Algorithmica
https://doi.org/10.1007/s00453-019-00565-w

A $o(n)$-Competitive Deterministic Algorithm for Online Matching on a Line

Antonios Antoniadis[1] · Neal Barcelo[2] · Michael Nugent[2] · Kirk Pruhs[2] · Michele Scquizzato[3]

Received: 9 September 2017 / Accepted: 13 March 2019

Abstract

Online matching on a line involves matching an online stream of requests to a given set of servers, all in the real line, with the objective of minimizing the sum of the distances between matched server-request pairs. The best previously known upper and lower bounds on the optimal deterministic competitive ratio are linear in the number of requests, and constant, respectively. We show that online matching on a line is essentially equivalent to a particular search problem, which we call *k-lost-cows*. We then obtain the first deterministic sub-linearly competitive algorithm for online matching on a line by giving such an algorithm for the k-lost-cows problem.

A preliminary version of this work [1] was presented at the *12th Workshop on Approximation and Online Algorithms (WAOA 2014)*.

Kirk Pruhs was supported, in part, by NSF Grants CCF-1421508 and CCF-1535755, and an IBM Faculty Award.

✉ Michele Scquizzato
scquizza@math.unipd.it

Antonios Antoniadis
aantonia@mpi-inf.mpg.de

Neal Barcelo
ncb30@pitt.edu

Michael Nugent
mpn1@pitt.edu

Kirk Pruhs
kirk@cs.pitt.edu

1 Max-Planck Institut für Informatik, Saarbrücken, Germany

2 University of Pittsburgh, Pittsburgh, PA, USA

3 University of Padova, Padua, Italy

Keywords Online algorithms · Competitive analysis · Metric matching · Search problems

1 Introduction

The classical Online Metric Matching problem (OMM) is set in a metric space (V, d), containing a set of servers $S = \{s_1, s_2, \ldots, s_n\} \subseteq V$. A set of requests $R = \{r_1, r_2, \ldots, r_n\} \subseteq V$ arrive one-by-one in an online fashion. When a request r_i arrives it must be immediately and irrevocably matched to some previously unmatched server s_j. The cost of matching request r_i to s_j is $d(r_i, s_j)$, and the objective is to minimize the total (equivalently, average) cost of matching all requests. For this problem there exists a deterministic $(2n-1)$-competitive algorithm, and this competitive ratio is optimal for deterministic algorithms [9,13].

The Online Matching on a Line problem (OML) is a special case of OMM where V is the real line and $d(r_i, s_j) = |r_i - s_j|$. The original motivation for considering OML came from applications where there is an online stream of items of various sizes, and the goal is to match each item as it arrives to a stored item of approximately the same size; for example, matching skiers, as they arrive in a ski rental shop, to skis of approximately their height. It is acknowledged that OML is perhaps the most interesting instance of OMM [14]. Despite some efforts, there has been no progress in obtaining a better deterministic upper bound for this special case, and thus the best known upper bound on the competitive ratio for deterministic algorithms is inherited from the upper bound for OMM, namely $2n - 1$.

In the classical cow-path problem, also known as the Lost Cow problem (LC), a cow is standing at a fence (formally represented by the real line) which contains one gate (the target) at some unknown location. Unfortunately the cow is short-sighted, which means that she will not know that she has found the gate until she is standing in front of it. The objective is to minimize the total distance that the cow has to walk until she finds the gate. There exists a 9-competitive algorithm for LC,[1] and this is optimal for deterministic algorithms [3]. Kalyanasundaram and Pruhs [10] observed that LC is a special case of OML where there is an optimal matching with only one positive cost edge, i.e., where only one request is matched with a cost greater than zero.

In 1996, Kalyanasundaram and Pruhs [10] conjectured that the hardest instances for OML are LC instances, and thus that there should be a 9-competitive algorithm for OML. In 2003, Fuchs et al. [7] refuted this conjecture by giving a rather complicated adversarial strategy that gives a lower bound of 9.001 on the competitive ratio of any deterministic algorithm for OML. This is currently the best known lower bound on the deterministic competitive ratio for OML.

[1] Recall that the competitive ratio for a search problem is defined as the largest ratio between the total distance traversed by the agent before reaching its target and the distance it would have traversed under perfect information of the search domain, for all possible locations of the target. A natural assumption is that the target is not located infinitesimally close (i.e., within an arbitrarily small distance ϵ) to the starting position of the cow and on the opposite side of her first move, as otherwise the competitive ratio can be arbitrarily large.

1.1 Our Results

As also noted by the authors, the lower bound in [7] can be intuitively understood as giving a lower bound on the competitive ratio for a search problem involving two lost cows (instead of one), and showing that the optimal deterministic competitive ratio for OML is at least the optimal deterministic competitive ratio for this two lost cows problem. This motivates us to ask the question of whether there is some natural search problem which is equivalent to OML. As search problems seem easier to reason about than online matching, we hypothesize that perhaps one can make progress on online matching by attacking an equivalent search problem. We introduce the following generalization of the LC problem.

Definition 1 (*k-Lost-Cows (k-LC)*) There are k short-sighted cows arriving at a fence, possibly at different locations and at different times. The fence contains k gates at locations unknown to the cows, and the goal of each cow is to find a gate through which to cross the fence. Once a gate has been used by a cow, it cannot be used by other cows. The objective is to minimize the total distance walked by the cows until they all have crossed the fence. We assume that (i) cows can coordinate, (ii) no two cows simultaneously arrive at the same unused gate, (iii) when a cow arrives at an unused gate she has to use it.

As will become clear later, assumptions (ii) and (iii) are without loss of generality. In the remainder of the paper we will think of the fence as the real line. We show that this search problem is essentially equivalent to OML. More precisely we show that:

- If there is a deterministic (resp., randomized) $f(k)$-competitive algorithm for k-LC then there is a deterministic (resp., randomized) $f(n)$-competitive algorithm for OML.
- If there is a deterministic (resp., randomized) $f(p)$-competitive algorithm for OML, where the parameter p is the minimum number of positive cost edges one can have in an optimal matching, then there is a deterministic (resp., randomized) $f(k)$-competitive algorithm for k-LC.

