Competitive Access Time via Dynamic Storage Rearrangement

(Preliminary Version)

Amos Fiat* Yishay Mansour † Adi Rosén ‡ Orli Waarts §

Abstract

We model the problem of storing items in some warehouse (modeled as an undirected graph) where a server has to visit items over time, with the goal of minimizing the total distance traversed by the server. Special cases of this problem include the management of a real industrial stacker crane warehouse [G], automatic robot run warehouses [Ho], disk track optimization to minimize access time [GS, P, W, YW], managing two dimensional memory (bubble memory and mass storage systems) [KMW, W, YW], doubly linked list management, and the process migration problem.

The *static* version of this problem assumes some known probability distribution on the access patterns. The study of special cases of this static version was first considered by Hardy, Littlewood and Pólya in 1932 [HLP].

In this work we initiate the study of the *dynamic* version of the problem, where the robot may rearrange the warehouse to deal efficiently with future events. We require no statistical assumptions on the access pattern, and give *competitive* algorithms that rearrange the warehouse over time to deal efficiently with the true access patterns. We give non-trivial upper bounds for the general problem, along with some interesting lower bounds. In addition, we model realistic data access patterns on disk storage by considering two practically significant scenarios: access to some database via dynamically changing alternative indices and access patterns derived from root to leaf traversals of some (unknown) tree structure. In both cases we give greatly improved competitive ratios.

1 Introduction

1.1 Motivation

We seek to model the fundamental class of problems of storage management to minimize costs over online access to items. We present a very general model, sub-cases of which include disk seek optimization [GS, P, W, YW], optimization of robot run warehouses [Ho] and the stacker crane problem [G], organization of two dimensional memory (bubble memory and mass storage systems) [KMW, W, YW], the doubly linked list [ST85a] and the process migration problem.

The study of very special cases of this class of problems was initiated in [HLP] (the rearrangement problem) in 1932. [HLP] and following studies of this problem or specific subproblems have all assumed some known probability distribution on the access patterns. The goal of these studies has been to compute an optimal static arrangement of the items.

One immediate problem with the classic approach used above is determining the appropriate distribution. Moreover, even if one is given a true initial distribution, dynamic changes in the request pattern might occur naturally. Consider the problem of minimizing seek times on disk storage, one would expect an airline reservation system to have many more accesses to this week's flights than next months flights. While this structured change can be taken into account, in many other applications there are unpredictable changes. *E.g.*, the access pattern on a World Wide Web server can change dramatically over time. Thus, it is important to achieve a technique for organizing the data on the tracks/cylinders of the disk so as to minimize seek time, that will be able to adapt to changes in the request

*Department of Computer Science, Tel Aviv University, Tel Aviv. Research supported in part by a grant from the Israel Academy of Sciences. E-Mail: fiat@math.tau.ac.il

†Department of Computer Science, Tel Aviv University, Tel Aviv. This research was supported in part by The Israel Science Foundation administered by The Israel Academy of Science and Humanities and by a grant of the Israeli Ministry of Science and Technology. E-Mail: mansour@math.tau.ac.il

‡Department of Computer Science, Tel Aviv University, Tel Aviv. Research supported in part by a grant from the Israel Academy of Sciences. E-Mail: adiro@math.tau.ac.il

§Computer Science Division, U.C. Berkeley. Research supported in part by an NSF Postdoctoral Fellowship. E-Mail: waarts@cs.berkeley.edu

pattern, irrespective of how complex these changes are. Implicitly, we are assuming a "virtual memory" mechanism for translation between the application address and the physical address.

This leads us to defining the *dynamic rearrangement problem.*

1.2 The Dynamic Rearrangement Problem

Consider a graph $G = (V, E)$ of $|V| = n$ vertices and a universe I of n *items.*

Every vertex can store exactly one distinct object at any given time. Imagine a *server* located at some vertex in the graph.

We consider two types of on-line requests, Access and Skip. Different applications make use of different subsets of the following:

1. Access(i), for some item $i \in I$. The server moves to the current location of item i. Repeatedly, and at a cost of one per edge, the server can move from its current vertex along one edge to some adjacent vertex, and may optionally swap the contents of these two vertices at no extra charge. This request represents either true physical motion such as robot motion/disk head seek time or communication costs for remote interprocess communications via a network.

2. Skip(i), for some item $i \in I$. The server moves from its current position to the current location of item i. The charge associated with this action is zero. No items may be rearranged. This request is used to reflect a logical switch such as a process I/O request at a remote computer.

We assume the server knows the structure of the graph and knows where all items are currently located[1]. Requests arrive one at a time and the server has to serve a request immediately upon its arrival. The aim is to minimize the total cost to serve the sequence of requests.

1.3 Problems that Reduce to the Dynamic Rearrangement Problem

The number of fundamental problem modeled by the dynamic rearrangement problem is large. Here are some of the most important examples.

Warehouse Organization The dynamic rearrangement problem models a *warehouse organization problem,* where items denote bins of identical parts and a vertex denotes a specific shelf on which the bin is placed. A typical sequence for a robot run warehouse problem would consist of picking up several objects, determined on-line, and then moving them to some shipping site. While the dynamic relocation problem is defined above without the "shipping site" option, one can add it to the model and algorithm.

