Online Minimum Cost Matching with Recourse on the Line

Nicole Megow
Department for Mathematics and Computer Science, University of Bremen, Germany
nicole.megow@uni-bremen.de

Lukas Nölke
Department for Mathematics and Computer Science, University of Bremen, Germany
noelke@uni-bremen.de

Abstract

In online minimum cost matching on the line, n requests appear one by one and have to be matched immediately and irrevocably to a given set of servers, all on the real line. The goal is to minimize the sum of distances from the requests to their respective servers. Despite all research efforts, it remains an intriguing open question whether there exists an $O(1)$-competitive algorithm. The best known online algorithm by Raghvendra [29] achieves a competitive factor of $\Theta(\log n)$. This result matches a lower bound of $\Omega(\log n)$ [3] that holds for a quite large class of online algorithms, including all deterministic algorithms in the literature.

In this work, we approach the problem in a recourse model where we allow to revoke online decisions to some extent, i.e., we allow to reassign previously matched edges. We show an $O(1)$-competitive algorithm for online matching on the line with amortized recourse of $O(\log n)$. This is the first non-trivial result for min-cost bipartite matching with recourse. For so-called alternating instances, with no more than one request between two servers, we obtain a near-optimal result. We give a $(1+\varepsilon)$-competitive algorithm that reassigns any request at most $O(\varepsilon^{-1.001})$ times. This special case is interesting as the aforementioned quite general lower bound $\Omega(\log n)$ holds for such instances.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Theory of computation → Online algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases min-cost matching in bipartite graphs, recourse, competitive analysis, online

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2020.37

Category APPROX

Related Version `https://arxiv.org/abs/2001.03107`

Funding *Nicole Megow*: Partially supported by the German Science Foundation (DFG) under contract ME 3825/1.

1 Introduction

Matching problems are among the most fundamental problems in combinatorial optimization with great importance in theory and applications. In the bipartite matching problem, we are given a complete bipartite graph $G = (R \cup S, E)$ with positive edge cost c_e for $e \in E$. Elements of R and S are called *requests* and *servers*, respectively, with $n := |R| \leq |S|$. A *matching* $M \subseteq E$ is a set of pairwise non-incident edges. A matching is called *complete* if every request in R is matched to a server in S, i.e. if it is incident to exactly one edge of M. The task is to compute a complete matching of minimum cost, where the *cost* of a matching M is $c(M) := \sum_{e\in M} c_e$. When all information is given in advance, the optimum can be computed efficiently, e.g., by using the Hungarian Method [22].

In the *online setting*, however, the set of requests is not known a priori. Requests arrive online, one by one, and have to be matched immediately and irrevocably to a unmatched server. As we cannot hope to find an optimal matching under these restrictions, we use standard competitive analysis to evaluate the performance of algorithms. An online algorithm is α-*competitive* if it computes for any instance a matching M with $c(M) \le \alpha \cdot \mathrm{OPT}$, where OPT is the cost of an optimal matching. For arbitrary edge costs, the competitive ratio of any online algorithm is unbounded [19,21]. For metric costs, there is a deterministic $(2n-1)$-competitive algorithm and this is optimal for deterministic online algorithms [19, 21]. A remarkable recent result by Nayyar and Raghvendra [27] is a fine-grained analysis of an online algorithm based on *t-net cost* [28] showing a competitive ratio of $O(\mu(G)\log^2 n)$, where $\mu(G)$ is the maximum ratio of minimum TSP tour and weighted diameter of a subset of G.

So far, the online matching problem has resisted all attempts for achieving an $O(1)$-competitive algorithm even for special metric spaces such as the line. In *online matching on the line* the edge costs are induced by a line metric; that is, we identify each vertex of G with a point on the real line and the cost of an edge between a request and a server equals their distance on the line. The competitive ratio of the aforementioned algorithm is then $O(\log^2 n)$, as $\mu(G) = 2$. This has been improved to $\Theta(\log n)$ [29], which is best possible for a large class of algorithms [2]. It remains a major open question whether there exists an $O(1)$-competitive online algorithm (deterministic or randomized) on the line.

In this paper, we consider online matching on the line with *recourse*. In the recourse model, we allow to change a small number of past decisions. Specifically, at any point, we may delete a set of edges $\{(r_i, s_i)\}_i$ of the current matching and rematch the requests r_i to different (free) servers. Online optimization with recourse is an alternative model to standard online optimization which has received increasing popularity recently; see, e.g., [1, 4, 6, 13, 18, 25]. Obviously, if the recourse is not limited then one can just simulate an optimal offline algorithm, and the online nature of the problem disappears. We say an algorithm requires *amortized recourse budget* β if it rematches requests at most βn times in total. The challenging question for online matching on the line is whether it is possible to maintain an $O(1)$-competitive solution with bounded recourse, i.e., with sublinear recourse budget.

Our results. We answer this question to the affirmative and give non-trivial results for the min-cost online matching problem with recourse. We show that with limited recourse, one can indeed maintain a constant competitive solution on the line.

▶ **Theorem 1.** *The online bipartite matching problem on the line admits an $O(1)$-competitive algorithm with amortized recourse budget $O(\log n)$.*

Our algorithm builds on the t-net-cost algorithm by Raghvendra [28, 29]; details follow later. It has the nice property that it interpolates between an $O(\log n)$-competitive online solution (without recourse) and an $O(1)$-approximate offline solution (with large recourse).

Furthermore, we investigate a special class of instances, called *alternating* instances, where between any two requests on the line there is at least one server. This class is interesting as the quite strong lower bound of $\Omega(\log n)$ for a large class of algorithms given in [3], that includes all deterministic online algorithms *without recourse* in literature, holds even on such instances. For alternating instances, we present a more direct and near-optimal algorithm with a scalable performance-recourse trade-off.

▶ **Theorem 2.** *For alternating instances of online matching on the line, there exists a $(1+\varepsilon)$-competitive algorithm that reassigns each request $O(\varepsilon^{-1.001})$ times.*

While the algorithm is quite simple, the proof requires a clever charging scheme that exploits the special structure of optimal solutions on alternating instances. We observe that a large number of recourse actions for a specific request involves large edges in the optimal solution elsewhere on the line.

As a byproduct we give a simple analysis of (a variant of) the algorithm in the traditional online setting without recourse. We show that it is $O(\log \Delta)$-competitive for alternating requests on the line, where Δ is the ratio between the largest and shortest request-server distance. This result compares to $\Theta(\log n)$ for the currently best known online algorithm [29].

▶ Remark. Simultaneously and independently of our work, Gupta, Krishnaswamy and Sandeep [17] obtained a similar result for online min-cost matching with recourse on the line. Their algorithm builds on the $O(n)$-competitive Permutation algorithm [19, 21] and adapts it for the recourse setting. On the line this is done by first matching edges according to Permutation and then asymmetrically applying recourse to arcs (r, s) of the current matching that overlap in a certain way. Both, their algorithm and analysis are completely different from ours. They further obtain a more general $O(\log n)$-competitive algorithm with amortized recourse $O(\log n)$ for arbitrary metrics.

Further related work. Extensive literature is devoted to online bipartite matching problems. The maximum matching variant is quite well understood. For the unweighted setting optimal deterministic and randomized algorithms with competitive ratio 2 and $e/(e-1)$ are known [20]. The weighted maximization setting does not admit bounded guarantees in general, but intensive research investigates models with additional assumptions; see, e.g., the survey [26]. The online min-cost matching problem is much less understood. It remains a wide open question whether a constant-competitive algorithm, deterministic or randomized, is possible for online min-cost matching on the line. In fact, the strongest known lower bound is $9+\varepsilon$ [10]. For a quite large class of algorithms, including all deterministic ones in the literature, there is lower bound of $\Omega(\log n)$ [3].