This shows that OML is essentially equivalent to a search problem involving multiple lost cows, instead of just one lost cow (modulo the difference in the parameters n and p).

We give the first sublinearly-competitive, $O\left(n^{\log_2(3+\epsilon)-1}/\epsilon\right)$-competitive for any $\epsilon > 0$ to be precise, deterministic online algorithm for OML, which we obtain by first giving a deterministic $O\left(k^{\log_2(3+\epsilon)-1}/\epsilon\right)$-competitive algorithm for k-LC. Our algorithm for k-LC is a reasonably natural greedy algorithm, but the resulting OML algorithm is not particularly intuitive. This provides mild support for the hypothesis that it is easier to reason about online matching via search rather than online matching directly. We also obtain a lower bound of $\Omega\left(n^{\log_2(3+\epsilon)-1}\right)$ for our algorithm, showing that this analysis is essentially tight.

1.2 Other Related Work

For OML, it had been conjectured [10] that the generalized Work Function Algorithm (WFA) of [15] is $O(1)$-competitive, but this was disproved in [14], where it was shown that the WFA has a competitive ratio of $\Omega(\log n)$. Expanded proofs for this and other results of [14] can be found in [22,23].

Randomized algorithms for OML have also been investigated. In 2006, Meyerson et al. [17] presented the first $o(n)$-competitive algorithm for general metric spaces (and thus for the line). More precisely, [17] obtained an $O\left(\log^3 n\right)$-competitive randomized algorithm using randomized embeddings into trees [6]. One could say that the procedure used to obtain a matching within such trees somewhat resembles a strategy for the Lost Cow problem. Bansal et al. [4] refined the approach in [17] to obtain an $O\left(\log^2 n\right)$-competitive randomized algorithm for general metrics. Finally, Gupta and Lewi [8] gave two different $O(\log n)$-competitive randomized algorithms for the line metric, once again using randomized embeddings, and one being the natural harmonic algorithm. The algorithms in [4,8,17] could in a way be imagined as having lost cows walk on the tree-embeddings. Kao et al. [12] gave a randomized algorithm for LC with competitive ratio of approximately 4.5911, and proved a matching lower bound. Many variants of searching problems such as the LC problem have been extensively studied (see, e.g., [16]).

The OMM problem has also been studied within the framework of resource augmentation, where the online algorithm is given additional servers. Kalyanasundaram and Pruhs [11] showed that a modified greedy algorithm is $O(1)$-competitive if the online algorithm gets twice as many servers. Chung et al. [5] showed that poly-log competitiveness is achievable if the online algorithm gets an additive number of additional servers. Raghvendra [19] recently presented a $(2n - 1 + 1/n)$-competitive deterministic algorithm for OMM which also achieves optimal $O(\log n)$ competitiveness in the random arrival model, where the adversary chooses the set of requests R at the beginning but their arrival order is chosen uniformly at random from all possible permutations.

Our algorithm for k-LC is similar to the natural offline greedy algorithm, which repeatedly matches the two closest points. More precisely, if all the cows arrived at the same time and ϵ was zero, then our algorithm for k-LC would give the same matching as the offline greedy algorithm. Reingold and Tarjan [21] showed that the approximation ratio of the offline greedy algorithm for *non-bipartite* matching is essentially the same as the competitive ratio of our algorithm for k-LC. The first step of the two analyses is the same, looking at the cycles formed by the algorithm's matching and the optimal matching, but they diverge from there. A corollary of our analysis is that the natural offline greedy algorithm is a $\Theta\left(n^{\log_2 3-1}\right)$-approximation for *bipartite* matching. We note that although the result of Reingold and Tarjan holds for (not necessarily bipartite) matching in general metric spaces, this greedy matching can of course not improve upon the tight $(2n-1)$-competitive ratio known for OMM. Intuitively, Sect. 4 crucially uses the straightforward fact that for any three ordered points x, y and z on the line, it holds that $|xy| + |yz| = |xz|$. Relaxing this to a triangle inequality would blow up the competitive ratio by a factor of n.

2 Overview

In this section we present an informal overview of both our algorithms and analyses for k-LC and OML.

In Sect. 3 we consider the k-Lost-Cows Without Arrivals (k-LCWA) problem, which is a restriction of k-LC in which each cow arrives at time $t = 0$. Our algorithm for OML is based on simulating an algorithm for k-LC, which in turn is based on simulating an algorithm for k-LCWA. Recall that in the optimal deterministic algorithm for the 1-LC problem the cow follows the classical "doubling strategy", whereby she changes her direction of movement at increasing powers of 2, i.e., at points $-1, 2, -4, \dots$ of the real line (assuming, without loss of generality, that the cow is initially located at the origin); for k-LCWA, we consider the algorithm A where each cow independently and in parallel follows a generalization of this strategy in which she switches direction of movement at increasing powers of $1 + \epsilon$, for some $\epsilon > 0$. For the case $k = 1$, it is well-known (see, e.g., [16, Lemma 2.1]) that the competitive ratio of algorithm A is $O(\epsilon + 1/\epsilon)$. To keep the paper self-contained, we provide this analysis in "Appendix A".

By Proposition 1 (see "Appendix A"), one nice feature of algorithm A is that, for any $0 < \epsilon \leq 1$, its cost is within a factor of $O(1/\epsilon)$ of the cost of the final matching M between cows' starting positions and the corresponding gates that the cows use in A. To analyze the cost of M we consider the union of M and the optimal matching OPT. As Reingold and Tarjan have already observed in [21] these edges can be decomposed into a set of disjoint cycles. We then prove some structural properties regarding the directions of edges in M and OPT. Finally, we can charge the cost of M's edges to the cost of OPT's edges based on the order in which A matches cows.

