On the online unit clustering problem*

Leah Epstein† Rob van Stee‡

June 17, 2008

Abstract

We continue the study of the online unit clustering problem, introduced by Chan and Zarrabi-Zadeh (*Workshop on Approximation and Online Algorithms 2006*, LNCS 4368, p.121–131. Springer, 2006). We design a deterministic algorithm with a competitive ratio of $7/4$ for the one-dimensional case. This is the first deterministic algorithm that beats the bound of 2. It also has a better competitive ratio than the previous randomized algorithms. Moreover, we provide the first non-trivial deterministic lower bound, improve the randomized lower bound, and prove the first lower bounds for higher dimensions.

1 Introduction

In clustering problems, a set of points need to be partitioned into groups, also called clusters, so as to optimize a given objective function. Clustering problems are fundamental and have many applications, this includes usage of clustering for computer related purposes, such as information retrieval and data mining, and various applications in other fields such as medical diagnosis and facility location.

In the online model, points are presented one by one to the algorithm, and must be assigned to clusters upon arrival. This assignment cannot be changed later. We measure the performance of an online algorithm $\mathcal{A}$ by comparing it to an optimal offline algorithm OPT using the competitive ratio, which is defined as

$$\sup_{\sigma} \frac{\mathcal{A}(\sigma)}{\mathrm{OPT}(\sigma)} .$$

Here, σ is the input, which is a sequence of points, and $\mathrm{ALG}(\sigma)$ denotes the cost of an algorithm ALG for this input, which is typically the number of clusters. For randomized algorithms, we replace $\mathcal{A}(\sigma)$ with $\mathbb{E}(\mathcal{A}(\sigma))$, and define the competitive ratio as $\sup_{\sigma} \frac{\mathbb{E}(\mathcal{A}(\sigma))}{\mathrm{OPT}(\sigma)}$. An algorithm with competitive ratio of at most $\mathcal{R}$ is called $\mathcal{R}$-competitive.

Charikar et al. [2] considered a problem which is called *the online unit covering problem*. In this problem, a set of n points needs to be covered by balls of unit radius, and the goal is to minimize the number of balls used. They gave an upper bound of $O(2^d d \log d)$ and a lower bound of $\Omega(\log d / \log\log\log d)$ on the competitive ratio of deterministic online algorithms in d dimensions. This problem is fully online in the sense that points arrive one by one, each point needs to be assigned to a ball upon arrival, and if it is assigned

*A preliminary version of this paper appeared in *Proceedings of the 5th Workshop on Approximation and Online Algorithms (WAOA 2007)*, Lecture Notes in Computer Science XXXX, Springer Verlag, 2008, pages 193–206.

†Department of Mathematics, University of Haifa, 31905 Haifa, Israel. lea@math.haifa.ac.il.

‡Department of Computer Science, University of Karlsruhe, D-76128 Karlsruhe, Germany. vanstee@ira.uka.de. Research supported by the Alexander von Humboldt Foundation.

to a new ball, the exact location of this ball is fixed at this time. The tight bounds on the competitive ratio for $d = 1$ and $d = 2$ are 2 and 4 respectively.

Chan and Zarrabi-Zadeh [1, 8] introduced the unit clustering problem. This problem is still an online problem and is similar to unit covering. However, it is more flexible and does not require that the exact position of the balls is fixed in advance. The algorithm needs to make sure that a set of points which is assigned to one cluster can always be covered by a unit ball. The goal is still to minimize the total number of clusters used. Therefore, the algorithm may terminate with clusters that still have more than one option for their location. In the offline model, this reduces to unit covering. However, in the online model, an algorithm now has the option of moving a cluster after a new point arrives, as long as this cluster still covers all the points that are assigned to it. In [1], the two-dimensional problem is considered in the L_∞ norm rather than the L_2 norm. Thus "balls", are actually squares. For $d = 1$ the two metrics are identical. In this paper, similarly to [1], we consider the L_∞ norm.

To implement the option of moving a previously defined cluster to a different location, we sometimes define smaller clusters in cases where a cluster can still be shifted. Specifically, if $d = 1$, this means that we may define a cluster of a length which is smaller than 1, and possibly extend it later to a total length of at most 1. If $d \geq 2$, a complete cluster is a square or cube. However, we may similarly define a cluster which is a rectangle or a box with sides of at most 1, where each side can be extended later to a length of at most 1. Thus, the output may contain clusters which were never extended to the full possible size.

Note that online clustering is an online graph coloring problem. If we see the clusters as colors, and the points are seen as vertices, then an edge between two point occurs if they are too far apart to be colored using the same color. The resulting graph for the one-dimensional problem is the complement of a unit interval graph (alternatively, the problem can be seen as a clique partition problem in unit interval graphs). See [5] for a survey on online graph coloring. Note that online coloring is a difficult problem that does not admit a constant competitive ratio already for trees [3, 6]. There is a small number of classes that admit constant competitive algorithms, one of which is interval graphs [4].

For the one-dimensional case, [1] showed that several naïve algorithms all have a competitive ratio of 2. Some of these algorithms are actually designed to solve already the unit covering problem and thus cannot be expected to overcome this bound (due to [2]). They also showed that any randomized algorithm for unit covering has a competitive ratio of at least 2. To demonstrate the difference between unit covering and unit clustering, they presented a randomized algorithm with a competitive ratio of $15/8 = 1.875$. Finally, they showed a lower bound of $4/3$ on the competitive ratio of any randomized algorithm. The deterministic lower bound that is implied by their work is $3/2 = 1.5$. A multi-dimensional extension of their algorithm, that they design, results in a $15/4 = 3.75$-competitive algorithm for two dimensions, or a $2^d \cdot 15/16$-competitive algorithm for general d. The randomized upper bound for one dimension was improved to $11/6$ by the same authors [8], implying corresponding improvements for higher dimensions.

We improve these results by presenting a relatively simple *deterministic* algorithm which attains a competitive ratio of $7/4 = 1.75$. Using the construction presented by Chan and Zarrabi-Zadeh [1], this implies an upper bound of $2^d \cdot 7/8$ in d dimensions. Moreover, we improve the randomized lower bound to $3/2 = 1.5$ and show a deterministic lower bound of $8/5 = 1.6$. Finally we give a deterministic lower bound of 2 and a randomized lower bound of $11/6 \approx 1.8333$ in two dimensions. The deterministic lower bound holds for the L_2 norm as well. A summary of previous and improved results can be found in Table 1.

We start the paper with additional definitions, afterwards, we present the new algorithm followed by its analysis. Finally, we prove lower bounds, first for one dimension and then for two dimensions.