Randomization allows an improvement upon the best possible deterministic competitive ratio of $(2n-1)$ for metric online bipartite matching [19, 21]; there is an $O(\log^2 n)$-competitive randomized algorithm [5]. On the line, no such improvement on the deterministic result by randomization is known; the competitive factor of $O(\log n)$ is the best known result for both, deterministic and randomized, algorithms [16, 29].

Interestingly, when assuming randomization in the order of request arrivals (instead of an adversarial arrival order), the natural Greedy algorithm is n-competitive [11] for general metric spaces. Furthermore, the online *t-net cost* algorithm is in this case $O(\log n)$-competitive [28] here. Very recently, Gupta et al. [14] gave an $O((\log \log \log n)^2)$-competitive algorithm in the model with online known i.i.d. arrivals.

Maintaining an online cardinality-maximal bipartite matching with recourse was studied extensively; see, e.g., [1, 6–8, 12, 30] and references therein. Bernstein et al. [6] showed recently that the 2-competitive greedy algorithm uses amortized $O(n \log^2 n)$ reassignments, leaving a small gap to the lower bound of $\Omega(n \log n)$. In contrast, for the min-cost variant, it remained a challenging question whether recourse can improve upon the competitive ratio. Even on the line, it remained open whether and how recourse can improve the bound of $O(\log n)$ [29].

The following two models address other types of matching with recourse. In a setting motivated by scheduling, several requests can be matched to the same server and the goal is to minimize the maximum number of requests assigned to a server. Gupta et al. [15] achieve an $O(1)$-competitive ratio with amortized $O(n)$ edge reassignments. A quite different two-stage robust model has been proposed recently by Matuschke et al. [24]. In a first stage one must compute a perfect matching on a given graph and in a second stage a batch of $2k$ new nodes appears which must be incorporated into the first-stage solution to maintain a low-cost matching by reassigning only few edges. For matching on the line, they give an algorithm that maintains a 10-approximate matching reassigning $2k$ edges.

Recourse in online optimization has been investigated also for other min-cost problems even though less than for maximization problems. Most notably seems the online minimum Steiner tree problem [13, 18, 23, 25]. Here, one edge reassignment per iteration suffices to maintain an $O(1)$-competitive algorithm [13], whereas the online setting without recourse admits a competitive ratio of $\Omega(\log n)$.

The recourse model has some relation to *dynamic algorithms*. Instead of minimizing the number of past decisions that are changed (recourse), the dynamic model focuses on the running time to implement this change (*update time*). A full body of research exists on maximum (weighted) bipartite matching; we refer to the nice survey in [9]. We are not aware of any results for min-cost matching.

2 Preliminaries

A path P is called *alternating* with respect to a matching M, if every other edge in P is contained in M. An alternating path is called *augmenting* with respect to M if it starts and ends at vertices are not covered by M. A common method for increasing the cardinality of an existing matching M is to augment along an augmenting path P. After augmentation, the resulting matching $\tilde{M}$ is given by the symmetric difference[1] $M \oplus P$. There may be a choice between different augmenting paths; typically, a path of minimum cost (w.r.t. some metric) is selected. Recently, Raghvendra [28] introduced the following metric. For $t > 1$, the *t-net-cost* of a path P w.r.t. a matching M is

$$\phi_t^M(P) := t \cdot c(P \setminus M) - c(P \cap M) \;=\; t \cdot c(P \cap \tilde{M}) - c(P \cap M). \tag{1}$$

Our algorithm maintains three matchings: the *recourse matching* M, the actual output of the algorithm, and two auxiliary matchings based on (online and offline versions of) the t-net-cost algorithm [28], namely, the *offline matching* M^* and the *online matching* M'. While M^* is a near-optimal offline matching that possibly requires a large amount of recourse, M' is an online matching that is $O(\log n)$-competitive [29] but uses no recourse. We describe how M^* and M' are obtained based on the above cost function; see also [27–29]. By speaking of the matching M_i, M_i^* or M_i', we refer to the state of the respective matching after serving the i-th request including possible reassignments.

On arrival of the i-th request r_i, the offline t-net-cost algorithm constructs the offline matching M_i^* by augmenting M_{i-1}^* along an alternating path P_i of minimum t-net-cost w.r.t. M_{i-1}^*. That is, $M_i^* := M_{i-1}^* \oplus P_i$. By definition, this path starts at r_i and ends at a free server, which we denote by s_i. While this procedure may require a large amount of recourse, the resulting matching has been shown to have bounded cost.

▶ **Lemma 3** (Raghvendra [28]). *For any i and $t > 1$, it holds that $c(M_i^*) \leq t \cdot \mathrm{OPT}_i$, where $\mathrm{OPT}_i := c(M_i^{\mathrm{OPT}})$ is denotes the cost of an optimal offline matching M_i^{OPT} of the first i requests.*

For constructing the online matching M_i', augmentation along a path is impossible since using recourse is not permitted. Instead, the online t-net-cost algorithm maintains M^* as an auxiliary matching and constructs M_i' by directly connecting the end points r_i and s_i of the augmenting path P_i. That is, $M_i' := M_{i-1}' \cup \{(r_i, s_i)\}$. In particular, M_i' and M_i^* utilize the same sets of servers.

[1] For two sets X, Y, their symmetric difference is given by $X \oplus Y := (X \cup Y) \setminus (X \cap Y)$. For matchings M_1, M_2, their symmetric difference $M_1 \oplus M_2$ consists of disjoint alternating paths and cycles.

Intuitively, in putting a higher weight on edges that would be added to M^* during augmentation, the parameter t in the t-net cost function discourages the offline t-net-cost algorithm from choosing long paths for augmentation (w.r.t. actual length). This allows for a trade-off between minimizing the cost of the underlying offline matching, and the connection costs in M' (with the latter in a greedy fashion). Looking at the extremal cases, this becomes even more clear. When $t = 1$, the offline t-net-cost algorithm is in fact equivalent to the Hungarian Method [22] which computes an optimal offline solution. The corresponding online matching, however, has a competitive ratio of $\Omega(n)$. In contrast, as t tends to infinity, the algorithm's behavior resembles that of the greedy online algorithm matching a request to the nearest free server. Its competitive ratio can be exponential [19]. Interestingly though, when $t = 3$, the t-net-cost algorithm has a competitive ratio of $O(\log n)$ [29].

3 A Constant-Competitive Algorithm

We start by giving an overview of our algorithm for the recourse model. It exploits the properties of the t-net-cost algorithm by carefully balancing between the offline matching M^* and the online matching M', simultaneously bounding competitive ratio and recourse budget. On a high level, this is done as follows. When a request arrives, we match it as in M' and locally group it with other recent requests into *blocks* that partition the line. Matching requests as in the online matching within a block somewhat increases total cost but requires zero recourse. A structural result, Lemma 8, allows us to bound this increase in cost up to a certain point at which the requests in the block are matched according to M^* causing a local update. During such an update, which we call a *recourse step*, the changes in M^* caused by the arrivals of the requests in the respective block are applied simultaneously, eliminating any redundant recourse actions. Intuitively, blocks can therefore be seen as input buffers for M^* that temporarily use edges from M'. The underlying structure of the blocks guarantees that recourse steps affect only the corresponding portion of the line. To prevent the recourse steps from causing too many reassignments, we additionally incorporate an edge freezing scheme that targets low-cost edges and at the same time keeps the overall cost low.

Given the precise value of OPT and n a priori, one could employ a very simple freezing scheme, which freezes all edges of M' with cost $\frac{\mathrm{OPT}}{n}$ or less. However, in the online model, we do not know OPT or n and, thus, need a dynamic freezing scheme. A typical guess-and-double approach may work concerning the costs. Yet, care has to be taken as $\frac{\mathrm{OPT}_i}{i}$ is not monotone. A major obstacle appears to be the bounding of the recourse budget. Details on our algorithm and dynamic freezing scheme are given in Section 3.