As an example, consider the base case of the first cow c that finds a gate in this cycle, and let ℓ denote the length of the edge corresponding to this matching. Since c's search is never biased more than $1 + \epsilon$ in either direction from its origin, we know that the closest gate to c is at least $\ell/(1 + \epsilon)$ away. Also, since A has all cows walking in parallel and no other cow has found a gate, we know that no other cow has a gate closer than $\ell/(1 + \epsilon)$. Using this argument we can charge the cost of this edge to any edge in OPT. As we proceed inductively, the inequalities become more complicated since we now may have to charge to multiple edges in both M and OPT. To aid our analysis we define a weighted binary tree for each cycle, with the property that the sum of the leaf costs is OPT's cost, and the sum of the internal nodes is an upper bound on M's cost. We show that if each tree is perfect (complete and balanced) then M is $O\left(k^{\log_2(3+\epsilon)-1}\right)$-competitive. The last step is to show that perfect trees are the worst case.

In Sect. 4 we then show how to extend the algorithm for k-LCWA to an algorithm for k-LC. Dealing with the online arrival of cows is a bit tricky since the charging argument used in the analysis of the algorithm A for k-LCWA is delicately based on the order in which cows find their gates. To cope with this, we simulate the state that A would be in had all the cows arrived at time 0, and use this to possibly change how the cows walk. Specifically, in the general k-LC problem, if a cow c is walking and finds a gate occupied by some other cow c', the algorithm determines which cow would have found this gate first had the two cows been released at the same time; if c' would have found the gate first, then c continues her own walk, otherwise c changes her walking strategy to the one that c' had at the time she reached the gate. It is relatively straightforward to see that the total distance walked by the k cows in this simulation is exactly the same as the total distance walked had all the cows arrived at time 0.

In Sect. 5 we show how to reduce k-LC to OML, and OML to k-LC. To convert an algorithm for k-LC into an algorithm for OML, one can release a new cow for every request r of the OML instance, wait until this cow hits an unoccupied server s, and then match r to s. To convert an algorithm for OML into an algorithm for k-LC one can continually issue requests at a cow's current location until a request is matched with a server corresponding to an unoccupied gate.

3 The Parallel Cows Algorithm for *k*-Lost-Cows Without Arrivals

We now define the $(1+\epsilon)$-Parallel Cows algorithm for the k-LCWA problem. The algorithm is to have every cow walk according to the $(1+\epsilon)$-cow algorithm independently and in parallel. In particular, this means that in her walk each cow ignores the other cows as well as all the gates already occupied by other cows. Throughout this section we assume ϵ to be some fixed parameter, and thus when possible we remove reference to it to lighten notation (e.g., Parallel Cows algorithm is a shorthand for $(1+\epsilon)$-Parallel Cows algorithm). In this section we analyze the Parallel Cows algorithm, and prove the following result.

Theorem 1 *For $\epsilon \leq 1$, the $(1+\epsilon)$-Parallel Cows algorithm for k-LCWA is $O\left(k^{\log_2(3+\epsilon)-1}/\epsilon\right)$-competitive.*

In Sect. 3.1 we discuss that we can view the matchings of cows to gates found by the Parallel Cows algorithm and by OPT as a set of disjoint weighted cycles. Properties of these cycles allow us to use weighted binary trees to analyze the total cost of the Parallel Cows algorithm in relation to OPT, where the leaves of a tree correspond to the edges in OPT and the internal nodes correspond to edges in the Parallel Cows matching. In Sect. 3.2 we analyze this tree in the case when it is a perfect binary tree and all of OPT's edges have the same cost, which intuitively seems like the worst case. In Sect. 3.3 we prove that we do indeed obtain the worst case matching for the Parallel Cows algorithm when the cycle analysis yields a perfect binary tree. The competitive ratio for the Parallel Cows algorithm then follows from observing that, by Proposition 1 in "Appendix A", when $\epsilon \leq 1$ the walking costs are only $O(1/\epsilon)$ times the matching costs.

3.1 Cycle Property

In this section we discuss some useful properties about the combination of the matchings produced by OPT and by the Parallel Cows algorithm, denoted with A.

We first introduce some notation. Let $C = \{c_1, c_2, \ldots, c_k\} \subseteq \mathbb{R}$ denote the set of cow starting locations. We use c_i to refer to the cow herself as well as her starting location, specifying it when it is not clear from context. Let $G = \{g_1, g_2, \ldots, g_k\} \subseteq \mathbb{R}$ denote the set of gate locations. When referring to a specific algorithm, we use $g(c_i)$:

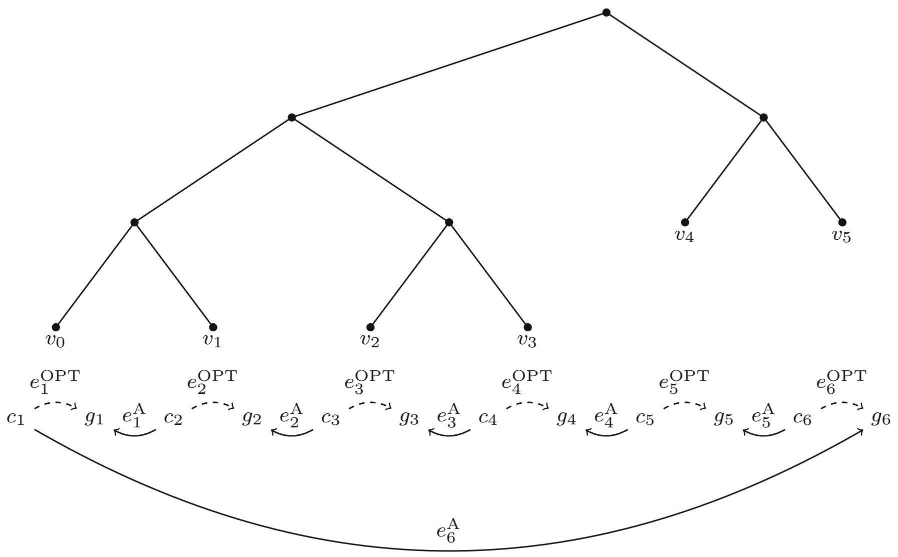

Fig. 1 A cycle and the corresponding MVST

$C \to G$ to denote the gate that cow c_i matches to. Consider the graph with vertices $V = C \cup G$. Let $E^{\mathrm{OPT}} = \{e_1^{\mathrm{OPT}}, e_2^{\mathrm{OPT}}, \dots, e_k^{\mathrm{OPT}}\}$ be OPT's edges in the graph, where $e_i^{\mathrm{OPT}} = (c_i, g(c_i))$, and whose weight is $|c_i - g(c_i)|$. E^{A} is defined analogously.

