Online Algorithms for Warehouse Management

Philip Dasler
Department of Computer Science, University of Maryland, College Park
daslerpc@cs.umd.edu

David M. Mount
Department of Computer Science, University of Maryland, College Park
mount@umd.edu

Abstract

As the prevalence of E-commerce continues to grow, the efficient operation of warehouses and fulfillment centers is becoming increasingly important. To this end, many such warehouses are adding automation in order to help streamline operations, drive down costs, and increase overall efficiency. The introduction of automation comes with the opportunity for new theoretical models and computational problems with which to better understand and optimize such systems.

These systems often maintain a warehouse of standardized portable storage units, which are stored and retrieved by robotic workers. In general, there are two principal issues in optimizing such a system: where in the warehouse each storage unit should be located and how best to retrieve them. These two concerns naturally go hand-in-hand, but are further complicated by the unknown request frequencies of stored products. Analogous to virtual-memory systems, the more popular and oft-requested an item is, the more efficient its retrieval should be. In this paper, we propose a theoretical model for organizing portable storage units in a warehouse subject to an online sequence of access requests. We consider two formulations, depending on whether there is a single access point or multiple access points. We present algorithms that are $O(1)$-competitive with respect to an optimal algorithm. In the case of a single access point, our solution is also asymptotically optimal with respect to density.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Warehouse management, online algorithms, competitive analysis, robotics

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.59

Funding *David M. Mount*: Research supported by NSF grant CCF–1618866.

Keywords: Warehouse management, online algorithms, competitive analysis, robotics

1 Introduction

Online shopping has grown rapidly in recent years and, as such, the efficiency of the warehouses and fulfillment centers that support it plays an increasingly important role. Several companies have developed automated systems to help streamline operations in these warehouses, drive down the costs of order fulfillment, and increase overall efficiency. The introduction of automation comes with the opportunity for new theoretical models and computational problems with which to better understand and optimize such systems.

These systems often maintain a warehouse of standardized portable storage units, which are stored and retrieved by robots [12, 14]. For example, Amazon's Kiva robots and Alibaba's Quicktron robots help to streamline the order-fulfillment process. The Amazon robots are 16 inches tall, weigh almost 145 kilograms, can travel at 5 mph, and carry a payload weighing up to 317 kilograms. These robots maneuver themselves under standardized shelving units, lift them from below, and carry them to a location in the warehouse where a human waits to complete an order with items from the shelf.

The frequency with which each storage unit is accessed varies, and so, intuitively, units that are accessed more often should be placed closer to the access points than those that are less frequently accessed. As access probabilities vary over time, there is a natural question of how to dynamically organize the warehouse's placement of storage units in order to guarantee efficient access at any time. In this paper we will develop a simple computational model for a "self-organizing warehouse," and we present online algorithms for solving them. We demonstrate that our algorithms are competitive with optimal algorithms in our model. Our work can be viewed as a geometric variant of online algorithms for self-organizing lists and virtual memory management systems [1, 19].

From a practical perspective there are many ways in which to model objects residing in a warehouse. In order to obtain meaningful theoretical results without imposing irrelevant technical details, we propose a very simple and general model, which encapsulates the most salient aspects of efficient self-organizing behavior. We model storage units, or *boxes*, as movable unit squares on a grid in the plane. In addition to the boxes, there are designated fixed points, called *access points*, where boxes are brought on demand. The input consists of a sequence of *access requests*, each specifying that a particular box in the system be moved to a given access point.

There are two natural ways in which to move boxes in a planar setting, picking them up (like cargo containers by an overhead crane) and sliding them along the ground (like the aforementioned robotic systems). The former is simpler to describe and analyze. The latter is more realistic and is consistent with other motion-planning models [11, 10]. Another issue is the geometrical configuration of the warehouse and the locations of the access points. We present clean and simple models based on infinite and semi-infinite grids and show how to generalize our solutions to rectangular warehouses.

We consider two versions of the problem: the *attic problem*, where there is a single access point and the *warehouse problem*, where there are multiple access points. In each version and for each motion type, we present an online algorithm that is competitive with respect to an optimal solution that has knowledge of the entire access sequence. Details of the problem formulations and results are given in the next section.

1.1 Model and Results

We model a warehouse as a rectangular subset Ω of $\mathbb{Z}^2$, the square grid in the plane. Throughout, distances are measured in the ℓ_1 metric (the sum of absolute differences in x and y coordinates). We are given a finite set $A = \{a_1, \ldots, a_m\} \subseteq \Omega$ of stationary *access points* and a (significantly larger) finite set $B = \{b_1, \ldots, b_n\}$ of portable storage units, called *boxes*. Each box is a unit square. At any time, its lower left corner coincides with a grid point in Ω, called its *location*. A point of Ω that contains a box is said to be *occupied*, and otherwise it is *unoccupied*. No two boxes may occupy the same location at the same time.

The initial layout of the boxes is specified in the input. This is followed by a sequence of *access requests*, each being a pair (b, a), which involves moving box $b \in B$ from its current location to access point $a \in A$. Access requests are processed *sequentially*, meaning that each request is completed before the next one is started. Since the access point may already be occupied, it will be necessary to reorganize box locations. This reorganization should be performed with care, keeping frequently accessed boxes near the access point and moving less frequently accessed boxes to the periphery. The challenge is that we do not know the future access sequence, and yet we wish to be competitive with an optimal algorithm that does.

In general, the reorganization following each access request will involve a sequence of box movements. The box at the access point is displaced to a nearby location, the box at this location is then displaced to a new location, and so on. This chain of box movements continues until the last box in the chain arrives at an unoccupied square of the grid, possibly the original location of the requested box. More formally, let p_0 denote the original location of b, and let p_1 denote the location of a. If a is not occupied, b is simply moved here, and we are done. Otherwise, the algorithm determines a chain $p_2, \ldots, p_k$ of locations, where $p_2, \ldots, p_{k-1}$ are occupied and p_k is unoccupied (see Fig. 1(a)). (Note that p_0 is considered to be unoccupied, because its box has been moved to the access point.) We call this a *reorganization chain*. If $p_k \neq p_0$, this is an *open chain* (see Fig. 1(b)), and otherwise it is a *closed chain* (see Fig. 1(c)).

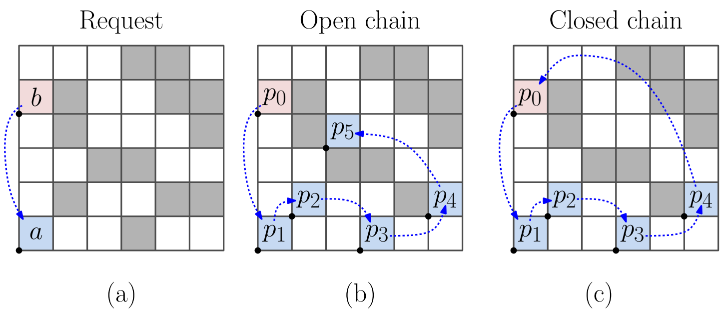

Figure 1 Processing a request.

For the sake of presenting our algorithms, it will be useful to describe the relocation process in terms of a sequence of *motion primitives*. In the case where boxes can be picked up (as by an overhead crane), the primitive operation is a *swap*, which exchanges the contents of two grid squares. The *cost* of the operation is the ℓ_1 distance between the two locations. The aforementioned reorganization involving a chain $\langle p_0, \ldots, p_k \rangle$ (whether open or closed) can be executed by swapping boxes in reverse order along the chain, that is, $p_k \leftrightarrow p_{k-1} \leftrightarrow \cdots \leftrightarrow p_0$ (see Fig. 2(a)).

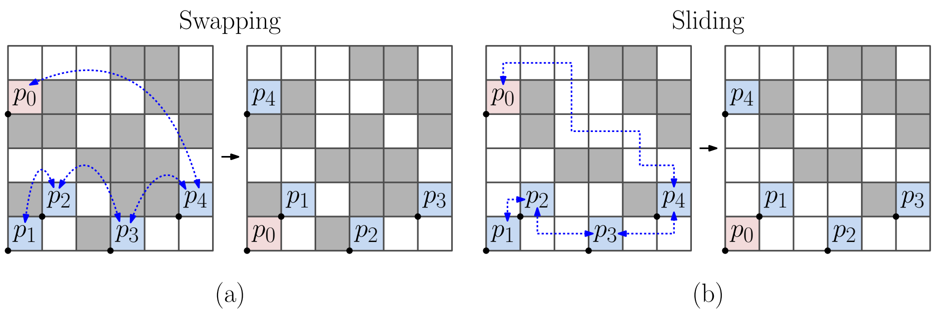

Figure 2 Motion primitives.

Alternatively, when boxes are moved along the ground the associated primitive operation is called *sliding*. As with swapping, the contents of two grid locations are swapped, but the boxes are moved along a rectilinear path of unoccupied grid locations (see Fig. 2(b)). The *cost* of the operation is the ℓ_1 length of the path, which may generally be much higher than the ℓ_1 distance between the two locations.

Sliding motion is more relevant in contexts where the boxes are being moved by robots, but it is complicated by the need to create empty space in which to move boxes. Our solutions will be based on first presenting a simple swapping-based solution and then showing how to adapt this to sliding motion without significantly increasing the cost. These two primitives provide a conceptually clear and simple model of motion costs. Of course, in practice, many other realistic issues would need to be considered.