In Section 4, we consider alternating instances. Again, we simulate the offline matching M^* and employ a simple freezing scheme. After a request reaches a certain threshold of reassignments, we freeze this request and the currently associated matching edge. We charge detours that are taken due to frozen edges to large non-frozen edges of M^{OPT}.

3.1 Further Definitions and Notation

Our algorithm classifies requests according to the structure of intervals that describe where on the line the t-net-cost algorithm searches for a free server. Define the *search interval* of a request r_i as the open interval $\bar{\mathrm{I}}_i = (s_i^{\mathrm{L}}, s_i^{\mathrm{R}})$, where s_i^{L} and s_i^{R} are points on the line farthest to the left and right of r_i, respectively, reachable from r_i with t-net-cost $\phi_t^{M^*_{i-1}}(P_i)$. One of $s_i^{\mathrm{L}}, s_i^{\mathrm{R}}$ is the server s_i (which r_i is matched to in M'), see Lemma 6, while the other may not necessarily be a point of $R \cup S$. For the purposes of this definition, we think of it as a (virtual) server. That is, we ask the question "If point p was a server in S, would

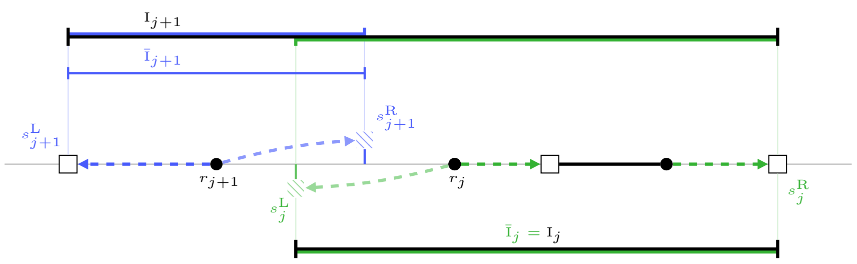

Figure 1 Construction of search intervals and aggregate search intervals of requests r_j and r_{j+1}. The points s^{L}_j and s^{R}_{j+1} (hatched) are points on the line that do not lie in S. We think of them as virtual servers for the purpose of defining $\bar{\mathrm{I}}_j$ and $\bar{\mathrm{I}}_{j+1}$ respectively.

we be able to reach it with t-net-cost $\phi_t^{M^*_{i-1}}(P_i)$?". In other words, $\bar{\mathrm{I}}_i$ is the convex hull of all points on the line, reachable from r_i via an augmenting path of t-net-cost (strictly) less than $\phi_t^{M^*_{i-1}}(P_i)$.

Define the *aggregate search interval* of r_i to be the maximal (open) interval I_i which contains r_i and is a subset of $\bigcup_{j\leq i}\bar{\mathrm{I}}_j$. Intuitively, $\bigcup_{j\leq i}\bar{\mathrm{I}}_j$ consists of the (disjoint) portions of the line, which the t-net-cost algorithm has considered, up to time i, in its search for free servers; the interval I_i is simply the portion containing r_i. See Figure 1 for an illustration. By definition, the portions of the line constituting $\bigcup_{j\leq i}\bar{\mathrm{I}}_j$ grow monotonously (and possibly merge). Thus, the aggregate search intervals inhibit a laminar structure as detailed in the following observation.

▷ Observation 4. Whenever $i < j$, then either $\mathrm{I}_i \cap \mathrm{I}_j = \emptyset$ or $\mathrm{I}_i \subseteq \mathrm{I}_j$.

Another important observation is the fact, that the arrival of a request r_i only affects requests and servers in its search interval $\bar{\mathrm{I}}_i$. This is due to the fact that the augmenting path used by the t-net-cost algorithm is entirely contained in the search interval. Outside of $\bar{\mathrm{I}}_i$, the matchings M^* and M' remain unchanged motivating the following observation.

▷ Observation 5. Altering the arrival order of requests via a permutation π for which $\mathrm{I}_i \subseteq \mathrm{I}_j$ and $i < j$ imply $\pi(i) < \pi(j)$ does not alter the final matching.

We say an aggregate search interval I_i has *level* k, if $(1+\varepsilon)^{k-1} \leq |\mathrm{I}_i| < (1+\varepsilon)^k$ and write $\ell(\mathrm{I}_i) = k$. Throughout, we set $t = 3$ and $\varepsilon = \frac{1}{32t}$. Further, two aggregate search intervals are said to belong to the same *block*, if they intersect with each other and are of same level. If the aggregate search intervals of a block do not intersect those of higher level, then this block is said to be a *top block*. With Observation 5 in mind, we note that the top blocks partition the line into portions that are compatible with the structure of M^*. A typical block-structure is depicted in Figure 2.

Our definition of search intervals was motivated by intuition and practicality (specifically for the proof of Lemma 12 later on). However, it describes intervals different from the search intervals defined in [29]. The following lemma shows that our definition of aggregate search intervals coincides with Raghvendra's definition of *intervals of a cumulative search region*. We may therefore use the corresponding results from [29].

▶ **Lemma 6.** *For a request r_i, the corresponding aggregate search interval I_i and interval of a cumulative search region C_i are equal. Further, for the search interval $\bar{\mathrm{I}}_i = (s^{\mathrm{L}}_i, s^{\mathrm{R}}_i)$, we have $s_i \in \{s^{\mathrm{L}}_i, s^{\mathrm{R}}_i\}$.*

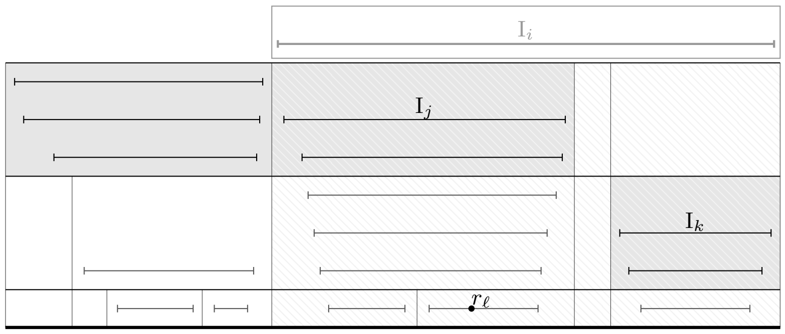

Figure 2 Illustration of a typical block structure. On arrival of r_i, its aggregate search interval I_i creates a top block (active) initiating a recourse step. Requests in the hatched area (now inactive) are reassigned w.r.t. $M^*_{i-1} \oplus M^{\mathrm{F}}_i$. On I_i, locally, $M^*_{i-1} = M^*_h$ for $j, k \leq h < i$.

Proof. To see that the first claim holds, note that the intervals C_i and I_i are constructed the same way. They are built by by taking the union of all known search intervals (for the respective definition) and choosing of the resulting new intervals that which contains the considered request. The definition of search intervals in [29], which, to avoid confusion, we call *dual intervals*, is as follows. The t-net-cost algorithm maintains dual values $y : S \cup R \to \mathbb{R}^+$ satisfying $y_s + y_r = c_{(s,r)}$ if $(s, r) \in M^*$ and $y_s + y_r \leq t \cdot c_{(s,r)}$ otherwise. Additionally, duals of free requests or servers are zero. When a request r_i arrives, a shortest t-net-cost path P_i is found and the duals of all vertices in the search tree (denoted by $A_i \subseteq S$ and $B_i \subseteq R$) are updated before augmentation so that the dual constraints on P_i are tight. This is true for both augmenting paths P^{L}_i and P^{R}_i that are used to reach s^{L}_i and s^{R}_i respectively. A dual interval of a request r_i is defined as $\mathit{interior}(\bigcup_{r \in B_i} \mathit{cspan}(r, i))$, where $\mathit{cspan}(r, i) = [r - \frac{y^i_{\max}(r)}{t}, r + \frac{y^i_{\max}(r)}{t}]$ and $y^i_{\max}(r)$ is the highest dual weight assigned to r until time i.