Stacker Crane The dynamic rearrangement problem models the stacker crane problem the same way as the warehouse because the work required to reach down into a stack of crates is proportional to the distance from the top to the crate in question.

Disk seek Optimization Choosing the graph to be a line graph models the disk problem, the items may be records or files, each of which is stored on one track (or cylinder in multi plate disks). The server is the read/write head of a disk and an Access request is a read or write request for a given record[2].

Doubly Linked List Another way to view the disk problem is that of maintaining a doubly linked list, where there is a single pointer (the server) that points to the current item. When viewed as managing a doubly linked list, our problem is a natural extension of the list management problem studied in [ST85a, IRSW91, Irani91].

Process Migration Another application is in distributed computing and deals with process migration. Assume that we have some network of processors, each of which can run exactly one process. Also, assume that processes communicate with each other by sending messages. We seek to minimize communications, adaptively dealing with changing communications patterns, and rearrange the processes in the processors over time so that communications are minimized. Here, every communications request from process p to process q is modeled by a Skip(p), Access(q), pair of requests.

1.4 Our Results

We evaluate the performance of our on-line algorithms in terms of the *competitive ratio* introduced by Sleator and Tarjan [ST85a], which is the supremum, over all possible request sequences, of the ratio between the cost achieved by the on-line algorithm and the cost of the optimal algorithm for this specific request sequence.

Not a single one of the above sub cases of the problem has a known nontrivial competitive solution. Even

[1] This is not really required if the graph is a line (the disk problem), as using techniques of [BCR] we can perform a doubling search.

[2] One argument against this model is that seek times obviously grow with the distance between source and target, but this growth is not necessarily linear (the same argument holds for all previous statistical studies). Variants of our results will hold for sub-linear functions (which are not "too small") and we can also introduce the added complexities of distinguishing between sub-linear seek times versus linear rearrangement times.

if we restrict the graph to be a line, the problem is surprisingly hard. Thus, the major result of the paper is providing a non-trivial upper bound for the dynamic rearrangement problem. In particular,

1 For arbitrary graphs and arbitrary request sequences we give an algorithm with a competitive ratio of $O(n^{1-\epsilon})$, where $\epsilon \approx 1/29$.

The algorithm is natural and deterministic. Its analysis involves several very novel techniques.

Our starting point was by restricting the request sequence. This yielded the following, second, contribution of the paper:

2 For arbitrary graphs, if we restrict the request sequence to be only access requests with an access pattern that is a series of permutations on the items, (*i.e.* we have rounds and in every round there are n requests, one request for every item in the universe) then we get a competitive ratio of $O(D^{6/7} \log n)$, where D is the diameter of the graph.

The proof of this result is nontrivial, and involves a few of the novel ideas used by the more general main result. In addition, as elaborated in the full paper, the permutation case is interesting on its own right.

As a third and fourth contributions of the paper, we give two nontrivial lower bounds, valid even for the line graph, that indicate that there is indeed an inherent difference in the complexity of maintaining a singly linked list versus a doubly linked list.

3 The first lower bound is for the setting where the adversary algorithm is restricted to hold a single fixed configuration (also known as a *static adversary* [ST85b]). We show a lower bound of

$$\Omega(\sqrt{\log n}/\log\log n)$$

on the competitive ratio of any on-line algorithm against such a static adversary.

4 For the more general dynamic adversary, usually considered in competitive analysis, we give a lower bound of $\Omega(\log n)$ on the competitive ratio.

Note that the first lower bound states that the on-line algorithm cannot "approximate" the best static adversary configuration well.

Motivated by the above lower bounds, and by the locality of reference model for paging of [BIRS91], we then consider the question of limited adversary sequences, to reflect more typical real life applications. As noted above, the permutations case can be considered as such a real life application. Another typical case is root-to-subtree traversal of some tree structure and thus we consider access patterns that are generated by such root-to-subtree traversals. This leads to the fifth and possibly quite practical result:

5 For a line graph (*i.e.* the disk seek problem), and access patterns generated by root-to-subtree traversals of some (unknown) tree[3], and subject to the constraint that the tree root is located at some fixed location, we give $O(1)$ competitive algorithms.

All our algorithms are deterministic.

Section 2 presents the general algorithm for arbitrary graphs and Access and Skip commands. Section 3 describes a simpler variant of the general algorithm for arbitrary graphs when the access patterns form a sequence of permutations. The algorithm and its analysis constitute the starting point for the general algorithm. A very high level description of the required changes in the analysis for the general problem is given in Section 4. In Section 5 we present some of the lower bounds. Section 6 deals with the simplified version of the general problem where the graph is a line-graph and the requests are limited to being root-to-subtree traversals of some tree structure.

For lack of space, in most of this abstract we consider sequences of requests that include Access requests alone (access sequences). Extending the description/analysis to include Skip commands is straightforward.

2 The Dynamic Rearrangement Algorithm

In this section we describe the algorithm for arbitrary graphs and arbitrary sequences of Access and Skip requests.

We define a round to be a sequence of n Access requests, possibly inter-spaced with Skip requests. We can ignore all Access requests that cost the on-line algorithm nothing, *i.e.*, Access to the item currently in the same vertex as the server.