As already observed in [21], the graph with vertices V and edges $E^{\mathrm{OPT}} \cup E^{\mathrm{A}}$ is a disjoint union of cycles, and furthermore it suffices to consider the ratio of the two costs when the union of the two matchings is a single cycle.

Figure 1 depicts one such cycle.

3.2 Perfect Tree Case

In this section, we show how to associate a cycle as described in the previous section with a weighted binary tree, where the sum of the values of the leaves of the tree is the cost of the optimal matching in that cycle, and the sum of the values of the internal nodes of the tree provides an upper bound on the cost of the matching found by the Parallel Cows algorithm. We additionally analyze the cost of this tree when the tree is perfect.

Definition 2 A *Full Weighted/Valued Binary Tree (FWVBT)* T is a full binary tree whose vertices are each associated to a value. The *weight* of T is defined as the sum of the values of all the vertices of T. The *cost* of T is defined as the sum of the values of all internal vertices of T. The *total leaf value* of T is defined as the sum of the values of all the leaves of T.

Definition 3 A $(1+\epsilon)$-*Minimum Value Subtree Tree* ($(1+\epsilon)$-*MVST)* is an FWVBT with the property that the value of a vertex i is equal to $(1+\epsilon)\min\{v_1, v_2\}$, where v_1 and v_2 are the weights of the subtrees of i rooted at the left and right child of i

respectively (there is no constraint on the values of leaves). A *perfect* $(1+\epsilon)$-MVST is a $(1+\epsilon)$-MVST where each leaf has the same depth and the same value.

Since we assume ϵ is a fixed parameter, throughout we abbreviate $(1+\epsilon)$-MVST by MVST. Given a cycle as described in the previous section, one can associate it with a MVST, as follows (see Fig. 1):

- Each edge e_i^{OPT} of OPT corresponds to a vertex/leaf of value $|c_i - g(c_i)|$. Each such leaf forms a distinct singleton (connected) component.
- Consider the edges of A, except the longest edge e_ℓ^A, in the order in which A adds them. For each edge e_i^A added, a vertex is introduced, and the two neighboring connected components become its children in the tree. This merges the two connected components into a single one. The value of this new vertex is set to be $(1+\epsilon)\min\{v_1, v_2)\}$, where v_1 and v_2 are the weights of the two subtrees of the vertex.
- It can be easily verified that for each edge e_i^A the total number of connected components decreases by one, and that the tree is indeed binary and full.

We have the following lemma:

Lemma 1 *With respect to a cycle and the corresponding MVST, the cost of the optimal matching OPT is the total leaf value of the tree, while the cost of A's matching is upper bounded by the cost of the tree.*

Proof The first part of the observation about the cost of OPT is straightforward. With respect to the cost of A, we claim that the cost of an edge added is upper bounded by $(1+\epsilon)$ times the cost of the cheapest of the two connected components that get merged as a result of the addition of this edge. We can show this inductively: by the definition of the algorithm any vertex of height 2 in the resulting tree cannot have a cost more than $(1+\epsilon)$ times the smallest value leaf among the two leaves that got merged into one component. Otherwise the smallest of the neighboring OPT edges would have been selected by the algorithm.

For the inductive step, assume the statement holds for edges corresponding to vertices up to some height h. Then, for an edge corresponding to a vertex of height $h+1$, we have that it has a cost that is upper bounded by $(1+\epsilon)$ times the cost of the cheapest connected component it merges, which is the sum of the weights of the edges in it; otherwise, by definition, A would "close the cycle" for this cheapest connected component instead of selecting the edge it did. However this connected component now consists of costs coming from edges of OPT (the values of the leaves) and from the edges of A (the internal nodes). For the edges of A we do not know the exact value but by the inductive hypothesis we have an upper bound that is their value in the tree. □

Let us now assume that the cycle produced by A has a length that is a power of two, and that the above construction produces a perfect binary tree, where each leaf has a value of 1. We prove the following lemma.

Lemma 2 *A perfect MVST with k leaves all of value* 1 *has cost* $\Theta\left(k^{\log_2(3+\epsilon)}\right)$.

Proof We first observe that all the leaves have the same value, and that the tree is perfect. Therefore for an internal node i with left and right subtrees of weight v_1 and v_2, respectively, we have $v_1 = v_2$. To simplify the analysis we will assume without loss of generality that the value of i is $(1+\epsilon)v_1$, i.e., the leaves of the left subtree are the ones that contribute to the weight of i. This implies that a leaf ℓ contributes to the weight of one of its ancestors i if and only if ℓ is a leaf on the left subtree of i, or in other words, if and only if the last edge on the path from ℓ to i is a *right-turn*. We can similarly define a *left-turn*.

Let $f(j)$ be the contribution of a leaf ℓ (whose value is 1) to the value of an internal node i that is an ancestor of ℓ, assuming that the path from ℓ to i contains exactly j right-turns. It can be verified that when the tree is perfect and each leaf has a value of 1, then f depends just on the number of right-turns j on the path from the leaf to the root.

