the line - at the moment, this algorithm remains the best established deterministic algorithm. The best known randomized algorithm, Harmonic, was presented by Gupta and Lewi and shown to be $O(\log n)$-competitive in [4]. However, only a lower bound of 9.001 has been established for deterministic algorithms on the line [3] and no lower bound has been established for the randomized algorithms on the line.

The best known deterministic algorithm for the bottleneck objective is also Permutation and has been shown to be $(2n-1)$-competitive in [6]. The highest known lower bound for the competitive ratio for deterministic algorithms was presented by Idury and Schaffer to be approximately $1.5n$ in [5]. To the best of our knowledge, there is also no established lower bound for the competitive ratio on the bottleneck objective for randomized algorithms.

2 Tools

In this section we will present a few useful tools and methods for approaching the analysis of this problem. The first tool we present allows us to make concrete restrictions on the type of algorithms we will consider. The second tool was established by Meyerson et al. in [9] and allows us to reduce the possible set of arrival locations of requests to the set of servers. The final tool we discuss are ρ-server-sets, which provides a reduction for the type of possible server sets that we will consider when establishing upper bounds for algorithms.

2.1 Nearest Neighbors Theorem

One way to reduce the problem of constructing lower bounds is to reduce the possible search pool of algorithms you are evaluating over. Recall that we construct lower bounds by considering the value of

$$\min_{A \in \mathcal{A}} \left(E\left[\frac{A(I)}{\textsc{Opt}(I)} \right] \right)$$

for any specific instance I. Hence, if you can prove that

$$\min_{A \in \mathcal{A}} \left(E\left[\frac{A(I)}{\textsc{Opt}(I)} \right] \right) = \min_{A \in \mathcal{A}_{\mathcal{P}}} \left(E\left[\frac{A(I)}{\textsc{Opt}(I)} \right] \right)$$

for some $\mathcal{A}_{\mathcal{P}} \subset \mathcal{A}$, then you are effectively reducing the type of algorithms which you are minimizing over. Hence we will establish that for any instance of the problem there is always an optimal matching that assigns each incoming request to either its closest left or right available server. By doing this, we need not consider algorithms that would match otherwise.

Theorem 1 (Nearest Neighbors Theorem). Given any instance of the problem for the minimum weight objective, there exists an optimal min-weight matching function $f : R \to S$ that satisfies $f(r_j) \in N_f(r_j)$, where $N_f(r_j)$ denotes the neighboring unmatched servers of r_j on the line. Formally, by letting

$$\begin{aligned} S_j &= S \setminus \{f(r_1), ..., f(r_{j-1})\}, \\ L_j &= \max\{s | s \in S_j, s \leq r_j\}, \text{ and} \\ R_j &= \min\{s | s \in S_j, s \geq r_j\}, \end{aligned}$$

then $N_f(r_j) = \{L_j, R_j\}$.

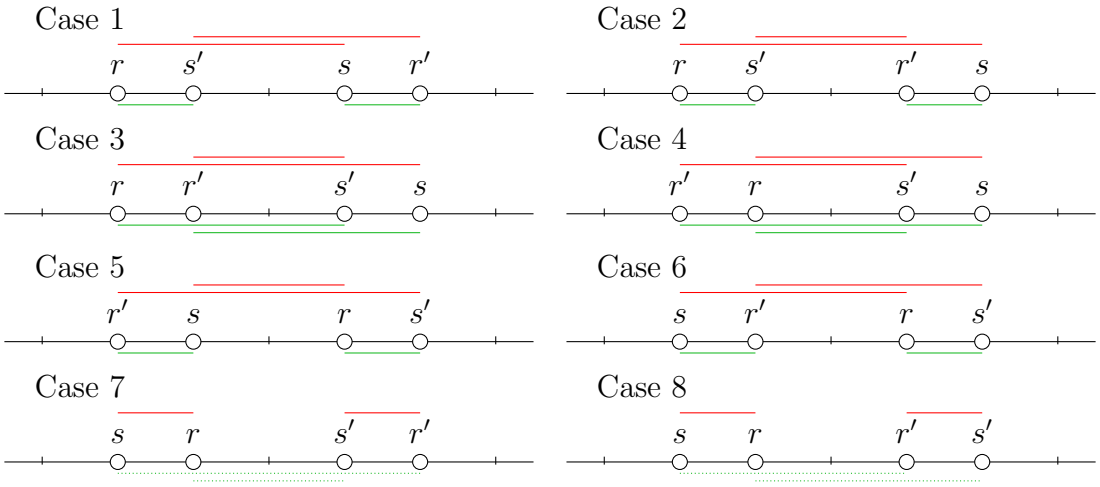
Proof. For the sake of contradiction, we will suppose there does not exist a matching that matches each request r_j to L_j or R_j. Let f denote any optimal matching that satisfies the above property up to some point. That is, $f(r_i) \in N_f(r_i)$ for all i up to some $n \leq k$, but $f(r_j) \notin N_f(r_j)$. We will use the following notation for the rest of the proof:

$$r = r_j, \quad s = f(r_j), \quad s' = R_j, \quad r' = f^{-1}(s').$$

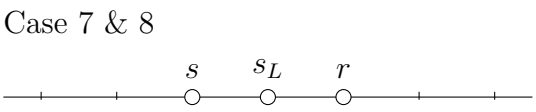
We will also let $\tilde{f}$ denote the matching that sends r to s' and r' to s and $\tilde{f}(r_j) = f(r_j)$ for all other requests r_j.

To see there are only 8 cases which need to be considered, note that we must have that $r < s'$, but cannot have that $r < s < s'$, since both s and s' must be available at the time of arrival for the jth request but s' is defined to be the closest right server. Thus, we have 3 possible cases where $r = \min\{r, s, s', r'\}$, 2 cases where $r' = \min\{r, s, s', r'\}$, and 3 cases where $s = \min\{r, s, s', r'\}$ (s' can't ever be the leftmost due to the restriction $r < s'$).

This gives us the following 8 possible cases (note that the figures are not drawn to scale, they are just to display ordering):



In case 1, case 2, case 5, and case 6, the cost of $\tilde{f}$ will actually be less than the cost of f, since $|r - f(r)| > |r - \tilde{f}(r)|$ and $|r' - f(r')| > |r' - \tilde{f}(r')|$. In cases 3 and 4, the cost of $\tilde{f}$ will be equal to the cost of f. Finally, in cases 7 and 8, we see that the cost of $\tilde{f}$ will actually be bigger than the cost of f. But this is not a problem – first, if $s = L_j$ (if s is r's closest left available server), then we have a contradiction because we originally supposed that r was not matched with either of its left or right neighboring servers. Thus, letting s_L denote L_j, we must have the following order:



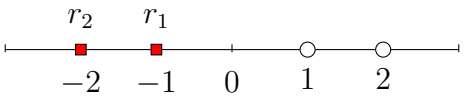
But now, we can see that the mirror of either case 1, 2, 3, or 4 must apply, since these are the cases where $r < s' < s$, and thus we know that constructing a new matching $\tilde{f}'$ such that $\tilde{f}'(r) = s_L$ and $\tilde{f}'^{-1}(s) = \tilde{f}'^{-1}(s_L)$ (where all other requests correspond to f) will satisfy that the total cost of $\tilde{f}'$ will be less than or equal to the total cost of f.

In all of these cases, we have constructed a new matching whose total cost is less than or equal to f where r is matched to one of its closest available neighbors. This is a contradiction with respect to the construction of f and completes the proof.

□

Hence, when analyzing the lower bound of the competitive ratio on a specific instance, we can restrict our search space to algorithms which always match to either the left or right closest servers.

Unfortunately, Theorem 1 doesn't hold for the bottleneck objective. To see this, consider an instance of just two servers:



Now, if Theorem 1 were also true for the bottleneck objective, then we would obtain an optimal matching by assigning r_1 to 1 and r_2 to 2. However, this will result in a maximal distance of $2-r_2 = 4$. But if instead we match r_1 to 2 and r_2 to 1, then the maximal distance is instead 3. This is a contradiction to the optimality of the first matching, confirming that Theorem 1 does not hold for the bottleneck objective.

2.2 Request Collocation

In [9], Meyerson et al. mentioned the following powerful result:

Lemma 2 (Meyerson et al.). The requests can be assumed to be collocated with servers. Specifically, there exists a constant c such that for any deterministic algorithm A and instance I with request sequence $R = (r_1, ..., r_n)$ and servers S, the following holds: consider the sequence of requests $\tilde{R} = (\tilde{r}_1, ..., \tilde{r}_n)$ where each request $\tilde{r}_i$ is the closest server to r_i and let $\rho(A)$ be the competitive ratio of A on the original instance and $\tilde{\rho}(A)$ be the competitive ratio of A on the new instance. Then $\frac{\tilde{\rho}(A)}{\rho(A)} \leq c$.