Raghvendra [28] shows, that the dual constraints on P^{L}_i and P^{R}_i are tight before augmentation. Therefore, we obtain our search intervals I_i by replacing $y^i_{\max}(r)$ with the dual weight of r before augmentation along P_i. Hence, a search interval is contained in the dual interval and therefore $\mathrm{I}_i \subseteq C_i$. At the same time, dual weights of requests are increased when the request is contained in some B_i as detailed above or reduced right after an augmentation. Thus, the maximal value $y^i_{\max}(r)$ is attained right before an augmentation, in which case the request is part of some P^{L}_h or P^{R}_h, implying $C_i \subseteq \mathrm{I}_i$. Therefore, the intervals I_i and C_i coincide and we may use the respective results from [29].

The second claim follows directly from Lemma 6 in [29], which states that there are no free servers in the dual interval. As seen above, search intervals are contained in dual intervals, so this statement is true for our definition as well. And lastly, since clearly $s_i \in [s^{\mathrm{L}}_i, s^{\mathrm{R}}_i]$, it follows that $s_i \in \{s^{\mathrm{L}}_i, s^{\mathrm{R}}_i\}$. ◀

3.2 The Algorithm

Our algorithm uses the above structures to partition the requests (dynamically) into three groups. For each group, we follow a different assignment procedure to construct our recourse matching M. We first give some intuition and then, further below, the precise description.

Group 1: The first group consists of all requests whose aggregate search intervals belong to a top block. We label requests in this group *active* and all other requests *inactive*. In M, we match an active request r_i exactly as in the online matching M'. That is, we have e_i

in M, where $e_i = (r_i, s_i) \in M'$ connects the endpoints of the minimum t-net-cost augmenting path P_i with respect to M^*_{i-1} (Algorithm 1, Step 1). In particular, any arriving request belongs immediately to a top block and is labeled active, by definition of aggregate search intervals. In the course of the algorithm's execution new blocks may appear (on top) rendering previously active requests inactive. We call this a *recourse step.*

Group 2: This group consists of inactive requests r_i, whose corresponding edge $e_i \in M'$ is of negligible size. The precise freezing scheme (Algorithm 1, Steps 2-4) is detailed below. Intuitively, it is not worth to spend recourse on them; it would ruin our recourse budget, see Figure 4. We call such a request or edge *frozen* and denote by $F_j \subseteq M'_j$ the set of frozen edges at time j. These requests, too, are to be matched in M according to the online matching M'. However, for technical reasons, the update on M for a newly frozen request is implemented with a subsequent recourse step. Denote by $M^{\mathrm{F}}_j \subseteq F_j$ the subset of frozen edges that is in $M_j \cap M'$. Their low costs ensures that frozen edges contribute in total at most OPT to the cost of M.

Group 3: The remaining requests (inactive and unfrozen) are in this group. Ideally, we would like to match these as in the offline matching M^*. This may not be possible, as M^* could assign a request r to a server s that is already covered by M^{F}. In such a case, we match r via a detour of low additional cost as follows; we call this *detour matching.*

Detour Matching. Consider an unfrozen request r and the symmetric difference $M^* \oplus M^{\mathrm{F}}$ consisting of alternating paths and cycles. There exists a (unique) alternating path P from r to some server s' not yet matched via M^{F}. To see this, note that the path starts with the edge $(r, s) \in M^*$ which cannot be contained in an alternating cycle as r is not frozen. Additionally, it cannot end at a request, since every request on P is reached via an edge of M^{F}. But since a request on P matched in M^{F} is certainly also matched (via a different edge) in M^*, it can be used to extend P. Thus, P ends at a server s' not matched in M^{F}. In M, we match r directly to this server s' (Alg. 1, Step 7). While the online matching M' does not change but only gets revealed over time, the offline matching M^* can change its structure drastically at any point. As this causes a lot of recourse, the first group is used as an input buffer and the detour matching is updated only periodically and locally, whenever a new top block appears. Specifically, at time i, an inactive request r_j is matched with respect to the detour matching $M^*_{h(i,j)} \oplus M^{\mathrm{F}}_i$, where $h(i, j) = \max\{k \leq i \mid r_j \in \mathrm{I}_k$ and I_k is not in a top block$\}$. That is, $h(i, j)$ is the last time before i when r_j participated in a recourse step. For an example, see Figure 2, where $h(i, \ell) = j$.

When considering active, inactive or frozen objects, or membership in a block, we identify a request r_i, the edge $e_i \in M'$ and the interval I_i. For instance, we say r_i is of level k and belongs to a certain block, when this holds for I_i, or, e_i is active when this is the case for r_i.

We describe Algorithm 1 in a step-by-step manner and detail the precise freezing scheme. On arrival of a request r_i, label it active and match it according to the online matching M'.

Freezing/unfreezing inactive requests. We update the set of frozen edges as follows. If the cost of an inactive edge $e_j \in M'$ falls below $\frac{\mathrm{OPT}_i}{i^2}$, then freeze it and add it to F_i. Note that while we use e_j to determine whether r_j is frozen, r_j may use another edge in M_i.

If for a previously frozen e_j, we have $c_{e_j} > \frac{\mathrm{OPT}_i}{i}$, unfreeze and delete it from F_i and M^{F}_i. Matching it according to M' is now too costly. Reassign it w.r.t. $M^*_{h(i,j)} \oplus M^{\mathrm{F}}_i$ as follows.

Consider an edge $e = (s, r)$ right before it is unfrozen. If e is already matched according to the offline matching, then there is nothing to do. Otherwise, it must be part of an alternating path or cycle in $M^* \oplus M^{\mathrm{F}}$. In the latter case, again, no recourse action needs to be taken as

Algorithm 1 for computing an online bipartite matching with recourse on the line.

```
   On arrival of the $i$-th request $r_i$:                 ▷ $r_i$ matched in $M'$ via $e_i = (r_i, s_i)$
1: match $r_i$ to $s_i$ as in $M'$                                                    ▷ active
   Freezing/Unfreezing
2: determine the set $F_i$ of frozen edges
3: for inactive requests $r_j$ that become unfrozen do
4:     remove $e_j$ from $M_i^{\mathrm{F}}$ and repair assignments on corresponding path of $M^*_{h(i,j)} \oplus M_i^{\mathrm{F}}$
   Recourse step (new top block)
5: if there is no $j < i$ such that $\mathrm{I}_j \subseteq \mathrm{I}_i$ and $\ell(\mathrm{I}_j) = \ell(\mathrm{I}_i)$ then
6:     for $r_j \in \mathrm{I}_i$ recently frozen do add $e_j = (r_j, s_j)$ to $M_i^{\mathrm{F}}$ and reassign $r_j$ to $s_j$
7:     for unfrozen $r_j \in \mathrm{I}_i \setminus \{r_i\}$ do reassign $r_j$ according to $M^*_{i-1} \oplus M_i^{\mathrm{F}}$   ▷ now inactive
```

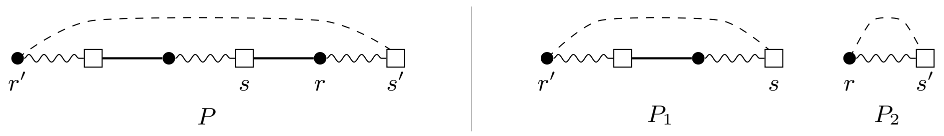

Figure 3 Illustration of Step 4 in Algorithm 1. Edges of M^* are snaked, edges of M^{F} solid and edges of $M \setminus M^{\mathrm{F}}$ dashed. After unfreezing r, the removal of (r, s) splits the path in $M^* \oplus M^{\mathrm{F}}$ in two parts.

the removal of e results in a path from r to s. In the detour matching, we want to connect the ends of this path, which is already accomplished by the edge $e = (s, r)$. Consider the case that $e \in P$ for some alternating path P in $M^* \oplus M^{\mathrm{F}}$ that starts in a request r' and ends in a server s'. Unfreezing e and matching according to $M^* \oplus M^{\mathrm{F}}$ decomposes P into the r'-s-path P_1 and the r-s'-path P_2. In Algorithm 1, we implement these changes via two recourse actions: we reassign r to s' and r' to s; see Figure 3.