It holds that $f(0) = 1$, and for $j \geq 1$ we have

$$\begin{aligned} f(j) &= (1+\epsilon)\sum_{i=0}^{j-1} f(i) \\ &= (1+\epsilon)(1+(1+\epsilon))\sum_{i=0}^{j-2} f(i) \\ &= (1+\epsilon)(1+(1+\epsilon))^2\sum_{i=0}^{j-3} f(i) \\ &\ \ \vdots \\ &= (1+\epsilon)(1+(1+\epsilon))^{j-1}. \end{aligned}$$

Also, since each leaf only contributes to a vertex if it lies after a right-turn on the path from the leaf to the root, we have that each leaf whose path to the root has j right-turns contributes to the total cost of the tree a value of

$$\sum_{i=1}^{j} f(i) = \sum_{i=1}^{j}(1+\epsilon)(1+(1+\epsilon))^{i-1} = (2+\epsilon)^j - 1.$$

Note that on a perfect binary tree with k leaves, the path from each leaf to the root has length $\log_2 k$, and that the k different paths from the leaves to the root are exactly all possible configurations of left-turns and right-turns on a path of length $\log_2 k$. It follows that the total cost of a perfect tree with k leaves of value 1 is

$$\sum_{i=0}^{\log_2 k} \binom{\log_2 k}{i} \left((2+\epsilon)^i - 1\right) = (3+\epsilon)^{\log_2 k} - k = k^{\log_2(3+\epsilon)} - k,$$

which concludes the proof of the lemma. □

The above lemma also implies that the cost of the matching returned by the algorithm for this particular class of cycles (the ones corresponding to a perfect binary tree with leaf values of 1) is a $\Theta\left(n^{\log_2(3+\epsilon)-1}\right)$-approximation with respect to the optimal matching. As we will see in the next subsection, this particular class of cycles is actually the worst case.

3.3 Analysis

The following lemma implies that the competitive ratio of the algorithm is at most

$$\left(\frac{3+\epsilon}{2}\right)^{\log_2 k} = k^{\log_2 \frac{3+\epsilon}{2}},$$

where $k \leq n$ is the number of leaves in the tree. This is the case because the cost of the optimal solution is the total leaf value L of the tree. The lemma furthermore implies that the perfect trees analyzed in the previous subsection are indeed the worst case.

Lemma 3 *Let T be a $(1+\epsilon)$-MVST with the values of its k leaves summing to L. Then the cost of T is upper bounded by*

$$L \cdot \left(\frac{3+\epsilon}{2}\right)^{\log_2 k}.$$

Proof We prove the lemma by induction on the levels of the tree. For the base case, consider a leaf of the tree. The induced subtree clearly has only one vertex which is simultaneously a leaf, and therefore $k = 1$. The statement directly follows.

For the inductive step, let T_1 and T_2 be the left and right subtrees in T, respectively. Let $L_1 = \ell L_2$ for $\ell \geq 0$, and $k_1 = ck_2$ for $c > 0$. Note that c, in contrast to ℓ, cannot be 0, since by the definition of MVST the tree is full. We have $L = (1+\ell)L_2$, and $k = (1+c)k_2$. Without loss of generality, assume that

$$L_1 \left(\frac{3+\epsilon}{2}\right)^{\log_2 k_1} \leq L_2 \left(\frac{3+\epsilon}{2}\right)^{\log_2 k_2},$$

which by substitution implies that $\ell \leq \left(\frac{3+\epsilon}{2}\right)^{\log_2 \frac{1}{c}}$. If we let $c(T)$ denote the cost of T, then we have

$$\begin{aligned}
c(T) &= c(T_1) + c(T_2) + (1+\epsilon)\min\{c(T_1), c(T_2)\} \\
&\leq (2+\epsilon)L_1 \left(\frac{3+\epsilon}{2}\right)^{\log_2 k_1} + L_2 \left(\frac{3+\epsilon}{2}\right)^{\log_2 k_2} \\
&\leq (2+\epsilon)\frac{\ell}{\ell+1}L \left(\frac{3+\epsilon}{2}\right)^{\log_2 \frac{c}{c+1}k} + \frac{1}{\ell+1}L \left(\frac{3+\epsilon}{2}\right)^{\log_2 \frac{1}{k+1}k} \\
&= L \left(\frac{3+\epsilon}{2}\right)^{\log_2 k} \left[(2+\epsilon)\frac{\ell}{\ell+1}L \left(\frac{3+\epsilon}{2}\right)^{\log_2 \frac{c}{c+1}} + \frac{1}{\ell+1} \left(\frac{3+\epsilon}{2}\right)^{\log_2 \frac{1}{c+1}}\right].
\end{aligned}$$

In order to prove the lemma it therefore suffices to show that

$$\begin{aligned}
&(2+\epsilon)\frac{\ell}{\ell+1}\left(\frac{3+\epsilon}{2}\right)^{\log_2\frac{c}{c+1}}+\frac{1}{\ell+1}\left(\frac{3+\epsilon}{2}\right)^{\log_2\frac{1}{c+1}}\leq 1\\
&\Leftrightarrow (2+\epsilon)\ell\left(\frac{3+\epsilon}{2}\right)^{\log_2 c}\left(\frac{3+\epsilon}{2}\right)^{\log_2\frac{1}{c+1}}+\left(\frac{3+\epsilon}{2}\right)^{\log_2\frac{1}{c+1}}\leq \ell+1\\
&\Leftrightarrow (2+\epsilon)\ell\left(\frac{3+\epsilon}{2}\right)^{\log_2 c}+1\leq(\ell+1)\left(\frac{3+\epsilon}{2}\right)^{\log_2(c+1)}\\
&\Leftrightarrow (\ell+1)(c+1)^{\log_2(\frac{3+\epsilon}{2})}-(2+\epsilon)\ell c^{\log_2(\frac{3+\epsilon}{2})}-1\geq 0.
\end{aligned}$$

Let

$$f(\ell,c)=\ell\left[(c+1)^{\log_2\frac{3+\epsilon}{2}}-(2+\epsilon)c^{\log_2\frac{3+\epsilon}{2}}\right]+(c+1)^{\log_2\frac{3+\epsilon}{2}}-1.$$

Since $f(\ell,c)$ is linear in ℓ, and furthermore for any fixed $c>0$ we are interested in values of ℓ such that $0\leq\ell\leq\left(\frac{3+\epsilon}{2}\right)^{\log_2\frac{1}{c}}$, in order to prove that $f(\ell,c)\geq 0$ for all such values of c and ℓ it suffices to show that:

(i) $f(0,c)\geq 0$, for all $c>0$, and
(ii) $f\left(\left(\frac{3+\epsilon}{2}\right)^{\log_2\frac{1}{c}},c\right)\geq 0$, for all $c>0$.

We now show these two inequalities.

