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

Antonios Antoniadis[2⋆], Neal Barcelo[1⋆⋆], Michael Nugent[1⋆⋆⋆], Kirk Pruhs[1†], and Michele Scquizzato[3‡]

[1] Department of Computer Science, University of Pittsburgh
[2] Max-Planck Institut für Informatik
[3] Department of Computer Science, University of Houston

Abstract. Online matching on a line involves matching an online stream of items of various sizes to stored items of various sizes, with the objective of minimizing the average discrepancy in size between matched items. The best previously known upper and lower bounds on the optimal deterministic competitive ratio are linear in the number of items, and constant, respectively. We show that online matching on a line is essentially equivalent to a particular search problem, that 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.

1 Introduction

The classic 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 online. When a request r_i arrives it must irrevocably be 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. There is a deterministic $(2n-1)$-competitive algorithm, and this competitive ratio is optimal for deterministic algorithms [7,11].

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

⋆ Part of the work was done while the author was at the University of Pittsburgh, supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD). E-mail: aantonia@mpi-inf.mpg.de.

⋆⋆ This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1247842. E-mail: ncb30@pitt.edu.

⋆⋆⋆ E-mail: mpn1@pitt.edu.

† Supported in part by NSF grants CCF-1115575, CNS-1253218, CCF-1421508, and an IBM Faculty Award. E-mail: kirk@cs.pitt.edu.

‡ Work done while at the University of Pittsburgh, supported in part by a fellowship of "Fondazione Ing. Aldo Gini", University of Padova. E-mail: michele@cs.uh.edu.

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 (see, e.g., [12]). 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 short-sighted cow is standing at a fence (formally, the real line) that contains a single gate at some unknown distance. The cow needs to traverse the fence until she finds the gate (formally, the algorithm needs to specify a walk on the real line). The objective is to minimize the distance traveled until the gate is found. There is a 9-competitive algorithm for LC, and this is optimal for deterministic algorithms [1]. [8] observed that LC is a special case of OML where there is an optimal matching with only one positive cost edge.

In 1996, [8] conjectured that the hardest instances for OML are LC instances, and thus that there should be a 9-competitive algorithm for OML. In 2003, [5] 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.

1.1 Our Results

Upon further reflection, the lower bound in [5] 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 that 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 the equivalent search problem. We show that the following search problem is essentially equivalent to OML:

k-Lost Cows (k-LC): k short-sighted cows arrive at a fence (formally, the real line) at potentially different times. The fence contains k gates in unknown positions. At each point in time, the online algorithm can specify a particular cow that has already arrived, and a direction, and then that cow will move one unit in that direction.[4] When a cow finds a gate, she will cross the fence, and this gate cannot be used by other cows. Each cow must cross the fence through a gate. The objective is to minimize the total distance traveled by the k cows.

[4] Allowing the cows to instead move simultaneously would not affect our results.

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 many lost cows, instead of one lost cow (modulo the difference in the parameters n and p).

We give the first sublinearly-competitive, $O\big(n^{\log_2(3+\epsilon)-1}/\epsilon\big)$-competitive for any $\epsilon > 0$ to be more precise, deterministic online algorithm for OML, which we obtain by first giving a deterministic $O\big(k^{\log_2(3+\epsilon)-1}/\epsilon\big)$-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 $\Omega\big(n^{\log_2(3+\epsilon)-1}\big)$ for our algorithm, showing that this analysis is essentially tight.

1.2 Other Related Work

For OML, it had been conjectured [8] that the generalized Work Function Algorithm (WFA) of [13] is $O(1)$-competitive, but this was disproved in [12], where it was shown that the WFA has a competitive ratio of $\Omega(\log n)$.

Randomized algorithms for OML have also been investigated. In 2006, [15] gave the first randomized algorithm and analysis giving a competitive ratio of $o(n)$ for general metric spaces (and thus for the line). More precisely, [15] obtained an $O\big(\log^3 n\big)$-competitive randomized algorithm using randomized embeddings into trees [4]. [2] refined the approach in [15] to obtain an $O\big(\log^2 n\big)$-competitive randomized algorithm for general metrics. Finally, [6] gave two different $O(\log n)$-competitive randomized algorithms for the line metric, one again using randomized embeddings, and one being the natural harmonic algorithm. [10] 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 (e.g., [14]).

OMM also has been studied within the framework of resource augmentation, where the online algorithm is given additional servers. [9] showed that a modified greedy algorithm is $O(1)$-competitive if the online algorithm gets twice as many servers. [3] showed that poly-log competitiveness is achievable if the online algorithm gets an additive number of additional servers.