	Lower bound of [1]	Lower bound (this paper)	Upper bound (this paper)	Upper bound of [8]
$d = 1$ deterministic	1.5	1.6	1.75	2
$d = 1$ randomized	1.33	1.5	1.75	1.83
$d = 2$ deterministic	1.5	2	3.5	4
$d = 2$ randomized	1.33	1.83	3.5	3.67

Table 1: Summary of new and previous results for one and two dimensions

2 A deterministic algorithm

2.1 Definitions

Throughout the design and analysis of our algorithm, we consider a fixed optimal offline algorithm.

As noted in [1], it is trivial to provide an optimal solution for a given input offline: starting from the left, repeatedly define a cluster of length 1 that has as its left endpoint the leftmost unserved point. It can be seen that in this solution, no two clusters overlap (not even at their endpoints). We will compare our algorithm, which would also not let clusters overlap, to this solution. We call the clusters used by our algorithm "online clusters" and the clusters of the optimal solution are called "optimal clusters".

For a cluster C, denote the leftmost request point contained in it by ℓ_C and the rightmost request point by r_C. A cluster is *single* if there is no cluster which has a common endpoint with it. A cluster is *fixed* if we have defined both its endpoints.

The distance of p to a cluster C is denoted by $d(p, C)$ and is determined as the distance from p to the closest point in C. For a fixed cluster C, this closest point is not necessarily a request point. The distance between two single clusters C and D is defined as the distance between their closest points.

We now define several kinds of pairs of clusters. In these definitions we discuss two clusters, where a cluster C is to the left of cluster D, without overlap. We call a pair of clusters close or far only if there is no cluster between them and there is no fixed cluster 'nearby'. Below, we specify what nearby means in this context.

Definition 1 *A* close pair *consists of two consecutive* single *clusters C and D such that one of the following two properties holds:*

- $d(\ell_C, \ell_D) \leq 1$ *and there is no fixed cluster which overlaps with the interval* $(\ell_D - 1, \ell_D + 1)$
- $d(r_C, r_D) \leq 1$ *and there is no fixed cluster which overlaps with the interval* $(r_C - 1, r_C + 1)$

Definition 2 *A* far pair *consists of two* single *clusters C and D that do not form a close pair and for which* $d(r_C, \ell_D) \leq 1$. *Moreover, there is no fixed cluster which overlaps with the interval* $(r_C - 1, \ell_D + 1)$.

Note that a cluster which contains a single point cannot be part of a far pair, only of a close pair, since for this cluster its left endpoint and its right endpoint are the same point.

Definition 3 *A* fixed pair *consists of two* fixed *clusters C and D that have a common endpoint which is a request point.*

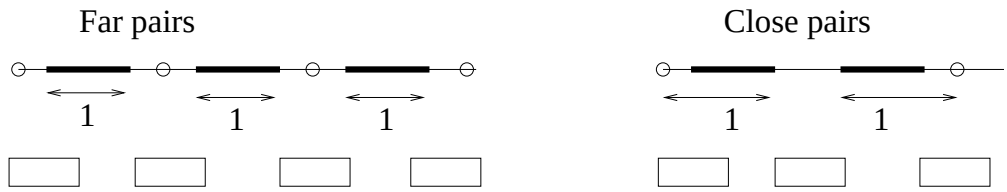

Figure 1: The difficulties caused by far pairs and close pairs. At the top are the online clusters (marked in bold) and the new requests (small circles) At the bottom is a possible optimal solution. Left, the new requests force the opening of new online clusters; right, there is a choice between opening new clusters or creating a far pair.

Our algorithm avoids close and far pairs to avoid bad examples and in particular, bad examples shown in [1]. Intuitively, the problem with far pairs is that the gap between them may be covered by an optimal cluster, but neither cluster in the far pair might be able to serve some requests in that gap, forcing the online algorithm to open a new cluster in that gap later. The problem with close pairs is that they may be pairs of clusters which are short and might turn into far pairs later, leading to the same problem as before, if the online algorithm assigns additional points to one the clusters (or to both of them). See also Figure 1.

Generally, we should try to avoid situations where there may be very little overlap between optimal clusters and online clusters. Therefore, such pairs are turned into fixed pairs using the *attach* operation which we now define.

Our algorithm attaches one cluster to another cluster in one of two ways, left-to-right-attach and right-to-left-attach. Let C, D be a close pair and assume again that D is to the right of C. The algorithm sometimes attaches cluster C to cluster D and sometimes it attaches cluster D to cluster C. In the first case, cluster C is attached to cluster D as follows. Fix the location of D to be the interval $[\ell_D, \ell_D + 1]$ and the location of C to be the interval $[\ell_D - 1, \ell_D]$. This attach operation is only performed if $\ell_C \geq \ell_D - 1$, i.e., in the first case of the definition of a close pair, and called left-to-right-attach, since the rightmost point of the left cluster is fixed to be the leftmost point of the right cluster. If C or D overlaps with an existing cluster as a result of these definitions, we *truncate* it at the point where it starts to overlap. Since there is no cluster between C and D, the overlap can happen only at the right hand side of D or at the left hand side of C.

In the other option, we can attach D to C. To do that, we fix C at $[r_C - 1, r_C]$ and D at $[r_C, r_C + 1]$. This is only done if $r_D \leq r_C + 1$, i.e., in the second case of the definition of a close pair, and called right-to-left-attach. Again, we truncate C or D if this is necessary to avoid overlap.

Thus it can be seen that if a cluster is attached to another, they form a fixed pair. The clusters in a fixed pair always have length 1 unless this would make them overlap with some other cluster. Single clusters are never fixed by our algorithm, and thus their right and left endpoints are request points (possibly, both endpoints are the same request point). See Figure 2 for examples of these attach operations.

2.2 The algorithm

The idea of this algorithm is to try and avoid gaps between clusters if requests occur 'close' to one another.

A request inside a cluster is assigned to that cluster. Let a *good cluster for* p be a single cluster C such that p can be assigned to C without creating a new far or close pair. Let a *feasible cluster for* p be a single cluster C such that there is no cluster between C and p and the distance of p to the furthest request point in C is at most 1.

