# Self-Adjusting Grid Networks

Chen Avin[a], Ingo van Duijn[b,*], Maciej Pacut[c], Stefan Schmid[c]

[a]*School of Electrical and Computer Engineering, Ben Gurion University of the Negev*
[b]*Department of Computer Science, Aalborg University*
[c]*Faculty of Computer Science, University of Vienna*

**Abstract**

Emerging networked systems become increasingly flexible, reconfigurable, and "self-∗". This introduces an opportunity to adjust networked systems in a demand-aware manner, leveraging spatial and temporal locality in the workload for *online* optimizations. However, it also introduces a tradeoff: while more frequent adjustments can improve performance, they also entail higher reconfiguration costs. This paper studies self-adjusting *grid* networks in which frequently communicating nodes (e.g., virtual machines) are moved topologically closer in an online and demand-aware manner, striking a balance between the benefits and costs of reconfigurations. We show that the underlying algorithmic problem can be seen as a generalization of the classic dynamic list update problem known from self-adjusting data structures: in a network, requests can occur between *node pairs*. This pairwise optimization turns out to be significantly harder than the classical problem it generalizes. Our main result is a general $\Omega(\log n)$ lower bound even in a scenario where not only the self-adjusting network topology forms a grid but also the communication pattern (the demand graph); hence, in principle, in this scenario the demand can be served at constant cost, once it is learned. To demonstrate the challenge of adapting a network to pair-wise communication requests, we also discuss the 1-dimensional grid in more details and present an online algorithm that is $O(\log n)$-competitive in this setting.

*Keywords:* Self-∗, self-adjusting data structures, self-adjusting networks, competitive analysis, distributed algorithms, communication networks.

## 1. Introduction

Communication networks are becoming increasingly flexible, along three main dimensions: routing (enabler: software-defined networking), embedding (enabler: virtualization), and topology (enabler: reconfigurable optical technologies). In particular, the possibility to quickly reconfigure communication networks, e.g., by migrating (virtualized) communication endpoints (9) or by reconfiguring the (optical) topology (16; 11), allows these networks to become *demand-aware*: i.e., to adapt to the traffic pattern they serve, in an online and "self-∗" manner. In particular, in a *self-adjusting* network, frequently communicating node pairs can be moved *topologically closer*, saving communication costs (e.g., bandwidth, energy) and improving performance (e.g., latency, throughput).

However, today, we still do not have a good understanding yet of the algorithmic problems underlying self-

*Corresponding author: ingovanduijn@gmail.com

adjusting networks. The design of such algorithms faces several challenges. In particular, as the demand is often not known ahead of time, *online* algorithms are required to react to changes in the workload in a clever way; ideally, such online algorithms are "competitive" even when compared to an optimal offline algorithm which knows the demand ahead of time. Furthermore, online algorithms need to strike a balance between the benefits of adjustments (i.e., improved performance and/or reduced costs) and their costs (i.e., frequent adjustments can temporarily harm consistency and/or performance, or come at energy costs).

The vision of self-adjusting networks is reminiscent of self-adjusting data structures such as *self-adjusting lists* and *splay trees*, which optimize themselves toward the workload. In particular, the *dynamic list update problem*, introduced already in the 1980s by Sleator and Tarjan in their seminal work (22), asks for an online algorithm to reconfigure an unordered linked list data structure, such that a sequence of lookup requests is served optimally and at minimal reconfiguration costs (i.e., pointer rotations). It is well-known that a simple *move-to-front* strategy, which immediately promotes each accessed element to the front of the list, is *dynamically optimal*, that is, has a constant competitive ratio.

This paper initiates the study of a most basic self-adjusting *grid network*, which can be seen as a generalization of the dynamic list update problem, along two dimensions: first, instead of considering a 1-dimensional list, we consider a $d$-dimensional grid; second and more importantly, while data structures serve requests originating from the front of the list (the "root") to access data items, networks serve *communication* requests between *pairs of nodes*. The objective in this *pairwise* generalization, is to move nodes (e.g., virtual machines) which currently communicate frequently, closer to each other, while accounting for reconfiguration costs.

### *1.1. Formal Model*

We study the design of a self-adjusting network which optimizes itself toward the pairwise communication requests it serves. At the core of this model is a set $V$ of communicating nodes and a static network $N$. One can think of the nodes in $V$ as virtual machines, and the nodes of $N$ as physical hosts; the virtual machines are hosted on the nodes of $N$ (one virtual machine per host), and their communication is facilitated by the network links (edges) of $N$. An embedding (injection) of $V$ into the nodes of $N$ is called a *host configuration* and is denoted $h$; the set of all configurations is denoted $C_{V \hookrightarrow N}$.

A sequence of communication requests $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_m)$ is called *demand*, and can be served at a cost of the sum of its constituent requests. Specifically, a communication request is a pair of communicating nodes, and a configuration $h \in C_{V \hookrightarrow N}$ can *serve* a communication request $(u, v) \in V \times V$ at *cost* $d_N(h(u), h(v))$, where $d_N$ denotes the (hop) distance in $N$. See Figure 1 for an illustration of a network with communicating nodes configured. The cost of serving communication $(4, 6)$ in the figure's configuration is 2. Note that the demand (dashed lines) in the figure can actually be configured so that every communication request can be served at cost 1.