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. [16] showed that the approximation ratio of the offline greedy algorithm for *non-bipartite* matching is essentially the same as the competitive ratio as 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 $\Theta\big(n^{\log_2 3-1}\big)$-approximation for *bipartite* matching.

2 Overview

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

In Section 3 we consider the problem of k-Lost Cows Without Arrivals (k-LCWA), 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 the optimal deterministic algorithm for 1-LC switches directions at increasing powers of 2 (i.e., switch directions at points -1, 2, -4, etc.). For k-LCWA, we consider the algorithm A where each cow independently and in parallel uses this optimal single cow algorithm, but switches directions at powers of $1+\epsilon$ instead of 2. One nice feature of A is that for any $\epsilon \leq 1$, the cost for A will be 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 used in A. To analyze the cost of M we consider the union of M and the optimal matching OPT. It is easy to see that these edges can be decomposed into a set of disjoint cycles. We give directions to edges in M and OPT based on whether a cow's starting position is on the left or on the right of the matched gate. We then prove some structural properties regarding the directions of edges in M and OPT. We can then 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 distance $\ell/(1+\epsilon)$ 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 $O\big(k^{\log_2(3+\epsilon)-1}\big)$-competitive. The last step is to show that perfect trees are the worst case. Although this is somewhat intuitive, this is by far the most technical aspect of the analysis, and involves showing that given an arbitrary tree, we can make a sequence of transformations such that the resulting tree is perfect, and at each step we do not decrease the competitive ratio.

In Section 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 state to change how the cows walk. More specifically, if a cow c is walking in the k-LC setting and finds a gate occupied by some cow c', the algorithm determines which cow would have found this gate first in the no arrivals case, and allows this cow to stay there, "kicking out" the other cow to continue walking. It is then relatively straightforward to see that the matching produced by the simulation is identical to the matching that A would have produced had all the cows arrived at time 0.

In Section 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 cow for every request r in the OML instance, and wait until this cow hits an unoccupied server s. Then, by matching r to s, the matchings in the two settings are equivalent, and the number of cows k will be equal to the number of servers n. 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.

Due to space constraints, many of the proofs are deferred to the full version of the paper.

3 Analysis of 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 move according to the $(1+\epsilon)$-cow algorithm independently and in parallel (i.e., simulate moving all cows at the same time by choosing cows to move in a round-robin fashion, and have each cow make one step to their left, then $1+\epsilon$ steps to the right of their starting location, etc.). In particular,this means that every cow ignores every other cow or any used gates that she finds. Throughout this section, we assume ϵ is some fixed parameter and so remove reference to it when possible 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 theorem:

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

In Subsection 3.1 we prove that we can view the matchings of cows to gates found by the Parallel Cows algorithm and OPT as a set of disjoint weighted cycles. We show these cycles have special structure, which allows 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 Subsection 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 worst case. In Subsection 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 the walking costs are only $O(1/\epsilon)$ times the matching costs.

3.1 Cycle Property

In this subsection we will show some useful properties about the combination of the matchings produced by OPT and by the Parallel Cows algorithm, denoted with A. In addition to building some intuition for the problem, these properties will show that our analysis is tight.

We first define some notation. Let $C = \{c_1, \ldots, c_k\} \subseteq \mathbb{Z}$ denote the set of cow starting locations. We use c_i to both refer to the cow as it is walking and as the starting location, specifying when it is not clear from context. Let $G = \{g_1, \ldots, g_k\} \subseteq \mathbb{Z}$ denote the set of gate locations. When referring to a specific algorithm, we use $g(c_i) : C \to G$ to denote the gate that cow c_i matches to. Consider the graph with vertices $V = C \cup G$. Let $E^{\mathrm{OPT}} = \{e_1^{\mathrm{OPT}}, e_2^{\mathrm{OPT}}, \ldots, e_k^{\mathrm{OPT}}\}$ be OPT's edges in the graph, where $e_i^{\mathrm{OPT}} = (c_i, g(c_i))$ and has weight $|c_i - g(c_i)|$. We can similarly define E^{A} for A. It is easy to see that the graph with vertices V and edges $E^{\mathrm{OPT}} \cup E^{\mathrm{A}}$ is a disjoint union of cycles.

We now argue that there exists an optimal solution OPT where, loosely speaking, edges e^{OPT} do not *cross*, where by crossing we mean a pair of edges (e_i, e_j) such that exactly one of the two endpoints of one edge lies between the two endpoints of the other edge. Formally, we have the following lemma.