1. If there exists a good cluster C for p, assign p to C.

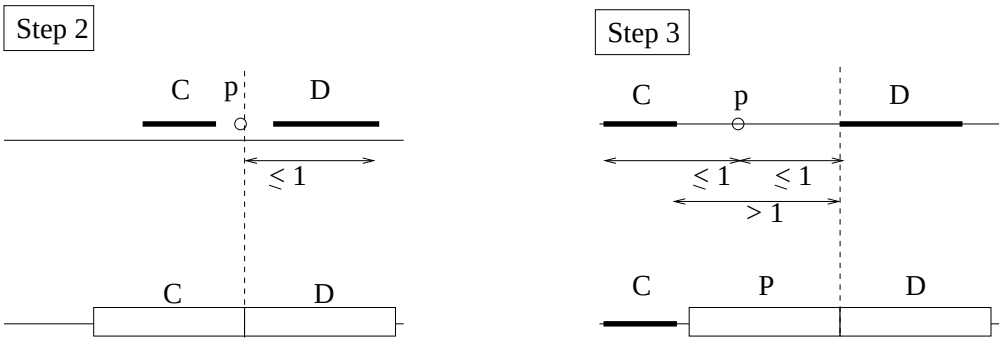

Figure 2: Creation of a fixed pair in Step 2 and 3 of our algorithm. On the left is a right-to-left attach operation (p is assigned to C and then D is attached to C, on the right a left-to-right attach operation (P is attached to D).

2. Else, if there exists a feasible cluster C for p such that assigning p to C creates a close pair C and D, assign p to C and attach D to C (perform a right-to-left-attach operation if D is to the right of C and otherwise a left-to-right-attach operation).

3. Else, if there exists a feasible cluster C for p such that assigning p to C creates a far pair C and D, define a new cluster P for p and attach it to D. (P, D form a close pair.)

4. Else, define a new cluster P for p. If there exists a cluster C such that P and C form a close pair, attach P to C.

For an illustration, see Figure 2. Our algorithm does not allow overlap between clusters (except for endpoints, and even that can only happen if two clusters are attached and fixed, or if a cluster is truncated at the point where another cluster begins). Our algorithm avoids the creation of close and far pairs. A close or far pair can be created if a request point is being assigned to a cluster C and thus making it closer to the closest cluster on the same side of C as the request point. If the point is indeed assigned to C then a single close pair may be created and this pair is fixed right away. Otherwise, if the point is finally assigned to a new cluster, this cluster may form a close pair with each one of two clusters on its both sides. This happens in Step 3 or in Step 4. If it happens in Step 3, it must form a close pair with each one of them, since in this case any fixed cluster is located too far. The algorithm fixes the new cluster with one of the two previously existing clusters.

Therefore, when a close pair or far pair appears, our algorithm immediately fixes at least one half of the pair, possibly leaving the other half unchanged. Thus if there were no close or far pairs before some request, they still do not occur afterwards. Note also that the creation of a new cluster P cannot create a far pair, since the new cluster consists of a single point at this time.

2.3 Analysis

We begin our analysis by proving that single online clusters cannot be "too close together" (Lemmas 1 and 2). We then prove a number of lemmas about the relationship between online clusters and optimal clusters, using the concept of a group (Definition 4). We consider the inside structure of a group and conclude that there are only two possibilities for this structure in the proof of Theorem 1. For both structures, we give an upper bound of $7/4$ on the competitive ratio, and conclude by showing that this ratio is tight.

2.3.1 Single clusters

Lemma 1 *There can be no interval of length 1 which contains two single clusters.*

Proof Suppose the two single clusters A and B are contained in an interval of length 1. Without loss of generality, denote by A the cluster that is defined earlier by the algorithm. Let b be the first request point in B. We consider the step in which b is assigned to a cluster. Since the point b fits in A (or in the cluster which is closest to b between b and A), it is not assigned by our algorithm in Step 4. In Steps 2 and 3, b is placed in a fixed cluster. In Step 1, b is placed in an existing cluster. But then B has more than one point. In all cases, we find a contradiction. □

Note that this lemma holds even if there are fixed clusters nearby. Specifically, the lemma shows that for two single clusters A and B that both contain only one request point, we have $d(A, B) > 1$.

In the following, we will repeatedly discuss sets of clusters, and denote the elements of such a set by $C_1, C_2, \ldots$, numbered from left to right. In such cases, denote the leftmost request point contained in C_i by ℓ_i and the rightmost request point by r_i.

Lemma 2 *Consider three consecutive single online clusters, C_1, C_2 and C_3. If there is an optimal cluster X which serves requests from all three clusters, then*

- *there exists a fixed cluster F which overlaps with the interval $(r_1 - 1, \ell_3 + 1)$,*
- *there is no single online cluster between F and C_j, where C_j is the cluster among C_1 and C_3 that is closest to F, and*
- *there exists a request point in C_j which is served by an optimal cluster Y which does not serve requests from any other single cluster.*

Proof Suppose there is no such fixed cluster F. The assumption implies that $d(r_1, \ell_3) \leq 1$. Let q be the oldest request point in C_2. If q is newer than r_1 and ℓ_3, C_1 and C_3 formed a close or far pair before q arrived, which our algorithm does not allow. Otherwise, without loss of generality, let r_1 be newer than ℓ_3. Then C_2 and C_3 form a close pair as soon as both q and ℓ_3 have arrived (since $d(\ell_2, \ell_3) \leq d(r_1, \ell_3) \leq 1$), which our algorithm also does not allow.

This proves the existence of the cluster F. Suppose that F is to the left of C_1. (The case where F is to the right of C_3 is symmetric.) By Lemma 1, C_1 and C_2 are not contained in an interval of length 1. This implies that the optimal cluster X which serves r_2 does not serve ℓ_1. By the same Lemma, there is no online single cluster between F and C_1 since $d(F, r_1) < 1$.

Consider the optimal cluster Y which serves ℓ_1. By these observations and the fact that Y does not overlap with X, we have that Y does not cover any point from any single cluster besides C_1 (possibly it covers some points of F). □

This Lemma shows that an optimal cluster X can only serve requests from three consecutive single clusters if these online clusters are the first or last three clusters in a sequence of consecutive single clusters (or the only three, of course).

Definition 4 *A* group *of online clusters is a maximal set of consecutive clusters $C_1, \ldots, C_m$ such that there is an optimal cluster which contains both r_i and ℓ_{i+1} for $i = 1, \ldots, m - 1$. (These optimal clusters are not necessarily all distinct.)*

If there is more than one group, for each group we have that the leftmost point of the leftmost online cluster is not to the right of the leftmost point of the leftmost optimal cluster by the way we construct our optimal solution.

Lemma 3 *For $m \geq 3$, at least $m-1$ offline clusters are needed to serve all the request points in m consecutive single clusters that are in one group.*

Proof Denote these single clusters by $C_1, \ldots, C_m$. For $m = 3$, even if there is an optimal cluster X which serves requests from all three single clusters, it cannot cover two of them completely by Lemma 1.