We define an interval to be a sequence of $r = n^d$ rounds ($d = 9/29$). We further partition a round to consist of $c = n^{2\delta}$ periods, ($\delta = 1/29$), each consisting of n/c Access requests possibly inter-spaced with Skip requests.

We define a tree-adversary to be an adversary that deals optimally with a sequence of requests, but is con-

[3] We assume that given the request sequence there exists a tree T such that the sequence can be divided to subsequences each starting with the root, and every other request in the subsequence is the son, in T, of the previous request in the subsequence.

strained to move along the edges of some spanning tree of the graph G. Given a graph and a sequence of requests, the tree-adversary may choose both some spanning tree of the graph and the initial configuration (the arrangement of items in the vertices). The tree-adversary chooses a spanning tree and an initial configuration so as to minimize the cost of the sequence when starting from the initial configuration, constrained to moving only along the edges of the spanning tree, with the server initially at any vertex of it's choice. The tree-adversary is free to rearrange the items in the graph during the course of dealing with the event sequence but is constrained to swapping items along edges of the spanning tree when the server traverses these edges.

It follows from ([AKPW91], Theorem 4.1, 4.2) that the cost of an optimal tree adversary to deal with a sequence of requests is no larger than $\exp(O(\sqrt{\log n \log\log n}))$ times the cost of an optimal (non-constrained) adversary to deal with the same sequence. While [AKPW91] make use of their construction to give a *randomized* k-server algorithm, we use these theorems in a different fashion and our dynamic rearrangement algorithm for general graphs is deterministic.

Our algorithm (called DR for Dynamic-Rearrangement), restarts following every interval.

In Figure 1 we define the algorithm for one such interval.

Theorem 2.1 *DR is $O(n^{1-\epsilon})$ competitive for the dynamic rearrangement problem, $\epsilon \approx 1/29$.*

3 Sequence of Requests is a Series of Access-Permutations

This section considers the case where the sequence of requests is a series of Access requests where every item is requested by exactly one of the first n Access requests (requests $1, \ldots, n$), and requested by exactly one of the second batch of n Access requests (requests $n+1, \ldots, 2n$), and so on. We call every such batch a *round*. We use the permutation analysis as a step towards the general case but it is also of independent interest.

The algorithm is $O(D^{6/7} \log n)$-competitive, where D is the diameter of the graph.

3.1 The π-DR Algorithm

The π-DR algorithm is a simplification of the DR algorithm presented above (Figure 1). It does not reorganize the items during a round, does not partition a round into periods, has no use for the T_i trees, and does not make use of tree-adversaries, it considers only unconstrained adversaries.

We consider here π-DR only for Access requests. As for DR, we partition the sequence of requests into intervals, each of which consists of D^d consecutive rounds (d will be determined later). (Recall that in this context a round is n consecutive Access requests which form a permutation.) Our algorithm, follows the rounds and intervals structure as follows.

Focus on a certain interval. Within a round π-DR serves the requests without changing the arrangement of the items. A round in which π-DR's cost for serving the requests contained in this round exceeds some bound $B = 21nD^{1-\delta}\log n$ is called a *bad round* (δ will be determined later).

We give the bad rounds sequence numbers such that bad round number i is preceded by exactly $i-1$ bad rounds in this interval. Let $\textsc{opt}(i)$ be some optimal algorithm for serving the requests issued from the start of the interval and until the end of the i'th bad round, starting with an initial configuration of its choosing. (Ties are broken arbitrarily.) At the end of bad round number i, π-DR rearranges the items into a configuration identical to the initial configuration of $\textsc{opt}(i)$.

3.2 Analysis of Algorithm π-DR

We will prove the competitive ratio of π-DR for any sequence of requests that forms a single interval; this clearly implies a bound for any sequence. Thus, from now on we focus on a single interval.

First note that the maximum cost that π-DR can incur in an interval is $O(D^d nD) = O(nD^{1+d})$. Therefore, if the adversary cost for the same interval is at least $\Omega(nD^{d+\epsilon})$, we have a competitive ratio of $O(D^{1-\epsilon})$ (ϵ will be determined later). On the other hand, even if the adversary cost is less than $nD^{d+\epsilon}$, it still has a cost of at least one for every request, therefore at least nD^d per interval. The main idea in the algorithm and its analysis is our ability to bound the number of bad rounds in an interval, under the assumption that the adversary cost in the interval is less than $nD^{d+\epsilon}$ (*i.e.* on the average the adversary pays at most D^ϵ per request), and thus give an upper bound on the competitive ratio.

3.2.1 Bounding the Number of Bad Rounds

This section obtains a bound on the number of bad rounds in intervals in which the adversary cost is at most $nD^{d+\epsilon}$. We proceed in two steps. The first step (Lemma 3.1) shows that in any bad round, say bad round number i, we can find a "large" set of pairs of items, such that every item appears in at most one pair,

Algorithm *DR*: Input: Graph $G = (V, E)$, $|V| = n$.
Throughout this algorithm we ignore zero-cost Access requests.
Set T_1 to be an arbitrary initial spanning tree;
Set C_1 to be an arbitrary initial configuration
(the mapping of the items to the vertices).
Set $i = 1$.