In our model, we are interested in minimizing the cost of the demand by allowing *host reconfigurations*. A recon-

figuration is a transition from a configuration $h$ to $h'$ by a sequence of *host migrations*, where one migration comprises two neighboring network nodes exchanging their embedded communicating nodes; the cost of a reconfiguration is the minimal number of migrations necessary to implement it. In summary, this yields:

**Definition 1.** Given a finite demand $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_m)$ and a sequence of configurations $h_0, h_1, \ldots, h_m \in C_{V \hookrightarrow N}$. The cost of serving $\sigma$ is the sum of serving each $\sigma_i$ in $h_i$ plus the reconfiguration cost between subsequent configurations $h_i, h_{i+1}$.

*The Pairwise Grid Update Problem.* In particular, we study the problem of designing a self-adjusting *grid network*: a network $N$ whose topology forms a $d$-dimensional grid, or *$d$-grid*, for which we use the following notation:

- A $d$-grid is a graph $N = (V_N, E)$ where $V_N \subset \mathbb{N} \times \cdots \times \mathbb{N}$, with $E = \{((n_1, \ldots, n_d), (m_1, \ldots, m_d)) \mid \exists i \text{ s.t. } |n_i - m_i| = 1 \text{ and } \forall j \neq i, n_j = m_j\}$.
- The *length* of a grid is its largest dimension, i.e., $\max_{(n_1,\ldots,n_d) \in V} n_i$.
- A *subgrid* of a $d$-grid is any subset of the grid which forms a $d'$-grid, with $1 \leq d' \leq d$

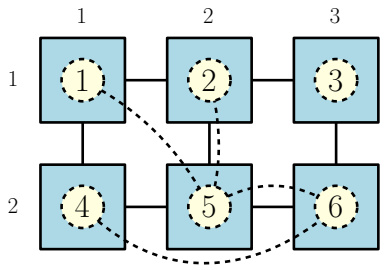



Figure 1: A $2 \times 3$ grid network $N$ (solid squares), with communicating nodes (dashed circles) configured on it.

**Definition 2** (Pairwise Grid Update)**.** Let $V$, $h$, and $\sigma$ be as before, with $N$ representing the $d$-dimensional grid (for some constant $d$), with all dimensions having length $n^{1/d}$.

The cost of serving a request $\sigma_i = (u, v) \in \sigma$ is given by $d_N(h(u), h(v)) = \|h(u), h(v)\|_1$, where $\|\cdot\|_1$ is the $\ell_1$ norm (taxicab distance).

As a first step, we in this paper consider the Pairwise Grid Update problem for the case where the demand has the same topology as the network: the communication requests between nodes also form a grid topology. We will refer to this graph as the *demand graph* or, synonymously, *request graph*. More formally, to denote the demand as a graph, we let $E_i = \{\sigma_1, \ldots, \sigma_i\}$ be the first $i$ requests of $\sigma$ interpreted as a set of edges on $V$, so that the request graph $R(\sigma) = (V, E_m)$ models the entire demand $\sigma$.

Since the self-adjusting network and the demand have the same topology, in principle, there exists a configuration of $V$ into the grid network such that any request can be served at cost 1. However, initially, the embedding may be arbitrarily far from optimal, and hence needs to be learned.

*Online Competitive Ratio.* Recall that the cost incurred by an algorithm A on $\sigma$ is the sum of communication and reconfiguration costs. In the realm of online algorithms and competitive analysis, we compare an online algorithm *ON* to an offline algorithm *OFF* which has complete knowledge of $\sigma$ ahead of time. We want to devise online algorithms *ON* which minimize the competitive ratio $\rho$:

$$\rho = \max_{\sigma} \frac{\text{cost}(ON(\sigma))}{\text{cost}(OFF(\sigma))}$$

### 1.2. Related Work

One important area of related work arises in the context of the dynamic list update problem. Since the groundbreaking work by Sleator and Tarjan on amortized analysis and self-adjusting data structures (22), researchers have explored many interesting variants of self-adjusting data structures, also using randomized algorithms (20), studying the power of lookaheads (1; 3), or devising offline algorithms (5; 19). The deterministic Move-To-Front (MTF) algorithm is known to optimally solve the standard formulation of the list update problem: it is constant competitive (22; 4). To the best of our knowledge, the exact competitive ratio in the randomized setting (against an oblivious adversary) is still an open problem (3; 23). The state-of-the-art randomized algorithm (3) makes an initial random choice between two known algorithms that have different worst-case request sequences, relying on the BIT (20) and TIMESTAMP (2) algorithms.

