Permutation Strikes Back: The Power of Recourse in Online Metric Matching

Varun Gupta
University of Chicago, IL, USA
guptav@uchicago.edu

Ravishankar Krishnaswamy
Microsoft Research India, Bangalore, India
rakri@microsoft.com

Sai Sandeep
Carnegie Mellon University, Pittsburgh, PA, USA
spallerl@andrew.cmu.edu

Abstract

In this paper, we study the online metric matching with recourse (OMM-Recourse) problem. Given a metric space with k servers, a sequence of clients is revealed online. A client must be matched to an available server on arrival. Unlike the classical online matching model where the match is irrevocable, the recourse model permits the algorithm to rematch existing clients upon the arrival of a new client. The goal is to maintain an online matching with a near-optimal total cost, while at the same time not rematching too many clients.

For the classical online metric matching problem without recourse, the optimal competitive ratio for deterministic algorithms is $2k-1$, and the best-known randomized algorithms have competitive ratio $O(\log^2 k)$. For the much-studied special case of line metric, the best-known algorithms have competitive ratios of $O(\log k)$. Improving these competitive ratios (or showing lower bounds) are important open problems in this line of work.

In this paper, we show that logarithmic recourse significantly improves the quality of matchings we can maintain online. For general metrics, we show a deterministic $O(\log k)$-competitive algorithm, with $O(\log k)$ recourse per client, an exponential improvement over the $2k-1$ lower bound without recourse. For line metrics we show a deterministic 3-competitive algorithm with $O(\log k)$ amortized recourse, again improving the best-known $O(\log k)$-competitive algorithms without recourse. The first result (general metrics) simulates a batched version of the classical algorithm for OMM called Permutation. The second result (line metric) also uses Permutation as the foundation but makes non-trivial changes to the matching to balance the competitive ratio and recourse.

Finally, we also consider the model when both clients and servers may arrive or depart dynamically, and exhibit a simple randomized $O(\log n)$-competitive algorithm with $O(\log \Delta)$ recourse, where n and Δ are the number of points and the aspect ratio of the underlying metric. We remark that no non-trivial bounds are possible in this fully-dynamic model when no recourse is allowed.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases online algorithms, bipartite matching

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.40

Category APPROX

Related Version A full version of the paper is available at `https://arxiv.org/abs/1911.12778`.

1 Introduction

The classical online metric matching (OMM) problem is defined on a metric space $(\mathcal{X}, d)$, where $\mathcal{X}$ denotes a set of n points where the servers and clients are located, and a distance function (metric) $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$. A set $S \subseteq \mathcal{X}$ of servers, $|S| = k$, is given offline, and a sequence of client requests $C = (c_1, \ldots, c_k)$ is revealed in an online manner. The algorithm is required to match each client request to an available (previously unmatched) server on arrival irrevocably. The objective is to minimize the total cost of the matching, which is the sum of distances between matched client-server pairs. The performance of an algorithm is measured using the notion of competitive ratio – the worst-case over all instances of the ratio of the cost of the online algorithm and the cost of an optimal offline matching.

This problem was first studied in two independent works [17, 18]. Both these works present a $(2k-1)$-competitive deterministic algorithm called PERMUTATION, and also show that *this bound is tight among deterministic algorithms*. Subsequently, the work of [22] shows that randomization can overcome this lower bound (for oblivious adversaries) by giving a $\mathcal{O}(\log^3 k)$-competitive[1] randomized algorithm, which was later improved in [4], where the authors show an $\mathcal{O}(\log^2 k)$-competitive algorithm. In contrast, the best known lower bound for randomized algorithms is a factor of $\Omega(\log k)$[22]. Resolving this gap has remained a challenging and long-standing open problem in this area.

The OMM problem has also elicited much interest in specialized metrics such as the line metric (OMM-LINE). It was long conjectured that OMM-LINE should admit a 9-competitive algorithm, by virtue of the connections between this problem and another classical problem in online algorithms known as the cow-path problem. However, [10] disproved this particular conjecture by presenting a lower bound of 9.001. In terms of upper bounds, no algorithms with better competitive ratios than those for general metrics were known until a recent line of work [15, 23, 25] gave improved algorithms for the line metric. The current best algorithm is a deterministic $\mathcal{O}(\log k)$-competitive algorithm [25]. It is still an open question whether constant-competitive algorithms exist for OMM-LINE. Intriguingly, there are $\Omega(\log k)$ lower bounds on natural families of algorithms [1, 19].

Given these barriers for OMM, we study whether we can obtain strictly better performance if we are *allowed to re-match a few clients* upon the arrival of a new client.

▶ **Problem 1** (OMM-RECOURSE). *An instance consists of a metric space $(\mathcal{X}, d)$, and a set $S \subseteq X$ of servers with $|S| = k$. A sequence of client requests $C_k = (c_1, \ldots, c_k)$ is revealed in an online manner. At time t, after the algorithm observes c_t, it must maintain a matching $\mathcal{M}_t$ such that every client is matched to exactly one server, and each server is matched to at most one client. The algorithm can re-match some earlier clients, and the number of times clients are re-matched is called the recourse.*

▶ **Definition 2.** *We say that an online algorithm is α-competitive with β-amortized recourse for* OMM-RECOURSE *if for all $t \in [k]$, the cost of the algorithm's matching for $C_t := (c_1, \ldots, c_t)$ is at most α times the cost of the optimal matching for C_t, and the total number of recourse steps taken so far is at most βt. Additionally, the algorithm is said to have a per-client recourse of β if no client is rematched more than β times.*

While our main theoretical motivation is in understanding the power of recourse in the classical OMM problem, often it is also the case in practice that matching decisions *are not irrevocable*, and instead, there is simply a cost (or) penalty for re-assignments. For example, in a video streaming setting, users arrive online and want to stream a video. The ISP must choose a server to stream from, preferring a server closer to the user. Of course, this decision can be changed over the time horizon, but this will cause a temporary disruption that must be minimized. The recourse model then naturally captures the competing goals of minimizing cost and the number of re-assignments. Moreover, the stronger notion of *per-client* recourse guarantees a fairness property by bounding the inconvenience for each client.

[1] Throughout the paper, logarithms are with respect to base 2.

The main results of this paper affirmatively answer the question of whether limited recourse can help in the OMM problem.

▶ **Theorem 3.** *There is an efficient deterministic* $2\log k$*-competitive algorithm with* $\log k$ *per-client recourse for* OMM-Recourse *on general metrics. (Section 3)*

Theorem 3 coupled with the $(2k-1)$ lower bound for deterministic algorithms without recourse highlights the exponential improvement in competitive ratio possible with limited recourse. We complement the above result by noting that the guarantees given above are tight for our algorithm. We also show that *any deterministic algorithm* which has constant per-client recourse must have a logarithmic competitive ratio.

▶ **Theorem 4.** *No deterministic algorithm for* OMM-Recourse *with per-client recourse at most an absolute constant* C *can have a competitive ratio* $o(\log k)$*. (Appendix C)*

Our algorithm for Theorem 3 is an adaptation of the classical Permutation algorithm for OMM [17, 18] which recomputes the offline optimal matching on the arrival of a new client, and matches the arriving client to the server that is part of the new offline matching but as yet unmatched in the online matching. While Permutation is a very natural and elegant algorithm, it fares poorly in the standard model without recourse where it has a competitive ratio of $\Omega(k)$ *even on line metrics*! At a high level, the worst-case behavior for the classical Permutation algorithm for OMM occurs when clients arrive one-by-one, and the competitive ratio improves if the clients arrive in batches. Our algorithm then *uses recourse to mimic* the output of such a batched version of Permutation.

We next turn our attention to the special case of the line metric, where we present a special-purpose algorithm that significantly improves on Theorem 3:

▶ **Theorem 5.** *There is a deterministic* 3*-competitive algorithm with* $\mathcal{O}(\log k)$*-amortized recourse for* OMM-Line-Recourse*. (Section 4)*

Note that obtaining constant-competitive algorithms for the line metric has been a challenging open problem for OMM– our algorithm achieves such a guarantee when equipped with a small amount of recourse. This result is also the main technical contribution of our work. Once again, our algorithm builds on the classical Permutation algorithm for OMM. Exploiting the nature of the line metric, we view the online matching determined by Permutation as a collection of directed forward or backward arcs from the clients to servers. Noting that such a matching may be sub-optimal only if *overlapping* forward and backward arcs are present, our algorithm tries to re-match some clients to eliminate such overlaps. However, blindly re-matching to eliminate all overlapping arcs can lead to a large recourse. Instead, we propose a clever *asymmetric uncrossing method* to balance the competitive ratio and recourse. Our analysis is somewhat non-standard in that we first identify a *family of algorithms* for OMM-Line-Recourse, all of which share the same asymmetric uncrossing criterion, and incur the same cost. Of this family, we first choose one algorithm (whose cost analysis is most intuitive) to bound the competitive ratio of *all the algorithms* in the family. However, this algorithm can demonstrably incur large recourse. Hence, our actual algorithm is another one from this family which is designed for minimizing the recourse but whose cost analysis in a direct manner is not apparent to us.