Our problem formulations involve a *problem instance*, which consists of a specification of the domain Ω and the locations of the m access points A. An input to a given instance consists of the initial locations of the n boxes followed by a sequence S of access requests. For each access request, the output consists of the sequence $\langle p_0, \ldots, p_k \rangle$ along which motion primitives are applied (either swapping or sliding, depending on the model). Since our focus is on reorganization strategies, we ignore a number of issues needed for a complete model, such as how to coordinate the movement of multiple robots. We focus on two versions of the problem depending on the number of access points (see Fig. 3):

Attic Problem: Ω is an axis-aligned rectangle containing a single access point.

Warehouse Problem: Ω is an axis-aligned rectangle with access points on its bottom side.

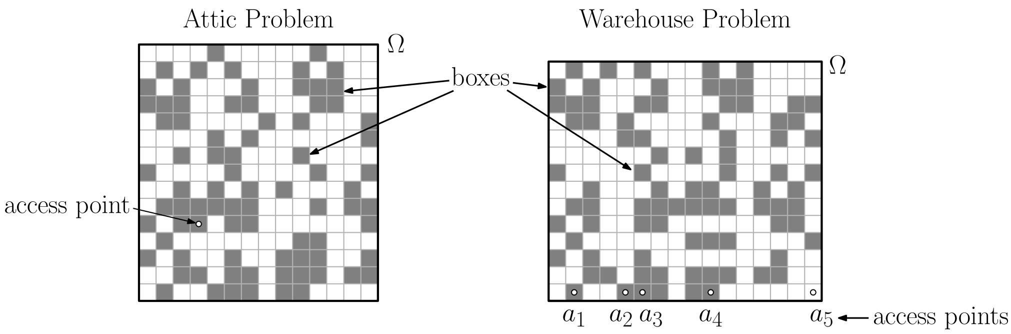

Figure 3 Problem versions.

We consider the above problems in an *online* setting, which means that each access request is processed without knowledge of future requests. In contrast, in an *offline* setting the entire access sequence is known in advance. An online algorithm is said to be *c-competitive* for a constant $c \geq 1$, called the *competitive ratio*, if for all sufficiently long access sequences S, the total cost of this algorithm is at most a factor c larger than the cost of an optimal offline solution for the same sequence. We say that an algorithm is *competitive* if it is c-competitive for some constant c, independent of m, n, the size of the domain, and the length of the access sequence. (The competitive ratios that result from our analyses are relatively high, and we suspect that they are far from tight. Reducing them will involve establishing better lower bounds on the optimum algorithm, and this seems to be quite challenging.) The notion of "sufficiently long access sequence" allows us to ignore start-up issues, such as the initial locations of the boxes.

Our main results are competitive online algorithms for these two problems in both the swapping- and sliding-motion models (presented in Theorems 1, 9, 10, and 12). Our result for the attic problem has the additional feature of being asymptotically optimal with respect to box density. (The precise definition will be given in Section 2.3.) These online algorithms exploit an intriguing connection between these problems and the task of maintaining hierarchical memory systems [1]. Hierarchical memory systems are linear in nature, and the geometric context of the our problems introduces novel challenges, since the reorganization must take into account the 2-dimensional locations of the boxes. Also, when sliding is involved, it is necessary to manage the set of unoccupied squares to guarantee short access paths.

1.2 Prior Work

There have been a number of papers devoted to the problem of organizing storage units in warehouses. Much of the prior work has focused on solving the various engineering challenges involved.

For example, Amato *et al.* [2] study control algorithms for the warehouse robots, assuming a continuous distribution of item locations throughout the warehouse and ignoring the benefits of intelligent item placement. In a similar vein, Chang *et al.* [5] attempt to minimize unnecessary task repetition using genetic algorithms, thus shortening robot travel times, but assume a fixed storage scheme regardless of differing access frequencies. Sarrafzadeh and Maddila [17] use a discrete grid-based model, as we will, but their focus is still an engineering one, concerned primarily with robot path-finding and constructing clearings through which to move. Closer to our work, Pang and Chan [16] address the question of where certain items should be stored in the warehouse, proposing a data-mining approach to determine the relationships between products and co-locating those that are often purchased together. Experimental analysis shows that their methodology outperforms a simple greedy policy, but they do not present any proofs on the performance of their approach.

The word "warehouse" has been used for various optimization problems. In the context of operations research, the *warehouse problem* was proposed by Cahn [3] and later refined and extended by Charnes and Cooper [6] and Wolsey and Yaman [20]. This work may sound related to ours, but its focus is on the logistics of managing a warehouse's stock in the face of changing demand. The word is also used in the context of coordinated motion planning under the name of the *warehouseman's problem*. This is a multi-agent motion planning problem amidst obstacles. It has been shown to be PSPACE-hard [11, 10], but efficient solutions exist for restricted versions (see, e.g., [18]).

While our approach is theoretical in nature, we avoid the high complexity of the warehouseman's problem by restricting shapes of boxes (to unit squares) and the allowed layout of boxes (by introducing additional empty working space throughout to facilitate easy motion). The problems we study are less focused on motion planning and more on how to organize the warehouse's contents to ensure efficient processing of a series of access requests.

More closely related to our work is the *dial-a-ride problem* [7]. In this problem, a set of users must be conveyed from source locations to specified destinations in a metric space. The goal is to plan a route (or routes, in the case of multiple vehicles or the more general k-server problem [13]) that satisfies all transportation requests while minimizing total distance traveled. One key difference is that the source locations are fully specified by the problem input, whereas in the warehouse problem the location of requested boxes can be adjusted according to need, and how best to do so is central to the problem.

As mentioned earlier, our work is similar in spirit to online algorithms for self-organizing memory structures [1, 19]. Another example is the work of Fekete and Hoffmann [8], who consider the online problem of packing variously sized squares into a dynamically sized square container.

2 Online Solution to the Attic Problem

In this section we present an online algorithm for the attic problem (single access point). We will show that the resulting scheme is competitive with respect to an optimal algorithm. As mentioned above, we exploit ideas from hierarchical memory systems. In such systems, memory consists of objects called *pages*, which are organized into blocks, called *caches*. Successive caches have higher storage capacity but higher access times. A common method for organizing such memory structures involves a block-based version of the least-recently used (LRU) policy, called *Block-LRU* of Aggarwal *et al.* [1]. In this policy, whenever a page is accessed it is brought to the lowest level cache, and the page that has resided in this cache for the longest time is evicted to the next higher level cache. The process is repeated until reaching the lowest cache that has space to hold this page, possibly the cache that contained the originally requested page. We next describe how the Block-LRU algorithm can be adapted to our geometric setting.

2.1 Hierarchical Model

In hierarchical memory systems, the cost of accessing an object is purely a function of each cache's speed. In our geometric context, the cost depends on the total cost of the motion primitives, which depends on the ℓ_1 distances between the locations of the boxes in the reorganization chain. The principal challenge is adapting the cache-based cost to the geometric setting. Our approach to the attic problem is based on surrounding the access point by collection of nested regions, called *containers*. Analogous to caches in the hierarchical memory systems, containers that are closer to the access point provide faster access but have lower storage capacity compared with those farther out.

It will simplify matters to describe the solution first for the infinite grid. We define a *hierarchical model*, which is based on an infinite sequence of nested *containers*, $C_0, C_1, \ldots$, where C_0 consists only of the origin (the access point), and for $k \geq 1$, C_k consists of the points of $\mathbb{Z}^2$ that whose ℓ_1 distance from the origin varies from $2^{k-1} + 1$ to 2^k (see Fig. 4 below). Whenever a box b is requested, it is first moved to the access point, and then a series of *evictions* takes place, where, for $k = 0, 1, \ldots$ a box from container C_k is moved to container C_{k+1}. The precise manner in which this is done for swapping and sliding motions is explained in Sections 2.2 and 2.3, respectively.

2.2 Online Algorithm for Swapping Motion

In this section we present an online algorithm solving the attic problem in the case of swapping motion, called Block-LRU$_A$. Consider a request for a box b. If the access point is unoccupied, we simply move the box there. Otherwise, in order to make space for b, we evict the least-recently accessed box from C_0, C_1, and so on until we encounter the first container C_k that has at least one unoccupied location (including possibly b's location at the time of the request). More formally, let p_b denote b's location, let p_0 denote the access point (origin), and let $p_1, \ldots, p_{k-1}$ denote the locations of the least-recently used boxes of containers C_1 through C_{k-1}, respectively. Finally, let $p_k \in C_k$ denote the final unoccupied location (possibly the former location of b). As described in Section 1.1, we achieve this by performing swaps in reverse order $p_k \leftrightarrow p_{k-1} \leftrightarrow \cdots \leftrightarrow p_0 \leftrightarrow p_b$ (see Fig. 4(a)). The cost is the sum of the ℓ_1 distances between consecutive pairs.

In order to apply this for a rectangular domain Ω, we simply clip the boundary of the containers at the limits of Ω (see Fig. 4(b)). We show next that this is competitive.

▶ **Theorem 1.** *For any instance of the attic problem and any sufficiently long access sequence R, the cost of Block-LRU$_A(S)$ is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Due to space limitations, the full proof and competitive analysis appear in Appendix A.1. In essence, the containers are treated as the caches of a memory hierarchy and then the standard LRU analysis of [19] and the Block-LRU analysis of [1] are adapted to our case.