The power of the statement lies in its ability to significantly reduce the 'search space' of instances when we prove upper bounds for the competitive ratios for algorithms. Instead of considering all possible instances, where requests could potentially land anywhere on the real line, we only need to consider requests which land in the set of servers. In particular, when considering one specific set of servers, we can assume there are only a finite number of places the requests can land. This will prove very useful when constructing our Minimax Matching algorithm later.

2.3 ρ Server Sets

When constructing lower bounds for the competitive ratio of specific algorithms or for the problem as a whole, one specific choice of servers can be found that no algorithm can perform well against in the worst case. In this section, we formalize this principle and provide some basic new results.

Definition. We call a set $S \subseteq \mathbb{R}$ a *symmetric server set* if there exists some $x \in \mathbb{R}$ such that $2s - x \in S$ for all $s \in S$.

This really just means that for every server $s \in S$ there is another server reflected about x. That is, S is symmetric about x. For example, the set $\{-2, -1, 1, 2\}$ is a symmetric server set, since it is symmetric about $x = 0$, whereas there is no x which the set $\{1, 2, 4, 8\}$ is symmetric about.

Definition. We call a set $S \subseteq \mathbb{R}$ a ρ *server set* if for every algorithm A there exists an instance $I_A = (S, R_A)$ such that $\frac{A(I_A)}{\text{OPT}(I_A)} \geq \rho$.

Of course, the costs of $A(I)$ and $\text{OPT}(I)$ will depend on what objective is being considered, which we have purposefully left vague in the definition along with whether the algorithm is deterministic or randomized so that the definition can be used with reference to either. Note that the existence of any ρ server set necessarily implies a lower bound of ρ on the competitive ratio of all algorithms.

The following lemma establishes the strength of symmetric server sets:

Lemma 3. If S is a ρ server set, then there exists a symmetric ρ server set $\tilde{S}$ under either objective.

Proof. Let $S = \{s_1, ..., s_n\}$ with $s_1 \leq ... \leq s_n$ and $x = 2s_1 - s_n < s_1 \in \mathbb{R}$. Then by defining $\tilde{S} = \{x - s_n, ..., x - s_1, s_1, ..., s_n\}$, we have that $\tilde{S}$ is a symmetric server set. Hence for any algorithm A we know that there exists some $I_A = (S, R_A)$ such that $\frac{A(I_A)}{\text{OPT}(I_A)} \geq \rho$. For any algorithm A, let $\tilde{A}$ be the algorithm that A becomes having matched the start of a request sequence $(x - s_1, ..., x - s_n)$ into $\tilde{S}$. Then we know that $\tilde{A}$ has a corresponding request sequence $R_{\tilde{A}} = (r_1^A, ..., r_n^A)$ on which its competitive ratio with the server set S is at least ρ. We now want to show that regardless of the objective A still has a competitive ratio no less than ρ on $\tilde{S}$. To do this, consider the request sequence $\tilde{R}_A = (x - s_1, ..., x - s_n, r_1^A, ..., r_n^A)$.

1. **Minimum Weight:** By the Nearest Neighbors Theorem, we know that the cost of A matching $\tilde{R}_A$ into $\tilde{S}$ is then at least the cost of matching the first n requests to their collocated servers and then the cost $\tilde{A}$ had on R_A, meaning that the competitive ratio of A with the server set $\tilde{S}$ is at least ρ.

2. **Bottleneck:** By the choice of x, we know that if any of the first n requests are matched to the positive servers the bottleneck distance will be strictly greater than the bottleneck distance of A on I_A. Thus after the first n requests we can assume that the only servers left are S. But then we know that the bottleneck edge of A on $\tilde{R}_A$ must be at least the bottleneck edge of $\tilde{A}$ on $R_{\tilde{A}}$ and hence the competitive ratio of A with the server set $\tilde{S}$ is at least ρ.

□

This lemma tells us two useful things: 1) that when we are constructing upper bounds for a specific algorithm, we only need to consider how our algorithm performs on symmetric servers, and 2) that when we are coming up with instances for lower bounds, we can really just stick to symmetric instances. In fact, we conjecture that you only need to consider symmetric server sets where the sequence of requests are *expanding*, where a request sequence $R = (r_1, ..., r_n)$ is called expanding if there is never a $i < j < k$ satisfying $r_i < r_k < r_j$ or $r_j < r_k < r_i$. If such a claim were proven, this would be a powerful limit to the types of instances one would need to consider when establishing the competitive ratio of an algorithm.