Recourse step. If there is no $j < i$ such that $\mathrm{I}_j \subseteq \mathrm{I}_i$ and $\ell(\mathrm{I}_j) = \ell(\mathrm{I}_i)$, then the arrival of r_i produces a new top block and may render a number of previously active requests inactive. This triggers, what we call a *recourse step* involving a number of recourse actions to accommodate the newly inactive requests as follows.

First, assign requests in I_i that were recently frozen according to M' and add the corresponding edge to M_i^{F}. Next, reassign all other requests (non-frozen, inactive) that lie in I_i according to $M^*_{i-1} \oplus M_i^{\mathrm{F}}$. We described this as detour matching above; see Figure 2.

Delaying the reassignment of frozen requests according to the frozen edge until their next recourse step, prevents reassignments when a request repeatedly alternates between frozen and unfrozen. This "delayed reassignment" accounts for the difference between M_i^{F} and F_i.

To see that the freezing scheme is indeed necessary, consider the example in Figure 4 showing an alternating instance with exponentially increasing edge costs. Without freezing low-cost edges, at the arrival of any request r_i with $i > 2$, all requests r_j with $j < i$ are reassigned. Thus, the recourse budget is linear.

Bounding the competitive ratio. The work already put into structuring M enables us to bound its cost rather easily. Recall that $c(M^*) \leq t \cdot \mathrm{OPT}$ due to Lemma 3. Frozen edges satisfy $c(M^{\mathrm{F}}) \leq \mathrm{OPT}$ as, after arrival of the last request, there are at most n frozen edges of cost at most $\frac{\mathrm{OPT}}{n}$ each. By triangle inequality, the contribution of inactive non-frozen requests to the cost of M is at most $c(M^* \oplus M^{\mathrm{F}}) \leq (t+1) \cdot \mathrm{OPT}$.

▷ Observation 7. The cost of inactive edges is at most $(t+2) \cdot \mathrm{OPT}$.

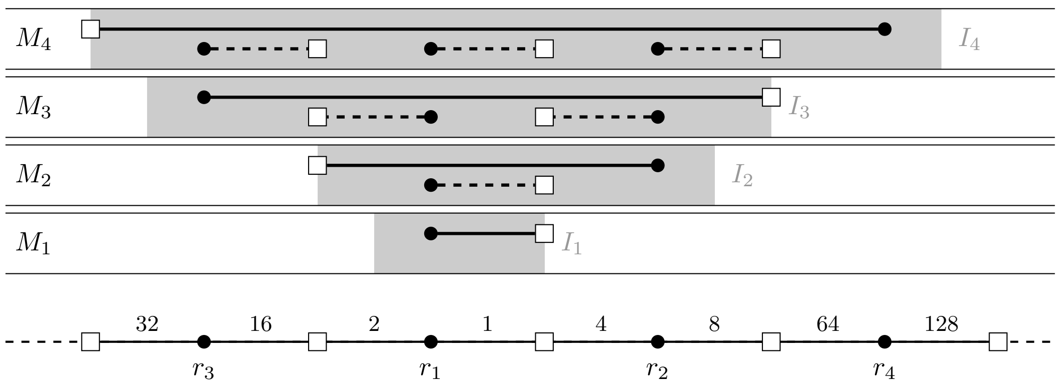

Figure 4 An alternating instance with exponentially increasing connection costs. Edge costs are indicated above the corresponding portion of the line (drawing not to scale). Note that the aggregate search intervals grow exponentially in size. Therefore, a (top) block consists of a single aggregate search interval only and every arrival of a request triggers a recourse step. Active edges are drawn solid, inactive edges dashed.

To bound the cost of active edges, we build on the analysis of Raghvendra [29]. We refine his technical propositions and perform a slightly more fine-grained analysis. Instead of simultaneously bounding the cost of all blocks of the same level, we argue more generally on the cost of any set of disjoint blocks at different levels. In particular, we are interested in bounding the cost contribution of the top blocks.

▶ **Lemma 8.** *For $t = 3$, the cost of all active edges of M is bounded by $O(\mathrm{OPT})$.*

To prove this lemma, we may closely follow the argumentation given in [29] for the online problem without recourse. In fact, there is given essentially the same statement for all edges of blocks of a particular level. We can adapt the approach, crucially using Observation 5, to make the arguments work for (disjoint) blocks of different levels. We leave the detailed proof for the full version.

▶ **Corollary 9.** *Algorithm 1 has a constant competitive ratio.*

Bounding the recourse. Due to delayed reassignments, freezing a request r does not cause any immediate recourse actions. Unfreezing r, on the other hand, may cause reassignments (Step 4), namely to r and possibly one additional request, see Figure 3. We charge these two recourse actions to r, in particular to the last recourse step it participated in. However, recourse due to unfreezing r happens at most once between two consecutive recourse steps of r. This implies, that a request is charged at most three times per recourse step it participates in, once for the recourse step itself and possible two times more for a subsequent unfreezing.

▷ Observation 10. A request is charged at most three times per recourse step it is involved in.

For a request getting frozen at time i and unfrozen at time $j > i$, we have $c_e \leq \frac{\mathrm{OPT}_i}{i^2}$ and $c_e > \frac{\mathrm{OPT}_j}{j}$ for the corresponding $e \in M'$. As $\mathrm{OPT}_i \leq \mathrm{OPT}_j$, this implies the following.

▷ Observation 11. A request frozen at time i stays frozen until time at least i^2.

On the other hand, the number of recourse steps in which a continuously non-frozen request takes part in can be bounded from above.

▶ **Lemma 12.** *If a request $r \in \mathrm{I}_i \subseteq \mathrm{I}_j$ is not frozen from time i to time j, then it holds that $\ell(\mathrm{I}_j) - \ell(\mathrm{I}_i) = O(\log j)$. In particular, between time i and j, there are $O(\log j)$ recourse steps in which r can participate.*

Proof. Consider the search interval $\bar{\mathrm{I}}_j = (s_j^{\mathrm{L}}, s_j^{\mathrm{R}})$. By definition of search intervals, there exist augmenting paths $P_j^{\mathrm{L}}, P_j^{\mathrm{R}}$ connecting r_j to s_j^{L} and s_j^{R} with the same t-net-cost as P_j, the path used by the t-net-cost algorithm. We bound the length of $P_j \in \{P_j^{\mathrm{L}}, P_j^{\mathrm{R}}\}$ by

$$c(P_j) = c(P_j \cap M_{j-1}^*) + c(P_j \cap M_j^*) \le 2t \cdot \mathrm{OPT}_j. \tag{2}$$

Without loss of generality, assume $P_j = P_j^{\mathrm{L}}$. Interpreting the point s_j^{R} as a virtual server and assuming that it is a contained in S, we could augment M_{j-1}^* also along P_j^{R} yielding a different matching $\tilde{M}_j^*$. By definition of P_j^{R} and Equation (1), we have

$$t \cdot c(P_j \cap M_j^*) - c(P_j \cap M_{j-1}^*) = \phi_t^{M_{j-1}^*}(P_j) = \phi_t^{M_{j-1}^*}(P_j^{\mathrm{R}}) = t \cdot c(P_j^{\mathrm{R}} \cap \tilde{M}_j^*) - c(P_j^{\mathrm{R}} \cap M_{j-1}^*).$$