Suppose $m \geq 4$. Clearly, an optimal cluster cannot cover requests from four (or more) different online clusters $C_i, \ldots, C_{i+3}$, because then the two clusters C_{i+1} and C_{i+2} would have to be contained in an interval of length 1, which is impossible by Lemma 1. Lemma 2 shows that if an optimal cluster serves requests from three consecutive single clusters, then these are the clusters C_1, C_2, C_3 or C_{m-2}, C_{m-1}, C_m (or both), since there must be a fixed cluster immediately next to them on one side. Suppose it happens to the first three clusters (the other case is symmetric). Lemma 2 also shows that in this case, there is an optimal cluster Y which serves only requests from C_1. So whether this case occurs or not, the first (and last) three clusters are served by at least two optimal clusters. No other three consecutive clusters can be served by one optimal cluster by Lemma 2. We see that on average, at least one optimal cluster is required to serve requests from each two consecutive single online clusters. Since we have $m-1$ pairs of consecutive single clusters, the lemma is proved. □

2.3.2 Fixed clusters

Lemma 4 *In each pair of fixed clusters where none of the clusters is truncated, there is at least one optimal cluster which is completely contained inside the pair.*

Proof This follows immediately from the fact that in any pair of fixed clusters, the shared endpoint of these two clusters is an actual request point. If both fixed clusters are not truncated, each one of them has length 1. The claim follows from the fact that the optimal solution needs to serve the point in the middle. □

Lemma 5 *Not all requests in a pair of fixed clusters are served by a single optimal cluster.*

Proof We need to show that there is always a pair of request points which is more than 1 apart in a pair of fixed clusters. If the pair of fixed clusters is created in Step 2, denote the fixed pair by C and D, where C is to the left of D. We must have had $d(\ell_C, r_D) > 1$ before these clusters were fixed by Lemma 1, due to the following. These clusters both existed before they became close, and were single clusters. The request that caused them to be fixed appears between the previous positions of these clusters and does not change the left endpoint of C or the right endpoint of D, that are the endpoints of these clusters when they are still single. These two points are requests points since the endpoints of single clusters are always request points.

If a pair of fixed clusters is created in Step 3, suppose C is a feasible cluster for p, such that if p is assigned to C, C and D would become a far pair, and assume that C is to the left of D (the other case is symmetric). Then p would have become the rightmost point of C, and D is to the right of p. Moreover, since C and D do not become a close pair, we have $d(p, r_D) > 1$. At this time, D is single so r_D is a request point. The fixed clusters are D and a new cluster that contains p, so the claim holds in this case.

If a pair of fixed clusters is created in Step 4, if the new request point does not fit in the old single cluster C, which forms a close pair with the new cluster, then clearly the two clusters contain a pair of request points of distance more than 1. These two points are p and an endpoint of C (the one that is further away from p). □

Lemma 6 *If a fixed cluster T is truncated, there is an existing fixed cluster F within a distance of less than 1 of the newly fixed cluster T, and exactly one single cluster E between T and F. The clusters T and E have a shared endpoint. The algorithm does not create additional clusters between E and F.*

Proof We consider the steps of our algorithm separately. Without loss of generality, let p be the new request point, which is to the right of a cluster C in what follows.

Step 2: Let C be the cluster to which p is assigned. Assigning p to C affects r_C, but leaves ℓ_C unchanged. If this creates a new close pair, D must be to the right of p, and we have $d(p, r_D) \leq 1$. This shows that p could (in principle) also be assigned to D. The two fixed clusters if not truncated in this case are $[p-1, p]$ and $[p, p+1]$. There can be no fixed cluster which overlaps with $(p-1, p+1)$ by the definition of a close pair. So for each of the fixed clusters there may only be one single cluster that has an overlap with it (due to Lemma 1). We consider the case that there is such a single cluster E to the right of D. There must be a fixed cluster F which overlaps with $(\ell_E, \ell_E + 1)$, else D and E formed a close pair before p arrived. In this case, D is truncated, but all future requests between D and F can and will be assigned to E, since F is of distance less than 1 from ℓ_E.

Similarly, if a single cluster E to the left of C overlaps with $(p-1, p+1)$, there must be a fixed cluster F which overlaps with $(r_E - 1, r_E)$, and all future requests between F and C will be assigned to E, since F is of distance less than 1 from r_E.

Step 3: Let C be the feasible cluster for p and D the cluster with which C forms a far pair if p is assigned to C, and P the new cluster created for p. Since we would get a far pair, D must be to the right of p. Before assigning p, we have $d(r_C, \ell_D) > 1$, and $d(p, \ell_D) \leq 1$ because we get a far pair, but $d(\ell_C, p) \leq 1$ since C is feasible for p . Also, $d(p, r_D) > 1$ since otherwise assigning p to C would create a close pair. Thus p could not be assigned to D. There can be no cluster in the interval $[r_C, \ell_D]$ (before P is created) by the definitions of far pairs and feasible clusters; there is no fixed cluster overlapping with $(r_C - 1, \ell_D + 1)$, so if a cluster Y contained in $[r_C, \ell_D]$ existed, then its left endpoint cannot be in $[r_C, p]$, otherwise it creates a close or far pair with C, and its right endpoint cannot be in $[p, \ell_D]$, otherwise it creates a close or far pair with D, so Y cannot exist. So the new cluster $P = [\ell_D - 1, \ell_D]$ contains p and does not overlap with an existing cluster. There is also no fixed cluster in the interval $(r_D, \ell_D + 1]$ by the definition of a far pair. If there is a single cluster E which overlaps with $(r_D, \ell_D + 1]$, there must be a fixed cluster F which overlaps with $(\ell_E, \ell_E + 1)$ as in the proof for Step 2. All future requests between D and F will be assigned to E.

Step 4: If p is assigned to a new cluster P, and thus P and a cluster C become a close pair, $d(p, r_C) \leq 1$. If there is a single cluster D to the right of p that truncates the newly created fixed cluster $P = [r_C, r_C + 1]$, then automatically $d(r_C, \ell_D) \leq 1$. Since C and D were not a far pair until p arrived, there must be a fixed cluster F which overlaps with $(\ell_D, \ell_D + 1)$. In this case, all future requests between P and F will be assigned to D.

If C is truncated, there must be a single cluster E to the left of C and a fixed cluster F which overlaps with $(r_E - 1, r_E)$ as in the proof for Step 2. Any future requests between C and F are assigned to E. □

Note that when a fixed cluster T is truncated, both its endpoints are request points by Lemma 6. We consider T to be in the same group as both its neighbors, even if there exists an optimal cluster which only serves requests from one of these three clusters.

Lemma 7 *If a fixed cluster T is truncated, at least three optimal clusters are required to serve all request points in T and the two clusters with which T shares an endpoint.*

Proof T shares one endpoint with another fixed cluster F, and one with a single cluster E by Lemma 6. These three clusters used to be single clusters. Wlog, let the order of them by E, T, F from left to right. Denote the rightmost endpoint of T before it became fixed by t. If the requests in these three clusters are served by only two optimal clusters, one optimal cluster must serve t and all request points in F, since no optimal cluster can serve E and T entirely by Lemma 1. But if this were possible, then T and F would have formed a close pair already before the request p arrived which caused T and F to become fixed. (If they did not form a close pair then, it was because there was a fixed cluster nearby, and in this case the appearance of p would also not have made them close, so T and F would not have become fixed.)