3 Minimum Weight Objective

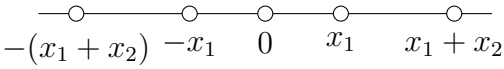
In this section we will discuss the problem under the minimum weight objective. We will first construct our lower bounds for the competitive ratio of deterministic and randomized algorithms, 4 and 2 respectively. We will then move on to deterministic algorithms, where we will establish that the PERMUTATION algorithm is strictly linearly competitive. We will then show our analysis of the algorithm presented in [1] and discuss our MINIMAX MATCHING algorithm. To wrap up the minimum weight objective we will discuss the HARMONIC randomized algorithm.

3.1 Lower Bounds – Deterministic

First we will consider the lower bounds of deterministic and randomized algorithms under the minimum weight objective. In [3], Fuchs et al. establish a lower bound of 9.001 for all deterministic algorithms in an admittedly complicated proof via an adversarial strategy for the related Cow Path Problem. We present a relatively simple lower bound of 4 via a singular instance of the Minimum Weight Objective and show that no deterministic algorithm can achieve a better ratio than 4.

Theorem 4. No deterministic algorithm for the minimum weight objective has a lower competitive ratio than 4.

Proof. Consider the following instance:



The servers are located at $\{-(x_1+x_2), -x_1, 0, x_1, x_1+x_2\}$ for some $0 < x_1, x_2 \in \mathbb{R}$. The first two requests will be located at 0 (hence $r_1 = r_2 = 0$). We can assume that by the Nearest Neighbors Theorem that r_1 will be matched to 0, and r_2 will be matched to either x_1 or $-x_1$. Without loss of generality, let's assume that r_2 will be matched to x_1. Then we will place our next request r_3 at x_1 and we will be left with two cases.

1. If r_3 is matched to $x_1 + x_2$, then we place r_4 at $x_1 + x_2$. By our Nearest Neighbors Theorem, we can assume that r_4 will then match to $-x_1$. Then we place our final request r_5 at $-(x_1 + x_2)$ which will be collocated with the only remaining server. The sum of all the matches made will be $0 + x_1 + x_2 + (x_1 + x_2 + x_1) + 0 = 3x_1 + 2x_2$. In this case, the optimal cost will be just x_1, so the ratio of the online cost to the offline will be $3 + 2\frac{x_2}{x_1}$.

2. If r_3 is instead matched to $-x_1$, then we place r_4 at $-x_1$. If r_4 is matched back up to $x_1 + x_2$, then by placing our final request r_5 at $x_1 + x_2$ we will have a total cost of $x_1 + (x_1 + x_1) + (x_1 + x_1 + x_2) + 2(x_1 + x_2) = 7x_1 + 3x_2$ whereas the optimal cost would have been $x_1 + x_2$. This gives a ratio of $\frac{7x_1+3x_2}{x_1+x_2}$. On the other hand, if r_4 were matched to $-(x_1 + x_2)$, then by placing r_5 at $-(x_1 + x_2)$ we have the total $x_1 + x_1 + x_1 + x_2 + 2(x_1 + x_2) = 5x_1 + 3x_2$ whereas the optimal cost would be $x_1 + x_2$, giving a ratio of $\frac{5x_1+3x_2}{x_1+x_2}$. Since this ratio is strictly less than $\frac{7x_1+3x_2}{x_1+x_2}$, we will ignore the case where r_4 is matched up to $x_1 + x_2$.

Hence the lower bound for any deterministic algorithm is the minimum of $\{3+2\frac{x_2}{x_1}, \frac{5x_1+3x_2}{x_1+x_2}\}$. This is maximized where $x_2 = \frac{\sqrt{5}-1}{2}x_1$ at which point the ratio is $2+\sqrt{5} > 4$. □

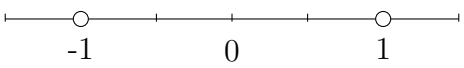
This lower bound is only for 5 servers, however. A higher lower bound would likely exist by analyzing these types of symmetric sets of servers for larger n, since by the symmetric-server-set theorems from above we see that any competitive ratio by a single server set can be replicated by a symmetric server set. Coming up with a lower bound for general n for these types of server-sets would likely be very powerful.