Finally, we focus on another limitation of classical OMM– due to the irrevocable nature of assignments, the competitive ratio would be unbounded when both clients and/or servers can arrive or depart the system. Hence, the classical model only considers the setting when all servers are known ahead of time and clients arrive in an online manner. We show that by allowing recourse, we can, in fact, handle arrivals and departures of clients and servers:

▶ **Theorem 6.** *There is a randomized $\mathcal{O}(\log n)$-competitive algorithm with $\mathcal{O}(\log \Delta)$ amortized recourse for* OMM-Recourse *when clients and servers can arrive and depart, where Δ is the aspect ratio of the metric space.*

We defer the details of this dynamic model to [16].

Related Work. To the best of our knowledge, the only work which considers recourse for online min-cost matching is the recent work [20] where the authors consider a *two-stage* version of the uni-chromatic problem (where there is no distinction between servers and clients): In the first stage, a perfect matching between $2n$ given nodes must be selected; in the second stage $2k$ new nodes are introduced. The goal is to produce α-competitive matchings at the end of both stages, and such that the number of edges removed from the first stage matching is at most βk. The authors show that $\alpha = 3, \beta = 1$ and $\alpha = 10, \beta = 2$ are possible when k is known or unknown, respectively. Our results can be seen as a multi-stage generalization of this two-stage model, although the two models are slightly different in terms of the distinction between servers and clients.

A related model capturing a different kind of flexibility in online matching is that of *matching with delays* [8, 5]: here, the requests need not be matched at the time of arrival, but accrue a delay penalty until the algorithm matches them. The algorithm must minimize the total matching cost plus total delay penalty. The current best known randomized algorithms are $\mathcal{O}(\log n)$ competitive [2], which also proves a lower bound of $\Omega\left(\frac{\log n}{\log\log n}\right)$. The best known deterministic algorithms are $\mathcal{O}(k^{0.59})$-competitive [3]. Another class of beyond-worst case models are stochastic models, such as *i.i.d.* and *random order* settings. The majority of work in this vein has been done in the reward maximization objective (see e.g., [11, 7, 6] and references therein). For OMM, [24] gave a deterministic algorithm that is simultaneously $\mathcal{O}(\log k)$-competitive in the random order model and $(2k-1)$-competitive in worst case. Recently, [12] show $\mathcal{O}((\log\log\log k)^2)$-competitive algorithms in the known *i.i.d.* model.

Finally, online algorithms with recourse have also been studied in other settings such as scheduling and set cover (see, e.g., [14, 13, 9] and the references within.).

Recently, and independent of our work, [21] have obtained an $\mathcal{O}(1)$-competitive algorithm with $\mathcal{O}(\log k)$-amortized recourse for OMM-Line-Recourse with a very different approach of extending the t-net framework of [25].

2 Preliminaries

For most of the paper (except for the fully dynamic setting), we consider the setting where the servers $\mathcal{S}$ are known up front. The clients arrive online, and we denote by $C_t = (c_1, \ldots, c_t)$ the set of the first t clients. An optimal matching between C_t and $\mathcal{S}$ is denoted by $\mathcal{M}_t^*$, and similarly, the algorithm's matching between C_t and $\mathcal{S}$ will be denoted by $\mathcal{M}_t$. We denote by OPT_t, the cost of the optimal matching $\mathcal{M}_t^*$. For any matching $\mathcal{M}$, we use $\mathcal{M}(c)$ and $\mathcal{M}(C)$ to denote the server and the set of servers matched to the client c and the set of clients C, respectively. We define $\mathcal{M}(s)$ and $\mathcal{M}(S)$ similarly.

The Permutation algorithm. As mentioned in Section 1, [17] and [18] independently proposed a $(2k-1)$-competitive algorithm Permutation for OMM. Since our algorithms build extensively on this algorithm, we first summarize Permutation and its key properties. The algorithm maintains two matchings: the current online matching $\mathcal{M}_t$, and the optimal offline matching $\mathcal{M}_t^*$ of the clients C_t that have arrived so far. The main observation behind

Algorithm 1 PERMUTATION (metric $(\mathcal{X}, d)$, server-optimal matching $\mathcal{M}_{t-1}$ for C_{t-1}).

```
1: for new batch of clients $C_{\text{cur}} = C_{t+\ell} \setminus C_{t-1} = \{c_t, c_{t+1}, \dots, c_{t+\ell}\}$ that arrives do
2:     let $\mathcal{M}^*_{t-1}$ and $\mathcal{M}^*_{t+\ell}$ be optimal matchings for $C_{t-1}$ and $C_{t+\ell}$ from Lemma 7.
3:     let $S_{\text{cur}} = S^*_{t+\ell} \setminus S^*_{t-1}$ be the set of $\ell + 1$ servers matched in $\mathcal{M}^*_{t+\ell}$ but not in $\mathcal{M}^*_{t-1}$
4:     let $\mathcal{M}_{\text{cur}}$ denote the minimum cost matching between $C_{\text{cur}}$ and $S_{\text{cur}}$
5:     augment $\mathcal{M}_{t-1}$ using $\mathcal{M}_{\text{cur}}$ to obtain the new matching $\mathcal{M}_{t+\ell}$
6: end for
```

the algorithm is that, when a new client c_{t+1} arrives, there exists an optimal matching of C_{t+1} to $\mathcal{S}$ which uses *exactly the servers used in $\mathcal{M}^*_t$ plus one extra server.* PERMUTATION simply identifies the extra server s_{t+1} and matches c_{t+1} with s_{t+1}. This property can be formalized as follows.

▶ **Lemma 7** (Lemma 2.2 in [17]). *There exists a sequence of optimal matchings $\mathcal{M}^*_1, \dots, \mathcal{M}^*_k$ matching client sets $C_1, \dots, C_k$ to $\mathcal{S}$ such that the sets of servers used in these matchings, $S^*_i := \mathcal{M}^*_i(C_i)$ are nested, i.e., $S^*_1 \subseteq S^*_2 \subseteq \dots \subseteq S^*_k$.*

▶ **Definition 8** (Server-optimal matching). *At time t, a matching $\mathcal{M}_t$ of client-set C_t is said to be* server-optimal *if it uses the same servers as $\mathcal{M}^*_t$, i.e., $\mathcal{M}^*_t(C_t) = \mathcal{M}_t(C_t)$.*

▶ **Proposition 9** ([17]). PERMUTATION *always maintains a server-optimal matching.*

The notion of server-optimality will be crucial throughout this paper. Intuitively, it says that the algorithm has identified the right set of servers from $\mathcal{S}$ to match to, and is only sub-optimal w.r.t to the actual matching maintained between C_t and $\mathcal{M}_t(C_t)$.

Algorithm 1 gives a more general version of PERMUTATION which we will use later, where clients arrive in *batches*, and we add a *minimum-cost matching* between the arriving clients in the batch and the additional servers an optimal solution uses (from Lemma 7).

▶ **Lemma 10** ([17]). *After the arrival of a batch of clients $C_{t+\ell} \setminus C_{t-1}$, the cost of the matching $\mathcal{M}_{\text{cur}}$ computed in Algorithm 1 is at most $2\mathsf{OPT}_{t+\ell}$.*

▶ **Theorem 11** (Theorem 2.4 in [17]). *Algorithm 1 is $(2m-1)$-competitive for online weighted matching if the requests arrive in m batches.*

3 Online matching with recourse for general metrics

In this section, we prove Theorem 3 by showing that Algorithm 2 is a $(2\log k, \log k)$-competitive algorithm for OMM-RECOURSE in a general metric. To motivate the algorithm, note that Theorem 11 says that in order to minimize competitive ratio, it is best to feed the client sequence to PERMUTATION in as few batches as possible. However, we are also constrained in matching clients immediately on arrival. One way of balancing the two goals is to run PERMUTATION incrementally on each client arrival, and use recourse to periodically unmatch a suffix of client sequence and re-introduce these clients as *a single batch*. As an example, assume that we create B batches of k/B clients, with the jth batch consisting of clients $\text{Batch}_j = \{(j-1)k/B+1, \dots, jk/B\}$. As clients in batch j arrive, we first match them via vanilla PERMUTATION. After the (jk/B)th client arrives, we unmatch all clients in Batch_j and re-introduce them as one single batch. The amortized recourse of this algorithm is 1. Moreover, the matching at any time t may be viewed as the output of running PERMUTATION with at most B batches of k/B clients each and k/B batches of 1 client each. Setting $B = \sqrt{k}$ then gives us an $O(\sqrt{k})$-competitive algorithm with a per-client recourse of 1!