We also note that the self-adjusting network design problem for pairwise requests can be considered a special case of general online problems such as the online Metrical Task System (MTS) problems. However, given the exponential number of possible configurations, the competitive ratio of generic MTS algorithms will be high if applied to our more specific problem (at least according to the existing bounds). Furthermore, we note that in case the demand graph and the network have the same topology, the problem can be seen as a learning problem and is hence related to bandits theory (13); however, in our model, reconfigurations come at a cost.

There also exists work on self-adjusting networks whose (bounded-degree) topology can be adjusted, e.g., (8; 10; 21; 18; 15). However, these approaches cannot be applied in our context, where only the mapping between the virtual machines and the physical servers can be changed, but not the network itself.

The paper closest to ours are by Avin et al. (6; 7) and by Olver et al. (17). In (6), the authors consider a problem related to self-adjusting grid networks, for a special type of input sequences which are chosen *i.i.d.* from a given pairwise distribution, namely a symmetric product distribution. For this model, the authors propose a local, distributed algorithm that is constant competitive for sufficiently long sequences; they also show that the problem is NP-hard. In (7), the authors present different swapping heuristics for migrating virtual machines on a hypercubic networks, aiming to aggressively collocate communicating nodes. Olver et al. (17) introduced the Itinerant List Update (ILU) problem: a relaxation of the classic dynamic list update problem in which the pointer no longer has to return to a home location after each request. The authors present an $\Omega(\log n)$ lower bound on the randomized competitive ratio and also describe a polynomial-time offline algorithm and prove that it achieves an approximation ratio of $O(\log^2 n)$. In contrast, we in our paper focus on online algorithms and demand graphs (where the edges are communication requests) which form a grid. In fact, we show that the lower bound $\Omega(\log n)$ even holds in this restrictive case, at least for deterministic algorithms. We also present an online algorithm which matches this bound in our model.

*Bibliographic note.* A preliminary version of this paper was presented at SSS 2019 (12). For this journal version we significantly revised the presentation, and also include additional technical results such as an NP-

hardness proof.

### 1.3. Contributions

This paper initiates the study of a class of basic self-adjusting networks ($d$-grids), which optimize themselves toward the dynamically changing demand (restricted to the same topology), while amortizing reconfiguration cost. The underlying algorithmic problem is natural and motivated by emerging reconfigurable communication networks where virtual machines can be migrated in a demand-aware manner. In the special case of a 1-dimensional grid (a line), the problem can also be seen as a pairwise generalization of the fundamental dynamic list update problem. Our main result is a negative one: we show that unlike the classic dynamic list update problem, which admits for constant-competitive online algorithms, there is an $\Omega(\log n)$ lower bound on the competitive ratio of any deterministic online algorithm for the Pairwise Grid Update problem, even if the demand forms the same grid topology as the physical network. We further show that the offline variant of the problem is NP-complete. We also demonstrate the challenges of designing online algorithms by presenting an online algorithm which is $O(\log n)$-competitive for long enough sequences on 1-grids (i.e., *lines*), where the demand has the same topology.

### 1.4. Organization

The remainder of this paper is organized as follows. In Section 2, we put the problem and its challenges into perspective with respect to the list update problem. In Section 3 we derive the lower bound for Pairwise Grid Update. Then, in Section 4 we present the upper bound for Pairwise List Update and the NP-completeness of its offline variant. We conclude in Section 5.

## 2. From Classic to Pairwise List Update

To provide an intuition of the challenges involved in designing online algorithms for Pairwise Grid Update problems and to put the problem into perspective, we first revisit the classic list update problem and then discuss why similar techniques fail if applied to communicating node *pairs*, i.e., scenarios where requests do not come from fixed points in the network (e.g. the head of a list network).

The *(dynamic) list update problem* (22) introduced by Sleator and Tarjan over 30 years ago is one of the most fundamental and oldest online problems: Given a set of $n$ elements stored in a linked list, how to update the list over time such that it optimally serves a request sequence $\tau = (\tau_1, \tau_2, \ldots)$ where for each $i$, $\tau_i \in V$ is an arbitrary element stored in the list? The cost incurred by an algorithm is the sum of the access costs (i.e. scanning from the *front* of the list to the accessed element) and the number of migrations (*swaps* in their terminology). As accesses to the list elements start at the front of the list, it makes sense to amortize high access costs by moving frequently accessed elements closer to the front of the list. In fact, the well-known *Move-To-Front* (MTF) algorithm even moves an accessed element to the front *immediately*, and is known to be *constant competitive*: its cost is at most a constant factor worse than that of an optimal offline algorithm which knows the entire sequence $\tau$ ahead of time (22). Throughout the literature, slightly different cost models have been used for the list update problem. Generally, a *cursor* is located at the head of the list at each request. Then, the algorithm can perform two operations, each operation incurring unit cost. i) *Move* the cursor to the left, or to the right, one position; the element in the new position is referred to

as *touched*. ii) *Swap* the element at the cursor with the element one position to the left or right; the cursor also moves.

In the 1-dimensional pair-wise update problem, upon a request $\sigma_i = (s_i, t_i)$, the cursor is placed at $s_i$ instead of the head of the list, and $t_i$ needs to be looked up. To demonstrate the significance of this difference, we first present a paraphrased version of the proof by Tarjan and Sleator showing the dynamic optimality of MTF. After that, we showcase a simple access sequence differentiating the two problems.