3.2 Lower Bounds – Randomized

Next, we will present a simple lower bound of 2 for the competitive ratio of any randomized algorithm on the minimum weight objective.

Theorem 5. The lower bound of any randomized algorithm on the Min-Weight objective is 2.

Proof. To achieve this, we only need 2 servers:



In this instance, the first request will arrive at 0 and the second request will arrive at either -1 or 1 with uniform probability. Hence, the optimal offline assignment will be to match the second request with whichever server it lands on and the first request with the other. This makes the optimal total weight a constant 1. However, any algorithm will have a 50% chance to match the first request with the server at the location of the second request. This means that any algorithm will have an expected cost of $\frac{1}{2}(1) + \frac{1}{2}(3) = 2$. Hence, the competitive ratio of any randomized algorithm must be at least 2. □

This lower bound unfortunately does not increase easily. It can be shown that increasing the number of servers while still maintaining the same format of the instance does not increase the lower bound. See the full version of this work for details.

3.3 Algorithms – Deterministic

Now we will discuss the current 'state of the art' deterministic algorithms for the minimum weight objective. The first algorithm we present is PERMUTATION, discovered by [6] and [7]. The main crux of the algorithms relies on an interesting fact about the sequence of optimal partial matchings.

Lemma 6. Let $R = (r_1, ..., r_n)$ be a sequence of requests and S a set of servers. Let $S_i \subset S$ be a set of servers that minimizes the cost of matching only the first i requests $(r_1, ..., r_i)$. Then there is a set of servers $T_{i+1} \subseteq S$ that minimizes the cost of matching only the first $i+1$ requests $(r_1, ..., r_{i+1})$ and satisfies $|T_{i+1} - S_i| = 1$.
For the proof see [6].

The pseudo code for the algorithm is as follows:

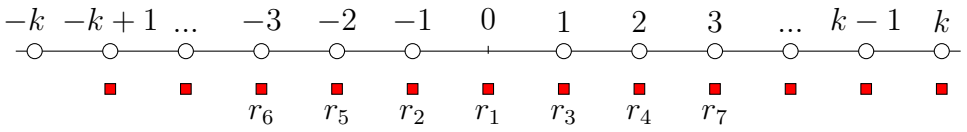
procedure PERMUTATION(R, S)
    for all requests r_i **do**
        Calculate the optimal partial matching of the first i requests into S.
        Let $\{s\} = T_i - S_{i-1}$ be the new server used via lemma 6.
        Match r_i to s.
    end for
end procedure

This algorithm achieves a $(2n-1)$-competitive ratio. It should also be noted that each request can be matched in polynomial time via this algorithm, as the optimal partial matchings can be calculated via the Hungarian algorithm or other algorithms designed for the assignment problem.

However, despite this algorithm's 'slick' structure and strong relationship with the sequence of optimal partial assignments, the PERMUTATION algorithm has a linear lower bound, even in the case of the line metric.

Lemma 7. PERMUTATION has a linear lower bound for its competitive ratio.

Proof. Consider the following instance with $n = 2k$ servers:



In this instance, we will assume that between tiebreakers under the PERMUTATION algorithm the server that minimizes the distance to the incoming request will be chosen, with the understanding that even while using a different method of tie-breaking a corresponding instance with a linear ratio can still be found. Hence when $r_1 = 0$ arrives, it will be matched to the server at 1. When r_2 arrives, the optimal matching for the first n requests will be using the servers $\{-1, 1\}$. Then when r_3 arrives, the choice of tiebreaker matches r_3 to 2. Then r_4 must be matched to -2, r_5 to -3, r_6 to 3, and so on, creating a pattern of switching off between crossing over to the other side of servers and going to the next highest server for every other request. This means that each odd request r_{2i+1} has a cost of 1 while each even request r_{2i} has a cost of $2i$. Thus the total cost of the matching is $\sum_{i=1}^{k}(2i+1) = k^2+k+k = k^2+2k$, while the optimal cost is always just k. Thus PERMUTATION's competitive ratio is lower bounded by $\frac{k^2+2k}{k} = \frac{n}{2} + 2$. □

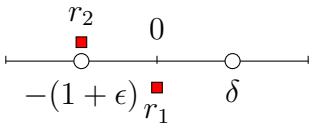
While this doesn't completely close the gap for PERMUTATION on the line, it does show that it has a strictly linear competitive ratio.