Algorithm 2 MultiScalePermutation (metric (X, d) and server set $\mathcal{S} \subseteq X$).

```
1: initialize matching $\mathcal{M}_0 = \emptyset$
2: for each new client $c_t$ that arrives at time-step $t$ do
3:     let $i(t) = \arg\max_i$ s.t $t$ is divisible by $2^i$
4:     unmatch the latest $2^{i(t)}$ clients $\{c_{t-2^{i(t)}+1}, \dots, c_t\}$ and revert to matching $\mathcal{M}_{t-2^{i(t)}}$
5:     introduce a block of clients $\{c_{t-2^{i(t)}+1}, \dots, c_t\}$ to Algorithm 1 with the current matching being $\mathcal{M}_{t-2^{i(t)}}$ and update $\mathcal{M}_t$ to be the resulting matching for the clients $C_t$
6: end for
```

To get a smaller competitive ratio at the expense of slightly higher recourse, we employ the following natural extension: imagine the first t clients as the leftmost t leaves of a balanced binary tree of depth $\lceil \log_2 k \rceil$. If an arriving client is the last leaf of some subtree, we unmatch all clients in the largest such subtree and re-introduce them as a single batch. The competitive ratio would then be bounded by $2 \log k$ since the matching at time t is simply the combination of at most $\log k$ batches (based on the binary decomposition of t) and by using Theorem 11. Further, any client is rematched at most $\log k$ times, since the size of the batch it is part of doubles on every rematch.

Proposition 12 generalizes the result in Theorem 3 to give a trade-off between the cost and recourse, and Proposition 13 proves that our analysis of Algorithm 2 is tight. The proofs of these propositions appear in Appendix B.

▶ **Proposition 12.** *Algorithm 2 with the constant 2 replaced by d, gives an $(d-1)\log_d k$-competitive algorithm with $\log_d k$-per client recourse. In particular, for any $d = \mathcal{O}(1)$ we get $\mathcal{O}(\log k)$-competitive algorithm with $\mathcal{O}(\log k)$-per client recourse, and for $d = k^\alpha$ ($\alpha \le 1$), we get an $\tilde{\mathcal{O}}(k^\alpha)$-competitive algorithm with $1 + 1/\alpha$-per client recourse.*

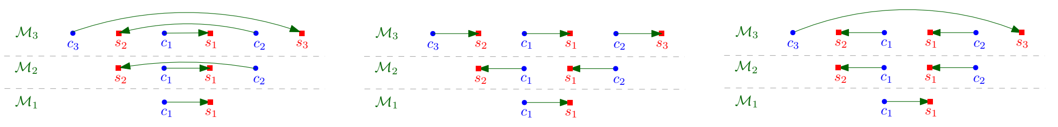
▶ **Proposition 13.** *The cost-recourse tradeoff of Theorem 3/Proposition 12 is tight: for $d = 2$, there is a sequence of instances where Algorithm 2 is $\Omega(\log k)$-competitive and has $\Omega(\log k)$ per-client recourse. Further, there is an increasing sequence $\{d_i\}$, such that with $d = d_i$ there is a sequence of instances where Algorithm 2 is $\Omega(d \log_d k)$-competitive with $\Omega(\log_d k)$ per-client recourse.*

4 Online Matching on the Line Metric

In this section, we focus on the special case of a line metric, where, for all points $x \in \mathcal{X}$, we associate a location $\ell : \mathcal{X} \to \mathbb{R}$ such that $d(x, y) = |\ell(x) - \ell(y)|$. We also assume without loss of generality that all the clients and servers are in distinct locations on the line.

Our starting point is again the Permutation algorithm which, by itself can have $\Omega(k)$ competitive ratio even on line metrics: see Figure 1a, where the distance between any consecutive client and server is 1[2]. Permutation would first match c_1 to s_1, and then c_2 to s_2 (the set $\{s_1, s_2\}$ is server-optimal as it admits an optimal matching $(c_1, s_2), (c_2, s_1)$). Continuing in this manner, Permutation would incur a total cost of $\Omega(k^2)$, whereas the optimal matching would have a cost of k. A natural fix would seem to be to re-match the clients and servers in the existing matching to maintain an optimal matching at all times, but as illustrated in the example in Figure 1b, this can lead to $\Omega(k)$ amortized recourse.

[2] Even though in this example it seems that Permutation can perform well by breaking ties correctly, we can modify the edge lengths very slightly to force the matchings, thus proving that no clever tie-breaking can help the algorithm.



(a) Permutation without re-match. **(b)** Permutation with full re-match. **(c)** *Asymmetric* Permutation with re-match.

Figure 1 Illustrative examples of OMM-Line-Recourse.

One of our main observations is that we can view a matching on a line metric as composed of *forward* and *backward* arcs between the matched clients and servers based on their relative location (as in Figure 1a), and, as we formalize later, a matching between a given client and server set is sub-optimal *if and only if* it contains overlapping forward and backward arcs. Indeed, Permutation does no recourse but has large overlaps, and always re-matching overlapping arcs output by Permutation yields an optimal solution but with large recourse. Our idea is to balance the overlap and recourse by *re-matching overlapping pairs asymmetrically.* When a new client c_t arrives, let s_t be the new server that Permutation brings in to the system. Then, if (c_t, s_t) is a forward arc i.e. if $\ell(c_t) \leq \ell(s_t)$, our algorithm simply adds it as is, even if it overlaps with existing backward arcs. On the other hand, when (c_t, s_t) is a backward arc, we use rematches to *maximally cancel* portions of this backward arc with overlapping portions of existing forward arcs. See Figure 1c for an example where $\mathcal{M}_2$ has undergone a re-matching, while $\mathcal{M}_3$ has not.

While this re-matching process is unambiguous for the example in Figure 1c, in general, there could be multiple ways of re-matching overlapping arcs in $\mathcal{M}_{t-1} \cup \{c_t, s_t\}$. We, therefore, begin by defining a *family of asymmetric maximally canceling algorithms* in Section 4.1 and prove that all algorithms in this family incur the same cost. In Section 4.2, we study one special algorithm, RecursiveCancel in this family which is the most amenable for cost analysis and bound the competitive ratio of the entire family of algorithms by 3. However, RecursiveCancel can incur a large $\Omega(k)$ recourse, and so our final algorithm MinimumCancel in Section 4.3 further identifies a way of re-matching backward arcs which *minimally changes the existing matching.* Using this property, we show that the MinimumCancel algorithm has $O(\log k)$ amortized recourse, thereby proving Theorem 5.

Before going into the details of the algorithms, we first introduce a property of the Permutation algorithm on the line metric that is useful:

▶ **Lemma 14.** *If* Permutation *maintains edges $(c_1, s_1), (c_2, s_2), \ldots, (c_t, s_t)$ at time t, then any server $s_{t'}$ added by the algorithm at time $t' > t$ will lie outside all these arcs.*

The proof follows from the server optimality of the Permutation algorithm and is presented in Appendix B.

4.1 Preliminary Concepts and Notation

We begin by introducing a few concepts that are important for cost analysis of matching on the line metric: forward and backward arcs, atomic intervals, and discrepancy. We end the section by defining the family of asymmetric maximally canceling algorithms. Unless otherwise stated, $C_t = \{c_1, c_2, \ldots, c_t\}$ will denote the set of clients which have arrived by time t, and $S_t = \{s_1, s_2, \ldots, s_t\}$ will denote the set of servers chosen by Permutation.

▶ **Definition 15** (Forward and Backward Arcs). *Let a client c be matched to server s in a given matching $\mathcal{M}$. We call the edge (c, s) a* forward arc *$\overrightarrow{c, s}$ if $\ell(c) \leq \ell(s)$ and a* backward arc *$\overleftarrow{s, c}$ otherwise.*

We now note that in an optimal matching, no pair of forward and backward arcs cross.

▶ **Proposition 16.** *Consider a set C of clients and a set S of servers with $|C| = |S|$. Then, a matching $\mathcal{M}$ between C and S is optimal for the client-server set (C, S) if and only if, for any forward arc $\overrightarrow{c_1, s_1} \in \mathcal{M}$ and backward arc $\overleftarrow{s_2, c_2} \in \mathcal{M}$, the intervals $[\ell(c_1), \ell(s_1)]$ and $[\ell(s_2), \ell(c_2)]$ are disjoint.*

Now, to compute the cost of any matching $\mathcal{M}_t$ between $C_t \cup S_t$, our approach will be to decompose the total cost into the contribution of what we call *atomic intervals* corresponding to $C_t \cup S_t$. Informally, the atomic intervals partition the line into open intervals between *two consecutive points* in $C_t \cup S_t$ in the metric space.

▶ **Definition 17** (Atomic intervals). *For two nodes $p_1, p_2 \in C_t \cup S_t$ in the matching $\mathcal{M}_t$ at time t, we call the open interval $I = (\ell(p_1), \ell(p_2))$ an* atomic interval *if and only if for any other $p \in C_t \cup S_t$, it holds that $\ell(p) \notin (\ell(p_1), \ell(p_2))$. We denote the set of all atomic intervals at time t by $\mathcal{AI}_t$. We denote by $|I| = |\ell(p_1) - \ell(p_2)|$ the length of the interval I.*

Note that $\mathcal{AI}_t$ depends only on the set $C_t \cup S_t$, and grows in t as this set expands. We call $I' = (\ell(p'_1), \ell(p'_2))$ a subinterval of $I = (\ell(p_1), \ell(p_2))$, denoted $I' \subseteq I$ if $\ell(p_1) \leq \ell(p'_1) \leq \ell(p'_2) \leq \ell(p_2)$.