Since our algorithm avoids the creation of close pairs, we have found a contradiction. □

Lemma 8 *Consider a group where no clusters are fixed. Our algorithm is 3/2-competitive on this group.*

Proof Consider a group consisting of the clusters $C_1, \ldots, C_m$. If $m = 1$, then since at least one optimal cluster is needed for this group we are done. If $m = 2$, then due to Lemma 1, which implies that no two single clusters can be completely served by a single optimal cluster, at least two optimal clusters are needed for this group. For $m \geq 3$, the statement follows immediately from Lemma 3. □

Theorem 1 *Our algorithm has a competitive ratio of 7/4.*

Proof By Lemma 5, the request points of a fixed pair are served by at least two optimal clusters. If they are served by three different optimal clusters, and both clusters in the pair are not truncated, we allow the optimal algorithm to move the leftmost of these clusters to the left until it no longer intersects the pair. This may mean that some request point is no longer served by the optimal algorithm, and thus can only make the competitive ratio higher. The request points that remain outside of optimal clusters are only points that the algorithm assigns to fixed clusters. From now on, we will derive bounds on the number of optimal clusters by considering only points that are assigned to single clusters by our online algorithm, and taking into account that any fixed pair is served by at least two optimal clusters. Thus our proof remains valid.

We consider the groups that exist after this shifting. Note that the endpoint of a fixed pair which is an inner point of its group (i.e., the left endpoint of the pair if this pair is on the right end of its group, and vice versa) is always covered by some optimal cluster.

For each fixed untruncated pair, by Lemma 4 there is an optimal cluster X which does not serve any request from any cluster outside the pair, and an optimal cluster Y which might. We are going to bound the number of optimal clusters needed to serve all the request points that are not in the fixed untruncated pair(s) at the end(s). We then add 2 to the online cost and 1 to the offline cost for each fixed pair (the cluster Y has already been counted).

Truncated clusters By our definition of groups, truncated clusters occur only in the middle of groups. By Lemma 6, if a cluster T is truncated, there is another (older) fixed cluster F within a distance of less than 1 of it, and one single cluster E between them. Also there is a fixed cluster G on the other side of T which shares an endpoint with it. We call three such clusters E, T, G a triplet. The cluster F is either part of the next triplet or one half of a pair of fixed untruncated clusters.

There might be a triplet E, T, G such that G is also truncated. In this case, there is a single cluster E' immediately next to G, followed by the next fixed cluster G' which is older than G and not truncated. In this case we call the set $\{E, T, G, E'\}$ a *quartet*. As above we have that G' is either part of the next triplet or one half of a pair of fixed untruncated clusters.

This leaves only two possibilities for the inside structure of a group (ignoring the possible fixed pairs at the ends):

- $(sTF)^k s^m (FTs)^\ell$, where $k \geq 0, m \geq 0, \ell \geq 0$
- $(sTF)^k (sTTs)(FTs)^\ell$, where $k \geq 0, \ell \geq 0$

In this list, s represents a single cluster, T is a truncated cluster, and F is an untruncated fixed cluster.

Group elements We see that we have three structural elements inside a group: triplets, quartets and sequences of single clusters. To calculate the number of optimal clusters required to serve the request points in such a group, we lower bound the number of optimal clusters for each element separately, going from left to right, and then add these together. Here we need to take into account that whenever we move from one element to the next, we need to subtract one from the optimal cost, because one optimal cluster gets counted double (once for each element).

By our results so far, we have the following table for the offline cost of each structural element.

Element	Contribution to online cost	Contribution to offline cost
Triplet	3	3 (Lemma 7)
Quartet	4	3 (Lemma 7)
One single cluster	1	1
Two single clusters	2	2 (Lemma 1)
$m \geq 3$ single clusters	m	$m - 1$ (Lemma 3)

We want to show an upper bound of $7/4$. There are only a few cases we need to check. To begin with, we only need to check groups with fixed untruncated pairs at both ends. We are going to add the optimal cost for each element, and subtract one from the optimal cost for each element beyond the first. Note that any triplet beyond the first does not help to show a competitive ratio above $3/2$, and that a sequence of single clusters cannot occur in combination with a quartet.

- No clusters in the group apart from the fixed untruncated pairs. If there is one fixed pair, we have a ratio of 1 by Lemma 5. Else, we have a ratio of at most $3/2$, again by using Lemma 5 (on both pairs), and noting that we are counting at most one optimal cluster double by Lemma 4.
- If the group starts with a quartet, the ratio is at most $8/5$ (two fixed pairs, there can be no sequence of single clusters in this group, any triplets decrease the competitive ratio).
- If the group contains at least one triplet followed by a quartet, the ratio is at most $11/7$ (two fixed pairs, one triplet, one quartet; no sequence of single clusters possible).
- Else, there is no quartet. If there is also no triplet, the ratio is at most $7/4$, given by $m = 3$ single clusters and two fixed pairs.
- If there is a triplet, we find a ratio of $7/5$ for $m = 0$ single clusters, $8/5$ for $m = 1$, $9/6$ for $m = 2$ and at most $10/7$ for $m \geq 3$.

Lower bound We can show that the analysis is tight using the following bad example for our algorithm (Table 2, Figure 3). The third column indicates the step of our algorithm in which the point is assigned.

We find that our algorithm creates 7 clusters, where it is possible to cover all requests by the following four clusters: $[0, 1], [2, 3], [4, 5], [6, 7]$. □

Point	Cluster	Step	Explanation
0	A	4	
1	A	1	The point fits in A.
3	B	4	
4	B	1	The point fits in B.
6	C	4	
7	C	1	The point fits in C.
2	D	2	D is attached to A.
			(this is a close pair, our algorithm does not distinguish between A and B).
5	E	2	E is attached to C
			(this is a close pair, our algorithm does not distinguish between B and C).
2.5	F	4	
4.5	G	4	

Table 2: Lower bound construction for our algorithm

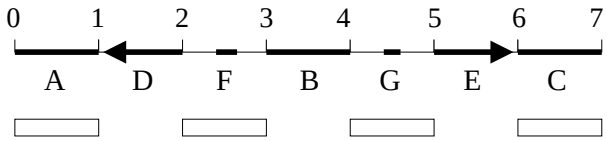

Figure 3: The lower bound for our algorithm. Arrows denote clusters which are attached to other clusters.