The best established deterministic algorithm was presented by Antoniadis et al. in [1]. This algorithm, here dubbed BRANCHING GREEDY, has no established lower bound and [1] claimed that the algorithm was $o(n)$-competitive. The pseudo-code for this algorithm is as follows:

procedure BRANCHING GREEDY(R, S, ϵ)
    for all requests r_i **do**

Initiate a counter $c = 1$ and store $r = r_i$.
Place a marker on the line at r_i.
while r_i is unmatched **do**
Move the marker to $r + (-(1+\epsilon))^c$
if a server is encountered **then**
if the server is unmatched **then**
Match r_i to the server.
end if
if the server is matched but the request r' it is matched to is further from the server than r **then**
Label the server as being matched to r.
Now continue with the marker where r' had left off, hence setting $r = r'$ and updating c.
end if
if r was not updated this iteration **then**
c = c+1
end if
end if
end while
end for
end procedure

This algorithm's complicated structure is due to its original format being written for a similar problem known as the k lost cows problem. We present an example of the algorithm's run through below:



Here ϵ and δ are real numbers with $0 < \delta < \epsilon$. The 2 servers are at $\{-(1+\epsilon), \delta\}$. The requests are $R = (0, -(1+\epsilon))$.

Since as cows each request will start by looking left, the first request r_1 will go to the server just below it $(-(1+\epsilon))$. Request r_2 will then arrive and 'knock out' the first request that had been matched there. More specifically the second request will then 'take over' as the first request that had been matched there and turn around, moving up to the server at δ. Hence, the online cost of this matching via the Branching Greedy algorithm will be $2(1+\epsilon) + \delta$, whereas the optimal cost would always be δ. Thus the competitive ratio of Branching Greedy is at least $\frac{(1+\epsilon)}{\delta}$. But for any $c > 0$, by letting $\delta = \frac{1+\epsilon}{c}$ (here assuming c is large enough to ensure that $\delta < \epsilon$), then the ratio must be at least c. This seems to show that no upper bound could be placed on the Branching Greedy algorithm. This currently appears to be a direct counter example to the results of [1], and we are in conversation with the authors to better understand their algorithm.

3.4 Minimax Matching

The next algorithm we consider is our MINIMAX MATCHING algorithm. So far, every algorithm (perhaps aside from BRANCHING GREEDY) could be implemented in polytime. Here our goal is to get the best possible approximation of some optimal solution - not necessarily to do it in some efficient time frame. In fact, it is conceivable that the best possible deterministic algorithm is only achievable in non-polynomial time. Perhaps it is the case that there exists some constant-competitive algorithm that runs in exponential time, but the best polytime algorithm is logarithmically competitive. So with that in mind, we present our algorithm, simply based on the principles of minimaxing (where we assume via the request collocation tool that requests arrive in the set of servers):

1: **procedure** MINIMAX MATCHING(R, S)
2: **for each** request r_i
3: **for all** possible future request sequences $(r_i, ..., r_n)$
4: Consider all possible strategies of matching $(r_i, ..., r_n)$ into the remaining available servers
5: Choose the strategy that minimizes the maximal competitive ratio a request sequence could force
6: Assign r_i to the corresponding server in this strategy.
7: **end procedure**

Theorem 8. Up to a constant factor in the competitive ratio, MINIMAX MATCHING (MM) is the best deterministic algorithm for the minimum weight objective.

Proof. Suppose that there is some better deterministic algorithm A. First, we may assume by Theorem 2 that all requests arrive in the set of servers. Since we are assuming that A is a different algorithm, then at some point for a given set of servers S and request sequence $R = (r_1, ..., r_n)$ it differs with MM, say at request r_i. That is, A matches r_i to a different server than MM. But since MM did not chose to match r_i to that server, then there must be some choices of sequences (all starting with $(r_1, ..., r_i)$) of requests that would have forced any matching to have a ratio no less than MM on this instance. Hence even if this algorithm A did achieve a lower ratio than MM on this sequence of requests then there exists some sequence of requests R' corresponding to the choices of sequences which A must have a ratio at least the value of MM's on R.