### 2.1. Expositional Proof for Optimality of MTF

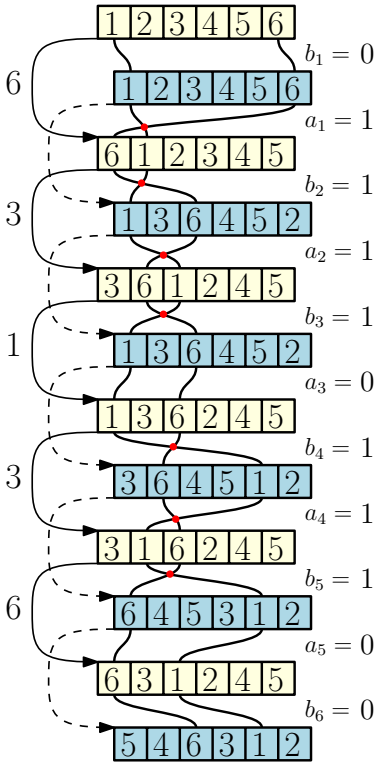



Figure 2: MTF (yellow) and $A$ (blue) on $\tau = 6, 3, 1, 3, 6$

While the potential argument used to show dynamic optimality of the move-to-front strategy for the list access problem yields a very elegant and succinct proof (22), it lacks intuition which makes it difficult to generalize the argument. The key idea in the potential argument is to compare the execution of MTF to the execution of an arbitrary algorithm $A$. The algorithm is fixed for the analysis, but any valid algorithm can be used, e.g. the optimal offline algorithm. The state (represented by a list) of MTF and $A$ are juxtaposed at every access, comparing how the order of elements in both lists differ. In fact, it is sufficient to only consider the relative order of two arbitrary but fixed elements, call them $u$ and $v$. Consider the order of $u$ and $v$ in the state of $A$ before it performs the $i$th access. If this order is the same as in MTF *before* it performs the $i$th access, let $b_i = 0$ and otherwise $b_i = 1$. Similarly, if their relative order is the same in MTF *after* its *ith* access, let $a_i = 0$ and otherwise $a_i = 1$. This describes an inversion sequence $b_1 a_1 b_2 a_2 \dots b_m a_m$. Figure 2 illustrates this for MTF and an arbitrarily chosen algorithm $A$ on a sequence $\tau = 6, 3, 1, 3, 6$, with the inversions of 1 and 6 described by the sequence 01111011100.

Suppose that $\tau_i \in \{u, v\}$ and that MTF touches $u$ and $v$ while accessing $\tau_i$. The proof by Tarjan and Sleator boils down to three observations.

**Observation 1.** MTF inverts $u$ and $v$ relative to $A$ by accessing $\tau_i$, i.e. $b_i \neq a_i$.

**Observation 2.** If $b_i = 0$, MTF and $A$ agree on the order of $u$ and $v$ before $\tau_i$. Since MTF touches both, $A$ also touches both in order to access $\tau_i$.

**Observation 3.** For $b_i = 1$, let $j < i$ be the largest index such that $b_j = 0$ or $a_j = 0$ (note that $j$ exists because $b_1 = 0$). When $a_j = 0$, and thus $b_{j+1} = 1$, $A$ inverts $u$ and $v$ and therefore must have touched both. When $b_j = 0$, and thus $a_j = 1$, MTF inverts $u$ and $v$ and one of them is $\tau_j$. By Observation 2, if $b_j = 0$ and MTF touches

$u$ and $v$ to access $\tau_j$, then A does as well.

The last observation is essentially the amortized argument rephrased as a charging argument. We can now easily prove the dynamic optimality of MTF.

**Theorem 1** (Tarjan & Sleator)**.** MTF is 4-competitive.

*Proof.* We prove that for all $\tau_i = v$ where MTF touches $u$, there is a move by A touching $u$. MTF first moves the cursor to $\tau_i$, and then swaps $\tau_i$ to the front. Along the way it touches $u$ twice, once with a move and once with a swap, incurring a cost of 2.

For $b_i = 0$ (resp. $b_i = 1$), we use Observation 2 (resp. 3) to charge the cost to A touching $u$ while accessing $\tau_i$ (resp. $\tau_j$). By Observation 1, $b_i \neq a_i$, and thus for any $\tau_k \in \{u, v\}$ with $i < k$, the largest index $j' < k$ with $b_{j'} = 0$ or $a_{j'} = 0$ must be at least $i$, and therefore $j < i \leq j'$. This guarantees that MTF charges at most a cost of 4 to one move of A. Since all the cost incurred by MTF is charged to some move of A, the claim follows. □

In the original work by Tarjan and Sleator, MTF is shown to be 2-competitive. This is because their cost model allows accessed elements to be moved to the front 'for free'. If we allow this as well, the cursor touches $u$ only once to access $v$, resulting in a factor 2.