▶ **Definition 18** (Discrepancy). *For any atomic interval $I \in \mathcal{AI}_t$ with its left end-point at location l, we define its* discrepancy *at time t to be the excess number of servers to the left of l in the currently used set of servers S_t: $\mathrm{disc}_t(I) := |S_t \cap (-\infty, l]| - |C_t \cap (-\infty, l]|$.*

The next lemma shows that the discrepancy of atomic intervals immediately gives the cost of the optimal matching.

▶ **Lemma 19.** *At any time t, and for any atomic interval $I \in \mathcal{AI}_t$, exactly $|\mathrm{disc}_t(I)|$ arcs cross I in an optimal matching between C_t and S_t. Further, if $\mathrm{disc}_t(I)$ is positive (respectively, negative), the direction of the crossing arcs is backward (resp., forward). Consequently, the cost of an optimal matching is $\mathsf{OPT}_t = \sum_{I \in \mathcal{AI}_t} |\mathrm{disc}_t(I)| \cdot |I|$.*

While $\mathrm{disc}_t(I)$ quantifies the minimum number of arcs that must cross I in any feasible matching, there could be many more arcs crossing I in the sub-optimal matching maintained by our algorithm. To this end, for a given matching $\mathcal{M}_t$ between $C_t \cup S_t$, and any subinterval $I' \subseteq I \in \mathcal{AI}_t$, let $n_t^f(I')$ and $n_t^b(I')$ denote the number of forward and backward arcs, respectively, crossing I' at time t. The following is then easy to see.

▷ Claim 20. For a matching $\mathcal{M}_t$, and any atomic interval $I \in \mathcal{AI}_t$, we have $n_t^b(I) - n_t^f(I) = \mathrm{disc}_t(I)$. Furthermore, the cost of $\mathcal{M}_t$ is $\mathsf{cost}(\mathcal{M}_t) = \sum_{I \in \mathcal{AI}_t} |n_t^f(I) + n_t^b(I)| \cdot |I|$.

We are now ready to define the family of asymmetric maximally canceling algorithms for OMM-LINE-RECOURSE.

▶ **Definition 21** (Asymmetric maximally canceling algorithm). *We call an algorithm an* asymmetric maximally canceling algorithm *if the sequence of matchings $\{\mathcal{M}_t\}$ produced by the algorithm satisfies:*

1. *The server sets $\{S_t\}$ are given by* PERMUTATION *(with a deterministic tie breaking rule);*
2. *Denoting s_t as the new server added by* PERMUTATION *at time t:*
   - *If (c_t, s_t) is a forward arc, then no rematches are made. In other words, for all atomic intervals $I \in \mathcal{AI}_t$ which overlap with $(\ell(c_t), \ell(s_t))$ we have: $n_t^f(I) = n_{t-1}^f(I) + 1$.*

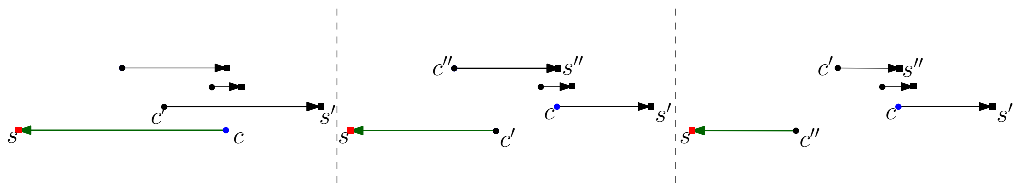


Figure 2 Illustration of RECURSIVECANCEL: The leftmost figure shows the initial matching – (c, s) is the new backward arc and (c', s') is the forward arc chosen at step 9 of first iteration. The center figure shows the matching at step 11 after the rematching. The edge (c', s) becoems the new backward arc, and (c'', s'') is the forward edge picked in step 9 for the next iteration where (c', s) and (c'', s'') are replaced by (c'', s) and (c', s'') as in the rightmost figure.

- *If (c_t, s_t) is a backward arc, we use it to cancel existing forward arcs as much as possible. In other words, in the resulting matching $\mathcal{M}_t$ (after recourse), for all atomic intervals $I \in \mathcal{AI}_t$ which overlap with $(\ell(s_t), \ell(c_t))$ we have: $n_t^f(I) = \max\{n_{t-1}^f(I) - 1, 0\}$.*
- *$n_t^f(I)$ is unchanged for all atomic intervals that do not overlap with $(\ell(c_t), \ell(s_t))$.*

Note that it suffices to define just n_t^f as $n_t^b(I)$ can be derived from $n_t^f(I)$ for all I using Claim 20. Furthermore, the definition does not restrict how the rematches are performed, but only that all asymmetric maximally canceling algorithms have the same values of $n_t^f(I)$ and $n_t^b(I)$, for all the atomic intervals. Claim 20 immediately gives:

▷ Claim 22. All asymmetric maximally canceling algorithms incur the same matching cost.

The next section gives a bound on this cost by focusing on a specific member in this family by exploiting a useful invariant.

4.2 Algorithm RECURSIVECANCEL for Bounding Cost

We now present our algorithm RECURSIVECANCEL (Algorithm 3) and bound its cost. When a new backward arc overlapping with existing forward arcs is added, RECURSIVECANCEL chooses the overlapping forward arc with the rightmost server and rematches the clients and servers of these two arcs to elimate overlap. This procedure results in at most one remnant backward arc (a strict suffix, i.e., server end, of the original backward arc) which can overlap with existing forward arcs. RECURSIVECANCEL then recurses on this remnant until there is no overlap with a forward arc.

We begin with a couple of simple but useful observations about the behavior of rematches made by RECURSIVECANCEL.

▶ **Lemma 23.** *Let (c, s) and (c', s') be two arcs defined in steps 8 and 9, respectively, of an iteration of* RECURSIVECANCEL*. Then, (i) the new edge (c', s) added in step 11 is a backward arc, thereby proving that the while loop is well-defined; and (ii) the edge (c, s') added in step 10 is either a backward arc, or it is a suffix of the original forward arc (c', s').*

Proof. From Lemma 14, we know that there are no free servers inside an arc added by PERMUTATION. Furthermore, since any arc in the current matching under RECURSIVECANCEL is a sub-arc of some original arc added by PERMUTATION, throughout the execution of RECURSIVECANCEL algorithm there is never a free server inside an arc in the matching $\mathcal{M}_t$. Thus, c' is to the right of s, otherwise the server s which is free in $\mathcal{M}_{t-1}$, is inside the arc (c', s'). The second part of the lemma can be proved by noting that the forward arc (c', s') intersects the backward arc (c, s), and thus $\ell(c') < \ell(c)$. ◀

Algorithm 3 Algorithm RecursiveCancel.

```
1: set $\mathcal{M}_0 = \emptyset$
2: for each client $c_t$ arriving at time $t \geq 1$ do
3:     let $s_t$ be the server Permutation matches $c_t$ to, and let $a := (c_t, s_t)$
4:     if $a$ is a forward arc, i.e., $\ell(c_t) < \ell(s_t)$ then
5:         $\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{a\}$
6:     else                                          ▷ $(c_t, s_t)$ is a backward arc
7:         while there exists a forward arc in $\mathcal{M}_{t-1}$ which overlaps with $a$ do    ▷ $a$ is the current backward arc
8:             let $(c, s) := a$
9:             let $(c', s')$ be the forward arc overlapping with $a$ with the *rightmost* server $s'$
10:            $\mathcal{M}_{t-1} = \{\mathcal{M}_{t-1} \setminus \{(c', s')\}\} \cup \{(c, s')\}$
11:            set $a := (c', s)$  ▷ From Lemma 23, $a$ will be a backward arc for loop recursion
12:        end while
13:        $\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{a\}$ ▷ $a$ now has no overlapping forward arcs, and is added to $\mathcal{M}_t$
14:    end if
15: end for
```

▶ **Lemma 24.** *Suppose that at some time t, the algorithm adds a backward arc (c, s) to the matching $\mathcal{M}_t$ (either by rematching an older edge in step 10 during the recursion, or as a new edge in step 13) at the end of the recursion. Then, this arc does not overlap with any existing forward arcs in $\mathcal{M}_t$.*

Proof. The proof follows directly from the fact that we use the forward arc with the rightmost server in the step 9 of the algorithm. ◀

We now try to quantify the excess cost of RecursiveCancel over the optimal solution using the above-defined quantities. Indeed, if RecursiveCancel adds an edge (c, s) which is a backward arc at some time step t, then from Lemma 24, we know that it does not have any existing forward arc overlapping with it, and hence would not contribute to any sub-optimality at this time. Hence, RecursiveCancel is sub-optimal only due to the addition of forward arcs which have overlaps with existing backward arcs. These could happen either in step 5 or in step 10. However, by Lemma 23, we also know that the forward arcs added in step 10 are only suffixes of the original forward arcs, and so the excess cost due to the rematched forward arc is only at most that of the original forward arc. Using this intuition, we label each atomic interval of each forward arc as either *redundant* (i.e., cost avoided by OPT_t) or *non-redundant* (cost incurred by OPT_t) as follows:

▶ **Definition 25** (Redundant/non-redundant forward arcs with respect to atomic intervals). *Suppose at time t, client c_t is matched to server s_t with a forward arc $a := (c_t, s_t)$ in step 5 of Algorithm 3. Then, the forward arc a is said to be redundant with respect to $I \in [\ell(c_t), \ell(s_t)]$ if $n^b_{t-1}(I) > n^f_{t-1}(I)$, and* non-redundant *with respect to I otherwise. Alternatively, if a new forward arc (c, s') is added in step 10 of the recursion then it must be the suffix of some forward arc $(c', s') \in \mathcal{M}_{t-1}$ (Lemma 23), and for any atomic interval I (or its subinterval), the new arc simply inherits its status (redundant/non-redundant) from the status of (c', s') with respect to I.*

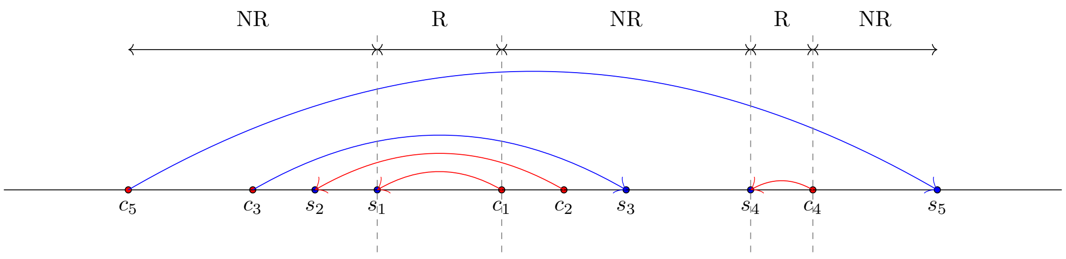


Figure 3 Illustration of Definition 25. In this example, when forward arc $a = (c_5, s_5)$ is added, a is redundant w.r.t. atomic intervals in $[\ell(s_1), \ell(c_1)]$ and $[\ell(s_4), \ell(c_4)]$ and *non-redundant* with respect to others.

We are now ready to bound the cost of RecursiveCancel. Indeed, we first show in Lemma 26 that the semantic meaning of redundant or non-redundant forward arcs with respect to atomic intervals is preserved throughout the algorithm. That is, for any atomic interval I, it will hold that there are exactly $\min(n_t^f(I), n_t^b(I))$ many forward arcs that are redundant with respect to I, since these can be avoided in an optimal matching.

▶ **Lemma 26.** *At any time t and for any atomic interval I, there are exactly* $\min(n_t^f(I), n_t^b(I))$ *forward arcs crossing I that are redundant w.r.t I.*

Proof. We prove the following two claims inductively on the number of client-server pairs added:

1. In any atomic interval $I \in \mathcal{AI}_t$, the number of forward arcs that are redundant with respect to I is equal to the minimum of the number of forward arcs and the number of backward arcs crossing I.
2. In any atomic interval I, if there exist two forward arcs $a_1 = (c_1, s_1)$ and $a_2 = (c_2, s_2)$ crossing the interval I such that a_1 is redundant w.r.t. I, and a_2 is non-redundant w.r.t. I, then $\ell(s_2) \geq \ell(s_1)$.

Let $a = (c, s)$ be the client-server pair given by Permutation. Consider the two cases, adding a forward arc and a backward arc:

1. Suppose that s is to the right of c i.e. the case when we add the forward arc directly. If in an atomic interval I between c and s, there are fewer forward arcs than backward arcs before adding c and s, then a is redundant w.r.t. I. Observe that this ensures that in such an interval I, the number of forward arcs that are redundant w.r.t. I increases, and is still equal to the minimum of the number of forward and the number of backward arcs crossing the interval. In intervals I where the number of forward arcs crossing I is at least the number of backward arcs crossing I before adding c and s, a is non-redundant w.r.t. I. In this case, the minimum of forward and backward arcs does not increase, and thus the claim continues to remain valid.
   For the second claim: If a is redundant w.r.t. an atomic interval I, then using claim 1 on the instance before adding a, we can infer that all the forward arcs crossing I are redundant w.r.t I. Thus, claim 2 is void in this case. If a is non-redundant w.r.t. an atomic interval I, we need to show that the new server is to the right of any server whose forward arc is redundant w.r.t. I. This follows directly from the fact that an unmatched server cannot be present in the middle of an arc (Lemma 14), and hence, s is to the right of the server of any forward arc that intersects I.
2. Suppose that s is to the left of c i.e. the case when we recursively add backward arc(s). In this case, in an atomic interval $I \in [\ell(s), \ell(c)]$, either a backward arc is added if there is no forward arc crossing I, or if there is at least one forward arc crossing I, the number of

forward arcs crossing I reduces by one. The intervals outside $[\ell(s), \ell(c)]$ are not affected. From Lemma 23, we know that the forward arcs corresponding to a server only shorten. Thus, the second claim trivially follows.

The algorithm deletes a forward arc from the atomic interval $I \in [\ell(s), \ell(c)]$ if there exists at least one forward arc crossing I before adding the new client c. If the number of forward arcs crossing I is at most the number of backward arcs crossing I, then all the forward arcs crossing I are marked redundant w.r.t. I, and we delete one such arc interval. The property of claim 1 still holds. Similarly, the property holds if there are no backward arcs are crossing I in which case, all the forward arcs crossing I are marked non-redundant. Thus, it remains to show that if there are both forward arcs that are labeled non-redundant and also ones that are marked redundant w.r.t. I cross I, our algorithm makes sure after the changes, one of the forward arc that is labeled non-redundant no longer crosses I. We use claim 2 here. If there are forward arcs a_1 and a_2 cross I and a_1 is redundant w.r.t I while a_1 is non-redundant w.r.t I, note that the server of a_2 is to the right of the server of a_1.

Recall that for any atomic interval I, there is at most one forward arc that crosses I is deleted. If a forward arc a that is redundant w.r.t. I is deleted, when the arc is selected in Line 9 of the algorithm, it has the farthest server among all arcs that intersect the backward arc. This combined with the above fact implies that if a forward arc that is redundant w.r.t. I is deleted, then there is no forward arc that is marked non-redundant crosses I. Thus, if there are forward arcs that are labeled non-redundant crossing I, our algorithm deletes one of them, which completes the proof of claim 1 in the case when we add a backward arc.

The both cases together complete the proof of the two claims, and in particular, of the original lemma. ◀

In order to analyze the cost of the algorithm, we define the *redundant cost* of a forward arc $a = (c, s)$ at time t as the sum of the lengths of all the atomic intervals $I \in \mathcal{AI}_t$ such that a is redundant w.r.t. I. Similarly, the *non-redundant cost* of a forward arc $a = (c, s)$ is the sum of the lengths of all the atomic intervals $I \in \mathcal{AI}_t$ such that a is non-redundant w.r.t. I. In Lemma 27, we bound the total redundant cost of all the forward arcs in terms of their non-redundant cost.

▶ **Lemma 27.** *At any time t, the redundant cost of any forward arc is at most the non-redundant cost of that forward arc.*

Proof. We will prove that for any forward arc a, at any time t, the redundant cost of a is at most the non-redundant cost of a at time t. Summing over all forward arcs will then complete the proof of Lemma 27.

To show this, we will in fact show the following: if RECURSIVECANCEL adds a forward arc $a = (c_t, s_t)$ in step 5 at time t, then in *any suffix of* a, the redundant cost is at most the non-redundant cost. This suffices for us since we know from Lemma 23 that re-matched forward arcs are only suffixes of existing forward arcs, and their redundant/non-redundant labels remain the same from Definition 25.

Henceforth, we assume that at time t, the client c_t has arrived, and PERMUTATION has chosen to match it using the server s_t by a forward arc, and RECURSIVECANCEL has done the same. Let $A = C_{t-1} \cup S_{t-1}$ denote the set of clients and servers prior to adding c_t and s_t. Let $x \in [\ell(c_t), \ell(s_t)]$. Consider the suffix $[x, \ell(s_t)]$ of the arc (c_t, s_t), and introduce a *virtual client* c' at x. We claim that the optimal cost of matching clients and servers in $A \cup \{c', s_t\}$ is at least the optimal cost of matching clients and servers in A. Indeed, if this is not the case, then we can produce a cheaper matching for the clients in C_{t-1} using a subset of servers in $S_{t-1} \cup \{s_t\}$ by ignoring the client c', which is a contradiction to Proposition 9 that PERMUTATION maintains a solution that is server-optimal.