Now, let T be the instance which MM maximizes its competitive ratio on. Then every algorithm A has some corresponding instance T_A on which its ratio must be higher than MM's ratio on T - thus MM is the best deterministic algorithm under competitive analysis under instances in which all requests land on servers. □

Unfortunately, as mentioned earlier, this algorithm does not have an obvious polytime implementation - as it is written, its implementation would be $O(n^n)$, though this could possibly be reduced by using the Nearest Neighbors Theorem and ideas such as alpha-beta pruning. This algorithm can also be useful in other ways: by virtue of it necessarily being the best deterministic algorithm up to a constant factor, any established super constant lower bound for MM would effectively establish a superconstant lower bound for all deterministic algorithms!

3.5 Algorithms – Randomized

One difficulty in constructing deterministic algorithms is that adversarial request sequences can be placed knowing exactly how the algorithm will match them. Such an adversary is difficult to construct against randomized algorithms, where you don't necessarily know where a specific algorithm will match a given request. The best randomized algorithm established to date has been HARMONIC, established in [4], which acts like GREEDY but with an element of randomness. After manipulating the server set to ensure that the maximal distance between any two servers is $O(n^3)$, the algorithm is as follows:

1: **procedure** HARMONIC(R, S)
2: **for all** requests r_i **do**
3: Let s_ℓ and s_r be the closest available servers on the left and the right respectively.
4: Match to s_r with probability $\frac{r_i - s_\ell}{s_r - s_\ell}$, and to s_ℓ with probability $1 - \frac{r_i - s_\ell}{s_r - s_\ell} = \frac{s_r - r_i}{s_r - s_\ell}$.
5: **end for**
6: **end procedure**

Gupta and Lewi showed that HARMONIC is $O(\log n)$-competitive, making Harmonic the best previously established algorithm for minimum metric matching on the line under the minimum weight objective. There is no established lower bound for Harmonic.

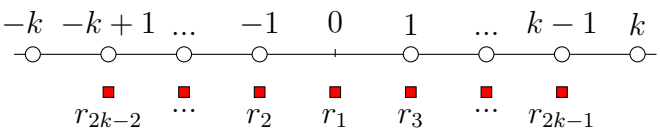
Again, however, one could apply the ideas of minimax to randomized algorithms. The algorithm would be similar in structure to its deterministic version, while now strategies can be mixed in principle. Since this is now equivalent to calculating mixed Nash equilibria, the algorithm's structure would be rather complex, but again its existence could again be used to construct lower bounds for randomized algorithms in a similar way.

4 Bottleneck Objective

In this section we will move away from the minimum weight objective and now consider the bottleneck objective. Much less research has been done previously for this objective, but we will discuss the known lower bounds for deterministic and randomized algorithms. In doing so, we will show that up to a constant factor difference in competitive ratio that PERMUTATION is the best randomized algorithm for the problem under the bottleneck objective.

4.1 Lower Bounds – Deterministic

The highest known lower bound for deterministic algorithms for the bottleneck objective is $1.5n$, established in [5]. However, a lower bound of $\frac{n}{2}$ is relatively easy to construct:



In this instance the servers are located at $\{-k, -(k-1), ..., -1, 1, ..., k-1, k\}$ and the request sequence is $R = (0, -1, 1, ..., -(k-1), k-1, r_{2k})$, where r_{2k} is placed according to the following rule: after the first $2k-1$ requests, there will be one server s remaining. If $s > 0$,

then assign $r_{2k} = -k$, otherwise assign $r_{2k} = k$. Note that this forces the last match to have a cost of at least $k = \frac{n}{2}$, whereas the optimal assignment will always have a bottleneck edge of 1 (k of the requests will be collocated, and the other k requests can be shifted by 1 to then all be collocated with servers, with no overlaps). Hence we have the immediate corollary:

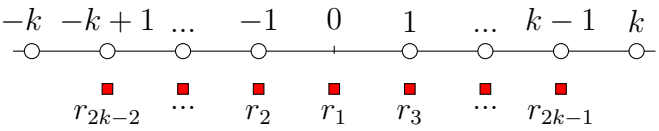
Theorem 9. No deterministic algorithm can achieve a better ratio that $O(n)$ for the bottleneck objective.

4.2 Lower Bounds – Randomized

Although there has been no established lower bound for randomized algorithms under the bottleneck objective, we can use a similar instance for the deterministic lower bound to construct a lower bound for randomized algorithms:

Theorem 10. The competitive ratio of any randomized algorithm for the bottleneck objective with n servers is no better than $\frac{n}{2} = \Omega(n)$.