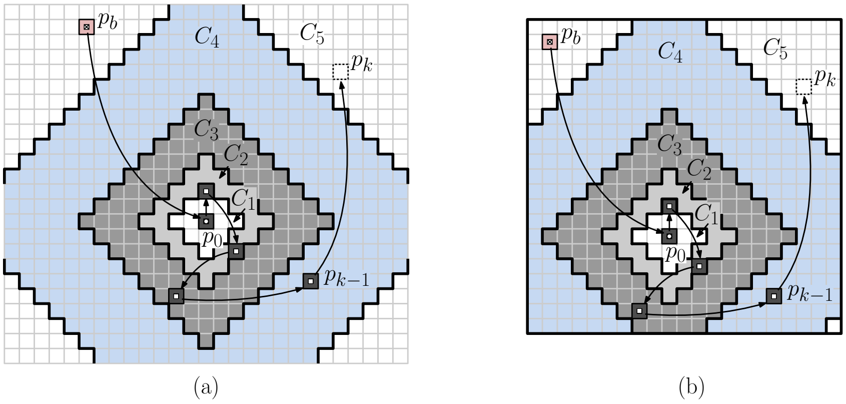

Figure 4 (a) Nested containers for the attic problem and (b) restriction to a rectangular domain.

2.3 Online Algorithm for Sliding Motion

In order to accommodate the added constraints involved in sliding boxes around the space, we constrain the manner in which boxes are arranged throughout the domain in order to retrieve them efficiently. An obvious solution would be to arrange the boxes in rows connected by empty corridors, as in typical warehouses. However, this is not efficient asymptotically, because it implies that the number of unoccupied squares in any region of space is at least a constant fraction of the available space. We will adopt a more space-efficient approach by packing distant boxes more densely. While these distant boxes will require more cost to access, this cost can be amortized against the cost incurred by their distance from the access point.

To make this formal, we define a *layout scheme* to be a subset of the integer grid $\mathbb{Z}^2$, which we will think of as a subset of the unit squares. For each integer s, define $n(s)$ to be the number of squares of the layout that lie within an $s \times s$ square that is centered about origin. Define the *asymptotic density* to be the limiting ratio of the fraction of squares in the layout lying within such origin-centered squares, that is, $\lim_{s\to\infty} n(s)/s^2$. For example, the layout that places boxes at every point of the grid has an optimal asymptotic density of 1, and a layout that places boxes only on the white squares of an infinite chessboard has an asymptotic density of 1/2.

In this section, we describe a layout that achieves the optimal asymptotic density of 1 and show how to convert our swapping-based Block-LRU$_A$ algorithm to the sliding context at the expense of an additional constant factor in cost.

2.3.1 The Nicomachus Layout

Out layout scheme inspired by a well-known visual proof of Nicomachus's Theorem [15], which is shown in Fig. 5(a).[1] The grid is partitioned into expanding concentric *rings* of square regions, denoted $r_1, r_2, \ldots$. The innermost ring, r_1, consists of 4 unit squares. Ring r_2 consists of eight copies of a 2×2 square region surrounding r_1. In general, r_k consists of

[1] Nicomachus's Theorem states that $\sum_{k=1}^{n} k^3 = \left(\sum_{k=1}^{n} k\right)^2$. If both sides of the equation are multiplied by 4, the layout of Fig. 5(a) provides a proof, where the left side arises by summing the number of blocks ring-by-ring (the kth ring has $4k$ blocks, each with k^2 squares) and the right side comes from the overall area (since the side length of the nth ring is $n(n+1) = 2\sum_{k=1}^{n} k$).

$4k$ copies of a $k \times k$ square region surrounding r_{k-1}.

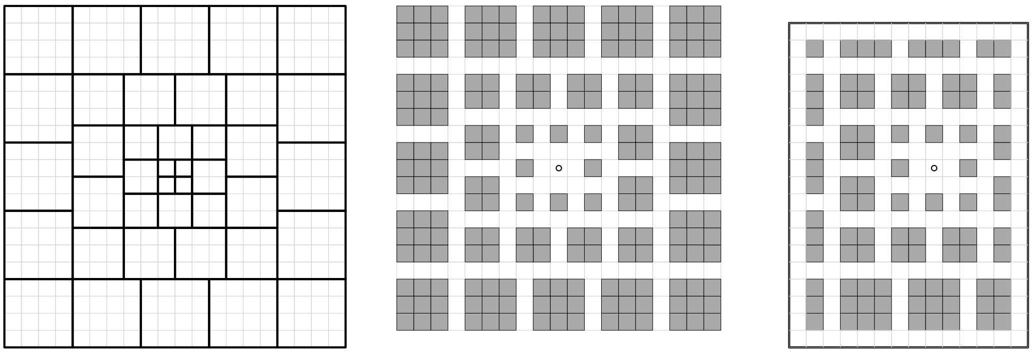

Figure 5 (a) A geometric tiling based on Nicomachus's Theorem, (b) the associated layout scheme, and (c) restricted to a rectangular domain.

Our layout for the warehouse problem, called the *Nicomachus layout*, is constructed as follows. For each ring r_k of the aforementioned structure and for each $k \times k$ square region of this ring, we include the $(k-1) \times (k-1)$ unit squares in the upper left corner in the layout (shaded in Fig. 5(b).) Each of these is called a *block*. We designate the upper-left cell of ring r_1 to be the access point. Finally, to accommodate a rectangular domain Ω, we clip the layout to the boundary of the rectangle and remove the layout squares touching the domain's boundary, thus creating corridors along the domain walls (see Fig. 5(c)). Observe that each block is surrounded by corridors that are one square wide. We show next that this layout achieves an optimal asymptotic density.

▶ **Lemma 2.** *The Nicomachus layout achieves an asymptotic density of* 1.

Proof. It suffices to show that the *asymptotic wastage*, that is, the asymptotic density of the complement of the Nicomachus layout is equal to zero. To see this, consider the first $\ell \geq 1$ rings of the layout. Each ring r_k, $1 \leq k \leq \ell$, consists of $4k$ blocks, each of size $(k-1) \times (k-1)$. The unused space per block is $k^2 - (k-1)^2 = 2k - 1$. Thus, the total wasted space for ring k is $4k(2k-1)$. Summing over all rings, the total wastage is $\sum_{k=1}^{\ell} 4k(2k-1) = 8\ell^3/3 + O(\ell^2)$. The first ℓ rings fill an origin-centered square of side length $\ell(\ell+1)$, which yields a total area of at least ℓ^4. Therefore, ignoring lower-order terms, the wastage for these rings is at most $(8\ell^3/3)/\ell^4 = 8/3\ell$. Clearly, this tends to zero in the limit. (Expressed as a function of n, the asymptotic density is the limit of $1 - 8/(3n^{1/4})$.) ◀

2.3.2 Accessing a Box

In order to access a box in the warehouse a robot must first travel to the block in which that box resides, retrieve it from the block, and then return it to the access point. The *depth* d of a box is defined to be the minimum number of boxes between it and the boundary of the block that contains it. So, a box on the perimeter of a block has depth $d = 0$, while one at the center of a block in ring r_i has depth $d = \left\lfloor \frac{i-2}{2} \right\rfloor$. (When the domain Ω is bounded, this is an upper bound since peripheral blocks may be clipped.)

In the Nicomachus layout, the cost of reaching a box in the arrangement and retrieving it from a block are both a function of the ring in which it resides. Let $M(r_i)$ denote the maximum cost of moving the robot from the access point to any cell adjacent to a block of ring r_i, and let $C(r_i)$ be the maximum cost of retrieving a box from a block in ring r_i. First, let us consider the travel cost of reaching a cell on the perimeter of a block of boxes.

▶ **Lemma 3.** *Travelling from the access point to any cell adjacent to a block in ring r_i requires at most $i^2 + i$ steps.*

Proof. To reach a box on the perimeter of a block in ring r_i from the access point a robot must traverse each ring $k \leq i$ by circumnavigating one of its blocks. It is easy to see that a robot can move between any two cells adjacent to a $(k-1) \times (k-1)$ block of ring r_k in $2k$ steps, from which we conclude that the total travel time is

$$M(r_i) \;\leq\; \sum_{k=1}^{i} 2k \;=\; i(i+1) \;\leq\; i^2 + i.$$

◀

An equivalent distance is traveled to return the requested box to the access point.

Next, let us define a primitive Replace(d) that allows for the swapping of a box b_i placed in the aisle adjacent to a block B with a box $b_j \in B$ at depth d. For now we will use this primitive to establish an upper bound on the cost of accessing a box, while the need for actually swapping boxes will not become apparent until later. Conceptually, the Replace primitive must unbury the target box by moving the d boxes in the way. It does so by moving them each $d+1$ spaces away, retrieving the target box, and then replacing them for a total cost $O(d^2)$. A more careful analysis yields the following.

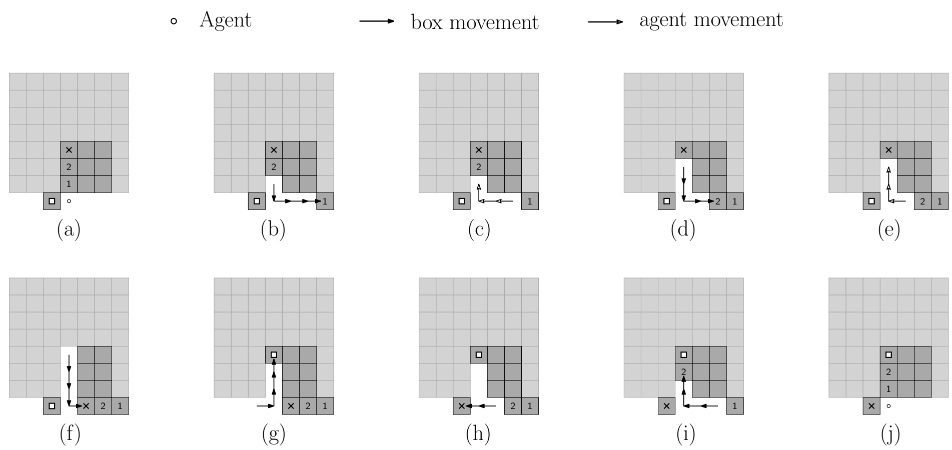

Figure 6 Swapping a pair of boxes, where the original box is at depth $d = 2$ within a 7×7 block in ring r_8.