When we add $\{c', s_t\}$ to A, by Lemma 19, the increase in the cost of optimal matching occurs precisely at the intervals where the number of clients to the left is greater than the number of servers (after adding c' and s_t). And in other intervals in $[x, \ell(s_t)]$, the cost paid by the optimal matching decreases. However, this exactly corresponds to the redundancy and non-redundancy of the forward arc (c_t, s_t) with respect to the intervals inside $[x, \ell(s_t)]$. The intervals where the cost of optimal matching increases are the ones with respect to which the arc is non-redundant, and the intervals where the cost of optimal matching decreases are the ones with respect to which the arc is redundant. ◀

▶ **Theorem 28.** *At any point of time, the cost of the matching of Algorithm 3 (*RECURSIVECANCEL*) is at most* 3 *times the cost of the optimal offline matching* OPT_t*.*

Proof. For the sake of analysis, for every atomic interval $I \in \mathcal{AI}_t$, let us label an arbitrary set of $\min(n_t^f(I), n_t^b(I))$ backward arcs as redundant with respect to I, and the rest as non-redundant with respect to I. Then, from Lemma 26, there will be an equal number of redundant backward arcs and redundant forward arcs with respect to any atomic interval $I \in \mathcal{AI}_t$. Using this, we conclude that, for any atomic interval $I \in \mathcal{AI}_t$, there are exactly $|\mathrm{disc}_t(I)|$ non-redundant arcs (including both forward and backward) w.r.t I. Indeed, if $n_t^f(I) \geq n_t^b(I)$, then there are $n_t^b(I)$ redundant forward (and redundant backward) arcs by Lemma 26, and the remaining $n_t^f(I) - n_t^b(I)$ forward arcs crossing I are non-redundant w.r.t I. But this is precisely the (absolute value of) the $\mathrm{disc}_t(I)$ by Claim 20. A similar argument holds when $n_t^b(I) > n_t^f(I)$.

For ease of notation, let us denote the total non-redundant cost of all the forward arcs (resp. backward) maintained by the algorithm at time t as $\mathsf{cost}(\mathcal{M}_t, NF)$ (resp. $\mathsf{cost}(\mathcal{M}_t, NB)$). Similarly, we denote the total redundant cost of all the forward arcs (resp. backward) as $\mathsf{cost}(\mathcal{M}_t, RF)$ (resp. $\mathsf{cost}(\mathcal{M}_t, RB)$). Now, from Lemma 19, and by noting that $|\mathrm{disc}_t(I)|$ is equal to the number of arcs crossing I which are non-redundant w.r.t. I, we have that

$$\mathsf{cost}(\mathcal{M}_t^*) = \sum_I |I||\mathrm{disc}_t(I)| = \mathsf{cost}(\mathcal{M}_t, NF) + \mathsf{cost}(\mathcal{M}_t, NB)\,.$$

On the other hand, the cost of $\mathcal{M}_t$ maintained by RECURSIVECANCEL is at most

$$\begin{aligned}
\mathsf{cost}(\mathcal{M}_t) &= \mathsf{cost}(M_t, RF) + \mathsf{cost}(M_t, RB) + \mathsf{cost}(M_t, NF) + \mathsf{cost}(M_t, NB)\\
&= 2 \cdot \mathsf{cost}(M_t, RF) + \mathsf{cost}(M_t, NF) + \mathsf{cost}(M_t, NB)\\
&\leq 2 \cdot \mathsf{cost}(M_t, NF) + \mathsf{cost}(M_t, NF) + \mathsf{cost}(M_t, NB)\\
&\leq 3\,(\mathsf{cost}(\mathcal{M}_t, NF) + \mathsf{cost}(\mathcal{M}_t, NB)) \leq 3\mathsf{cost}(\mathcal{M}_t^*) = 3\mathsf{OPT}_t\,.
\end{aligned}$$

The first equality is from the definition of redundant backward arc w.r.t. atomic intervals and the first inequality is due to Lemma 27. ◀

4.3 Algorithm MINIMUMCANCEL

Even though the RECURSIVECANCEL algorithm has a good competitive ratio, there are instances in which the algorithm performs $\Omega(k)$ rematches per client on average. We illustrate one such example in Appendix A. We, therefore, present another algorithm MINIMUMCANCEL which also satisfies Definition 21 and then bound its recourse.

Algorithm 4 Algorithm MinimumCancel.

```
1:  for each client $c_t$ arriving at time $t$ do
2:      let $s_t$ be the server Permutation matches $c_t$ to
3:      if $(c_t, s_t)$ is a forward arc then
4:          $\mathcal{M}_t = \mathcal{M}_{t-1} \cup (c_t, s_t)$
5:      else
6:          let $A_t$ denote all forward arcs $\overrightarrow{c,s} \in \mathcal{M}_{t-1}$ with $\ell(c) \in [\ell(s_t), \ell(c_t))$
7:          let $F_t = \cup_{\overrightarrow{c,s} \in A_t} [\ell(c), \ell(s)]$ denote the union of line segments contained within $A_t$,
    and let $O_t = F_t \cap [\ell(s_t), \ell(c_t)]$ denote the overlapped portion with $\overleftarrow{s_t, c_t}$
8:          let $A_t^*$ denote a minimal subset of $A_t$ whose union covers the overlapped region $O_t$
9:          order the edges in $A_t^*$ as $(c'_1, s'_1), (c'_2, s'_2), \ldots, (c'_m, s'_m)$ such that $\ell(c'_1) \le \ell(c'_2) \ldots \le$
    $\ell(c'_m)$
10:         let $\widetilde{A}_t = \{(c'_2, s'_1), (c'_3, s'_2), \ldots, (c'_m, s'_{m-1})\}$
11:         update $\mathcal{M}_t = \{\mathcal{M}_{t-1} \setminus A_t^*\} \cup \widetilde{A}_t \cup \{(c_t, s'_m)\} \cup \{(c'_1, s_t)\}$
12:     end if
13: end for
```

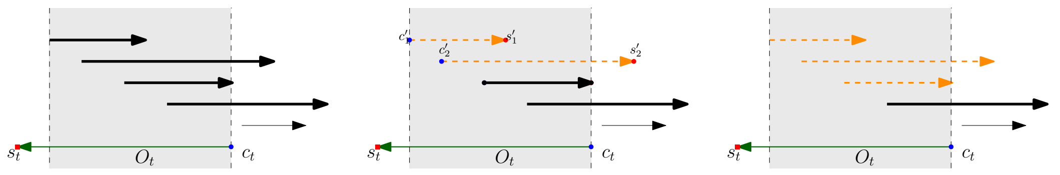
The crux of the algorithm is to cancel parts of forward arcs that overlap with the new backward arc in a manner that *minimally changes the existing solution.* We do this by identifying a minimal arc cover of the overlapping regions and perform the rematch only within the covering set of arcs. Figure 4a gives an illustration of the existing set of arcs when a new backward arc $\overleftarrow{s_t, c_t}$ is added. Figure 4b shows the minimal cover chosen in step 8 of the algorithm, and the final rematch will contain the arcs $\overleftarrow{s_t, c'_1}$, $\overrightarrow{c'_2, s'_1}$ and $\overrightarrow{c_t, s'_2}$. Figure 4c shows an example of a non-minimal cover where there are three overlapping forward arcs.

▶ **Lemma 29.** *When a new client c_t arrives, the re-matched set of forward arcs $\widetilde{A}_t$ computed in step 10 of Algorithm 4 are mutually disjoint.*

Proof. We first claim that in the minimum cover A_t^*, no three forward arcs intersect. Suppose for contradiction that there are forward arcs $a_1 = \overrightarrow{c_1, s_1}, a_2 = \overrightarrow{c_2, s_2}, a_3 = \overrightarrow{c_3, s_3}$ intersect at a point. Without loss of generality, let a_1 be the arc with left most client and a_3 be the arc with right most server. Then, the union of $[\ell(c_1, \ell(s_1))]$ and $[\ell(c_3), \ell(s_3)]$ fully contains $[\ell(c_2), \ell(s_2)]$, thus making the arc a_2 redundant in the cover, contradicting the minimality of the cover A_t^*. It is easy to conclude that the arcs $(c'_2, s'_1), (c'_3, s'_2), \ldots, (c'_m, s'_{m-1})$ are mutually disjoint. The minimal cover property is crucial as illutrated in Figure 4c; the disjointness property of residual forward arcs does not hold for the non-minimal cover. ◀

Using Lemma 29, we can bound the recourse of the MinimumCancel algorithm.

▶ **Theorem 30.** *After the arrival of k clients, the total recourse of* MinimumCancel *algorithm is at most $\mathcal{O}(k \log(k))$.*



(a) A backward arc and overlapping forward arcs. **(b)** A minimal cover of O_t (in dashed orange). **(c)** A non-minimal cover of O_t (in dashed orange).

Figure 4 Illustration for Algorithm 4. Forward arcs in A_t are shown with thick width.

Proof. Note that once a client is matched by a backward arc, it is not going to get re-matched later. We now bound the number forward-arc to forward-arc rematches.