3 Lower bounds

3.1 One dimension

Theorem 2 *No deterministic algorithm can have a competitive ratio below* $8/5$.

We use the instance shown in Table 3. In Table 3, $\mathcal{A}(\sigma)$ is the cost of an online algorithm and $\text{OPT}(\sigma)$ is the current cost of the optimal solution for the instance σ up to now. In each row, "point" is the location of a new point. "Cluster" is the cluster it must belong to, where a new name means that a new cluster must be opened. See Figure 4. It can be seen that the construction results in a lower bound of $8/5$.

Theorem 3 *No randomized algorithm can have a competitive ratio below 3/2.*

Proof We use an adaptation of Yao's principle [7] for proving lower bounds for *randomized algorithms*. It states that a lower bound on the competitive ratio of deterministic algorithms using a fixed distribution on the input, is also a lower bound for randomized algorithms and its value is given by $\frac{\mathbb{E}(\mathcal{A}(\sigma))}{\text{OPT}(\sigma)}$.

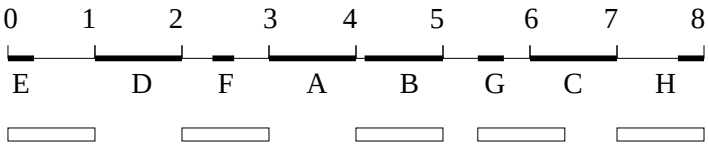

Figure 4: The general deterministic lower bound: at the top the online clusters (marked in bold), at the bottom the final optimal solution.

Point	Cluster	$\mathcal{A}(\sigma)$	$\text{OPT}(\sigma)$	Explanation
3	A	1	1	
4	A	1	1	Otherwise we get $\mathcal{A}(\sigma) = 2$ and $\text{OPT}(\sigma) = 1$.
5	B	2	2	
6	C	3	2	If the point is placed in B, the three new requests $2, 4.5, 7$ open three new clusters and give $\mathcal{A}(\sigma) = 5$, $\text{OPT}(\sigma) = 3$.
2	D	4	3	
1	D	4	3	If the point is not placed in D, then $\mathcal{A}(\sigma) = 5$, $\text{OPT}(\sigma) = 3$.
0	E	5	4	
2.5	F	6	4	
7	C	6	4	If the point is not placed in C, then $\mathcal{A}(\sigma) = 7$, $\text{OPT}(\sigma) = 4$.
4.1	B	6	4	Otherwise we get $\mathcal{A}(\sigma) = 7$.
5.5	G	7	5	
8	H	8	5	

Table 3: The general deterministic lower bound construction

We construct a sequence of points σ for which $\text{OPT}(\sigma) = 2$, with equal distances between points. However the exact location of points is given by a probability distribution. The sequence is given in two steps, where the first set of points is contained in an interval of length 1. In the second step, this interval is extended to have length slightly over 2, so that an optimal solution has cost 2. The algorithm does not know in advance where the border between the two clusters of an optimal solution will be.

Let N be a large integer. To simplify presentation, we apply scaling so that the length of a cluster is at most N instead of 1. We give requests only at integer points. The final set of requests is $2N + 2$ consecutive integer points, thus we can combine the first $N + 1$ into one cluster, and the remaining $N + 1$ into an additional cluster, and have $\text{OPT}(\sigma) = 2$. Specifically, the final set of points is the set $N - j, ..., 3N + 1 - j$ for some $j \in \{1, ..., N\}$ and this j is chosen uniformly at random. We would like to show that the expected number of clusters is $3 - \frac{1}{N}$, and thus the cost tends to 3 for $N \to \infty$.

The first part of the input is exactly $N + 1$ points, which are $N, N + 1, ..., 2N$. Consider the output of some deterministic algorithm at this time. If there are at least 3 clusters we are done, no matter which value of j is chosen. Otherwise, assume first that the points N and $2N$ belong to distinct clusters, A, B. Let B be the cluster of the rightmost point, that is, of point $2N$, and A of the leftmost point, that is, of point N. Let a be the rightmost point which is assigned to cluster A and let $b = a + 1 \leq 2N$. Clearly, b is assigned to B.

Given a fixed value of j, we claim that the online cost is 3 unless $j = 2N - a$. If $a < 2N - j$, then the point b was assigned to B and therefore, if the point $b + N + 1 = N + a + 2$ is introduced, a third cluster must be opened for it. This point exists if $3N + 1 - j \geq N + a + 2$ which holds if $a \leq 2N - j - 1$. On the other hand, if $a > 2n - j$, since a was assigned to cluster A, if the point $a - N - 1$ is introduced, a new cluster must be opened for it. This point exists if $N - j \leq a - N - 1$ or $a \geq 2N - j + 1$. For each value of $a \leq 2N - 1$, there is exactly one value of j for which the cost is 2. Thus, with probability $1/N$ the cost is at least 2 and otherwise it is at least 3. Thus the expected cost is at least $3 - 1/N$.

Now if N and $2N$ belong to a single cluster, then no point to the left of N and no point to the right of $2N$ can be assigned to the existing cluster. Moreover, if the input contains both points $2N + 2$ and $N - 1$, then two additional clusters are required. This situation occurs for all values of j and in this case, the expected cost is at least 3. □

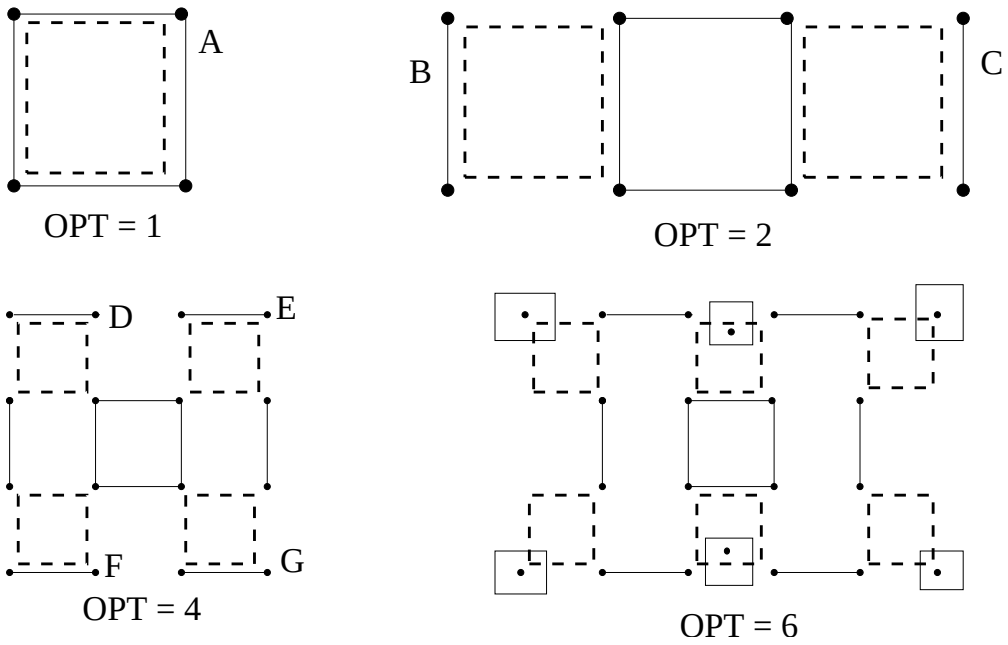

Figure 5: The deterministic lower bound in two dimensions: online clusters are represented by lines and boxes, an optimal solution for each case is represented by dashed boxes.