▶ **Lemma 4.** *The cost of* Replace(d) *is at most $4d^2 + 8d + 6$, where d is the depth of box b_j.*

Proof. First, number the boxes inward from box b_j's nearest boundary from 1 to d. We assume that the robot begins adjacent to box 1 and that box b_i is adjacent to the robot. Next, we iteratively move each of the $d+1$ boxes (the d labeled boxes plus b_j) to a location that is $d+2$ units away along the side of the block (see Fig. 6). Accounting for the time to reach each box, pick it up, move it, put it down, and return to a position adjacent to the next box to be moved, each iteration has a total cost of $2d+3$, except the last which does not require moving to the next box and so only costs $d+2$. In total, moving these boxes costs $d(2d+3)+(d+2) = 2d^2+4d+2$. Next, we reverse the process at the same cost, replacing box b_j with box b_i and restoring boxes 1 through d to their original positions. This process is briefly interrupted to move box b_j out of the way, adding a cost of 2 (Fig. 6(h)). Thus, in total, swapping a new box with an interior box comes at a cost of $2(2d^2+4d+2)+2 = 4d^2+8d+6$. ◀

The depth of a box is bounded by the radius of the block in which it resides. Specifically, a box in ring r_i has a depth $d \leq \frac{i-2}{2}$ and so, along with Lemma 4, we have the following corollary:

▶ **Corollary 5.** *Retrieving a box from a block in ring r_i has a cost of $C(r_i) \leq i^2+2$.*

Combining this corollary and Lemma 3, the total cost to move to a box in ring r_i, retrieve it, and return to the access point is at most

$$(i^2+i)+(i^2+2)+(i^2+i) \ = \ 3i^2+2i+2 \tag{1}$$

Next, let us consider retrieval cost as a function of distance from the access point.

▶ **Lemma 6.** *If a box is at ℓ_1 distance δ from the access point then it lies in a ring r_i, such that $i \leq \sqrt{3\delta}$.*

Proof. To reach the highest ring level possible at a distance δ, travel orthogonally in a straight line, traversing each ring's width in turn. As ring r_i has width i, the farthest ring that can be reached is the first ring r_i such that

$$\delta \ \leq \ \sum_{j=0}^{i} j \ = \ \frac{i^2+i}{2} \tag{2}$$

Solving for i yields $i \geq \sqrt{2\delta+\frac{1}{4}} - \frac{1}{2}$.

It is easily seen that for all $\delta \geq 1$, $\sqrt{3\delta} \geq \sqrt{2\delta+\frac{1}{4}} - \frac{1}{2}$, thus $i = \sqrt{3\delta}$ suffices as an upper bound for the greatest ring index at a distance no more than δ. ◀

By combining Eq. (1) and Lemma 6, we obtain the following.

▶ **Lemma 7.** *In the Nicomachus layout, retrieving a box at ℓ_1 distance δ from the access point is $O(\delta)$.*

Proof. Eq. (1) shows that retrieving a box in ring r_i has a maximum total cost of $3i^2+2i+2$ and Lemma 6 shows that a box at distance δ will be in some ring r_i, where $i \leq \sqrt{3\delta}$. So, retrieving a box at distance δ incurs at most a cost of $3(\sqrt{3\delta})^2 + 2\sqrt{3\delta} + 2 = 9\delta + 2\sqrt{3\delta} + 2$, which is $O(\delta)$. ◀

From this we find that trading the positions of two boxes can be done at a cost proportional to the sum of their ℓ_1 distances from the access point. A simple, naive algorithm could use the access point as an intermediary, accessing both boxes at cost $O(\delta)$, and returning them to their opposing rather than original positions. Thus, we have the following:

▶ **Corollary 8.** *If two boxes b_i and b_j are at ℓ_1 distances δ_i and δ_j from the access point, respectively, then the cost of swapping them is no more than $c(\delta_i+\delta_j)$, for some constant c.*

Given this corollary, we can now show that Block-LRU$_A$ is competitive in the sliding model. From the proof of Theorem 1 and the structure of Block-LRU$_A$, it suffices to bound the cost of evictions from each of the containers. For any $k \geq 0$, consider an eviction from container C_k to C_{k+1}. The contribution of this eviction to $W_{\mathrm{lru}}(S)$ is 2^k. By Corollary 8, the cost of sliding one to the other is at most $c(2^{k-1} + 2^k) < 2c2^k$, implying that the sliding cost is within a constant factor of the eviction cost (roughly 4). From the proof of Theorem 1 the eviction cost can be used as a proxy for its actual cost, and therefore the sliding cost is at most a constant factor more than the actual cost of Block-LRU$_A$ in the case of swapping motion. This implies that the cost of Block-LRU$_A$ in the sliding motion model is competitive with the optimum solution in the swapping motion model. The actual cost of the optimum algorithm in the sliding model cannot be lower than the actual cost of the optimum algorithm in the swapping model. With a roughly factor-4 cost ratio between the sliding and swapping models, the overall ratio is roughly 128. While this competitive ratio may be rather high, the analysis thus far has assumed worst case scenarios across multiple factors and the focus has been to prove the general competitiveness rather than finding the best competitive ratio. We are confident that an empirical experiment would likely show that the average case scenario has a much more favorable competitive ratio. Regardless, as a consequence of the above discussion, we have:

▶ **Theorem 9.** *For any instance of the attic problem and any sufficiently long access sequence S, the cost of Block-LRU$_A(S)$ is within a constant factor of the cost of an optimal solution, assuming sliding motion.*

3 Online Solution to the Warehouse Problem

In this section we present an online algorithm for the warehouse problem. As before, we will present the algorithm for swapping motion and then generalize to sliding motion. Recall that the warehouse problem differs from the attic problem in that there are multiple access points, all of which lie on the bottom side of the rectangular domain Ω, which we may assume lies on the x-axis. Our algorithm, which we call Block-LRU$_W$, will be similar in spirit to online algorithms for hierarchical memory systems, but the combination of spatial locations and multiple access points adds considerable complexity. As with the attic problem, it will simplify matters to describe the algorithm first in an infinite context, where boxes may be placed anywhere above the x-axis, and then adjust the solution to the case of a rectangular domain. Our approach will be to define containers based on a quadtree-like structure above the x-axis, and to evict boxes up the quadtree from child to parent. We will treat each quadtree cell as if it were a cache in the memory hierarchy, with the least-recently used box evicted whenever more space is needed.

3.1 Quadtree Model

As mentioned above, our online solution to the warehouse problem employs a quadtree subdivision over the positive-y halfspace. The leaves of the quadtree, or *level* 0, consist of the unit squares whose lower left corners are the grid points on the x-axis, that is, $(x, 0)$ for $x \in \mathbb{Z}$. Level 1 consists of the 2×2 squares lying immediately above whose lower left corners are located at $(2x, 1)$ for $x \in \mathbb{Z}$. In general, for $k \geq 0$, level-k consists of the $2^k \times 2^k$ squares whose lower left corners lie on $(2^k x, 2^k - 1)$, for $x \in \mathbb{Z}$. Each level-k node u has a parent $p(u)$ of twice the side length lying immediately above on level $k+1$ (see Fig. 7(a)), and two children each of half the side length lying immediately below on level $k-1$. The set of unit squares associated with each node of the quadtree is called its *cell*. This structure covers the infinite grid lying above the x-axis. Given a rectangular domain Ω whose lower side lies along the x-axis, we clip the above structure to this rectangle (see Fig. 7(b)).

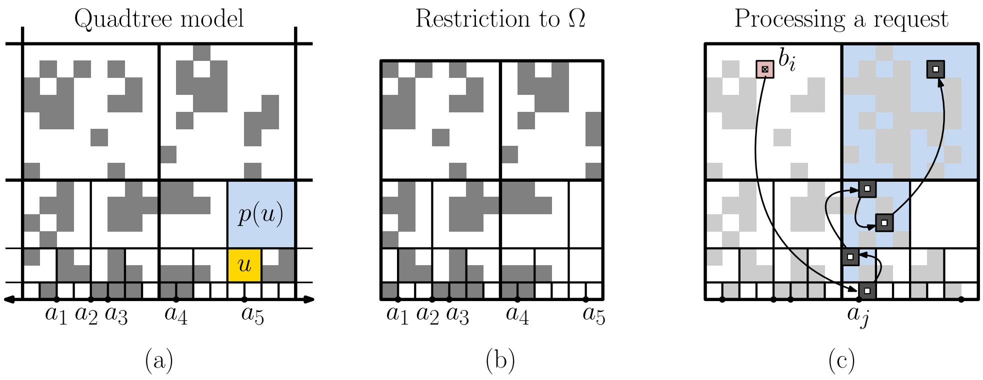

Figure 7 Quadtree layout.

To simplify the analysis of our solution, we first define a variant of the warehouse problem with an alternate cost function based on this quadtree structure, which we call the *quadtree model*. Of course, an optimal solution does not need to follow this model, and later, we will relate the cost of the standard solution to this variant. The processing of requests in this model differs from the standard model (described in Section 1.1) in that, after moving the box to the desired access point, the reorganization chain is allowed to move a box within its current quadtree cell, or it may move the box to the quadtree cell of an ancestor, but no other movements are allowed (see Fig. 7(c)).