For a vertex (either client or server) z, let us define a "length" $\text{len}(z)$ parameter which is equal to the number of vertices (all vertices which are part of the eventual matching after all the k clients have arrived) lying strictly inside the arc defined by z and its currently matched server/client. Therefore, before the start of the algorithm, the len value of every vertex is at most $2k$. We now define the level of vertex z as $\lfloor \log \text{len}(z) \rfloor$ so that the total initial level of all the vertices is at most $2k(1 + \log k)$. Suppose that a client c is currently matched to s by using a forward arc, and in an iteration gets re-matched to s' and s gets re-matched to c' both again using forward arcs. At least one of $\text{len}(c)$ or $\text{len}(s)$ should have decreased by at least a factor of 2 since these are now disjoint arcs from Lemma 29. And therefore the total level of c and s at least decreases by 1. In other words, on every re-match, the total level decreases by at least 1, which together with the bound on the total initial level gives the number of such re-matches to be at most $2k(1 + \log k)$. If at least one of c or s gets re-matched by a backward arc, its match does not change from then on. Thus, the number of these type of re-matches are at most $2k$. Thus, in total, the recourse of the algorithm is $\mathcal{O}(k \log k)$. ◀

5 Conclusion and Open Questions

The current work (together with the concurrent work [21]) represents the first attempt at exploring the trade-off between recourse and competitive ratio in online metric matching, and understanding the optimal trade-offs is an interesting direction to pursue. Concretely, can we get $o(\log k)$-competitive (even randomized) algorithms with $\text{polylog}(k)$-recourse for general metrics? This would be interesting given the $\Omega(\log k)$-lower bounds for algorithms without recourse. Similarly, obtaining or refuting $O(1)$-competitive algorithms with $O(1)$-recourse on line metrics is a very interesting question. Finally, extending our results to specialized inputs such as random order arrivals or unknown *i.i.d.* models would be interesting, as it better captures beyond-worst-case scenarios.

References

1 Antonios Antoniadis, Carsten Fischer, and Andreas Tönnis. A collection of lower bounds for online matching on the line. In *Latin American Symposium on Theoretical Informatics*, pages 52–65. Springer, 2018.

2 Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

3 Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. In *International Workshop on Approximation and Online Algorithms*, pages 21–35. Springer, 2018.

4 Nikhil Bansal, Niv Buchbinder, Anupam Gupta, and Joseph Seffi Naor. An $O(\log^2 k)$-competitive algorithm for metric bipartite matching. In *European Symposium on Algorithms*, pages 522–533. Springer, 2007.

5 Marcin Bienkowski, Artur Kraska, and Paweł Schmidt. A match in time saves nine: Deterministic online matching with delays. In *International Workshop on Approximation and Online Algorithms*, pages 132–146. Springer, 2017.

6 Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. New algorithms, better bounds, and a novel model for online stochastic matching. In *24th Annual European Symposium on Algorithms (ESA 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

7 Nikhil R Devanur, Balasubramanian Sivan, and Yossi Azar. Asymptotically optimal algorithm for stochastic adwords. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 388–404. ACM, 2012.

8 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 333–344. ACM, 2016.

9 Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 51:1–51:24, 2018.

10 Bernhard Fuchs, Winfried Hochstättler, and Walter Kern. Online matching on a line. *Theoretical Computer Science*, 332(1-3):251–264, 2005.

11 Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 982–991. Society for Industrial and Applied Mathematics, 2008.

12 Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 67:1–67:14, 2019.

13 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 537–550, 2017.

14 Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 468–479, 2014.

15 Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *International Colloquium on Automata, Languages, and Programming*, pages 424–435. Springer, 2012.

16 Varun Gupta, Ravishankar Krishnaswamy, and Sai Sandeep. PERMUTATION strikes back: The power of recourse in online metric matching. *arXiv preprint*, 2019. `arXiv:1911.12778`.

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

18 Samir Khuller, Stephen G Mitchell, and Vijay V Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

19 Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *International Workshop on Approximation and Online Algorithms*, pages 179–191. Springer, 2003.

20 Jannik Matuschke, Ulrike Schmidt-Kraepelin, and José Verschae. Maintaining perfect matchings at low cost. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 82:1–82:14, 2019.

21 Nicole Megow and Lukas Nölke. Online minimum cost matching on the line with recourse. *arXiv preprint*, 2020. `arXiv:2001.03107`.

22 Adam Meyerson, Akash Nanavati, and Laura Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 954–959. Society for Industrial and Applied Mathematics, 2006.

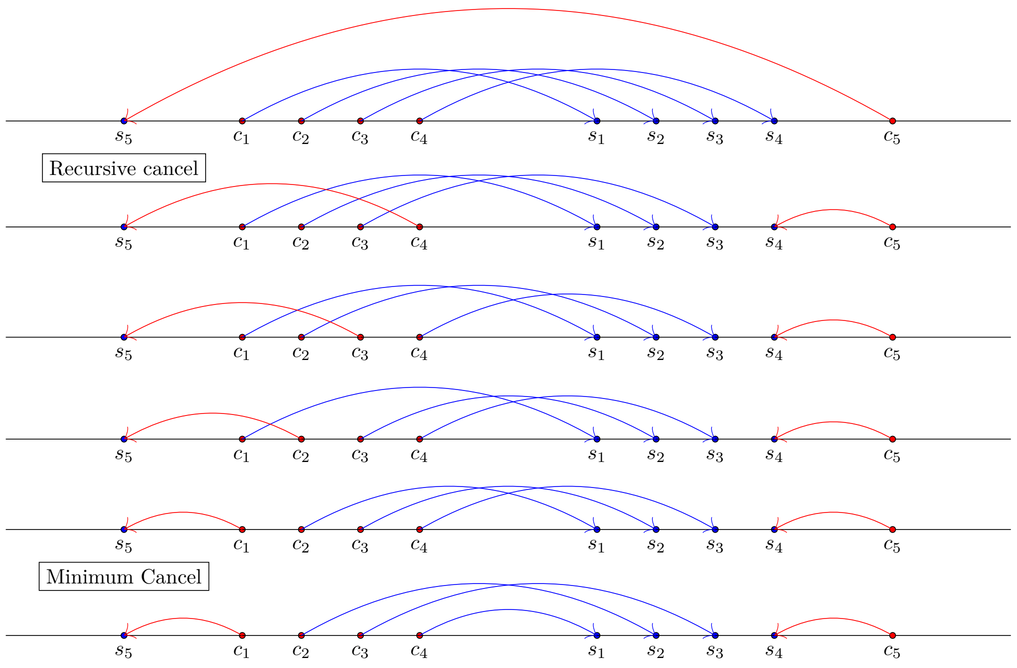
23 Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 505–515. IEEE, 2017.

24 Sharath Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

25 Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, pages 67:1–67:14, 2018.

A Bad Example for Recourse of RecursiveCancel

We illustrate the fact that RecursiveCancel algorithm can have bad recourse in the following example:



In this instance, there are four clients c_1, c_2, c_3 and c_4, $\ell(c_1) < \ell(c_2) < \ell(c_3) < \ell(c_4)$ currently matched using forward arcs to s_1, s_2, s_3 and s_4 respectively such that $\ell(c_4) < \ell(s_1) < \ell(s_2) < \ell(s_3) < \ell(s_4)$. A new client c_5 arrives to the right of s_4 and Permutation outputs s_5 to the left of c_1 as the new server. Since the arc (c_5, s_5) is backward, the algorithms try to fix the matching. The RecursiveCancel i.e. Algorithm 3 changes the matching completely to obtain $(c_2, s_1), (c_3, s_2), (c_4, s_3)$ as the new forward arcs, where as Algorithm 4 changes only c_4's matching and keeps c_2 and c_3 intact. If there are k such forward arcs, and if k backward arcs arrive, Algorithm 3 has a recourse of $\Omega(k^2)$ where as Algorithm 4 has only $O(k)$ recourse.

B Missing Proofs

Proof of Proposition 12. For the sake of simplicity, we stick with $d = 2$, and the same proof holds for larger d as well. At any time t, we view our algorithm as simulating the Permutation algorithm for a certain batch sequence. Indeed, note, the solution maintained in $\mathcal{M}_t$ is exactly what Permutation maintains when fed $O(\log t)$ batches of consecutive clients corresponding to the different powers-of-two 2^{i-1} (in decreasing order) such that the

i^{th} bit from right in the binary representation of t is 1. Theorem 11 then bounds the cost. The recourse is bounded since any client is involved in a re-matching of size 2^i at most once for all i. ◀

Proof of Proposition 13. For simplicity, we prove the proposition for the case $d = 2$. An alternate view of Algorithm MultiScalePermutation is the following: Let the leaves of a complete balanced tree of degree d denote the k client arrivals. Whenever the arrival of a client completes a subtree (that is, it is the rightmost leaf in some subtree), the matching for the clients and their currently matched serves is re-solved optimally.

Our lower bound instance will be on the line metric and consist of two parts: a *core* instance and a *auxiliary* instance. The subsequent client arrival will be chosen as the next arrival from either the core or the auxiliary instance so as to obtain a large recourse cost as we describe soon. The servers for the core instance will at locations $\pm 1, \pm 2, \pm 3, \ldots, \pm k/2$, and the servers for the auxiliary instance will be k servers at location $10k$. The client arrival sequence in the core instance will be $\epsilon, -1-\epsilon, 1+\epsilon, -2-\epsilon, 2+\epsilon, -3-\epsilon, 3+\epsilon, \ldots$; the client arrival sequence for the auxiliary instance is $10k, 10k, 10k, \ldots$.