Repeat $r = n^d$ times {for r rounds, $d = 9/29$ } :

Repeat $c = n^{2\delta}$ times {for c periods, $\delta = 1/29$}:

Arrange the items in G according to C_i;

Relocate the server to service the first request, Skip or Access:
For Skip, the server travels along the shortest path in G
between its current location and the new vertex. But it may perform no
rearrangement as it moves along.

Repeat until $t - 1 = n^{1-2\delta} - 1$ Access requests have been serviced:

If the request is Skip(j):
Move the server to the vertex currently containing item j. { At zero cost.}

If the request is $Access(j)$:
Consider the unique simple path $P \subset T_i$ between the vertex currently occupied
by the server, u, to the vertex currently containing item j, (v).
Move both the item currently at u, say j', and item j
as close as possible to the midpoint between their current
positions along P. (they become adjacent); {Cost $\approx 1.5|P|$.}
Server ends up at new vertex containing j approximately at midpoint of P.

END Repeat;
END Repeat ;

If the total cost in this round (number i) $> B = b \cdot n^{2-\delta} \log n$ (For some sufficiently large constant b)
Set (T_{i+1}, C_{i+1}) to be a spanning tree and initial arrangement, respectively,
chosen by an optimal tree-adversary on G given the event sequence
consisting of rounds $1, \ldots, i$.
If not, set (T_{i+1}, C_{i+1}) to be (T_i, C_i).
$i := i + 1$;

END Repeat ;

Figure 1: The DR Algorithm in an interval.

and the two elements in a pair are "very distant" on the configuration held by π-DR in bad round number i, but are "near" on any configuration held by it in any subsequent (bad) round in the interval.

The second step (Lemma 3.2) uses the existence of a "large" set of such pairs to prove that the number of bad rounds is "small", as follows. For every item we keep a set of items that have to be "near" it in all future rounds in this interval. Since the items in every pair we find are "very distant" from each other on the configuration at bad round i, and the items in the set of each item must be "near" it, the respective sets of the two items are disjoint. Thus, for every such pair we can show that for at least one of the items the size of its set is doubled. This implies that in the entire interval we can have at most $O(n \log n)$ such pairs. Given that every bad round provides many such pairs, the bound on the number of bad rounds will follow.

We start with a few definitions:

Definitions:

1. For two items s_1 and s_2, the *distance between* s_1 *and* s_2 on configuration C, is the length of the shortest path in G between the vertices on which s_1 and s_2 reside, according to C. We denote this distance by $d_C(s_1, s_2)$.

2. Two items s_1, and s_2 are said to be *very distant* on configuration C if $d_C(s_1, s_2) \geq 3D^{\epsilon+2d+\delta} \log n$.

3. The ϵ-*neighborhood* of item s on configuration C is the set of items $\{s' : d_C(s, s') \leq D^{\epsilon+2d+\delta}\}$.

4. The $k \cdot \epsilon$-*neighborhood* of item s on configuration C is the set of items $\{s' : d_C(s, s') \leq k \cdot D^{\epsilon+2d+\delta}\}$.

5. We say that a set of pairs of items is a *chosen set for bad round number* i, if it satisfies the following properties:

   (a) Every item is contained in at most one of the pairs of items in the set.

   (b) For every pair in the set, the two items contained in the pair are very distant on the configuration π-DR uses for the round.

   (c) For every pair in the set, the two items contained in the pair are in the ϵ-neighborhood of each other on the configurations held by π-DR in any of the subsequent rounds.

The following lemma shows that for every bad round there is a large chosen set of pairs of items.

Lemma 3.1 *If the adversary cost in the interval is at most* $nD^{d+\epsilon}$*, then for every bad round number* i*, there exists a chosen set of pairs of items,* C_i *such that*

$$|C_i| \geq 6nD^{-\delta} \log n \ ,$$

where $\epsilon + 2\delta + 2d \leq 1$.

Proof. Recall that we consider a bad round, i.e., a round in which the on-line cost is at least

$$B = 21nD^{1-\delta} \log n.$$

If the sequence of requests is $\mathcal{S} = s_{i_1} \ldots s_{i_n}$, and the configuration is C, then the cost of π-DR is

$$\alpha = \sum_{j=2}^{n} d_C(s_{i_{j-1}}, s_{i_j}).$$

Denote by $\mathcal{F}$ the set of indices j such that

$$d_C(s_{i_{j-1}}, s_{i_j}) \geq 3D^{\epsilon+2d+\delta} \log n.$$

Since this is a bad round we have that $21nD^{1-\delta} \log n \leq \alpha \leq |\mathcal{F}|D + 3nD^{\epsilon+2d+\delta} \log n$. Since $1 - \delta \geq \epsilon + 2d + \delta$ then $|\mathcal{F}| \geq (21-3)nD^{-\delta} \log n$.

We build a set of pairs of items, by choosing for every index in $\mathcal{F}$ the two items $(s_{i_{j-1}}, s_{i_j})$. We then take a subset such that no item appears twice. Since the sequence of requests is a permutation, the number of pairs left is at least $9nD^{-\delta} \log n$. Thus the pairs in the set we have picked satisfy the first two properties of the chosen set.