More formally, consider a request for a box b to access point a. Let $Q_0(a)$ denote the quadtree cell containing a, and let $Q_1(a), Q_2(a), \ldots$ denote the successive quadtree ancestor cells of $Q_0(a)$. If a is unoccupied, we simply move the box there. Otherwise, in order to make space for b_i, we perform a chain of swaps along some locations $p_0, p_1, \ldots, p_k$ such that $p_0 = a$, p_k is either unoccupied (possibly the former location of b), and if $p_i \in Q_j(a)$, then p_{i+1} is the same cell or an ancestor, that is, $p_{i+1} \in Q_{j'}(a)$ for $j' \geq j$. As described in Section 1.1, we perform swaps (in reverse order) along the resulting chain. Each swap that moves a box out of its current quadtree cell is called *eviction*.

Costs are defined as follows in this model. A box may be moved within its quadtree cell free of charge, but when it is moved to an ancestor cell, it is charged 2^k, where k is the level of the quadtree cell into which the box is moved. (The analogy with hierarchical memory systems should be evident, where we think of each quadtree cell as a cache, and eviction to an ancestor is analogous to moving a page to a larger cache in slower memory.)

3.2 Online Algorithm for Swapping Motion

Let us now present our algorithm for the warehouse problem, which we call Block-LRU$_W$. Consider a request (b, a) to bring box b to access point a. If this access point is unoccupied, we simply move the box there. Otherwise, in order to make space for b, we will perform a sequence of evictions from $Q_0(a)$, $Q_1(a)$, and so on until we encounter the first quadtree ancestor $Q_k(a)$ that has at least one unoccupied location (possibly b's location at the time of the request). More formally, let p_b denote b's location, let $p_0 = a$ denote the access point, and let $p_1, \ldots, p_{k-1}$ denote the locations of the least-recently used boxes of quadtree cells $Q_0(a)$ through $Q_{k-1}(a)$, respectively. Finally, let $p_k \in Q_k(a)$ denote the final unoccupied location (or former location of b). As described in Section 1.1, we perform swaps (in reverse order) along the chain $\langle p_b, p_0, \ldots, p_k \rangle$. The main result of this section is showing that this algorithm is competitive.

▶ **Theorem 10.** *For any instance of the warehouse problem and any sufficiently long access sequence S, the cost of Block-LRU$_W(S)$ is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Due to space limitations, the full proof and competitive analysis appear in Appendix A.2. It is a nontrivial extension of the single-container structure from the attic problem to a hierarchical container structure based on the quadtree, and showing how a general solution in the standard model can be transformed competitively into the quadtree model.

3.3 Online Algorithm for Sliding Motion

In this section, we show that the competitiveness of Block-LRU$_W$ in the case of swapping motion can be used to prove that the sliding version of the same algorithm is competitive. As in the attic problem, our approach will be to describe a layout of boxes that is amenable to efficient sliding motion.

We make use of a Nicomachus-like box layout. Rather than rings centered about the access point, we flatten these rings into layers stacked above the x-axis. As before, we begin with a layer of 1×1 cell regions. Above this is a row of 2×2 regions, then 3×3, and so on, with each $i \times i$ region containing a block of $(i-1) \times (i-1)$ boxes (see Fig. 8). We call this the *flattened Nicomachus layout*.

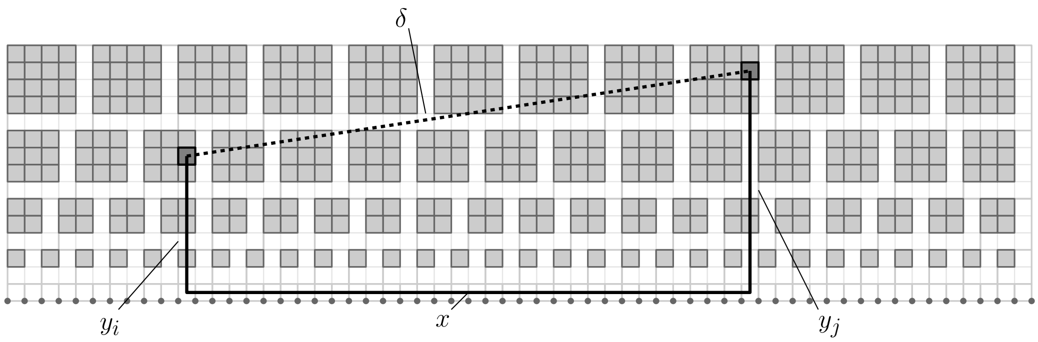

Figure 8 A flattened version of the Nicomachus layout for the warehouse problem, with a conceptual example of swapping two boxes. Pathfinding is ignored in this illustration, but accounted for in the supporting lemma.

Once again, we make use of a simple naive algorithm that can efficiently trade the positions of two boxes in the sliding model. More formally, we prove the following:

▶ **Lemma 11.** *If two boxes b_i and b_j are at ℓ_1 distances δ from each other and at vertical distances y_i and y_j from the x-axis, respectively, then the cost of swapping them in the flattened Nicomachus layout is no more than $c(\delta + y_i + y_j)$, for some constant c.*

Proof. A naive algorithm can swap the two boxes b_i and b_j by: (1) bringing them to the x-axis, (2) swapping their positions along the x-axis, and (3) returning them to their new vertical positions. Notice that the cost of retrieving/replacing a box and bringing it to the x-axis is equivalent to the retrieval cost of a box positioned directly above the access point in the Attic Problem with Sliding Motion. As per Lemma 7, this access cost in both contexts is

$O(y)$, where y is the distance to the x-axis or singular access point, respectively. Given this, both steps (1) and (3) of the algorithm occur at a constant factor of $(y_i + y_j)$. Clearly the horizontal distance traveled along the x-axis $x \leq \delta$, therefore, the total cost of swapping the two boxes must be no greater than $c(\delta + y_i + y_j)$, for some constant c. ◀

We can use this lemma to related the cost of swapping two elements in the swapping and sliding models. The following summarizes our main result.

▶ **Theorem 12.** *For any instance of the warehouse problem and any sufficiently long access sequence S, the cost of Block-LRU$_W(S)$ is within a constant factor of the cost of an optimal solution, assuming sliding motion.*

Proof. From Theorem 10 and the structure of Block-LRU$_W$, it suffices to bound the cost of evictions from one quadtree node to its parent. Assuming that the node is at quadtree level $k - 1$, and its parent is at level k, this swap incurs a cost of 2^k in the quadtree model. Letting y_1 and y_2 denote the vertical distances of these locations from the x-axis, we have $y_1 \leq 2^k$ and $y_2 \leq 2^{k+1}$. Also, they are separated from each other by an ℓ_1 distance of $\delta \leq 2^{k+2}$. By Lemma 11, the cost of sliding one to the other is at most $c(\delta + y_i + y_j) \leq c(2^{k+2} + 2^k + 2^{k+1}) = 7c2^k$, implying that sliding cost is within a constant factor of the quadtree cost. From the proof of Theorem 10 and the structure of Block-LRU$_W$, the quadtree cost of Block-LRU$_W$ can be used as a proxy for its actual cost, and therefore the sliding cost is at most a constant factor more than the actual cost of Block-LRU$_W$ assuming swapping motion. This implies that the cost of Block-LRU$_W$ in the sliding motion model is competitive with the optimum solution in the swapping motion model. The actual cost of the optimum algorithm in the sliding model cannot be lower than the actual cost of the optimum algorithm in the swapping model. With a roughly factor-7 cost ratio between the sliding and swapping models, the overall ratio is roughly 112. As before, this is based on many worst-case assumptions and can likely be improved upon. ◀

4 Concluding Remarks

In this paper we have presented a model for an automated warehouse management system containing a set of standardized portable storage units or boxes, a robot that moves these boxes around the warehouse in one of two ways (swapping or sliding), and a set of access points where requested boxes must be delivered. We then presented online algorithms for two natural instances of the warehouse problem, one involving a single access point within a rectangular domain and the other involving a sequence of access points along the bottom side of a rectangular domain. We prove that our algorithms are competitive with respect to an optimal (offline) algorithm with full knowledge of the access sequence. Our competitive ratios are relatively high, and we suspect that they are far from tight, but tightening these bounds will involve either significantly more complex algorithms or better lower bounds.

We leave for future work some interesting open problems. Recall that our model assumes that access requests are processed sequentially. This simplifying assumption allowed us to ignore the extremely difficult issue of motion coordination, which arises when multiple robots are present [11, 10, 18]. Clearly, any realistic solution should consider an environment with multiple robots where requests are processed concurrently. Because we control the layout of boxes in the domain, it may be possible insert additional *slack space* into the layout to facilitate efficient motion coordination. Another interesting question in this vein is how to handle the insertion/deletion of boxes from the collection. Perhaps we could further leverage memory management schemes such as [9], which efficiently handle the reallocation of 2D memory.

Also, how does the competitiveness of our schemes change, if at all, when the model becomes less uniform. In our current model, all actions taken by the robot are of unit cost, regardless of factors like whether or not the robot is laden or what sort of path a robot takes to retrieve a box. Çelik and Süral [4], for example, show that the number of turns a robot makes in a parallel-aisle warehouse can have a significant impact on retrieval efficiency. Fekete and Hoffmann [8] look at the online problem of packing differently sized squares into a dynamically sized square container, and applying this work to a warehouse which does not use standardized containers would be a natural continuation of the work presented here. Further generalizing our model to account for differing action costs and box dimensions would increase its real-world applicability and may lead to some interesting insights.

References

1 A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, STOC '87, pages 305–314, New York, NY, 1987. ACM. URL: `http://doi.acm.org/10.1145/28395.28428`, `doi:10.1145/28395.28428`.