### 2.2. *The Challenge of* Pairwise List Update

Generalizing dynamic list update to pairwise requests introduces a number of challenges already present in 1-dimensional grids, which render the problem more difficult. First, the natural inversion argument no longer works: a reference point such as the front of the list is missing in the pairwise setting. This makes it harder to relate algorithms to each other, and to perform a potential function analysis of the competitive ratio. Second, for general request graphs $R(\sigma)$ (describing the demand pattern), an online algorithm needs to be able to essentially "recognize" certain patterns over time.

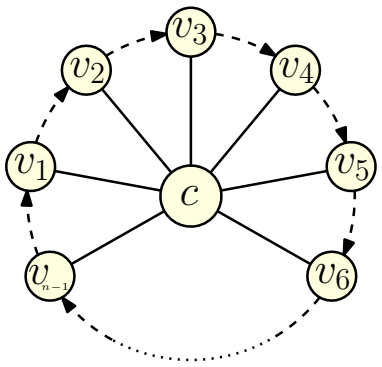



Figure 3: A star graph used to construct a cyclic sequence of requests $\sigma_c = (c, v_1), (c, v_2), \ldots, (c, v_{n-1}), (c, v_1), \ldots$

Regarding the latter, consider the set of nodes $V = \{v_1, ..., v_n\}$ and let $\tau_c$ be a cyclic sequence: for all $\tau_i, \tau_{i+1} \in \tau_c$ with $\tau_i = v_j$ and $\tau_{i+1} = v_k$ it holds that $j+1 = k \,(\mathrm{mod} n-1)$. From this we construct a similar sequence $\sigma_c$ for the pairwise problem, on the set of nodes $V \cup \{c\}$, with $\sigma_i = (c, \tau_i)$. This yields a star graph $R(\sigma_c)$ as denoted in Figure 3. An offline algorithm can efficiently serve the cyclic order by first embedding the elements in the order $v_1, ...v_k$, and then moving the element $c$ one position further after every request. If the cost of embedding the initial order is dominated by serving all requests, then the amortized cost is $O(1)$ per request (per cycle there are $n-1$ moves of cost $O(1)$ and once $c$ is moved a distance $n$). However, in the list update model, any sequence cycling through all elements is a worst-case sequence with $\Omega(n)$ per request. This demonstrates that a "dynamic cursor" can mean a factor $n$ difference in cost. What the sequence $\sigma_c$ also demonstrates, is that

aggregating elements around a highly communicative node is suboptimal; in the particular case of $\sigma_c$, it is this central node that needs to be moved.

Another pattern is a request sequence $\sigma$ that forms a connected path in the request graph $R(\sigma)$ (describing the demand). When restricted to only these patterns, the 1-dimensional pairwise problem corresponds to the *Itinerant List Update Problem* (ILU) studied in (17). In this work it is shown that deriving non-trivial upper bounds on the competitive ratio already seems notoriously hard (even offline approximation factors are relatively high). Note that the star example can be expressed as a path, i.e. $\sigma'_c = (c, v_1), (v_1, c), (c, v_2), (v_2, c), (c, v_3), \ldots$, demonstrating the significance of understanding simple request patterns for pairwise requests. This, among other, motivates us to specifically consider request graphs with a linear demand in this paper.

## 3. Lower Bound for Pairwise Grid Update

This section derives a lower bound on the competitive ratio of any algorithm for Pairwise Grid Update, where both the self-adjusting network as well as the demand feature a grid topology. The proof relies on a technical statement which can be independently understood of its application in the lower bound. We therefore first present the technical part in Section 3.1, and then present the adversarial lower bound strategy in Section 3.2.

### 3.1. Technical Proof

**Theorem 2.** Let $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ be sequences of $k$ nonnegative numbers, and let $x$ (resp. $y$) denote $\sum_{i=1}^{k} x_i$ (resp. $\sum_{i=1}^{k} y_i$). Let the *weight* of an *involution*[1] over the indices $1, \ldots, k$ be defined as $w(f) = \sum_{i=1}^{k} x_i y_{f(i)}$.

The average weight over all involutions is $\Omega(\frac{xy}{k})$.

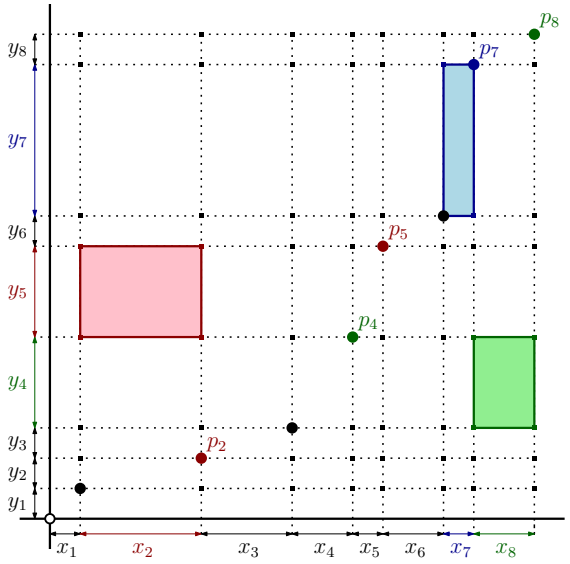



Figure 4: A staircase of 8 points based on the sequences $x_1, \ldots, x_8$ and $y_1, \ldots, y_8$. The values $w(2,5)$, $w(8,4)$, and $w(7,7)$ are visualized as the area of rectangles highlighted in red, green, and blue respectively.