To satisfy the third property recall that the configurations held by the on-line algorithm are the initial configurations of optimal algorithms that serve the request sequence up to the time of the rearrangement. Consider one such algorithm:

Consider a set of pairs of consecutive requests, such that every item appears at most once in the pairs. Consider a pair, and the two items appearing in it. Such two items are *distant* on the initial configuration of the optimal algorithm if their distance on this configuration is more than $D^{\epsilon+2d+\delta}$.

Claim 3.1 *The number of distant pairs in such a set is at most* $3nD^{-d-\delta}$.

Proof. At a cost of a single unit the algorithm can move once and swap at most two items, thus it can improve its performance on at most three requests. Thus, the cost of any algorithm to serve the requests presented in the set of pairs is at least 1/3 the sum of distance between the items in the pair, on the initial configuration of the algorithm. Since the total cost of the optimal algorithm is at most $nD^{d+\epsilon}$ we can conclude that the sum of distance is at most $3nD^{d+\epsilon}$. Thus at most $3nD^{d+\epsilon}/D^{\epsilon+2d+\delta} = 3nD^{-d-\delta}$ pairs can be distant. □

Given such a set of pairs of requests, we call a pair *corrupted* if the distance between the two items in the pair on the initial configuration of *at least one* future configuration of π-DR, is more than $D^{\epsilon+2d+\delta}$.

Recall that the on-line algorithm changes its configuration only upon a bad round, and then moves to the initial configuration of an optimal algorithm that serves the requests (up to this time) optimally. Thus the number of different configurations used by the on-line algorithm is bounded by D^d. Every one of those can contribute at most $3nD^{-d-\delta}$ pairs to the set of corrupted pairs, by Claim 3.1. Thus, there are at most $3nD^{-d-\delta}D^d = 3nD^{-\delta}$ *corrupted* pairs.

Consider the set of consecutive pairs that satisfy the first two properties, we now eliminate from this set *all* corrupted pairs. We are left with at least $9nD^{-\delta}\log n - 3nD^{-\delta}$ pairs, which is at least $6nD^{-\delta}\log n$. The third property of the chosen set is now satisfied since the items in any remaining pair are in the ϵ-neighborhood of each other on the initial configuration of *any* optimal algorithm whose initial configuration we use.

We have thus built a set of chosen pairs of size at least $6nD^{-\delta}\log n$. □

Based on the above Lemma, we can give a bound on the number of bad rounds as follows.

Lemma 3.2 *If the adversary cost in the interval is at most $nD^{d+\epsilon}$, then the number of bad rounds in the interval is at most D^δ.*

Proof. We construct a two dimensional table in which every item has a row. We will show that a filled entry in column k will hold a set of at least 2^{k-1} items. Thus, every row can have at most $\lceil \log n \rceil$ filled entries; and hence the whole table contains at most $n\lceil \log n \rceil$ filled entries. We will be able to add a new such entry for every pair in the set of chosen pairs, and hence by Lemma 3.1, we fill $6nD^{-\delta}\log n$ new entries per bad round. Thus the maximum number of bad rounds is at most $n\lceil \log n \rceil / 6nD^{-\delta}\log n \leq D^\delta$, for $n \geq 2$.

We first describe how we build the table. For the first bad round we fill entries in the table using the set of chosen pairs of this round: For every such pair (p, q), we fill the first entry of q with the set $\{p\}$, and the first entry of q with the set $\{p\}$. Clearly these satisfy the property of the entries stated above.

Inductively we fill new entries for every bad round. Consider bad round number i, and assume we have already updated the table according to the first $i-1$ bad rounds. Consider the set of chosen pairs of bad round number i, and a pair (p, q). Let k_p and k_q be the largest numbered column filled in the rows of p and q respectively. Without loss of generality assume that $k_q \leq k_p$. We fill the k_q+1 entry of q with a set which is the union of the sets of items in the k_q entry of q and the k_q entry of p. Thus we add a new entry in the table for every chosen pair.

In the rest of the proof we prove that in the kth column the sets are of size at least 2^{k-1}. We prove that by induction on k. For $k = 1$ this is clearly true. To prove the induction step we prove that the two sets that are joined to create a new entry are disjoint. We start with the following claim.

Claim 3.2 *If item c is in the k_a entry of item a and the k_a entry of item a was filled in before bad round i, then items c and a are in the $k_a \cdot \epsilon$-neighborhood of each other on the configuration we maintain in bad round i, i.e.* $\text{OPT}(i-1)$.

Proof. By the construction of the table, this implies that there is a sequence of k items, say items $e_1, \ldots, e_k$, such that (i) $k \leq k_a + 1$; (ii) $e_1 = a$; (iii) $e_k = c$; and (iv) for $j = 1, \ldots, k-1$, the pair e_j, e_{j+1} is a chosen pair of items in some bad round that was previous to the ith bad round.