2 F. Amato, F. Basile, C. Carbone, and P. Chiacchio. An approach to control automated warehouse systems. *Control Eng. Pract.*, 13(10):1223–1241, October 2005. URL: `http://www.sciencedirect.com/science/article/pii/S0967066104002345`, `doi:10.1016/j.conengprac.2004.10.017`.

3 A. S. Cahn. The summer meeting in Madison. *Bull. Amer. Math. Soc.*, 54(11):1073, November 1948. URL: `http://www.ams.org/journal-getitem?pii=S0002-9904-1948-09093-0`, `doi:10.1090/S0002-9904-1948-09093-0`.

4 M. Çelik and H. Süral. Order picking in a parallel-aisle warehouse with turn penalties. *Internat. J. Production Res.*, 54(14):4340–4355, July 2016. URL: `http://www-tandfonline-com.proxy-um.researchport.umd.edu/doi/abs/10.1080/00207543.2016.1154624`, `doi:10.1080/00207543.2016.1154624`.

5 F.-L. Chang, Z.-X. Liu, Z. Xin, and D.-D. Liu. Research on order picking optimization problem of automated warehouse. *Sys. Eng. - Theory & Pract.*, 27(2):139–143, February 2007. URL: `http://www.sciencedirect.com/science/article/pii/S1874865108600150`, `doi:10.1016/S1874-8651(08)60015-0`.

6 A. Charnes and W. W. Cooper. Generalizations of the warehousing model. *OR: Oper. Research Quarterly*, 6(4):131–172, 1955. URL: `http://www.jstor.org/stable/3006550`, `doi:10.2307/3006550`.

7 J.-F. Cordeau and G. Laporte. The dial-a-ride problem: Models and algorithms. *Ann. Oper. Res.*, 153(1):29–46, 2007. URL: `https://link.springer.com/article/10.1007/s10479-007-0170-8`, `doi:10.1007/s10479-007-0170-8`.

8 S. P. Fekete and H.-F. Hoffmann. Online square-into-square packing. *Algorithmica*, 77(3):867–901, 2017. URL: `https://link.springer.com/article/10.1007%2Fs00453-016-0114-2`, `doi:https://doi.org/10.1007/s00453-016-0114-2`.

9 S. P. Fekete., J.-M. Reinhardt, and C. Scheffer. An efficient data structure for dynamic two-dimensional reconfiguration. *J. Syst. Archit.*, 75(C):15–25, April 2017. URL: `https://doi.org/10.1016/j.sysarc.2017.02.004`, `doi:10.1016/j.sysarc.2017.02.004`.

10 R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theo. Comp. Sci.*, 343(1-2):72–96, 2005. URL: `https://www.sciencedirect.com/science/article/pii/S0304397505003105`, `doi:https://doi.org/10.1016/j.tcs.2005.05.008`.

11 J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem". *Internat.*

J. Robotics Res., 3(4):76–88, 1984. URL: https://doi.org/10.1177/027836498400300405, doi:10.1177/027836498400300405.

12 D. Jain. Adoption of next generation robotics: A case study on Amazon. *Perspectiva: A Case Research Journal*, III:15, 2017.

13 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. URL: http://www.sciencedirect.com/science/article/pii/S1574013709000197, doi:10.1016/j.cosrev.2009.04.002.

14 C. K. M. Lee. Development of an industrial internet of things (IIoT) based smart robotic warehouse management system. In *CONF-IRM 2018 Proceedings*, page 14, 2018.

15 R. B. Nelsen. *Proofs without words: Exercises in visual thinking.* Number no. 1 in Classroom resource materials. The Mathematical Association of America, Washington, D.C, 1993.

16 K.-W. Pang and H.-L. Chang. Data mining-based algorithm for storage location assignment in a randomised warehouse. *Internat. J. Production Res.*, 55(14):4035–4052, July 2017. URL: https://doi.org/10.1080/00207543.2016.1244615, doi:10.1080/00207543.2016.1244615.

17 M. Sarrafzadeh and S. R. Maddila. Discrete warehouse problem. *Theo. Comp. Sci.*, 140(2):231–247, April 1995. URL: http://linkinghub.elsevier.com/retrieve/pii/030439759400192L, doi:10.1016/0304-3975(94)00192-L.

18 R. Sharma and Y. Aloimonos. Coordinated motion planning: The warehouseman's problem with constraints on free space. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(1):130–141, February 1992. URL: http://ieeexplore.ieee.org/document/141317/, doi:10.1109/21.141317.

19 D. D. Sleator and R. E. Tarjan. Amorized eficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, February 1985. URL: https://dl.acm.org/citation.cfm?id=2793, doi:10.1145/2786.2793.

20 L. Wolsey and H. Yaman. Convex hull results for the warehouse problem. *Disc. Optimization*, 30:108–120, 2018. URL: http://www.sciencedirect.com/science/article/pii/S1572528617301482, doi:https://doi.org/10.1016/j.disopt.2018.06.002.

A Full Proofs

A.1 Competitiveness of Block-LRU$_A$ (Attic Problem) with Swapping

▶ **Theorem 1.** *For any instance of the attic problem and any sufficiently long access sequence R, the cost of Block-LRU$_A$(S) is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Proof. Consider an input S consisting of the initial box placement and a sequence of access requests. Let $T_{\text{opt}}(S)$ and $T_{\text{lru}}(S)$ denote the total cost of the optimum and Block-LRU$_A$ solutions, respectively, on this input. We will show that there exists a constant c and quantity $f(S)$ that does not grow with the length of the access sequence, such that $T_{\text{lru}}(S) \leq cT_{\text{opt}}(S) + f(S)$. Since $f(S)$ does not grow with the length of the access sequence, for all sufficiently long access sequences its impact on the total cost will be negligible compared to $T_{\text{opt}}(S)$.

Our analysis will be based on an auxiliary statistic. Given any container C_k, define an *eviction* to be an event in which a box lying within this container is moved to a location in an enclosing container $C_{k'}$, for $k' > k$. For the given access request sequence S, define $E_{\text{lru}}(S, k)$ to be the total number of evictions from container C_k performed by Block-LRU$_A$. Let $W_{\text{lru}}(S) = \sum_{k \geq 0} 2^k E_{\text{lru}}(S, k)$ denote the weighted cost of these evictions. We will show that there exist constants c_1 and c_2 and quantities $f_1(S)$ and $f_2(S)$ that do not grow with the length of the access sequence, such that the following two inequalities hold:

$$(1)\ T_{\text{lru}}(S) \leq c_1 W_{\text{lru}}(S) + f_1(S) \quad \text{and} \quad (2)\ W_{\text{lru}}(S) \leq c_2 T_{\text{opt}}(S) + f_2(S).$$

We first prove inequality (1). Observe that the cost of processing a request involving a box b in Block-LRU$_A$ consists of two parts, the cost of moving b to the access point (that is, the ℓ_1 distance of b to access point) plus the cost of performing the evictions caused by this move. We assert that it suffices to bound only the latter quantity. To see why, consider two consecutive requests to b. Just after the first request, b is located at the access point. When the second request occurs, if b is not at the access point, it has been moved away due to various evictions involving b that have occurred due to intervening access requests. By the triangle inequality, the sum of the costs of these evictions involving b is at least as large as the ℓ_1 distance of b from the access point at the time of the second request. Thus, the cost of moving b to the access point for the second request is not greater than cost of evictions involving b due to intervening requests. This allows us to account for all the requests for b except the first. Define $f_1(S)$ to be the sum of the ℓ_1 of every box's initial location to the access point. Clearly, $f_1(S)$ depends only on the initial box placements.

It remains to bound the cost needed to process the evictions. Each time Block-LRU$_A$ evicts a box from some container C_k to the enclosing container C_{k+1}, the cost is bounded above by the maximum distance between any point of C_k to any point in C_{k+1}. Clearly, this is not greater than the diameter of C_{k+1}, which is 2^{k+2}. Summing over all accesses and all containers, it follows that the total cost of Block-LRU$_A$ evictions is at most $\sum_{k\geq 0} 2^{k+2} E_{\mathrm{lru}}(S,k) = 4W_{\mathrm{lru}}(S)$. By our earlier observation that the cost of bringing boxes back to the access point is bounded above by the sum of $f_1(S)$ and the total eviction cost, it follows that $T_{\mathrm{lru}}(S) \;\leq\; c_1 W_{\mathrm{lru}}(S) + f_1(S)$, where $c_1 = 2 \cdot 4 = 8$, thus establishing (1).

To prove inequality (2), we will apply a technique similar to one given by Sleator and Tarjan [19] and Aggarwal *et al.* [1] for hierarchical memory systems. For any $k \geq 0$, define $\overline{C}_k = \bigcup_{j\leq k} C_j$ (that is, the set of points within distance 2^k of the origin). Also define $m_k = |C_k|$ and $\overline{m}_k = |\overline{C}_k|$ denote the total capacities of these sets. For each $k \geq 2$, we will relate the weighted eviction cost of Block-LRU$_A$ on container C_k with respect to the cost of box movements by the optimal solution within container C_k. The overall analysis comes about by summing over all container levels.