(i) We have $f(0,c)=(c+1)^{\log_2\frac{3+\epsilon}{2}}-1$, which is clearly nonnegative because $c+1>1$ when $c>0$.
(ii) We have

$$\begin{aligned}
&f\left(\left(\frac{1}{c}\right)^{\log_2\frac{3+\epsilon}{2}},c\right)\\
&\quad=\left(\frac{1}{c}\right)^{\log_2\frac{3+\epsilon}{2}}\left[(c+1)^{\log_2\frac{3+\epsilon}{2}}-(2+\epsilon)c^{\log_2\frac{3+\epsilon}{2}}\right]+(c+1)^{\log_2\frac{3+\epsilon}{2}}-1\\
&\quad=\left(\frac{c+1}{c}\right)^{\log_2\frac{3+\epsilon}{2}}-(2+\epsilon)+(c+1)^{\log_2\frac{3+\epsilon}{2}}-1.
\end{aligned}$$

It therefore suffices to show that

$$g(c):=\left(\frac{c+1}{c}\right)^{\log_2\frac{3+\epsilon}{2}}+(c+1)^{\log_2\frac{3+\epsilon}{2}}-(3+\epsilon)\geq 0,$$

for all $c > 0$. By using two times the inequality of arithmetic and geometric means, we have:

$$\begin{aligned} g(c) &= \left(\frac{3+\epsilon}{2}\right)^{\log_2(1+1/c)} + \left(\frac{3+\epsilon}{2}\right)^{\log_2(1+c)} - (3+\epsilon) \\ &\geq 2\sqrt{\left(\frac{3+\epsilon}{2}\right)^{\log_2(c+2+1/c)}} - (3+\epsilon) \\ &\geq 2\sqrt{\left(\frac{3+\epsilon}{2}\right)^{\log_2(2+2\sqrt{1})}} - (3+\epsilon) \\ &= 2\frac{3+\epsilon}{2} - (3+\epsilon) = 0. \end{aligned}$$

This concludes the proof of the lemma. □

3.4 Proof of Parallel Cows Competitiveness

We can now prove Theorem 1.

Proof (*of Theorem* 1) Fix some instance of k-LCWA, where the value of the optimal solution is OPT, and let A be the cost of the Parallel Cows algorithm on this instance and M_A be the cost of the matching found by the Parallel Cows algorithm on this instance. Note that the cost of the optimal matching and the cost of the optimal solution are the same. By Lemma 1, M_A is the sum of the costs of the MVSTs built on the cycles induced by the algorithm's and optimal's matchings, while OPT is the sum of the costs of the leaves of those same MVSTs. Fix one cycle, and let T be the MVST for that cycle. By Lemma 3 the cost of the algorithm's matching on this cycle is at most $O\left(k_c^{\log_2(3+\epsilon)-1}\right)$ times that of the optimal's, where k_c is the number of cows and gates in the cycle. Thus $M_A = O\left(k^{\log_2(3+\epsilon)-1}\mathrm{OPT}\right)$.

Since each cow only stops when it finds an unused gate, the walking cost of each cow in the Parallel Cows algorithm is the same as if that cow and the gate it finds were the only ones present. Thus when $\epsilon \leq 1$ its walking cost is $O(1/\epsilon)$ times its matching cost, and thus $A = O\left(k^{\log_2(3+\epsilon)-1}\mathrm{OPT}/\epsilon\right)$. □

4 From *k*-Lost-Cows Without Arrivals to *k*-Lost-Cows

In this section we show how to extend the solution for the k-LCWA problem of the previous section to the k-LC problem. The algorithm for k-LC is a suitable simulation of the $(1+\epsilon)$-Parallel Cows algorithm for the k-LCWA problem. The resulting algorithm for k-LC inherits exactly the same competitive ratio of the $(1+\epsilon)$-Parallel Cows algorithm for k-LCWA (but outputs a possibly different matching of cows to gates).

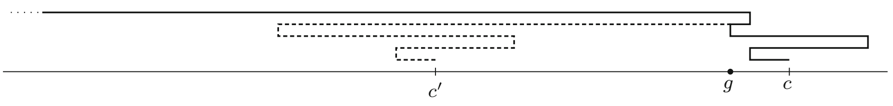

Fig. 2 Simulation algorithm for k-LC. Cow c reaches gate g, which is already occupied by cow c'. However, c would have reached gate g first, had c and c' been released at the same time; in this case, when c reaches gate g she updates her walking strategy (solid) to follow the walking strategy of c' (dashed)

The algorithm is as follows. As soon as a cow c is released, she starts walking following the $(1+\epsilon)$-cow strategy. Whenever, during her walk, c finds a gate g, there are two possible cases. If g does not yet have a cow matched to it, then c is matched to g. Otherwise, there is already some cow c' matched to g. In this case, the algorithm determines which one between c and c' would have reached location g first if both c and c' were released at time 0.[2] If c' would have found g first, then c continues her walk. Otherwise, c would have found g earlier than c' would. Ideally, in this case we wish c to "kick" c' out of gate g, and then c' to continue her walk. However, since c' cannot be removed from gate g once she has been matched to it, we have c simulate the rest of the walk of c'. This means that c changes its walking strategy to follow, from that point on, the walking strategy that c' had at the time she reached gate g. See Fig. 2. (For this case it might be helpful to think that, when they meet at gate g, cows c and c' act as if they swapped identities.)

Clearly, the optimal (offline) matching of cows to gates does not change in case of arbitrary release times associated to the cows. Therefore, in order to prove that the above algorithm has the same competitive ratio as the Parallel Cows algorithm for k-LCWA we only need to show that the total walking cost of this simulation is equal to the total walking cost if all the cows were released at time 0.

Lemma 4 *The total distance walked by the cows in the above simulation algorithm for k-LC is exactly the same as the total distance walked by the cows in the* $(1+\epsilon)$*-Parallel Cows algorithm for k-LCWA.*

Proof The lemma follows by construction: the algorithm simulates the $(1+\epsilon)$-Parallel Cows algorithm for the k-LCWA problem in such a way that any path walked by a cow in the latter is also walked in the simulation algorithm, possibly with different disjoint portions of the same path being walked by different cows, and paths walked in the $(1+\epsilon)$-Parallel Cows algorithm are the only paths walked in the simulation algorithm. □

A direct consequence of Theorem 1 and Lemma 4 is the following.