*Proof.* Let $\mathcal{I}_k$ denote the set of all involutions on a set of $k$ elements, where $|\mathcal{I}_k| = T(k)$ is given by the recurrence $T(k) = T(k-1) + (k-1)T(k-2)$ with $T(0) = T(1) = 1$. For every pair of distinct indices $i, j$, there are $T(k-2)$ involutions $f \in \mathcal{I}_k$ such that $f(i) = j$ (namely for all involutions on the remaining $k-2$ indices). Similarly for every index $i$, there are $T(k-1)$ involutions such that $f(i) = i$. Thus, for every ordered pair of (not necessarily distinct) indices $i, j$ there are *at least* $T(k-2)$ involutions with $f(i) = j$.

For convenience we define a *staircase* of points $p_j = (\sum_{i=1}^{j} x_i, \sum_{i=1}^{j} y_i)$. Observe that we can subdivide the

[1] A function $f$ such that $f(f(x)) = x$ for all $x$. One can think of it as a matching that allows self-pairings.

rectangle defined by $p_k$ and the origin into $k^2$ axis-aligned rectangles, so that the area of every such rectangle corresponds to the weight of one ordered pair of indices (see Figure 4). Since every ordered pair of indices appears in at least $T(k-2)$ involutions, their weight (and thus the corresponding rectangle), contributes at least $T(k-2)$ times in the sum of weights over all involutions. This means that the area $xy$ of the complete rectangle contributes $T(k-2)$ times to that sum:

$$\begin{aligned}\frac{\sum_{f\in\mathcal{I}_k} w(f)}{|\mathcal{I}_k|} &\geq \frac{T(k-2)\cdot\sum_{i=1}^{k}\sum_{j=1}^{k} w(i,j)}{T(k)}\\ &= \frac{T(k-2)}{T(k)}\cdot xy\end{aligned}$$

To lower bound $\frac{T(k-2)}{T(k)}$, we first define $R(n) = \frac{T(n)}{T(n-1)}$ and observe that this definition is equivalent to the one in Lemma 1:

$$\begin{aligned}R(n) &= \frac{T(n)}{T(n-1)}\\ &= \frac{T(n-1)+(n-1)T(n-2)}{T(n-1)}\\ &= 1+(n-1)\frac{T(n-2)}{T(n-1)}\\ &= 1+\frac{n-1}{R(n-1)}\end{aligned}$$

Since $\frac{T(n)}{T(n-2)} = R(n)R(n-1)$, we can use Lemma 1 to lower bound $\frac{T(k-2)}{T(k)}$ by:

$$\begin{aligned}\frac{T(k-2)}{T(k)} &= \frac{1}{R(k)R(k-2)}\\ &\geq \frac{1}{(1+\sqrt{k+1})(1+\sqrt{k-1})} = \Theta\left(\frac{1}{k}\right)\end{aligned}$$

thus yielding an average weight of $\Theta(\frac{xy}{k})$ over all involutions. □

**Lemma 1.** Let $R(n) = 1 + \frac{n-1}{R(n-1)}$ with $R(1) = 1$; for all $n \geq 1$ it holds that:

$$\sqrt{n} \overset{(i)}{\leq} R(n) \overset{(ii)}{<} 1+\sqrt{n+1}$$

*Proof.* The proof is by induction on $n$, with base case $\sqrt{1} \leq R(1) < 1+\sqrt{1+1}$. From $R(n) < 1+\sqrt{n+1}$ we conclude:

$$\begin{aligned}\sqrt{n+1} &= 1+\frac{n}{1+\sqrt{n+1}}\\ &\overset{(ii)}{<} 1+\frac{(n+1)-1}{R(n)} = R(n+1)\end{aligned}$$

And from $\sqrt{n} \leq R(n)$ we conclude:

$$\begin{aligned}R(n+1) &= 1+\frac{(n+1)-1}{R(n)}\\ &\overset{(i)}{\leq} 1+\frac{n}{\sqrt{n}} = 1+\sqrt{n}\\ &< 1+\sqrt{(n+1)+1}\end{aligned}$$

□

### 3.2. Adversarial Lower Bound

**Theorem 3.** The competitive ratio $\rho = \max_\sigma \frac{\text{cost}(ON(\sigma))}{\text{cost}(OFF(\sigma))}$ for Pairwise Grid Update, with $|\sigma| = \Omega(n^{1+\frac{1}{d}})$, is at least $\Omega(\log n)$. This bound holds for arbitrarily long sequences, but if $|\sigma| = \Theta(n^{1+\frac{1}{d}})$, it even holds if the request graph is a $d$-grid.