Lemma 1. *There exists an optimal matching* $e_1^{\mathrm{OPT}}, e_2^{\mathrm{OPT}}, \ldots, e_k^{\mathrm{OPT}}$ *where for each* $i, j \in \{1, 2, \ldots, k\}$*, with* $i \leq j$*, the following holds:*

$$\begin{aligned} &\min\{c_j, g(c_j)\} \leq \min\{c_i, g(c_i)\} \Rightarrow \\ &\text{either } \max\{c_j, g(c_j)\} \leq \min\{c_i, g(c_i)\} \text{ or } \max\{c_j, g(c_j)\} \geq \max\{c_i, g(c_i)\}. \end{aligned}$$

Henceforth, we may assume that OPT does not have crossings.

It is easy to show that, by the definition of the algorithm, the same non-crossing property holds for the Parallel Cows algorithm. Given an input instance, consider both the solutions produced by OPT and by A. By leveraging the above property, we can partition the arcs of the two solutions into single cycles, where by cycle we mean a set of arcs from both A and OPT such that the undirected graph induced by them is a cycle. Figure 1 depicts one such cycle.

Finally, we will argue that any such cycle looks like the one in Figure 1. We say that two edges e_i and e_j are in the *same direction* if either (i) $c_i \leq g(c_i)$ and $c_j \leq g(c_j)$ or (ii) $c_i \geq g(c_i)$ and $c_j \geq g(c_j)$, and the *opposite direction* if this is not the case.

Lemma 2. *Fix a cycle in the graph of A's and OPT's matchings. Let A's edges be* $e_1^A, e_2^A, \ldots, e_\ell^A$ *and OPT's edges be* $e_1^{\mathrm{OPT}}, \ldots, e_\ell^{\mathrm{OPT}}$*. Then* $e_1^A, \ldots, e_{\ell-1}^A$ *are all in one direction and* $e_\ell^A, e_1^{\mathrm{OPT}}, \ldots, e_\ell^{\mathrm{OPT}}$ *are all in the opposite direction.*

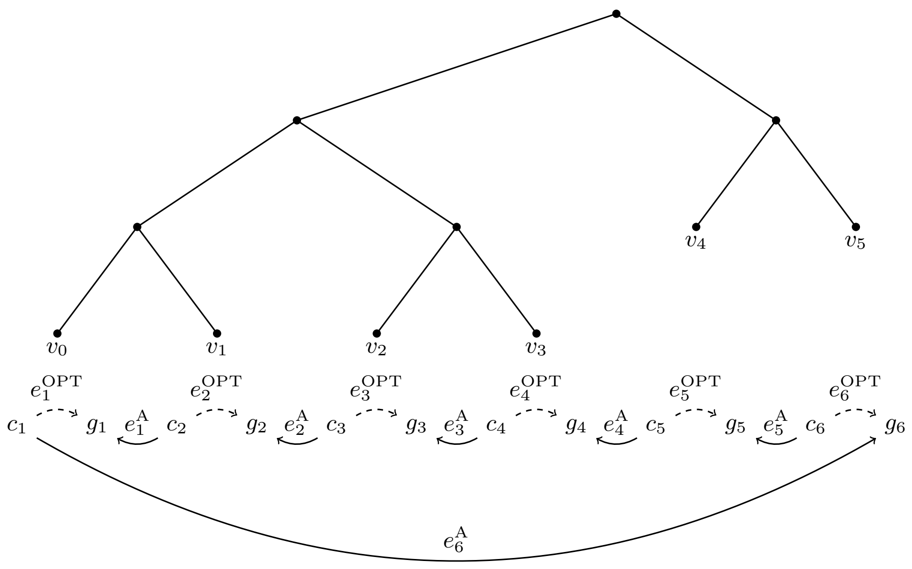

Fig. 1. A cycle and the corresponding MVST.

3.2 Perfect Tree Case

In this subsection, we show how to associate a cycle as described in the previous section with a weighted binary tree, where the sum of the weights of the leaves of the tree is the cost of the optimal matching in that cycle, and the sum of the weights 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 1. *A* Full Weighted/Valued Binary Tree (FWVBT) *T is a full binary tree, whose vertices are each associated with a weight and a value.*

- *The* weight *of T is defined as the sum of the values of all the vertices in T.*
- *The* cost *of T is defined as the sum of the values of all internal (non-leaf) vertices of T.*
- *The* total leaf value *of T is defined as the sum of the values of all the leaves of T.*

Definition 2. *A* $(1+\epsilon)$-Minimum Value Subtree Tree ($(1+\epsilon)$-MVST) *is an FWVBT T with the following property: The value of a node 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 Figure 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 (connected) component.
- Consider the edges of A, except 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 gets decreased by one, and that the tree is indeed binary and full.