Recall that $\text{OPT}(i-1)$ is the configuration held by π-DR during bad round i. By the third part of the chosen set property, the two items in a chosen pair of bad round j, where $j < i$, must be in the ϵ-neighborhood of each other in $\text{OPT}(i-1)$. Thus, for all $j = 1, \ldots, k-1$, items e_j, e_{j+1} must be in the ϵ-neighborhood of each other in $\text{OPT}(i-1)$. Hence, items a and c must be in the $k_a \cdot \epsilon$-neighborhood of each other in $\text{OPT}(i-1)$. □

Now, by the properties of the set of chosen pairs, p and q are *very distant* on the configuration used by the on-line algorithm, therefore (considering the values of "very distant" and $k_p \cdot \epsilon$-neighborhood and $k_q \cdot \epsilon$-neighborhood) the two joined sets are disjoint. □

3.2.2 The Competitive Ratio

Theorem 3.1 *π-DR is $O(D^{6/7}\log n)$-competitive for the rearrangement problem on a general graph, where D is the diameter of the graph.*

Proof. As argued at the beginning of the section, if the adversary cost is at least $nD^{d+\epsilon}$, we have a competitive ratio of $D^{1-\epsilon}$.

If the adversary cost is at most $nD^{d+\epsilon}$, then Lemma 3.2 implies that the number of bad rounds is at most D^δ. Therefore the on-line cost is at most $O(D^\delta nD + D^d nD^{1-\delta}\log n)$. Since the adversary cost is at least $D^d n$, the competitive ratio in the interval is at most $O(D^{1+\delta-d} + D^{1-\delta}\log n)$.

To optimize the competitive ratio we should thus minimize $D^{1-\epsilon}; D^{1-\delta}\log n; D^{1+\delta-d}$, under the requirements that $\epsilon + 2d + 2\delta \leq 1$. If we set $\epsilon = 1/7, \delta = 1/7$, and $d = 2/7$, we get a competitive ratio of $O(D^{6/7}\log n)$. □

4 Analysis of the DR Algorithm.

One obvious advantage for the π-DR algorithm is that it is guaranteed that the input consists of consecutive permutations. Obviously, π-DR will fail miserably if this is not true. π-DR changes configurations at most once every n requests. It suffices that the adversary repeatedly request the furthest pair of items on the configuration that π-DR is (temporarily) committed to holding. This leads to a lower bound on the competitive ratio of the diameter of the graph.

Because of [AKPW91] it suffices to consider a tree-adversary without losing much in the competitive ratio. To argue that the DR algorithm is $O(n^{1-\epsilon})$ competitive against a tree-adversary, we need analogues to Lemmata 3.1 and 3.2.

Analogously to the definition of a bad round above, define an *expensive round* as a round that costs the online algorithm at least $b \cdot n^{2-\delta} \log n$. As there are $n^{2\delta}$ periods per round, it follows that in an expensive round there must be some *expensive* period that costs $b \cdot n^{2-3\delta} \log n$.

Lemma 4.1 plays the same role in the analysis of DR as Lemma 3.1 plays in the analysis of π-DR:

Lemma 4.1 *For every expensive period there exists a set S of pairs of requests for items, all items distinct, of size $m = \Omega(n^{1-8\delta} \log n)$, such that*

1. *Every such requested pair of items is "very distant" on the online configuration at the beginning of the period. (Here, very distant means at least $\Omega(n^{\epsilon+2d+8\delta} \log n)$ items apart on the on-line tree of the period.)*
2. *Every algorithm that serves the requests included in S, in the order they arrive, starting from some initial configuration C, must pay at least $1/6$'th of the sum of distances between the paired items on C.*

The analogue to Lemma 3.2 is

Lemma 4.2 *If the adversary cost in the interval is at most $n^{1+d+\epsilon}$, then the number of expensive rounds in the interval is at most $n^{8\delta}$.*

The proof of Lemma 4.1 for a tree graph and general sequences is much more complex than the proof of Lemma 3.1 where we assume that the request sequence consists of successive permutations.

To do so we make use of a so-called "token game" used to trace the positions of the various items in the course of an expensive period.

5 Lower Bounds

We first prove a lower bound of

$$\Omega(\sqrt{\log n}/ \log\log n)$$

on the competitive ratio of any on-line deterministic algorithm for managing a doubly linked list, even against an adversary that maintains the same configuration during the whole sequence of requests, i.e., never rearranges its items.

Regard the n items, numbered 0 to $n-1$, as the vertices of a $(\log n)$-dimension hypercube, i.e., an edge exists between two items if and only if their numbers, in the binary representation, differ in exactly one bit. We say that requests are *restricted to a hypercube*, if after a request for item u, the next request is restricted to one of the neighbors of u in the hypercube. We first show an upper bound on the cost of the off line adversary, for any such sequence of requests.

Lemma 5.1 *For any sequence of m requests that are restricted to a hypercube, there is a linear embedding of the items such that the total cost of accessing the items in the order in which they are requested is bounded by $O(mn/\log n)$.*

We prove the lower bound by showing that the adversary can continuously present to the on-line algorithm a sequence of requests σ on which the on-line algorithm will pay

$$\Omega(\frac{n}{\sqrt{\log n}\log\log n})$$

per request on the average.

Definition 5.1 *For every two vertices of the hypercube v, u, $d(u, v)$ is the distance in hypercube-edges between v and u, i.e., the Hamming distance between v and u in their binary representation.*

We regard the configuration held by the on-line algorithm as an embedding π of the hypercube vertices, on the line.