To prove this, we consider an arbitrary online algorithm *ON* for Pairwise Grid Update. The main idea is to have an adaptive online adversary construct a sequence $\sigma_{ON}$ that depends on the algorithm *ON*. The adversary constructs $\sigma_{ON}$ so that the resulting request graph $R(\sigma_{ON})$ is a $d$-grid. Because an offline algorithm knows $R(\sigma_{ON})$ in advance, it can immediately configure it at cost $O(n^{1+\frac{1}{d}})$ and serve all requests at optimal cost of 1; since $|\sigma| = \Omega(n^{1+\frac{1}{d}})$, the configuration cost of $O(n^{1+\frac{1}{d}})$ is negligible. We show that the online algorithm is forced to essentially reconfigure its layout $\log n$ times, resulting in the desired ratio. To facilitate our analysis, we use a notion of the *distortion* of an embedding reminiscent of the one used in the Minimum Linear Arrangement (MLA) (14) problem.

**Definition 3.** Given a request graph $(V, E)$ with $E \subseteq V \times V$, let

$$E^+ = \{(u, v) \mid d_G(u, v) < \infty\}$$

denote the transitive closure of $E$.

For $h \in C_{V \hookrightarrow N}$, let $d_h(E)$ denote the *distortion* of $E$, which is defined as:

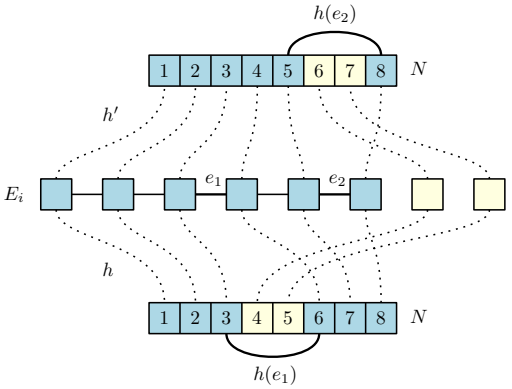
$$d_h(E) = \sum_{(u,v) \in E^+} d_N(h(u), h(v))$$

To build $\sigma_{ON}$, the adversary gradually commits to the edges of $R(\sigma_{ON})$. Having already requested $\sigma_1, \ldots, \sigma_i$, then depending on the distortion the adversary:

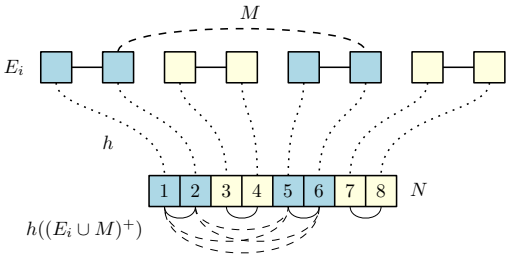
**Option** 1: requests $\sigma_{i+1} = \arg\max_{(u,v) \in E_i} d_N(h(u), h(v))$ (largest distorted previous request)

**Option** 2: reveals a new batch of edges $M \subset V \times V$

From these two options, the adversary's strategy becomes clear; Option 1 forces the highest possible cost to *ON* based on $E_i$ and $h$, and Option 2 introduces new communication edges to force an increase in distortion. What is left to show is how the value of $d_h(E_i)$ comes into play, and which edges the adversary reveals in Option 2. The adversary reveals at most $dn$ edges (since the final request graph is a $d$-dimensional grid), and the size of every subsequent batch of edges is essentially halved, resulting in $\Theta(\log n)$ batches. After each batch, for *ON* to remain optimal it must permute its layout at cost $\Omega(n^{1+\frac{1}{d}})$, totaling a cost of $\Omega(n^{1+\frac{1}{d}} \log n)$ for all batches combined. To ensure that $R(\sigma_{ON})$ is a subgrid, the partial request graph $E_i$ (i.e., the set of revealed edges) always comprises a set of disjoint subgrids. Therefore, the adversary only reveals sets of edges that concatenate two subgrids in $E_i$ along their length. Initially $E_i$ is empty and the corresponding subgrids are all singleton sets of $u \in V$.




(a) Two embeddings $h$ and $h'$ of a set of edges. Even though both embeddings embed only a single edge suboptimally, the distortion of $d_h(E)$ is bigger than $d_{h'}(E)$ because more paths in $E_i$ cross $e_1$ than $e_2$.




(b) A visualization of $d_h(E_i \cup M)$: the list graph N, $E_i$ (solid) and $M$ (dashed) are sets of edges, configured on $N$ by $h$ (dotted). The sum of length of the configured edges $h((E_i \cup M)^+)$ is the distortion $d_h(E_i \cup M)$.

Figure 5: Illustrations of distortion on a 1-grid (list)

To help decide which edges to reveal, we use the distortion to associate a cost to batches of edges that the adversary can commit to. Let $M \subseteq V \times V \setminus E_i$ be any set of edges such that the graph $(V, E_i \cup M)$ comprises a set of disjoint subgrids. For a configuration $h$ of *ON*, the set $M$ induces a distortion of $d_h(E_i \cup M)$, as shown in Figure 5b. We show that for any embedding that *ON* chooses, the adversary can find a set $M$ so that the distortion is large.