We have the following lemma:

Lemma 3. *With respect to this 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.*

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 4. *A perfect MVST where each leaf has a value of* 1 *has cost* $\Theta\left(k^{\log_2(3+\epsilon)}\right)$.

Combined with Lemma 2, 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 Perfect Trees are the Worst Case

The result of this section is the following lemma:

Lemma 5. *The cost of any MVST T is at most the cost of the smallest (in terms of number of nodes) perfect MVST with size at least that of T and total leaf value the same as T.*

To prove this, one can show that any MVST that has maximum cost for a fixed tree with fixed total leaf value must have certain structure. One can then show that a series of transformations can be performed on the tree such that the final tree is a perfect MVST and that each transformation results in a new tree with same or greater cost. The first transformation removes any nodes in the tree with value 0, and the second transformation creates a balanced tree by replacing leaves of maximum depth in the tree with leaves at a minimum possible depth in the tree. The final transformation creates a perfect tree by adding leaves until the tree is perfect.

3.4 Proof of Parallel Cows Competitiveness

We can now prove Theorem 1.

Proof (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 3, 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 leaves of those same MVSTs. Fix one cycle, and let T be the MVST for that cycle. By Lemma 5, the cost of T can be upper bounded by the cost of the minimum perfect MVST larger than T, while the cost of the optimal solution on that cycle remains fixed. Thus we have by Lemma 4 that the cost of the algorithm's matching on this cycle is at most $O\Big(k_c^{\log_2(3+\epsilon)-1}\Big)$ times that of the optimal's, where k_c is the number of cows and gates in the cycle. Thus M_A is at most $O\big(k^{\log_2(3+\epsilon)-1}OPT\big)$.

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 we obtain that $A = O\big(k^{\log_2(3+\epsilon)-1}OPT/\epsilon\big)$. □

4 Extending the Algorithm for *k*-Lost Cows Without Arrivals to *k*-Lost Cows

In this section we show how to extend the solution for the k-LCWA problem to the k-LC problem. The basic idea is to maintain the state of our Parallel Cows algorithm assuming all cows arrived at time 0. When a cow finds a gate (occupied or unoccupied) we stop the cow and figure out who would match there in the Parallel Cows algorithm. Based on this we update our state and continue searching. We formalize this below.

While there is a cow c that is released and not matched to a gate, have c walk as a $(1+\epsilon)$-biased cow. If during this walk c finds a gate g there are two cases to consider. If g does not yet have a cow matched to it, then match c to g. Otherwise, if there is some cow c' currently matched to h, we calculate which cow would have reached location h first if both c and c' were released at time 0. If c' would reach this location first, then c continues her walk. If c would find h first, then c "kicks" c' out who continues his walk. Since we cannot actually remove c' from this gate once it is matched, we simply have c walk as c', and record that c is really matched to g.

The first observation is that the offline optimal matching does not change if there are release times associated with the cows. So to show that this algorithm has the same competitive ratio of the Parallel Cows algorithm, we only need to show that the matching cost of this simulation is equal to the matching cost if all cows were released at time 0. The result then follows from Theorem 1.

Lemma 6. *Let $I = (C, G)$ be an instance to the generalized cow problem with release time a_i for cow c_i. The walking cost of the above simulation algorithm on I is equal to the walking cost of the parallel cow algorithm on I with zero release times.*

Proof. Let $M \subset C \times G$ be the matching found by the Parallel Cows algorithm on I with zero release times. For each $c \in C$, recall that $g(c) \in G$ is the gate that c matches to in M, that is $(c, g(c)) \in M$. We first show that if c ever reaches the point $g(c)$ in the simulation, then it will remain there for the rest of the simulation, i.e., if c reaches its gate it will match there. To see this, assume by contradiction that some cow c reaches $g(c)$ but later leaves this location. Further, assume c is the first cow to do this, and let c' be the cow that causes c to leave $g(c)$. By definition of the simulation, this means that c' would reach location $g(c)$ first in the parallel cows algorithm. However, since c' does not match to $g(c)$, this means that c' has already reached and left location $g(c')$ contradicting that c was the first cow to leave its gate.