Definition 5.2 *Given some embedding π, and two vertices v, u, $d_\pi(u, v)$ is the distance on the line-graph graph between u and v, according to the embedding π.*

Lemma 5.2 *Let π be any embedding held by the on-line algorithm. There exists a sequence*

$$v_1, u_1, v_2, u_2, \ldots, v_k, u_k, \quad k = \Omega(\log n),$$

of vertices, such that for every i,

$$d_\pi(v_i, u_i) = \Omega(\frac{n}{\sqrt{\log n}}),$$

in π, while in the hypercube, $d(v_i, u_i) = 1$, and for every i, j, $d(v_i, v_j) = O(\log\log n)$.

Combining the last two lemmata gives us the following:

Theorem 5.1 *Any on-line algorithm for the rearrangement problem has a competitive ratio of at least*

$$\Omega(\sqrt{\log n}/\log\log n),$$

even against a static adversary.

Now we state the lower bound for regular (dynamic) adversaries:

Theorem 5.2 *Any on-line algorithm for the rearrangement problem has a competitive ratio of at least* $\Omega(\log n)$.

6 Access Request Patterns Generated from Tree Traversal on Line-Graph

This section describes a result that is considerably simpler than the rest of the results in the paper, but seems to be of immediate practical value. We consider a line-graph (the disk seek problem) and sequences of requests that are generated by root to subtree paths on some tree.

We assume that the adversary stores the root of the tree at some fixed location, and that all access requests are the result of some root to subtree path traversal. Let $\pi(x)$ denote the address of item x. We define the algorithm `PUSH`:

Let r denote the item in the root of the tree. The algorithm starts with an embedding π which is some embedding of the tree by levels, that is, first the root r, then the items at depth 1, etc. As a consequence, $\pi(r) = 1$, and if x is a descendent of y in the tree then $\pi(x) \geq \pi(y)$.

Upon a request to some item $v \neq r$, the algorithm simply moves the server to v. Upon a request to item r (the root), let v be the last item requested (our pointer currently points to v). Let $r = v_0, v_1, \ldots, v_k = v$ denote the sequence of items along the path in the tree from r to v (all of which must have been requested in succession up to v).

In response to the request r, algorithm `PUSH` starts moving back to location 1, dragging item v back with it (by swapping items) until it reaches item v_{k-1}. This process then continues with item v_{k-1} until item v_{k-2} is reached, and so on, until item v_1 is placed adjacent to the root r, i.e., $\pi(v_1) = 2$.

Theorem 6.1 *Algorithm* `PUSH` *is* $O(1)$ *competitive versus an adversary that holds the root of the tree at a fixed location.*

Proof. The rearrangements performed by PUSH maintain the invariant that if π is the embedding held by PUSH, then $\pi(r) = 1$, and if x is a descendant of y then $\pi(x) \geq \pi(y)$.

We partition the request sequence, as algorithm PUSH does, into blocks, each ending at a request for the root r. Clearly PUSH incurs a cost of at most $2n$ is any such block, thus restricting our attention to the times of the end of those blocks does not effect the competitive ratio.

Let v be the item requested just before the root is requested again. As a result of the invariant maintained by the algorithm, the cost incurred by PUSH during the whole block until it reaches v is $O(\pi(v))$. Note that the rearrangement procedure (and going back to r) just doubles the cost.

We prove that PUSH has a constant competitive-ratio against an adversary which holds the root of the tree at a fixed location. We first prove a constant competitive ratio against an adversary A that maintains the root r at address 1.

Following Sleator and Tarjan [ST85a], we define for every pair of items $x \neq y$ the *relative order function* $\phi_{x,y}$:

$$\phi_{x,y} = \begin{cases} 0 & \text{if the relative order of } x \text{ and } y \text{ in the configuration held by PUSH and in the adversary configuration are the same.} \\ 1 & \text{otherwise.} \end{cases}$$

We use the following potential function, used previously in [ST85a]:

$$\Phi = 2\sum_{x\neq y} \phi_{x,y} \ .$$

Consider a block in which the last request before the root is to item v. Let $C_P(v)$ denote the cost PUSH pays in the block until it reaches v. Let $C_A(v)$ denote the adversary cost for accessing v and all predecessors to v in the path from the root to v (this is the block we consider).

Lemma 6.1 *For every block of requests such that the last request before the root is request* v, $C_P(v) + \Delta\Phi \leq 6C_A(v)$.

Proof. Let the request sequence leading to v be

$$r = v_0, v_1, \ldots, v_k = v,$$

as defined above, define $\mathcal{V} = \{v_j\}_{j=0}^{k}$.

Let π and π^*, be the embeddings held by PUSH and A, respectively, just before the request block starts. We first analyze the change in the potential function due to the rearrangement done by PUSH, as if no action is taken by A, and then consider A's actions.

During this procedure all items z, such that $1 \leq \pi(z) \leq \pi(v)$, except the items in $\mathcal{V}$, change their relative order with exactly one item (the one they are swapped with). Let X denote the set of items that have a relative order function changed from 0 to 1, and let Y denote the set of those items that have a relative order function change from 1 to 0. No other changes in the potential function occur during the rearrangement procedure, therefore $\Delta\Phi = 2|X| - 2|Y|$.