Proof. We will construct an instance with $n = 2k$ servers, where the servers are located at $\{-k, -k+1, ..., -1, 1, ..., k-1, k\}$ and the requests will arrive at $r_1 = 0, r_2 = -1, r_3 = 1, r_4 = -2, r_5 = 2, ...,\ r_{2k-2} = -k+1$, and $r_{2k-1} = k-1$ with the final request r_{2k} landing either on $-k$ or k with uniform probability.



Note that after r_{2k-1} has been matched by any algorithm, there will always be exactly one server remaining, which we will denote by $s \in \mathbb{Z}$. The expected cost of the bottleneck edge in this algorithm will be at least the expected cost of the last edge created with s, since the maximal distance pairing will necessarily be bigger than or equal to $d(r_{2k}, s) = |r_{2k} - s|$. Since $s \geq -k$, the edge cost if $r_{2k} = -k$ will be $k + s$. If $r_{2k} = k$, then the edge cost will be $k - s$, since $s \leq k$. Hence, the expected cost of any algorithm will be at least $\frac{1}{2}((k+s) + (k-s)) = k$.

The offline optimal matching, however, would have a maximal pairing cost of 1. To achieve this, simply match the leftmost available server with the leftmost unassigned request and repeat. If $r_{2k} = k$, then all the nonpositive requests will have a distance of 1 to their respective server while all the positive requests will have a cost of 0. Thus the competitive ratio of any algorithm will be no less than $\frac{k}{1} = k$, where k is half the number of servers used. □

Thus for the bottleneck objective on n servers, the gap for randomized algorithms is between $\frac{n}{2}$ and $2n - 1$.

4.3 Upper Bounds

Unfortunately, very little analysis has been done for randomized algorithms under the bottleneck objective. In fact, our only upper bounds for the randomized bottleneck objective are simply inherited from the deterministic algorithms for the bottleneck objective. Yet in some ways this is sufficient - we have shown that the lower bound for randomized algorithms under the bottleneck objective is linear and the best known algorithm is PERMUTATION, which has been shown to be linearly competitive. Perhaps most interesting, this shows that the natural extension of the MINIMAX MATCHING algorithm to the randomized bottleneck objective does not do drastically better than any other algorithms.

5 Conclusions and Further Work

At the end, we are still left with many open questions. Even with the definitively best deterministic algorithm for the minimum weight objective, we still don't know whether there exists a constant-competitive algorithm or not. We do, however, know that the Bottleneck objective is a strictly "harder" problem than the Minimum weight, as we have a logarithmic upper bound for Minimum weight and a linear lower bound for the Bottleneck objective. To close this work, we have a short list of objectives that may prove useful for the adventurous reader:

1. Establish that any bound on the competitive ratio ρ can be transformed to a bound on *expanding* request sequences on ρ-server sets, as this would greatly reduce the types of instances to consider.

2. Establish a lower bound on the competitive ratio of the MINIMAX MATCHING algorithm, as this would also be a lower bound on the competitive ratio of any deterministic algorithm. In fact, the entire problem rests on the competitive ratio of the MINIMAX MATCHING algorithm.

3. Reformat the 'ranking' of algorithms to include the run time, so that non-polynomial time algorithms might not achieve more than polynomial time algorithms (note that this only matters if there doesn't exist a polytime algorithm that can compare to MINIMAX MATCHING).

References

[1] A. Antoniadis, N. Barcelo, M. Nugent, K. Pruhs, and M. Scquizzato. A o(n)-competitive determinstic algorithm for online matching on a line. In *WAOA14*, page to appear, 2014.

[2] N. Bansal, N. Buchbinder, A. Gupta, and J. Naor. A randomized $o(\log^2 k)$-competitive algorithm for metric bipartite matching. *Algorithmica*, 68(2):390–403, 2014.

[3] B. Fuchs, W. Hochstättler, and W. Kern. Online matching on a line. In *Electronic Notes In Discrete Mathematics*. Elsevier, 2003.

[4] Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming*, volume 7391 of *Lecture Notes in Computer Science*, pages 424–435. Springer Berlin Heidelberg, 2012.

[5] R. Idury and A. A. Schäffer. A better lower bound for on-line bottleneck matching. http://www.ncbi.nlm.nih.gov/core/assets/cbb/files/Firehouse.pdf. Accessed: 2015-09-7.

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

[7] S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127:251–264, 1994.

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

[9] A. Meyerson, A. Nanavati, and L. J. 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.