Theorem 2 *For* $\epsilon \leq 1$*, the above simulation algorithm for k-LC is* $O\left(k^{\log_2(3+\epsilon)-1}/\epsilon\right)$*-competitive.*

5 Comparing *k*-Lost-Cows and Online Matching on a Line

In this section we explore the relationship between the k-LC and OML problems. In particular we show that upper bounds in the k-LC setting carry over to upper bounds

[2] We remark that we are implicitly assuming that all cows walk at the same speed.

in the OML setting (assuming the competitive ratio is defined in terms of the number of servers, n). We also show that lower bounds in the k-LC setting carry over to lower bounds in the OML setting; however, here the competitive ratio for OML is defined in terms of the minimum number of positive requests in an optimal matching.

Definition 4 Given a matching M, a request $r_i \in R$ is said to be *positive in* M if r_i is matched in M to a server s_j such that $d(r_i, s_j) > 0$.

Although one can perturb the input instance so that all the n requests are positive, we prefer to distinguish between the number p of positive requests in an optimal solution and the total number of requests n. This is because in our reduction: (i) we explicitly introduce a large number of requests that OPT should match with cost zero, and (ii) it better exemplifies the connection between the number of positive requests in an optimal solution for the OML instance and the number of cows k in the corresponding k-LC instance.

Theorem 3 *Let p be the minimum number of positive requests in an optimal solution to OML. The following two implications hold.*

1. *If there is an $f(k)$-competitive algorithm for k-LC then there is an $f(n)$-competitive algorithm for OML.*
2. *If there is an $f(p)$-competitive algorithm for OML then there is an $f(k)$-competitive algorithm for k-LC.*

Proof To prove the first claim we shall simulate the $f(k)$-competitive algorithm for k-LC, as follows: given an input instance for the OML problem, we create an input instance for k-LC by placing one gate at each server location, and by releasing a new cow at location r_i whenever a request r_i arrives. Once released, the cow walks according to the algorithm for k-LC, until she reaches her final gate g; the request r_i is then matched to the server found at the location of gate g. Since the cost of the final matching is clearly no more than the total distance walked by the cows, and since by construction the number of cows is equal to the number of requests, the claim follows.

For the second part, assume that we have an $f(p)$-competitive algorithm A for OML, and let I be an instance to the k-LC problem. To obtain an $f(k)$-competitive algorithm for k-LC, we create an instance I' for the OML problem with a server at every integer. When a cow arrives, at some location c we add one request for location c in I'. It is easy to see, using a simple exchange argument, that A can be converted into an algorithm that matches each request r with one of its two surrounding servers (i.e., the rightmost unmatched server on the left of r and the leftmost unmatched server on the right of r - note that one of these two servers could be collocated with request) with no increase of competitive ratio. Assuming this, we now have c walk to the server that A matches the request to, which is either one unit to the left, one unit to the right or staying put. While c has not yet found a gate, we continue to have a request in I' at c's current location, always having c walk to the server that is used to match the latest request. Once c finds a gate, we stop requesting to c's location. We continue to do this until all cows have been matched.

First note that by construction the total walking cost of the cows is equal to the matching cost of A. Further, we claim that there is an optimal solution to I' with k

positive requests. To see this assume by contradiction that the optimal matching to I' with the minimum number of positive requests has more than k positive requests. Since there are k gates, at least one positive request, say r_i, must be matched to a non-gate location, say $s(r_i)$. However every used server that is not a gate receives a request. So there is another request at location $s(r_i)$, say r_j, that is matched to some positive cost sever $s(r_j)$. Note that the optimal solution that matches r_i to $s(r_j)$ and r_j as cost 0 does not increase the cost of the matching and has one less positive request. This contradicts our choice of OPT. This shows that there is an algorithm A that is $f(k)$-competitive on I', and therefore the corresponding cow algorithm is $f(k)$-competitive on I.

Notice that in order to achieve the competitive ratio of $f(k)$ for instances with k cows, we have to reduce to an OML instance with $p = k$ positive requests and a potentially much larger n. □

The following is a direct consequence of Theorems 2 and 3.

Theorem 4 *There is an $O\left(n^{\log_2(3+\epsilon)-1}/\epsilon\right)$-competitive algorithm for OML.*

We note that, in a manner similar to that presented in Sect. 4, it is possible to extend the Parallel Cows algorithm for k-LCWA to OML directly and obtain an $O\left(p^{\log_2(3+\epsilon)-1}/\epsilon\right)$-competitive algorithm for OML.

6 Conclusions

This paper presents the first deterministic sub-linearly competitive algorithm for online matching on a line, by exploiting and analyzing the connections between this problem and a generalization of the well-known lost-cow problem.

Since the conference version of this paper appeared, there have been further advancements in the understanding of the problem. Nayyar and Raghvendra [18] provided a new analysis of the deterministic online algorithm presented in [19], showing that on a line its competitive ratio is $O(\log^2 n)$; very recently, Raghvendra [20] improved the analysis of this algorithm to a competitive ratio of $O(\log n)$. Despite these advancements, there is still a big gap between the best known upper and lower bounds for the problem. Furthermore, Antoniadis, Fischer, and Tönnis [2] have shown a lower bound of $\Omega(\log n)$ for a restricted class of algorithms. Since this class contains all the deterministic algorithms found in the literature, it would be interesting to try to either develop an algorithm that does not belong to this class, or extend their construction to show a lower bound for an even wider class of algorithms.