With this we can now prove the desired lemma, that the matching is indeed the same. Let $M_s \subset C \times G$ denote the matching found by the simulation. Let $c_1, c_2, \ldots, c_k \in C$ such that c_i is the i-th cow to find its gate in the parallel cows algorithm. We show by induction that, for all i, $(c_i, g(c_i)) \in M_s$. For the base case, note that since c_1 is the first cow to find a gate in the parallel algorithm, c_1 will find $g(c_1)$ before any other gate in G. However by the above argument, once c_1 finds $g(c_1)$ it will match there in M_s. Now for the inductive hypothesis, assume the first $j-1$ cows match the same in M and M_s. Note that since c_j is the j-th cow to find its gate in the parallel algorithm, it sees at most $j-1$ gates before reaching $g(c_j)$ in the parallel algorithm, and these all belong to $g(c_1), g(c_2), \ldots, g(c_{j-1})$. However, by the inductive hypothesis, $c_1, c_2, \ldots, c_{j-1}$ will match there in M_s, so c_j can not match to any of those. This means that c_j must reach $g(c_j)$ and by the above argument $(c_j, g(c_j)) \in M_s$. □

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 positive results in the k-LC setting carry over to positive results 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.

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

Proof. For the first implication, assume that we have an $f(p)$-competitive algorithm A for OML and let $I = (C, G)$ be an instance to the k-LC problem. To obtain an $f(k)$-competitive algorithm, we create an instance I' in the OML problem with a server at every integer. When a cow arrives, at some location c we have two requests for location c in I'. It can easily be shown that A can be converted to an algorithm where every request is matched with the left most or right most server with no increase in cost. Assuming this, we now have c walk to the server that A matches the second request with, which will either be one unit to the left or right. 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 within a constant factor of 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 minimum (in terms of positive requests) optimal matching to I' 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.

To prove the second implication assume that we have an $f(k)$-competitive algorithm for the k-LC problem and as input to the OML problem we are given an input I consisting of a set of servers S. Whenever a request r_i arrives, we release a cow c at location r_i and it begins to walk. Whenever c reaches a location corresponding to a server $s \in S$ that no previous request has been matched to s we match r_i to s and place a gate location at s so c stops walking. It is clear that the cost of the matching is at most the total walking cost of the cows and further the number of cows is equal to the number of servers. □

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

Proof. By Theorem 1 and Lemma 6, we have an $O\big(k^{\log_2(3+\epsilon)-1}/\epsilon\big)$-competitive algorithm for k-LC. The result follows from Theorem 2. □

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

References

1. R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993.
2. N. Bansal, N. Buchbinder, A. Gupta, and J. Naor. A randomized $O(\log^2 k)$-competitive algorithm for metric bipartite matching. *Algorithmica*, 68(2):390–403, 2014.
3. C. Chung, K. Pruhs, and P. Uthaisombut. The online transportation problem: On the exponential boost of one extra server. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN)*, pages 228–239, 2008.
4. J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.
5. B. Fuchs, W. Hochstättler, and W. Kern. Online matching on a line. *Theor. Comput. Sci.*, 332(1-3):251–264, 2005.
6. A. Gupta and K. Lewi. The online metric matching problem for doubling metrics. In *Proceedings of the 39th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 424–435, 2012.
7. B. Kalyanasundaram and K. Pruhs. Online weighted matching. *J. Algorithms*, 14(3):478–488, 1993.
8. B. Kalyanasundaram and K. Pruhs. Online network optimization problems. In *Online Algorithms: The State of the Art*, pages 268–280. Springer-Verlag, 1998.
9. B. Kalyanasundaram and K. Pruhs. The online transportation problem. *SIAM J. Discrete Math.*, 13(3):370–383, 2000.
10. M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Inf. Comput.*, 131(1):63–79, 1996.
11. S. Khuller, S. G. Mitchell, and V. V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theor. Comput. Sci.*, 127(2):255–267, 1994.
12. E. Koutsoupias and A. Nanavati. The online matching problem on a line. In *Proceedings of the 1st International Workshop on Approximation and Online Algorithms (WAOA)*, pages 179–191, 2003.
13. E. Koutsoupias and C. H. Papadimitriou. On the k-server conjecture. *J. ACM*, 42(5):971–983, 1995.
14. A. López-Ortiz. *On-line Target Searching in Bounded and Unbounded Domains.* PhD thesis, University of Waterloo, 1996.
15. A. Meyerson, A. Nanavati, and L. J. Poplawski. Randomized online algorithms for minimum metric bipartite matching. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 954–959, 2006.
16. E. M. Reingold and R. E. Tarjan. On a greedy heuristic for complete matching. *SIAM J. Comput.*, 10(4):676–681, 1981.