3.2 Two dimensions

Theorem 4 *No deterministic online algorithm can have a competitive ratio less than 2 in two dimensions.*

Proof The proof is illustrated in Figure 5. Consider an online algorithm $\mathcal{A}$, and assume by contradiction that it has a competitive ratio of less than 2. First, four points arrive on the corners of a unit square. $\mathcal{A}$ must assign them all to the same cluster A, otherwise, since there exists a feasible solution consisting of a single cluster, it has a ratio of (at least) 2 and the input stops. It can be seen that A cannot be assigned any further point outside this unit square.

Then, four additional points arrive, two to either side (top right of Figure 5), so that the input now consists of two rows of four points each, one above the other. $\mathcal{A}$ must open exactly one cluster for each new pair, since it is possible to cover all existing points with only two clusters. Both of these new clusters, B and C, cannot cover any request above or below them, since the points they contain are already of distance 1 apart from each other (vertically).

In the next phase, eight additional points arrive (bottom left of Figure 5). These are four new points above and below the previous points, so that the 16 points form a square, and the distance between every consecutive pair of points is 1, both vertically and horizontally. Since it is possible to serve all these points with only four clusters, $\mathcal{A}$ must open four new clusters for these points; less than four does not cover all the points, and more than four gives a competitive ratio of 2. It can be seen that these clusters D, E, F, G cannot serve any request which is to the left or to the right of them, since each cluster contains two points that are of distance 1 apart (horizontally).

Finally, six additional points arrive. Three of which are in the top row of points, between the two central points, and in distance 1 from the extremal points, and the other three are in the same positions in the bottom row. Algorithm $\mathcal{A}$ is forced to open six new clusters for them, since none of these points fit in an existing

cluster: they are to the side of D, E, F, and G, and above or below B and C (bottom right of Figure 5). Clearly, no two new points can be assigned to the same new cluster. Now $\mathcal{A}$ has opened 13 clusters in total while the optimal solution requires only six clusters. This is a contradiction and shows that $\mathcal{A}$ has a competitive ratio of at least 2. □

Note that this lower bound of 2 for two dimensions implies a lower bound of 2 for any higher dimension as well: we can let all the requests appear in a 2-dimensional subspace.

Theorem 5 *No randomized online algorithm can have a competitive ratio less than* $11/6$ *in two dimensions.*

Proof As in the randomized lower bound for one dimension, we scale the clusters so that they have size N, and requests occur only at integer points. Moreover, we apply Yao's method once again. There are four phases:

1. $(N+1)^2$ points: $\{N, \ldots, 2N\} \times \{N, \ldots, 2N\}$.
2. Choose an integer i uniformly at random, $0 \leq i \leq N-1$. In this phase $(N+1)^2$ points appear, so that the set of all points requested so far is now $\{N, \ldots, 2N\} \times \{i, \ldots, 2N+1+i\}$.
3. Choose an integer j uniformly at random, $0 \leq j \leq N-1$. In this phase $2(N+1)^2$ additional points appear in such a way that the set of points requested so far is $\{j, \ldots, 2N+1+j\} \times \{i, \ldots, 2N+1+i\}$.
4. Choose an integer k uniformly at random, $0 \leq k \leq N-1$. In this phase $2(N+1)^2$ final points appear in such a way that the set of points requested so far is $\{j, \ldots, 2N+1+j\} \times \{i-k-1, \ldots, 3N+1+i-k\}$.

Thus, the set of request points is first extended vertically, then horizontally, and finally vertically again. We will show that for $N \to \infty$, with high probability, $\mathcal{A}(\sigma) \geq 11$ for any deterministic online algorithm $\mathcal{A}$. It can be seen that the input σ can be covered using only six clusters. These clusters are defined by vertical lines trough the points $(j,0)$,$(N+j,0)$,$(N+j+1,0)$ and $(2N+j+1,0)$, and horizontal lines through the points $(0,i-k-1)$, $(0,N+i-k-1)$ $(0,N+i-k)$,$(0,2N+i-k)$,$(0,2N+i-k+1)$,$(0,3N+i-k+1)$.

We first focus on the last two phases in our construction. Consider the set of points $S_1 = \{(j,y) | y = i, \ldots, 2N+1+i\}$. Let C_1 be the cluster which contains (j,i). Let $p \leq i+N$ be the highest value such that (j,p) is in cluster C_1. Define the set $S_2 = \{(j,y) | y = i-k-1, \ldots, 3N+1+i-k\}$.

Claim: *With probability at least* $1 - 1/N$*, the online algorithm will require four clusters in the fourth phase to cover all the points in the set* S_2*.*

Proof of claim: Suppose $p = i+N$. The point $(j, i-1)$ is in S_2 for any $0 \leq k \leq N-1$, and this point cannot be in C_1, we call its cluster C_0. The points $(j, i+N+1)$, $(j, i+2N+2)$ are in S_2 for any $0 \leq k \leq N-1$ and cannot be covered by C_1 or by C_0, or even by the same cluster. Therefore, we get a total of at least four clusters.

If $p < i+N$, the online algorithm requires at least three clusters to cover only the points in S_1. This follows because the points $(j, i+N)$ and $(j, i+2N+1)$ cannot be covered by the same cluster, and are also not covered by C_1. If S_1 is covered by at least four clusters, we are done.

Otherwise, consider the cases where cluster C_1 cannot cover all points below (i,j). This happens if $i-k-1 < p-N$, which holds for $k \geq N+i-p$. In this case a fourth cluster is enforced since the clusters of $(j, i+N)$ and $(j, i+2N+1)$ cannot cover the point $(j, i-k-1)$ either. If $k \leq N+i-p-2$, let p_1 and p_2 be such that p_1 is the maximum value of y such that (j,y) is covered by the same cluster as $(j, p+1)$ and p_2 is the maximum value of y such that (j,y) is covered by the same cluster as (j, p_1+1).