Fix any $k \geq 2$. Partition the access request sequence into contiguous *segments*, such that within any segment (except possibly the last), Block-LRU$_A$ performs $\overline{m}_k$ evictions from container C_k. (The last segment will not be analyzed, but since there is only one such segment for each k from which an eviction was performed, it follows that for all sufficiently long access segments, the impact on the overall cost of these segments be negligible. See [19] for more details.) Consider any complete segment. The contribution of the evictions of this segment from C_k to the weighted eviction cost $W_{\mathrm{lru}}(S)$ is $2^k\overline{m}_k$. In Block-LRU$_A$ every container C_j for $j \leq k$ evicts the least recently accessed box, and this implies that any box evicted from container C_k is the least recently accessed box not only from C_k, but from $\overline{C}_k$ as well. We assert that during this segment, the number of distinct boxes accessed must be at least $\overline{m}_k$. To see why, observe that either all of the boxes evicted during this segment are distinct, or some box was evicted twice during the sequence. If there are $\overline{m}_k$ distinct evictions, then there are at least least $\overline{m}_k$ distinct boxes requested. On the other hand, if a box is evicted twice, then by the nature of Block-LRU$_A$, between these two evictions, every one of the $\overline{m}_k$ boxes in $\overline{C}_k$ must have been accessed in order for this box to transition from the most recent to the least recent.

Now, let us consider how the optimum algorithm deals with the $\overline{m}_k$ distinct box requests that have occurred during this segment. Intuitively, because of the exponential increase in container sizes, most of the $\overline{m}_k$ distinct accessed boxes cannot fit within $\overline{C}_{k-1}$, and hence they must spill out into the surrounding region. We will charge for the work needed for the spillover but limited to C_k (to avoid double counting).

It will simplify matters to ignore boundary issues for now and consider the unbounded case where $\Omega = \mathbb{Z}^2$. Define $\widehat{C}_k$ to be the set of points of the infinite grid that lie within ℓ_1 distance $(3/4)2^k$ of the access point. Since $k \geq 2$, we have $\overline{C}_{k-1} \subset \widehat{C}_k \subset \overline{C}_k$. Let $\widehat{m}_k = |\widehat{C}_k|$. We have $\widehat{m}_k \leq c'\overline{m}_k$, where $c' \approx (3/4)^2 \leq 2/3$. Thus, a fraction of $1 - c'$ or roughly one-third of the $\overline{m}_k$ distinct boxes accessed during this sequence must spill out from $\overline{C}_{k-1}$ to an ℓ_1 distance of at least $(3/4)2^k - 2^{k-1} = (1/2)2^{k-1} = 2^{k-2}$ beyond $\overline{C}_{k-1}$'s outer boundary. It follows that the contribution of to the cost of $T_{\text{opt}}(S)$ of these boxes is at least $(\overline{m}_k/3)2^{k-2} = 2^k\,\overline{m}_k/12$. Because all of these box motions are contained within C_k, there is no double counting of this cost between containers.

The generalization to the case of a bounded rectangular domain Ω is straightforward but tedious. The key difference is that, due to the bounded nature of Ω, the sizes of consecutive containers may grow only linearly, not quadratically with the ℓ_1 radius of the container. (This happens, for example, if the domain is a long, thin strip.) Further, the size of the last container may even be smaller than its predecessor as we approach the outer edges of the domain. However, the key is that, since the radius value grows exponentially, consecutive container sizes differ by a constant factor for all but a constant number containers, and this is all that the above analysis requires.

Let s_k denote the number of complete segments for level k. Summing all the segments and all the levels of the hierarchy, we obtain

$$T_{\text{opt}}(S) \;\geq\; \sum_{k\geq 2} s_k 2^{k-2}\,\overline{m}_k.$$

Adding in a term $f_2(S)$ to account for the final (incomplete) segments, noting that $\overline{m}_0$ and $\overline{m}_1$ are both constants, and combining with our earlier bound on $W_{\text{lru}}(S)$, we obtain the following, for a suitable constant c_3.

$$\begin{aligned} W_{\text{lru}}(S) \;&\leq\; \sum_{k\geq 0} s_k 2^k\,\overline{m}_k + f_2(S) \;=\; s_0\overline{m}_0 + s_1 2\overline{m}_1 + \sum_{k\geq 2} s_k 2^k\,\overline{m}_k + f_2(S) \\ &\leq\; c_3(s_0 + s_1) + 4T_{\text{opt}}(S) + f_2(S). \end{aligned}$$

The term $c_3(s_0 + s_1)$ is just a constant times the total number of access requests and is not dominant. It follows that there is a constant c_2 such that $W_{\text{lru}}(S) \leq c_2\,T_{\text{opt}}(S) + f_2(S)$, which establishes inequality (2). Note that $f_2(S)$ does not grow with the length of the access sequence.

Finally, by combining inequalities (1) and (2), we obtain

$$\begin{aligned} T_{\text{lru}}(S) \;&\leq\; c_1 W_{\text{lru}}(S) + f_1(S) \;\leq\; c_1(c_2 T_{\text{opt}}(S) + f_2(S)) + f_1(S) \\ &\leq\; c_1 c_2 T_{\text{opt}}(S) + (c_1 f_2(S) + f_1(S)) \;\leq\; cT_{\text{opt}}(S) + f(S), \end{aligned}$$

for some constant $c \geq c_1 c_2 \geq 32$ and quantity $f(S)$ that does not grow with the length of the access sequence. For all sufficiently long access sequences, this final term will be negligible. This completes the proof. ◀

A.2 Competitiveness of Block-LRU$_W$ (Warehouse) with Swapping

▶ **Theorem 10.** *For any instance of the warehouse problem and any sufficiently long access sequence S, the cost of Block-LRU$_W(S)$ is within a constant factor of the cost of an optimal solution, assuming swapping motion.*

Observe that Block-LRU$_W$ satisfies the requirements in quadtree model. For the sake of the above theorem, its cost is computed in the standard manner, as the sum of the ℓ_1 distances of all swaps performed. Later, we will show that this is proportional to its cost in the quadtree model.

The remainder of this section is devoted to proving this theorem. First, let us consider how we can simulate the behavior of a general solution to the warehouse problem in the quadtree model. Rather than focusing on individual access requests, we will do this on a box-by-box basis. Consider input sequence S and any box b. Let S' denote a contiguous segment of S, which starts and ends at two consecutive access requests involving b. Let us denote these access points by a_1 and a_2, respectively. (For the segment prior to b's first access, set a_1 the closest access point to b's initial location, and for the segment following b's last access, a_2 can be set arbitrarily to any access point.)

When the standard solution completes the processing of the first access request, b will reside at a_1. As a result of subsequent access requests in S', b may be moved to new locations in the domain as a result of swap operations. Let $\langle p_0, \ldots, p_k \rangle$ denote the sequence of locations through which b moves during S', so that $p_0 = a_1$, and p_k is the location of b just prior to the upcoming access request at a_2. Since this is in the standard model, the points of this sequence are arbitrary. To perform the simulation, we will define a function π that maps the location of b at any time to the cell of some quadtree ancestor of a_1 in a manner such that, under this function, b will move in accordance with the quadtree model. We present this mapping in the next section.

A.2.1 Container Structure for the Warehouse Problem

Before giving the details of the aforementioned mapping, let us start with an intuitive explanation. For each access point a let $Q_k(a)$ denote the quadtree cell associated with a's ancestor at level k. We define a collection of nested regions of exponentially increasing sizes called *containers* surrounding a, denoted $C_0(a) \subset C_1(a) \subset \cdots$ (see Fig. 9(a)). (Note that, unlike the containers of Section 2.1, which were pairwise disjoint, here each container includes all the squares of its predecessors.)

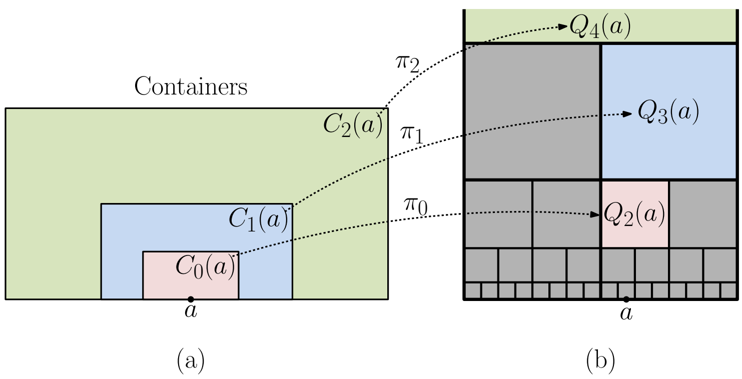

Figure 9 Intuitive structure of containers for the warehouse quadtree model.

For each container $C_k(a)$ we will define a 1–1 function π_k that maps each of point in $C_k(A)$ to a point within the cell of some quadtree ancestor of a. (For example, in Fig. 9(a), π_k maps boxes from $C_k(a)$ to $Q_{k+2}(a)$.) In order to simulate the movement of a box that has been accessed most recently by a, we will track its movement through these containers. On first entering a container $C_k(a)$ at some point p, we map the box to the associated point

$\pi_k(p)$ in the quadtree cell. When the box moves to a new point p' within the same container, we move the box to $\pi_k(p')$. Observe that because the containers are nested, even if the box moves into a location in a smaller container, it will still be considered as lying within C_k and so will remain in the same quadtree cell in the simulation. Recall that in the quadtree model, movements within the same quadtree cell are free of charge, and hence there is no need to account for movements within a given container. Whenever the box is first moved into a new larger container $C_{k'}$, it will be charged the eviction cost of $2^{k''}$, where $Q_{k''}(a)$ is the associated quadtree cell.

Let us now define the containers and the associated functions more formally. One complication that arises is that the functions π_k associated with two nearby access points may map locations to the same quadtree cell. When this happens, we must guarantee that two distinct locations in their containers are not mapped to the same location in this quadtree cell. To handle this, we will design our container structure carefully so that access points that map to the same quadtree cell will share the same container and the same mapping function.