Thus,

$$\begin{aligned} c(P_j^{\mathrm{R}}) &= c(P_j^{\mathrm{R}} \cap M_{j-1}^*) + c(P_j^{\mathrm{R}} \cap \tilde{M}_j^*) \\ &\le c(P_j^{\mathrm{R}} \cap M_{j-1}^*) + c(P_j \cap M_j^*) + \tfrac{1}{t} c(P_j^{\mathrm{R}} \cap M_{j-1}^*) \le 3t \cdot \mathrm{OPT}_j. \end{aligned}$$

From Equation (2), we get $|\bar{\mathrm{I}}_j| \le c(P_j^{\mathrm{L}}) + c(P_j^{\mathrm{R}}) \le 5t \cdot \mathrm{OPT}_j$, which implies $|\mathrm{I}_j| \le \sum_{k \le j} |\bar{\mathrm{I}}_k| \le 5t \cdot j \cdot \mathrm{OPT}_j$. On the other hand, the cost of $|\mathrm{I}_i|$ can be bounded from below by c_e, where e is the edge in M' incident to r, as both ends of e are contained in the interval. We then obtain

$$\frac{|\mathrm{I}_j|}{|\mathrm{I}_i|} \le \frac{5t \cdot j \cdot \mathrm{OPT}_j}{c_e} < 5t \cdot j^3.$$

The last inequality follows from the assumption that r is not frozen at time j, implying that $c_e > \frac{\mathrm{OPT}_j}{j^2}$. Recall that $|\mathrm{I}_i| < (1+\varepsilon)^{\ell(\mathrm{I}_i)}$ and $(1+\varepsilon)^{\ell(\mathrm{I}_j)-1} \le |\mathrm{I}_j|$. We conclude

$$\ell(\mathrm{I}_j) - \ell(\mathrm{I}_i) \le 1 + \log_{(1+\varepsilon)}(5t \cdot j^3) = 1 + \log_{(1+\varepsilon)}(5t) + \frac{\log j}{\log(1+\varepsilon)} \le \; c \cdot \log j\,, \tag{3}$$

for some constant c. Regarding the second claim, recall that r participates in a recourse step when an interval containing r opens a new top block (i.e. a new level) while r is not (freshly) frozen. Between time i and time j, this can only happen for intervals I_h of distinct levels for which $\mathrm{I}_i \subseteq \mathrm{I}_h \subseteq \mathrm{I}_j$. The claim follows. ◀

Lemma 12 and Observation 11 bound the total number of recourse actions taken by Algorithm 1.

▶ **Lemma 13.** *Algorithm 1 uses a recourse budget of at most $O(\log n)$.*

Proof. Consider a request r. By Observation 10, it suffices to bound the number of recourse steps that r is involved in. Let $[i_h^U, i_h^F]$, for $h = 0, 1, \ldots, k$, be maximal intervals of consecutive time points during which r is not frozen, i.e., r is not frozen at any time $i \in [i_h^U, i_h^F]$. We use induction on k to show that r participates in at most $2c \cdot \log(i_k^F)$ recourse steps, where c is the constant from Equation (3). The base case, $k = 0$, follows directly from Lemma 12. For $k \ge 1$, we have $(i_{k-1}^F)^2 \le i_k^U \le i_k^F$ due to Observation 11. By induction hypothesis, the number of reassignments that involve r in the first $k-1$ time intervals is at most

$$2c \cdot \log(i_{k-1}^F) \le 2c \cdot \log\left(\sqrt{i_k^F}\,\right) = c \cdot \log(i_k^F).$$

For the last time interval, we have at most $c \cdot \log(i_k^F)$ many such recourse steps by Lemma 12. Since $i_k^F \le n$, this concludes the proof. ◀

Corollary 9 and Lemma 13 together imply Theorem 1.

4 A Near-Optimal Algorithm on Alternating Instances

For alternating instances, we may assume that requests and servers alternate from $-\infty$ to ∞ on the line, with servers at $\pm\infty$. For such instances, an optimum matching matches all requests either to the server directly to their left or all requests to the server on their right. Denote these matchings by M^{L} and M^{R} respectively and call their edges *minimal*.

We describe a $(1+\varepsilon)$-competitive algorithm for alternating instances that reassigns each request at most a constant number of times. In addition to its output M, it maintain M^* and a set of frozen edges M^{F}. A request is frozen when it is reassigned the k-th time, for some k only depending on ε. The request remains matched to its current server perpetually and the corresponding edge is added to M^{F}. Non-frozen requests are matched according to the detour matching $M^* \oplus M^{\mathrm{F}}$ as described in Section 3. By design, the recourse budget per request is constant, only the competitive analysis remains.

Notation. We use a similar interval structure as before and keep the notation. Consider intervals $\mathrm{I}_i = [s_i^{\mathrm{L}}, s_i^{\mathrm{R}}]$, where $s_i^{\mathrm{L}}, s_i^{\mathrm{R}} \in S$ are the closest free servers on the line to the left and right of r_i respectively at the time of its arrival. Denote by $P_i^{\mathrm{L}}, P_i^{\mathrm{R}}$ the alternating paths connecting r to s_i^{L} and s_i^{R} respectively that have shortest t-net-cost.

Considered as line segments, the augmenting paths P_i have a laminar structure. We view them as nodes of a forest, where P_i is a child of the minimal augmenting path that properly contains it, or a root if no such path exists (see Figure 5). The *depth* of a path P_i denotes its distance to the root and determines the number of reassignments of the corresponding request r_i in M^*.

▶ **Lemma 14.**

(i) *The paths $P_i^{\mathrm{L}}, P_i^{\mathrm{R}}$ and the matching M_i^* only use minimal edges.*

(ii) *If $P_i = P_i^X$, for $X \in \{\mathrm{L}, \mathrm{R}\}$, then in the area of I_i, locally, we have that $M_i^* = M^{\mathrm{X}}$. Specifically, this implies $\phi_t^{M_{i-1}^*}(P_i) = \phi_t^{M^Y}(P_i)$ for $X \neq Y \in \{\mathrm{L}, \mathrm{R}\}$.*

(iii) *If P_j is a child of P_i, then $P_i = P_i^{\mathrm{L}}$ if and only if $P_j = P_j^{\mathrm{R}}$. In particular, we have $\mathrm{I}_j \subseteq P_i$ and $\mathrm{I}_j \cap M_j^* \cap M_i^* = \emptyset$.*

Proof. (i): We use induction on i. The base case, $i = 1$, is easy. Let $i \geq 2$. Without loss of generality $P_i = P_i^{\mathrm{R}}$. Assume P_i contains a non-minimal edge e. Let the notation (s, r) for an edge reflect that s is to the left of r on the line. Edges $(s, r) \in P_i$ are contained in M_{i-1}^* and by induction hypothesis minimal, so $e = (r, s) \in M_i^*$. Consider the server s' right of r. Since e is not minimal, $r < s' < r' < s$, with r' being the server right of s'. If s' was free, altering P_i to go from r directly to s' would yield a lower cost. This follows from the fact that the t-net-cost of a path from a request to a free server is always non-negative, see [28]. If s' is matched, it must be matched to r'. Therefore, replacing e by $(r, s'), (s', r'), (r', s)$ reduces the cost as well, contradicting the minimality of P_i.

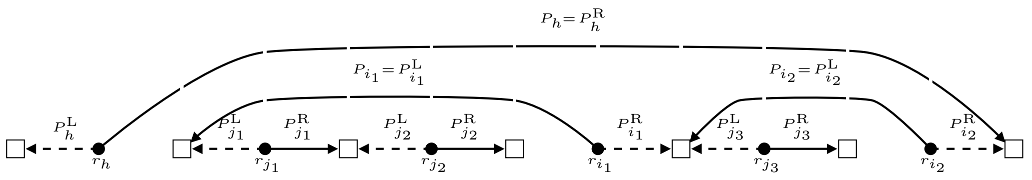

Figure 5 Illustration of a path tree in an alternating instance. Paths not chosen for augmentation are dashed. Servers are depicted as squares and requests as filled circles.