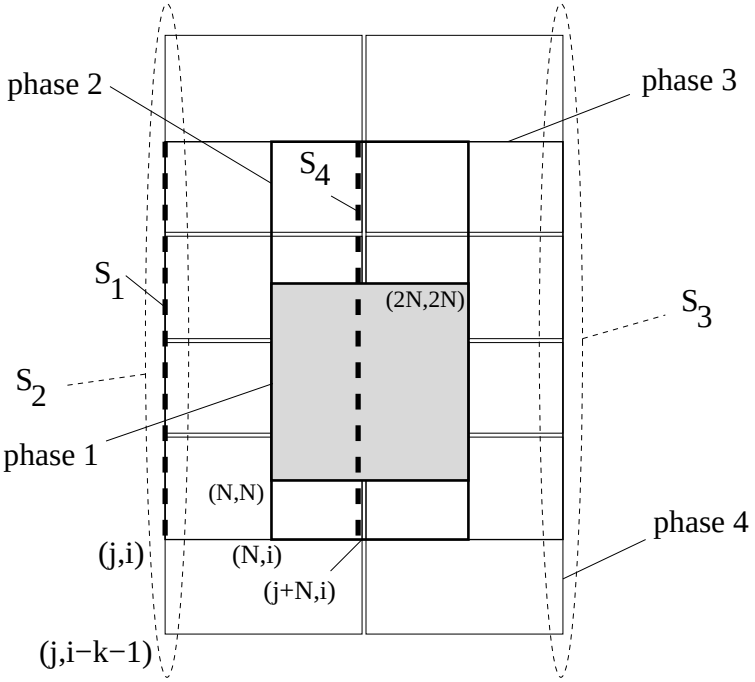

Figure 6: The randomized lower bound in two dimensions. The values i, j, and k are all chosen uniformly at random from the set $\{0, \ldots, N-1\}$. The sets $S_1, \ldots, S_5$ are defined in the proof.

Clearly, $p_1 \leq p + N + 1$ and $p_2 \leq p_1 + N + 1 \leq p + 2N + 2$. Consider the point $(j, p_2 + 1)$. This point is part of the input since $p_2 + 1 \leq p + 2N + 3 \leq (N + i - k - 2) + 2N + 3 = 3N + 1 + i - k$. This point requires a fourth cluster.

The only case that is not considered here is $k = N + i - p - 1$. In this case it is possible that there are only three clusters that cover S_2. However, since k is chosen uniformly at random from N possibilities, the probability that this happens is at most $1/N$. This proves the claim.

We can apply the same analysis to the point set $S_3 = \{(2N+1+j, y) | y = i-k-1, \ldots, 3N+1+i-k\}$, showing that with probability at least $1 - \frac{1}{N}$, four clusters are needed to cover this set as well. Note that there cannot be a cluster which contains points from both S_2 and S_3.

Finally, consider the set of points $S_4 = \{(j + N, y) | y = i, \ldots, 2N + 1 + i\}$. Note that the points in S_4 are requested already in the first two phases of our input sequence. An analysis as in the proof of Theorem 3 shows that with high probability, S_4 requires at least three clusters. We consider the clusters of S_4 and would like to show that with high probability, these are three clusters that are different from the eight clusters that we already found.

To show this, we consider the input after the first two phases. If already at this time, there are at least 11 clusters, we are done. Otherwise, there are at most ten clusters. We say that $N \leq x \leq 2N$ is a border of a cluster X if there exists a point (x, y) that the algorithm assigns to cluster X but no point (x', y') with $x < x' \leq 2N$ and $i \leq y' \leq 2N + 1 + i$ that is assigned to X exists. Consider the clusters that are used by the algorithm to cover S_4. Assume that a cluster C_4 is identical to one of the clusters found for S_2. Then $j + N$ is a border for C_4. Clearly, each cluster has one border. Since j is chosen uniformly at random such that $0 \leq j \leq N - 1$, the probability that $j + N$ is a border of C_4 is at most $\frac{1}{N}$. The probability that among

all (at most ten) clusters, at least one has $j + N$ as a border is at most $\frac{10}{N}$. Thus, with probability at least $1 - \frac{10}{N}$, the clusters of S_4 are all different from those of S_2. Clearly, they cannot be the same as these of S_3 (due to the distance).

Thus with high probability we find that the online algorithm requires at least 11 distinct clusters to cover all the requests in the input. Four for S_1, four for S_2, and three for S_4. □

4 Concluding remarks

This paper significantly improves the previously known bounds. However, many questions still remain open. Specifically, we would like to find out whether the competitive ratio grows with the dimension. Another unresolved issue is the relation between deterministic and randomized algorithms. It is known that for small dimensions ($d = 1, 2$), randomization does not help in the unit covering problem. However, we do not have clear evidence that this is the case for unit clustering as well.

Note that in this paper, the multi-dimensional variant of the problem is considered with respect to the L_∞ norm, as in [1]. This is done for simplicity, in order to make it possible to use the one-dimensional upper bound result as a black box in the design of an algorithm for multiple dimensions, rather than defining and analyzing an additional algorithm which is based on the same ideas. Note that our lower bound for two dimensions can be modified for the L_2 norm, by starting with a square that exactly fits in a unit ball.

References

[1] Timothy M. Chan and Hamid Zarrabi-Zadeh. A randomized algorithm for online unit clustering. In *Proc. 4th Workshop on Approximation and Online Algorithms (WAOA 2006)*, volume 4368 of *Lecture Notes in Computer Science*, pages 121–131. Springer, 2007. To appear in *Theory of Computing Systems* (doi:10.1007/s00224-007-9085-7).

[2] Moses Charikar, Chandra Chekuri, Tomás Feder, and Rajeev Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[3] A. Gyárfás and J. Lehel. On-line and first-fit colorings of graphs. *J. Graph Theory*, 12(2):217–227, 1988.

[4] H. A. Kierstead and William T. Trotter. An extremal problem in recursive combinatorics. *Congr. Numer.*, 33:143–153, 1981.

[5] Hal A. Kierstead. Coloring graphs on-line. In Amos Fiat and Gerhard J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*, pages 281–305. Springer, 1998.

[6] László Lovász, Michael E. Saks, and W. T. Trotter. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Math.*, 75(1–3):319–325, 1989.

[7] A. C. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th Symp. Foundations of Computer Science (FOCS)*, pages 222–227. IEEE, 1977.

[8] Hamid Zarrabi-Zadeh and Timothy M. Chan. An improved algorithm for online unit clustering. In *Proc. 13th Annual International Conference on Computing and Combinatorics (COCOON 2007)*, volume 4598 of *Lecture Notes in Computer Science*, pages 383–393. Springer, 2007. To appear in *Algorithmica*.