Let $v_m \in \mathcal{V}$ be such that $\pi^*(v_m) = \max_{v \in \mathcal{V}}(\pi^*(v))$. Clearly, $C_A(v) \geq \pi^*(v_m)$.

Since for every item $x \in X$ there is an item $v_i \in \mathcal{V}$ such that ϕ_{x,v_i} changes from 0 to 1, by changing π such that $\pi(x) > \pi(v_i)$, we know that $\pi^*(x) < \pi^*(v_i) < \pi^*(v_m)$. Thus, the adversary must hold all items in X and all v_i, $i \neq m$, before v_m. Therefore $|X| + k \leq \pi^*(v_m) \leq C_A(v)$.

We get $C_P(v) + \Delta\Phi = 2\pi(v) + 2|X| - 2|Y|$. Since $|X| + |Y| + k = \pi(v)$,

$$C_P(v) + \Delta\Phi = 4|X| + 2k \leq 4\pi^*(v_m) \leq 4C_A(v).$$

To add A's possible swaps we notice that Φ can increase by at most 2 per unit cost of A. Therefore, $C_P(v) + \Delta\Phi \leq 6C_A(v)$. □

We now eliminate the restriction that the adversary keeps the root of the tree at location 1. Assume the adversary A keeps the root at some location s. We show that an adversary that keeps the root at location 1 only increases its costs by a constant factor. One can simulate the actions taken by A simply be mapping addresses in the range $s+1 \ldots 2s-1$ to even addresses and addresses in the range $1 \ldots s-1$ to odd addresses. Addresses $\geq 2s$ get mapped to sequential addresses greater than $2s$. This only increases the cost by a constant factor. □

References

[AKPW91] N. Alon, R.M. Karp, D. Peleg, and D.B. West. Graph-Theoretic Game, and its Applications to the k-Server Problem. In *Proc. of DIMACS Workshop on On-line Algorithms*, 1991.

[B] P.P. Bergmans, "Minimizing expected travel time on geometrical patterns by optimal probability rearrangements", Information and Control, 20 (1972), pp. 331–350.

[BBKTW90] S. Ben-David, A. Borodin, R.M. Karp, G. Tárdos, and A. Wigderson. On the Power of Randomization in On-line Algorithms. In *Proc. of the 22nd Ann. ACM Symp. on Theory of Computing*, pages 379-386, May 1990.

[BCR] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the Plane. *Information and Computation*, 1990

[BIRS91] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. "Competitive Paging with Locality of Reference". In Proceedings of the 23rd Ann. ACM Symp. on Theory of Computing, pages 249-259, May 1991.

[BLS87] A. Borodin, N. Linial, and M. Saks. "An Optimal On-Line Algorithm for Metrical Task Systems". In *Proc. of the 19th Ann. ACM Symp on Theory of Computing*, pages 373–382, May 1987.

[G] M. Groetschel, "Theoretical and Practical Aspects of Combinatorial Problem Solving", Invited Talk, Proceedings of 3rd annual ACM-SIAM symposium on discrete algorithms, pp. 195, 1992.

[GS] D.D. Grossman and H.F. Silverman, "Placement of records on a secondary storage device to minimize access time", JACM, 20 (1973), pp. 429–438.

[HLP] G.H. Hardy, J.E. Littlewood, and G. Pólya, *Inequalities*, Cambridge University Press, Cambridge, England, 1952.

[Ho] O. Hounkonnou, Modelling and simulation of an automated warehouse in activity. Math. Comp. Model. 1990, vol. 14, pp. 1100-1102.

[IRSW91] S. Irani, N. Reingold, D. Sleator and J. Westbrook. "Randomized Competitive Algorithms for the List Update Problem". In *Proceedings of the 2nd ACM Symposium on Discrete Algorithms*, 1991.

[Irani91] S. Irani. "Two results on the list update problem". *Information Processing Letters* 38, pp. 301–306, 1991.

[KMW] R.M. Karp, A.C. McKellar, and C.K. Wong, "Near-Optimal solutions to a 2-dimensional placement problem". SIAM J. Comp., Vol. 4, No. 3, 1975, pp. 271–286.

[P] V.R. Pratt, "An $N \log N$ algorithm to distribute N records optimally in a sequential access file", Complexity of Computer Computations, R.E. Miller and J.W. Thatcher, eds., Plenum Press, New York, 1972, pp. 111–118.

[ST85a] D.D. Sleator and R.E. Tarjan. "Amortized Efficiency of List Update and Paging Rules". In *Communications of the ACM*, 28(2) pages 202-208, 1985.

[ST85b] D.D. Sleator and R.E. Tarjan. "Self-Adjusting Binary Search Trees". *Journal of the ACM*, 32:652–686, 1985.

[Stam84] John W. Stamos. "Static grouping of small objects to enhance performance of a paged virtual memory". In *ACM Transactions on Computer Systems*, volume 2, number 2, pages 155–180, May 1984.

[W] C.K. Wong, "Minimizing Expected Head Movement in One-Dimensional and Two-Dimensional Mass Storage Systems", Computing Surveys, Vol. 12, No. 2, June 1980, pp. 167–178.

[YW] P.C. Yue and C.K. Wong, "On the optimality of the probability ranking scheme in storage applications". JACM, 20 (1973), pp. 624–633.