(ii): By statement (i), P_i^{L} and P_i^{R} only consist of minimal edges. Then $P_i^{\mathrm{L}} \cap M_{i-1}^* = M^{\mathrm{R}}$ and $M_{i-1}^* \cap P_i^{\mathrm{R}} = M^{\mathrm{L}}$. After augmenting M_{i-1}^* along P_i, the matching is flipped (locally).

(iii): By (ii), we know that $M_j^* \cap \mathrm{I}_j = M^{\mathrm{X}}$, say $X = R$, so edges are of the form (r, s). From part (i), only augmenting paths $P_h = P_h^{\mathrm{L}}$ can traverse them. If this happens, all of I_j is traversed as there is no free server in its interior. As parent of P_j, P_i is the first path to properly contain P_j and thus $P_i = P_i^{\mathrm{L}}$ and in particular $\mathrm{I}_j \subseteq P_i$. Therefore, the equation $\mathrm{I}_j \cap M_j^* \cap M_i^* = \emptyset$ follows directly from (ii). ◀

The above lemma can be used to show that the sum of lengths of augmenting paths of some depth grows exponentially towards the root. For a path P_h, denote by $|P_h|$ the length of the corresponding line segment. Further, let $\mathcal{H}_k$ be the set of indices of paths at depth k in the induced subtree of P_h with the root P_h at depth 0.

▶ **Lemma 15.** *Consider a path P_h and its grandchildren P_j, for $j \in \mathcal{H}_2$. Then*

$$|P_h| \geq \left(2 - \tfrac{1}{t}\right) \cdot \sum_{j \in \mathcal{H}_2} |P_j|.$$

Proof. Denote by P_i, $i \in \mathcal{I} = \mathcal{H}_1$, children of P_h in the path-forest and by $\mathcal{J}_i \subseteq \mathcal{J} = \mathcal{H}_2$ the sets of indices of their respective children. Without loss of generality, assume $P_h = P_h^{\mathrm{R}}$. Lemma 14, (iii), implies $P_i = P_i^{\mathrm{L}}$, and $P_j = P_j^{\mathrm{R}}$, for $i \in \mathcal{I}, j \in \mathcal{J}$; see Figure 5. With again Lemma 14, (iii), and $\phi_t^{M_{i-1}^*}(P_i^{\mathrm{R}}) \geq \phi_t^{M_{i-1}^*}(P_i^{\mathrm{L}})$, we get

$$\begin{aligned} t \cdot |P_h| &\geq t \cdot \sum_{i \in \mathcal{I}} |I_i| = t \cdot \sum_{i \in \mathcal{I}} \left(|P_i^{\mathrm{L}}| + |P_i^{\mathrm{R}}|\right) \geq \sum_{i \in \mathcal{I}} \left(t \cdot |P_i^{\mathrm{L}}| + \phi_t^{M_{i-1}^*}(P_i^{\mathrm{R}})\right) \\ &\geq t \cdot \sum_{j \in \mathcal{J}} |P_j^{\mathrm{R}}| + t \cdot \sum_{i \in \mathcal{I}} |P_i \setminus (\cup_{j \in \mathcal{J}_i} P_j^{\mathrm{R}})| + \sum_{i \in \mathcal{I}} \phi_t^{M_{i-1}^*}(P_i^{\mathrm{L}}). \end{aligned} \tag{4}$$

Using Lemma 14, we get

$$\begin{aligned} \phi_t^{M_{i-1}^*}(P_i^{\mathrm{L}}) = \phi_t^{M^{\mathrm{R}}}(P_i^{\mathrm{L}}) &= \sum_{j \in \mathcal{J}_i} \left(\phi_t^{M^{\mathrm{R}}}(P_j^{\mathrm{L}}) + \phi_t^{M^{\mathrm{R}}}(P_j^{\mathrm{R}})\right) + \phi_t^{M^{\mathrm{R}}}(P_i \setminus (\cup_{j \in \mathcal{J}_i} \mathrm{I}_j)) \\ &\geq \sum_{j \in \mathcal{J}_i} \left(\phi_t^{M_{j-1}^*}(P_j^{\mathrm{L}}) + \phi_t^{M^{\mathrm{R}}}(P_j^{\mathrm{R}})\right) - t \cdot |P_i \setminus (\cup_{j \in \mathcal{J}_i} \mathrm{I}_j)|. \end{aligned}$$

Since $\phi_t^{M_{j-1}^*}(P_j^{\mathrm{L}}) \geq \phi_t^{M_{j-1}^*}(P_j^{\mathrm{R}}) = \phi_t^{M^{\mathrm{L}}}(P_j^{\mathrm{L}})$, the above together with Equation (4) implies

$$t \cdot |P_h| \geq \sum_{j \in \mathcal{J}} \left(t \cdot |P_j^{\mathrm{R}}| + \phi_t^{M^{\mathrm{L}}}(P_j^{\mathrm{L}}) + \phi_t^{M^{\mathrm{R}}}(P_j^{\mathrm{R}})\right) \geq (2t - 1) \cdot \sum_{j \in \mathcal{J}} |P_j^{\mathrm{R}}|,$$

where last inequality follows from the observation that $\phi_t^{M^{\mathrm{L}}}(P) + \phi_t^{M^{\mathrm{R}}}(P) = (t-1) \cdot |P|$. ◀

Proof of Theorem 2. We in fact prove a stronger result than in the theorem statement and show that the algorithm described in the beginning of this section is $(1+\varepsilon)$-competitive while reassigning each request at most $O(\varepsilon^{-(1+\lambda)})$ times for fixed $\lambda > 0$. Consider a path P_h. The intervals I_j, $j \in \mathcal{H}_{2k+2}$, are contained in paths $P_{j'}$, $j' \in \mathcal{H}_{2k+1}$, by Lemma 14, (iii). Lemma 15 implies

$$\textstyle\sum_{j \in \mathcal{H}_{2k+2}} |I_j| \leq \sum_{j' \in \mathcal{H}_{2k+1}} |P_{j'}| \leq \left(2 - \frac{1}{t}\right)^{-k} \cdot \sum_{i \in \mathcal{H}_1} |P_i|. \tag{5}$$

Raghvendra [28] shows that the t-net-cost of augmenting paths is always non-negative. In particular, $\phi_t^{M_{h-1}^*}(P_h) = t \cdot c(P_h \cap M_h^*) - c(P_h \cap M_{h-1}^*) \geq 0$, and thus

$$|P_h| = c(P_h \cap M_h^*) + c(P_h \cap M_{h-1}^*) \leq (t+1) \cdot c(P_h \cap M_h^*). \tag{6}$$

Similarly, $\sum_{i\in\mathcal{H}_1}|P_i| \le (t+1)\cdot\sum_{i\in\mathcal{H}_1} c(P_i\cap M_i^*) \le (t+1)\cdot c(P_h\cap M_i^*)$. In an interval I_i, locally, $M_h^* = M^{\mathrm{L}}$ if and only if $M_i^* = M^{\mathrm{R}}$ by Lemma 14 (ii). With (5) and (6), this implies

$$\left[\tfrac{1}{t+1}\left(2-\tfrac{1}{t}\right)^k - 1\right]\cdot \sum_{j\in\mathcal{H}_{2k+2}} |\mathrm{I}_j| \le c\left(P_h\setminus(\cup_{j\in\mathcal{H}_{2k+2}}\mathrm{I}_j)\cap M^{\mathrm{X}}\right), \quad X\in\{\mathrm{L},\mathrm{R}\}. \tag{7}$$

Denote by $\alpha(k,t)$ the term in square brackets. When a (minimal) edge is frozen, the remaining instance is again alternating and at most one request will take a detour due to (r,s) being frozen. The additional cost is bounded by $|\mathrm{I}_r|$ and can, via Equation (7), be charged to non-frozen parts of M^{OPT}. A part $P_h\cap M^{\mathrm{OPT}}$ is charged this way at most $2k+2$ times before P_h itself is frozen which leads to a competitive factor of $(t+\frac{2k+2}{\alpha(k,t)})$. Substituting $\alpha(k,t)$ from Equation (7) and setting $t = 1+\frac{\varepsilon}{2}$ and $k = \frac{4c(\lambda)}{\varepsilon}\cdot\frac{\varepsilon^{-\lambda}}{\lambda}$, for a constant $c(\lambda)$, we can show that this term is at most $1+\varepsilon$, yielding a recourse of $O_\lambda(\varepsilon^{-(1+\lambda)})$. ◀

As a byproduct, we show, for this special class of instances, a result in the online setting without recourse. It relates the competitive ratio to the cost metric, i.e., the maximum difference in edge cost for connecting a request to a server. This result compares to the best known competitive ratio of $O(\log n)$ by Raghvendra [29].