To make this precise, consider any access point a and any quadtree ancestor of a at level k. The function π_k for a will map points from a's container $C_k(a)$ to $Q_{k+2}(a)$. This implies that the four grandchildren of $Q_{k+2}(a)$ at level k will do the same. So, we will give them all a common container and a common function. (In Fig. 10(a), the container $C_2(a)$ is shared by four 4×4 quadtree cells drawn in heavy black lines.) The associated container is defined as follows. First, imagine a square grid of side length 2^k covering the plane that is aligned with the quadtree cells. The container consists of the 16 grid cells that are ℓ_1 neighbors of the four grandchildren. (In Fig. 10(a), this container $C_2(a)$ is shaded in dark gray and includes the squares of $C_0(a)$ and $C_1(a)$. Note that the lowest tier of these grid squares falls one unit below the x-axis, but we simply ignore these nonexistent squares in our mapping.) The number of squares is at most $16 \cdot 2^k = 2^{k+2}$, and so there is sufficient space to map the squares of the container into $Q^{k+2}(a)$ (see Fig. 10(b)). We define π_k for this container to be any such function. (We do not require that this function preserve distances because, according to the quadtree model, movements within a quadtree cell are free.)

A.2.2 Proving Competitiveness

In this section, we present a proof of Theorem 10. Given a access sequence S, define $T_{\mathrm{opt}}(S)$, $T_{\mathrm{lru}}(S)$ to be the (standard) costs for Opt and Block-LRU$_W$, respectively. Define $W_{\mathrm{lru}}(S)$ to be the cost of Block-LRU$_W$ in the quadtree cost model, and define $W_{\mathrm{opt}}(S)$ to be the cost of the quadtree-simulated version of Opt in the quadtree cost model.

The analysis follows a similar structure to the one given in Theorem 1, and so we will focus on just the major differences. The analysis is based on three inequalities, where c_1, c_2, and c_3 are constants and $f_2(S)$ and $f_3(S)$ are quantities that do not grow with the length of the access sequence:

$$(1) T_{\mathrm{lru}}(S) \leq c_1 W_{\mathrm{lru}}(S) \quad (2) W_{\mathrm{lru}}(S) \leq c_2 W_{\mathrm{opt}}(S) + f_2(S) \quad (3) W_{\mathrm{opt}}(S) \leq c_3 T_{\mathrm{opt}}(S) + f_3(S)$$

- $T_{\mathrm{lru}}(S) \leq c_1 W_{\mathrm{lru}}(S)$: Block-LRU$_W$ is running in the quadtree model, but it uses the standard (ℓ_1) costs, not the eviction costs. Also, it evicts from child to parent, never skipping ancestors. When moving a box from quadtree cell Q_{k-1} to Q_k the actual cost is at most the worst-case ℓ_1 distance between these cells, which is at most $2 \cdot 2^k = 2^{k+1}$, and the quadtree model assesses a charge of 2^k. Thus, setting $c_1 = 2$ yields the desired bound.

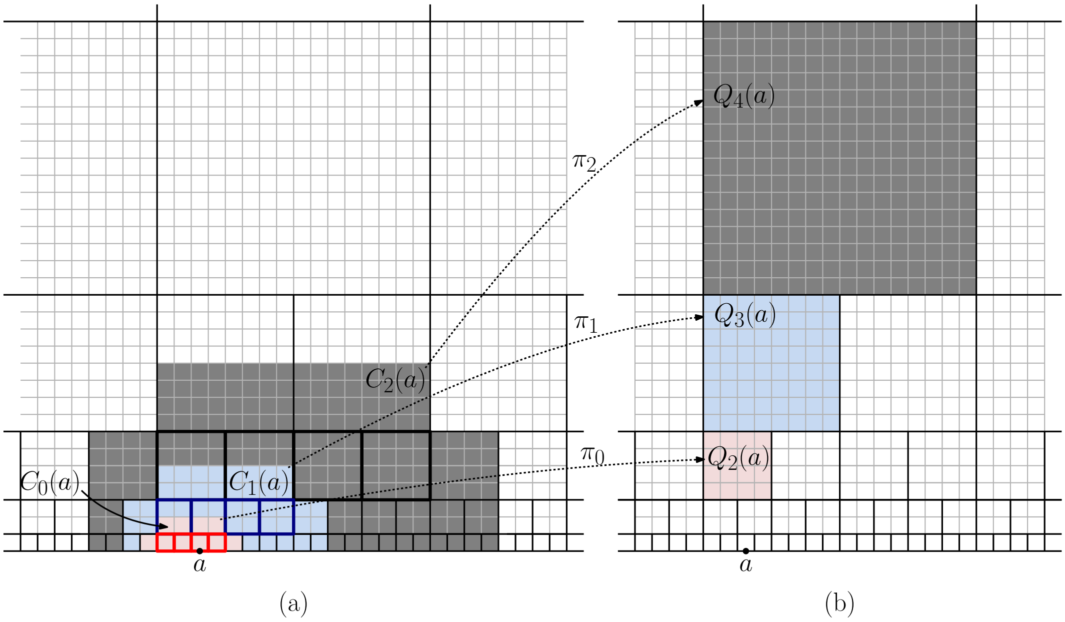

Figure 10 Actual structure of containers for the warehouse quadtree model.

- $W_{\rm lru}(S) \le c_2 W_{\rm opt}(S) + f_2(S)$: Let $m_k = 2^{2k}$ denote the number of boxes in a quadtree cell Q_k at level k. Let $\overline{m}_k$ the sum of m_j for a quadtree cell and all its descendants (which is roughly $2m_k$). Let us focus on a single quadtree cell at level k, call it Q_k. Consider the two child cells at level $k-1$, Q'_{k-1} and Q''_{k-1}. Let A' and A'' denote the subsets of access points descended from these two quadtree nodes, respectively. Now, break up the access sequence into contiguous segments, such that Q_k witnesses $\overline{m}_k$ evictions in the running of Block-LRU$_W$. Let us consider a single segment S'. Observe that, with respect to access points $A' \cup A''$, Block-LRU$_W$ is effectively running an LRU algorithm on the union of Q_k and the cells of all its children. (To see why, observe that the least-recently used boxes of each descendent are evicted to their parents and eventually up to to Q_k, and the least-recently used box within Q_k is evicted.)

  We assert that over segment S', at least $\overline{m}_k$ distinct box accesses have been processed by the access points $A' \cup A''$ combined. Now, let us consider how $W_{\rm opt}(S)$ handles the same requests, but from the perspective of Q'_{k-1} and Q''_{k-1}. These two together (and their descendant cells) have a total capacity of $\overline{m}_{k-1} + \overline{m}_{k-1} \approx \overline{m}_k/2$. Thus, the remaining roughly $\overline{m}_k/2$ boxes must be evicted from these children by Opt. They may be evicted up one level to Q_k or up multiple levels. For the sake of simplicity, let us consider the case where they are evicted up just one level to Q_k. (The other case involves splitting the charge among the nodes along the path according to a geometric series.) Each evicted box is assessed a charge of 2^k, for a total of roughly $2^k \overline{m}_k/2 = 2^{k-1}\overline{m}_k$. Therefore, the total charge assessed to $W_{\rm opt}(S)$ during this segment is at least $2^{k-1}\overline{m}_k$, while the total charge assessed to Q_k in $W_{\rm lru}(S)$ is $2^{k+1}\overline{m}_k$. Summing over all the levels (and letting $f_2(S)$ account for the charges in the partial segment at the end of S) we have $W_{\rm lru}(S) \le c_2 W_{\rm opt}(S) + f_2(S)$, where c_2 is roughly 4.

- $W_{\rm opt}(S) \le c_3 T_{\rm opt}(S) + f_3(S)$: We focus on the activity involving a single box b between two consecutive accesses to a and a', say. (The additional $f_3(S)$ term handles the cost prior to the initial request for b and after the final request.) Observe that $W_{\rm opt}(S)$ does not charge for movements within a quadtree cell, and (since we are in the quadtree model) it never demotes a box to a lower level of the quadtree. It charges an eviction cost of 2^k whenever the box enters a quadtree cell at level k. This event corresponds to an event in standard Opt when this box enters $C_k(a) \setminus C_{k-1}(a)$ for the first time. Let k^* denote the highest container index into which Opt moves this box (formally, the highest k such that the box enters $C_k(a) \setminus C_{k-1}(a)$). Since this box might be evicted into all the containers from level 1 up to k^*, this box contributes at most $\sum_{k=1}^{k^*} 2^k \leq 2^{k^*+1}$ to $W_{\text{opt}}(S)$. On the other hand, Opt has to move this box from the access point to some point in $C_{k^*}(a) \setminus C_{k^*-1}(a)$. It is easy to see that this involves a distance of at least $2^{k^*} + 1$. It follows that this box contributes more than 2^{k^*} to $T_{\text{opt}}(S)$ and at most 2^{k^*+1} to $W_{\text{opt}}(S)$. Therefore, setting $c_3 = 2$ yields the desired result.

Together, the three inequalities imply that

$$\begin{aligned} T_{\text{lru}}(S) \;&\leq\; c_1 W_{\text{lru}}(S) \;\leq\; c_1(c_2 W_{\text{opt}}(S) + f_2(S)) \\ &\leq\; c_1(c_2(c_3 T_{\text{opt}}(S) + f_3(S)) + f_2(S)) \;\leq\; c T_{\text{opt}}(S) + f(S), \end{aligned}$$

where $c = c_1 c_2 c_3 = 16$ and $f(S) = c_1 c_2 f_3(S) + c_1 f_2(S)$. This completes the proof of Theorem 10.