Appendix

A Analysis of the $(1+\epsilon)$-Cow Algorithm

Here we analyze the search strategy for the 1-LC problem whereby the cow, starting at the origin of the real line, changes her direction of movement at points

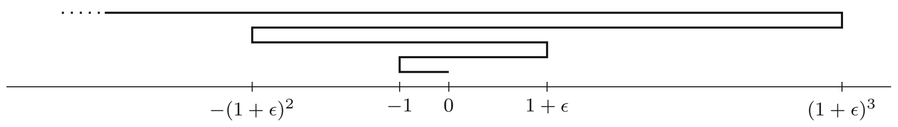

Fig. 3 Representation of the $(1+\epsilon)$-cow algorithm

$$p_1 = -1, p_2 = (1+\epsilon), p_3 = -(1+\epsilon)^2, p_4 = (1+\epsilon)^3, \dots, p_i = (-1)^i(1+\epsilon)^{i-1}, \dots,$$

for some $\epsilon > 0$. See Fig. 3. We refer to this strategy as the $(1+\epsilon)$-*cow algorithm.*

Proposition 1 *The* $(1+\epsilon)$*-cow algorithm is* $O(\epsilon + 1/\epsilon)$*-competitive for the* 1*-LC problem.*

Proof Let $p \in \mathbb{R}$ be the location of the gate, $A_\epsilon(p)$ be the total distance traversed by the cow following the $(1+\epsilon)$-cow algorithm until she finds the gate located at point p, and $c(A_\epsilon)$ be the competitive ratio of the $(1+\epsilon)$-cow algorithm. Then, by definition,

$$\begin{aligned}
c(A_\epsilon) &= \sup_{p\in\mathbb{R}} \left\{ \frac{A_\epsilon(p)}{|p|} \right\} \\
&= \sup_{p\in\mathbb{R}} \left\{ \frac{\sum_{i=1}^{k} 2|p_i| + |p|}{|p|} : p \in [p_{k-1}, p_{k+1}] \right\} \\
&= 1 + \sup_{p\in\mathbb{R}} \left\{ \frac{\sum_{i=1}^{k} 2|p_i|}{|p|} : p \in [p_{k-1}, p_{k+1}] \right\} \\
&= 1 + \sup_{k\in\mathbb{Z}} \left\{ \frac{\sum_{i=1}^{k} 2|p_i|}{|p_{k-1}|} \right\} \\
&= 1 + \sup_{k\in\mathbb{Z}} \left\{ \frac{\sum_{i=1}^{k} 2(1+\epsilon)^{i-1}}{(1+\epsilon)^{k-2}} \right\} \\
&= 1 + \sup_{k\in\mathbb{Z}} \left\{ 2 \cdot \frac{(1+\epsilon)^k - 1}{((1+\epsilon)-1)(1+\epsilon)^{k-2}} \right\} \\
&< 1 + 2 \cdot \frac{(1+\epsilon)^2}{\epsilon} \\
&= O\left(\epsilon + \frac{1}{\epsilon}\right),
\end{aligned}$$

as desired. □

References

1. Antoniadis, A., Barcelo, N., Nugent, M., Pruhs, K., Scquizzato, M.: A $o(n)$-competitive deterministic algorithm for online matching on a line. In: Proceedings of the 12th International Workshop on Approximation and Online Algorithms (WAOA), pp. 11–22 (2014)

2. Antoniadis, A., Fischer, C., Tönnis, A.: A collection of lower bounds for online matching on the line. In: Proceedings of the 13th Latin American Theoretical Informatics Symposium (LATIN), pp. 52–65 (2018)
3. Baeza-Yates, R.A., Culberson, J.C., Rawlins, G.J.E.: Searching in the plane. Inf. Comput. **106**(2), 234–252 (1993)
4. Bansal, N., Buchbinder, N., Gupta, A., Naor, J.: A randomized $O(\log^2 k)$-competitive algorithm for metric bipartite matching. Algorithmica **68**(2), 390–403 (2014)
5. Chung, C., Pruhs, K., Uthaisombut, P.: The online transportation problem: on the exponential boost of one extra server. In: Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN), pp. 228–239 (2008)
6. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. **69**(3), 485–497 (2004)
7. Fuchs, B., Hochstättler, W., Kern, W.: Online matching on a line. Theor. Comput. Sci. **332**(1–3), 251–264 (2005)
8. Gupta, A., Lewi, K.: The online metric matching problem for doubling metrics. In: Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP), pp. 424–435 (2012)
9. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. J. Algorithms **14**(3), 478–488 (1993)
10. Kalyanasundaram, B., Pruhs, K.: Online network optimization problems. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms: The State of the Art, pp. 268–280. Springer, Berlin Heidelberg (1998)
11. Kalyanasundaram, B., Pruhs, K.: The online transportation problem. SIAM J. Discrete Math. **13**(3), 370–383 (2000)
12. Kao, M.-Y., Reif, J.H., Tate, S.R.: Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. Inf. Comput. **131**(1), 63–79 (1996)
13. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theor. Comput. Sci. **127**(2), 255–267 (1994)
14. Koutsoupias, E., Nanavati, A.: The online matching problem on a line. In: Proceedings of the 1st International Workshop on Approximation and Online Algorithms (WAOA), pp. 179–191 (2003)
15. Koutsoupias, E., Papadimitriou, C.H.: On the k-server conjecture. J. ACM **42**(5), 971–983 (1995)
16. López-Ortiz, A.: On-Line Target Searching in Bounded and Unbounded Domains. Ph.D. thesis, University of Waterloo (1996)
17. Meyerson, A., Nanavati, A., Poplawski, L.J.: Randomized online algorithms for minimum metric bipartite matching. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 954–959 (2006)
18. Nayyar, K., Raghvendra, S.: An input sensitive online algorithm for the metric bipartite matching problem. In: Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS), pp. 505–515 (2017)
19. Raghvendra, S.: A robust and optimal online algorithm for minimum metric bipartite matching. In: Proceedings of the 19th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), pp. 18:1–18:16 (2016)
20. Raghvendra, S.: Optimal analysis of an online algorithm for the bipartite matching problem on a line. In: Proceedings of the 34th International Symposium on Computational Geometry (SoCG), pp. 67:1–67:14 (2018)
21. Reingold, E.M., Tarjan, R.E.: On a greedy heuristic for complete matching. SIAM J. Comput. **10**(4), 676–681 (1981)
22. van Stee, R.: SIGACT news online algorithms column 27: Online matching on the line, part 1. SIGACT News **47**(1), 99–110 (2016)
23. van Stee, R.: SIGACT news online algorithms column 28: Online matching on the line, part 2. SIGACT News **47**(2), 40–51 (2016)