▶ **Theorem 16.** *The online t-net-cost algorithm is $O(\log\Delta)$-competitive for online matching on an alternating line, where $\Delta = \max_{r,r'\in R, s,s'\in S}\frac{c(r,s)}{c(r',s')}$.*

Proof. Consider an edge e_i of M' and assume $P_i = P_i^{\mathrm{R}}$. By Lemma 14, (ii),

$$t\cdot c(P_i^{\mathrm{L}}\cap M^{\mathrm{L}}) - c(P_i^{\mathrm{L}}\cap M^{\mathrm{R}}) = \phi_t^{M_{i-1}^*}(P_i^{\mathrm{L}}) \ge \phi_t^{M_{i-1}^*}(P_i) = t\cdot c(P_i\cap M^{\mathrm{R}}) - c(P_i\cap M^{\mathrm{L}}).$$

Therefore,

$$|P_i| = c(P_i\cap M^{\mathrm{L}}) + c(P_i\cap M^{\mathrm{R}}) \le \left(1+\tfrac{1}{t}\right)\cdot c(P_i\cap M^{\mathrm{L}}) + c(P_i^{\mathrm{L}}\cap M^{\mathrm{L}}).$$

As in Equation (6), we obtain $|P_i| \le (t+1)\cdot c(P_i\cap M^{\mathrm{R}})$. Together with the above, this implies that $c_{e_i} = |P_i| \le c(\mathrm{I}_i\cap M^{\mathrm{OPT}})\cdot\max\{t+1, 1+\frac{1}{t}\}$. Since intervals corresponding to the same depth in the path-forest are disjoint, we can bound the lengths of paths which are of same depth by $\max\{t+1,1+\frac{1}{t}\}\cdot\mathrm{OPT}$. As there are at most $2\cdot\log_{(2-1/t)}\Delta$ levels in total, by Lemma 15, the theorem's statement follows. ◀

5 Conclusion

In this paper, we give non-trivial results for the min-cost online bipartite matching problem with recourse. The results were obtained simultaneously with and independently of Gupta et al. [17] who consider also more general metrics than the line. We confirm that an average recourse of $O(\log n)$ per request is sufficient to obtain an $O(1)$-competitive matching on the line. It remains open if such a result can be obtained in a non-amortized setting, where the recourse is available only per iteration. Our algorithm is clearly designed for the amortized setting as it buffers online matching decisions and repairs them in batches.

Further, it remains open whether constant recourse per request is sufficient for maintaining an $O(1)$-competitive matching on the line, as for the case of alternating requests. This may be very well possible as there is, currently, no lower bound that rules this out.

Finally, we remind of a major open question in this field: Does there exist an $O(1)$-competitive algorithm for online matching on the line without any recourse?

References

1 S. Angelopoulos, C. Dürr, and S. Jin. Online maximum matching with recourse. In *MFCS*, volume 117 of *LIPIcs*, pages 8:1–8:15, 2018.

2 A. Antoniadis, N. Barcelo, M. Nugent, K. Pruhs, and M. Scquizzato. A $o(n)$-competitive deterministic algorithm for online matching on a line. *Algorithmica*, 81(7):2917–2933, 2019.

3 A. Antoniadis, C. Fischer, and A. Tönnis. A collection of lower bounds for online matching on the line. In *LATIN*, volume 10807 of *LNCS*, pages 52–65, 2018.

4 T. Avitabile, C. Mathieu, and L. Parkinson. Online constrained optimization with recourse. *Inf. Process. Lett.*, 113(3):81–86, 2013.

5 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.

6 A. Bernstein, J. Holm, and E. Rotenberg. Online bipartite matching with amortized $o(\log^2 n)$ replacements. *J. ACM*, 66(5):37:1–37:23, 2019.

7 B. Bosek, D. Leniowski, P. Sankowski, and A. Zych. Online bipartite matching in offline time. In *FOCS*, pages 384–393, 2014.

8 K. Chaudhuri, C. Daskalakis, R. Kleinberg, and H. Lin. Online bipartite perfect matching with augmentations. In *INFOCOM*, pages 1044–1052, 2009.

9 R. Duan and S. Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1):1:1–1:23, 2014.

10 B. Fuchs, W. Hochstättler, and W. Kern. Online matching on a line. *Theor. Comput. Sci.*, 332(1-3):251–264, 2005.

11 M. Gairing and M. Klimm. Greedy metric minimum online matchings with random arrivals. *Oper. Res. Lett.*, 47(2):88–91, 2019.

12 E. Grove, M.-Y. Kao, P. Krishnan, and J. Vitter. Online perfect matching and mobile computing. In *WADS*, volume 955 of *LNCS*, pages 194–205, 1995.

13 A. Gu, A. Gupta, and A. Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45(1):1–28, 2016.

14 A. Gupta, G. Guruganesh, B. Peng, and D. Wajc. Stochastic online metric matching. In *ICALP*, volume 132 of *LIPIcs*, pages 67:1–67:14, 2019.

15 A. Gupta, A. Kumar, and C. Stein. Maintaining assignments online: Matching, scheduling, and flows. In *SODA*, pages 468–479, 2014.

16 A. Gupta and K. Lewi. The online metric matching problem for doubling metrics. In *ICALP*, volume 7391 of *LNCS*, pages 424–435, 2012.

17 V. Gupta, R. Krishnaswamy, and S. Sandeep. Permutation strikes back: The power of recourse in online metric matching. Accepted to appear in APPROX-RANDOM, 2020.

18 M. Imase and B. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

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

20 R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, pages 352–358, 1990.

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

22 H. W. Kuhn and B. Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

23 J. Lacki, J. Oćwieja, M. Pilipczuk, P. Sankowski, and A. Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In *STOC*, pages 11–20, 2015.

24 J. Matuschke, U. Schmidt-Kraepelin, and J. Verschae. Maintaining perfect matchings at low cost. In *ICALP*, volume 132 of *LIPIcs*, pages 82:1–82:14, 2019.

25 N. Megow, M. Skutella, J. Verschae, and A. Wiese. The power of recourse for online MST and TSP. *SIAM J. Comput.*, 45(3):859–880, 2016.

26 A. Mehta. Online matching and ad allocation. *Found. Trends Theor. Comput. Sci.*, 8(4):265–368, 2013.
27 K. Nayyar and S. Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *FOCS*, pages 505–515, 2017.
28 S. Raghvendra. A robust and optimal online algorithm for minimum metric bipartite matching. In *APPROX-RANDOM*, volume 60 of *LIPIcs*, pages 18:1–18:16, 2016.
29 S. Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *SoCG*, volume 99 of *LIPIcs*, pages 67:1–67:14, 2018.
30 Yongho Shin, Kangsan Kim, Seungmin Lee, and Hyung-Chan An. Online graph matching problems with a worst-case reassignment budget. *CoRR*, abs/2003.05175, 2020. arXiv:2003.05175.