Note that the above instance have been set up so that on the arrival of a client from the core instance, the server added by Permutation to the matching is also from the core instance, and similarly for a client from the auxiliary instance a server from the auxiliary instance is added. Further, the same is done by OPTso that it suffices to study the cost and recourse for the arrivals in the core instance.

To decide whether the next client arrival happens from the core or the auxiliary sequence, we first check whether the arrival completes any subtree. If it does, denote the largest subtree it completes by T, and by $T_1, \ldots, T_d$ the d subtrees of the root of T (so that the new arrival is the rightmost leaf of T_d). If the number of core arrivals so far in T_d is even, then the new arrival is also chosen from the core sequence. Otherwise the new arrival is chosen from the auxiliary chosen.

We first prove the recourse bound. The sequence in which Permutation adds servers when the clients arrive from the core sequence is $1, -1, 2, -2, \ldots$. In particular, the new client and server are added on the opposite sides of a central matching that is built online. The construction of the client arrival sequence ensures that when MultiScalePermutation resolves the optimal matching for subtree $T = (T_1, T_2)$ the number of client arrivals in T_2 is odd, and hence batch resolving ends up rematching all clients in $T_1 \cup T_2$ (except at most one). (In the general d case we have to assume d is odd, in which case it is easy to show that all the subtrees $T_1, T_2, \ldots, T_d$ have odd number of core clients, and thus a $\frac{d-1}{d}$ fraction of clients are rematched in the subtree T.)

To study cost, consider the matching immediately after the arrival of the ith client, and let $i = \sum_{j=0}^{\ell} d^j k_j$ $(0 \le k_j \le d-1)$ denote the base d representation of i. In particular, consider the case $k_j = 1$ for $1 \le j \le \ell$. The matching consists of one batch of d^ℓ clients each, followed by 1 batch of $d^{\ell-1}$ clients and so forth. The cost of OPTis at most i. However, the cost of MultiScalePermutation is $\Omega(i \cdot \ell) = \Omega(i \log_d i)$. ◀

Proof of Lemma 14. Recall that Permutation maintains an offline optimal matching $\mathcal{M}_t^*$ at time t, and when a client c_t arrives, we pair it with the server that is present in $\mathcal{M}_t^* \setminus \mathcal{M}_{t-1}^*$. In fact, the symmetric difference of $\mathcal{M}_t^*$ and $\mathcal{M}_{t-1}^*$ is an augmenting path starting at c_t and ending at s_t. Let it be denoted by $P = c_t, s_{p_1}, c_{p_1}, \ldots, s_{p_m} = s_t$. The edges $(c_t, s_{p_1}), (c_{p_1}, s_{p_2}), \ldots, (c_{p_{m-1}}, s_{p_m})$ are the new edges, and the rest $(s_{p_1}, c_{p_1}), (s_{p_2}, c_{p_2}), \ldots,$ $(s_{p_{m-1}}, c_{p_{m-1}})$ are the old edges. Recall that the cost of $\mathcal{M}_t^*$ is at least that of $\mathcal{M}_{t-1}^*$, and thus, in the augmenting path, the cost of new edges is at least that of the old edges.

We claim a stronger property that in any suffix of the augmenting path, the cost of the new edges is at least that of old edges. Consider a suffix $s_{p_i}, c_{p_i}, \ldots, s_{p_m}$. If the cost of new edges is less than the old edges, we can change the old matching from $(s_{p_i}, c_{p_i}), \ldots, (s_{p_{m-1}}, c_{p_{m-1}})$ to $(c_{p_i}, s_{p_{i+1}}), \ldots, (c_{p_{m-1}}, s_{p_m})$ while keeping the rest of the edges intact to get a matching with cost less than $\mathcal{M}^*_{t-1}$, contradicting the fact that $\mathcal{M}^*_{t-1}$ is an optimal matching for the first $t-1$ clients. Now, suppose that there is a free server s' in between c_t and s_t. Consider the prefix of the augmenting path P starting at c_t and ending at s'. Let this prefix be denoted by P'. Let $\mathcal{M}'_t$ be the matching obtained from $\mathcal{M}^*_{t-1}$ by augmenting with the new augmenting path P'. Since the cost of new edges is at least that of old edges in any suffix of the original augmenting path, the difference between new edges and old edges in P' is at most that of the original augmenting path P. Thus, the cost of the matching $\mathcal{M}'_t$ is at most that of $\mathcal{M}^*_t$. Furthermore, as we have assumed that the location of all the clients and servers are distinct, the cost of $\mathcal{M}'_t$ is strictly smaller than $\mathcal{M}^*_t$, contradicting the fact that $\mathcal{M}^*_t$ is an optimal matching of first t clients. This proves the claim that when we execute PERMUTATION on the line metric, there are no free servers inside any arc. ◀

C Lower bounds

In this section, we present our lower bound for OMM on general metrics.

▶ **Theorem 31.** *Suppose that there exists an algorithm for* OMM *such that for every client c, the number of servers s such that c is matched to s at some point of execution of the algorithm is at most C, for an absolute constant C. Then, the competitive ratio of the algorithm is at least $\Omega(\log(n))$.*

Proof. We first describe the hard instance for OMM that we use to prove the lower bound. The underlying metric space is the star metric i.e. there exists a node v_0 that is at the center of the star, and a set of nodes $v_1, v_2, \ldots, v_n$ such that $d(v_0, v_i) = 1$ for all $i \in [n]$, and $d(v_i, v_j) = 2$ for all $i, j \in [n], i \neq j$. For every $i \in [n]$, there is a server s_i at v_i. For each time $t = 0, 1, \ldots, n-1$ a single client c_t arrives at a point in the metric space. First, at $t = 0$, the client c_0 arrives at v_0. The next clients arrive at the location of the server just used by the algorithm. Suppose that the algorithm matches c_0 to s_i. Then, c_1 arrives at v_i. After t clients have arrived and have been matched by the algorithm, consider the server matched to c_0- let it be s_{i_1}. Let s_{i_2} be the server matched by the algorithm to the client at v_{i_1}, and so on till there is no client yet arrived at v_{i_k}. Then, in our instance, at time t, a new client arrives at v_{i_k}.

Note that all the clients arrive at different locations in the metric space. This implies that at any point of time t, the offline optimal algorithm cost is equal to 1. We can simply match each client c_i other than c_0 to the server s_i, and match c_0 to an arbitrary unused server.

Suppose that for each client c, the number of servers s such that (c, s) is part of the matching of the algorithm at some point, is at most C. Then, we claim that there is a time t such that the online algorithm has cost at least $\Omega(\log(n))$ at time t. Let M_t, $t = 0, 1, \ldots, n-1$ denote the matching maintained by the algorithm after time t. We consider a new algorithm that maintains a set of matchings M'_t, $t = 0, 1, \ldots, n-1$ after time t. For every time t, we obtain M'_t from M_t as follows: Let $M = M_t$. While there exists a client c located at v_i matched in M to a server s at $v_j \neq v_i$, but the server s_i is not used in M, we rematch c to s_i in M. Note that this process terminates in at most n steps. When this process can no longer proceed, we output $M'_t = M$. The cost of the matching M'_t is at most the cost of M_t, as every iteration of the above procedure only decreases the cost of the matching. For every client c, the number of servers s such that c is matched to s in some M'_t, is at most $C+1$. Finally, the new algorithm that maintains the matchings M'_t has a key property that at any time t: the matching M'_t can be described as a path: $(c_0, s_{i_1}), (c_1, s_{i_2}), \ldots, (c_t, s_{i_{t+1}})$ such that c_j and s_{i_j} are at the same location.

We now claim that there exist some time t such that the size of M'_t is at least $\Omega(\log(n))$, which proves the required lower bound. We define a directed graph $G = (V, E)$. The vertex set of the graph V is equal to $\{0, 1, \ldots, n\}$. There is an edge from i to j if for some time t, the client c_i is matched to the server s_j in M'_t. The out-degree of every node is at most $C+1$ in G. We also define the graphs $G_0, G_1, \ldots, G_{n-1}$ as follows: The vertex set of G_k is the same as G for every k. There is an edge from i to j in G_k if client c_i is matched to s_j in M'_k. It follows from the definitions that for every $k \in \{0, 1, \ldots, n-1\}$, G_i is a subgraph of G.

Note that for each k, the graph G_k is a path that starts at 0 and ends at the index of the location of the client c_k. Thus, all the graphs $G_0, G_1, \ldots, G_{n-1}$ are different path subgraphs of G all of which start at vertex 0 and end at a different vertex in G. As the out-degree of every vertex is at most $C+1$ in G, the number of distinct paths of length at most l in G starting at 0 is at most $(C+1)^l$. Thus, there should exist at least one path whose length is $\frac{\log(n)}{\log(C+1)} = \Omega(\log(n))$. As the length of the subgraph G_i denotes the cost of the matching M'_i, we get the required lower bound on the competitive ratio. ◀