**Lemma 2.** Let $N$ be a $d$-grid graph, and $E \subseteq V \times V$ a set of edges so that the graph $G = (V, E)$ induces $k$ disjoint subgrids of the same shape. For every $h \in C_{V \hookrightarrow N}$, there exists a set $M \subseteq V \times V$ such that $d_h(E \cup M) = \Omega(\frac{n^{2+\frac{1}{d}}}{k})$ and $(V, E \cup M)$ contains a set of at least $k/2$ disjoint

subgrids of the same shape.

To prove this lemma, we use Theorem 2.

*Lemma 2.* Let $L_1, ..., L_k \subseteq E$ be the uniform subgrids in $G$. For all pairs $(i, j)$, let $(L_i, L_j)$ denote any set of edges so that $L_i \cup L_j \cup (L_i, L_j) = L_i \oplus L_j$ is a concatenation of of $L_i$ and $L_j$ along their length, which is well defined since the subgrids have the same shape (if $L_i = L_j$ then $L_i \oplus L_j = \emptyset$). For any involution $f$ on the subgrids we have:

$$2d_h(E \cup \{(L_i, L_{f(i)}) \mid i \neq f(i)\}) \geq \sum_{i=1}^{k} d_h(L_i \oplus L_{f(i)}). \quad (1)$$

The factor 2 is necessary because for $i$ such that $i \neq f(i)$, the term $d_h(L_i \oplus L_{f(i)})$ appears twice in the sum. Now partition $N$ into three subgrid slabs: a bottom slab $X = \{(n_1, ..., n_d) \in N \mid n_1 \leq \lceil n/3 \rceil\}$, a top slab $Y = \{(n_1, ..., n_d) \in N \mid \lfloor 2n/3 \rfloor \leq n_1\}$, and the centre slab $C = N \setminus (X \cup Y)$. Let $h_X(L_i)$ (resp. $h_Y(L_i)$) denote the number of elements of $L_i$ that $h$ maps onto $X$ (resp. $Y$). Every two vertices $u, v$ so that $h(u) \in X$ and $h(v) \in Y$ are by construction at least $\Theta(n^{\frac{1}{d}})$ apart in $N$, and therefore we can lower bound $d_h(L_i \oplus L_j)$ by:

$$d_h(L_i \oplus L_j) \geq \Theta(n^{\frac{1}{d}}) \cdot h_X(L_i) h_Y(L_j) \quad (2)$$

For an involution $f$ drawn uniformly at random, Theorem 2 gives us a bound on the expected value of the following:

$$\mathbf{E}\left(\sum_{i=1}^{k} h_X(L_i) h_Y(L_{f(i)})\right) = \Omega\left(\frac{\lceil n/3 \rceil^2}{k}\right) \quad (3)$$

Therefore, there exists an involution $f$ for which we have:

$$\begin{aligned} 2d_h(E \cup \bigcup_i (L_i, L_{f(i)})) &\overset{(1)}{\geq} \sum_{i=1}^{k} d_h(L_i \oplus L_{f(i)}) \\ &\overset{(2)}{\geq} \Theta(n^{\frac{1}{d}}) \cdot \sum_{i=1}^{k} h_X(L_i) h_Y(L_{f(i)}) \\ &\overset{(3)}{=} \Theta(n^{\frac{1}{d}}) \cdot \Omega(n^2/k) = \Omega\left(\frac{n^{2+\frac{1}{d}}}{k}\right) \end{aligned}$$

Since this holds for any choice of $(L_i, L_j)$, we can pick them so that $(V, E \cup \bigcup_i (L_i, L_{f(i)})) \mid i \neq f(i)\})$ contains at least $k/2$ disjoint subgrids with the same shape.

□

This lemma (and the proof) partially reveals how the adversary commits to a new batch of edges in Option 2; it can essentially pick a random matching between subgrids and concatenate them so they form bigger subgrids. For a clean scheme of building up subgrids, we assume w.l.o.g. that $n = 2^{dp}$. This means that the size $n^{\frac{1}{d}}$ of the grid is $2^p$, meaning that the adversary can perfectly build up hypercube-shaped subgrids of size $2^1, 2^2, ..., 2^p$. However, a hypercube comprises $2^d$ half-sized hypercubes, whereas Lemma 2 matches *pairs* of subgrids. Since a hypercube needs to be cut in $d$ dimensions to yield said half-sized hypercubes, we can conversely also build up the hypercubes of size $2^{i+1}$ by $d$ rounds of concatenating subgrids composed of hypercubes of size $2^i$.

Next we show the precondition for the adversary to opt for Option 1, including a lower bound on the corresponding cost imposed on *ON*.

**Lemma 3.** Let $N$ be a $d$-grid, $h \in C_{V \hookrightarrow N}$ a configuration, and $E \subseteq V \times V$ a set of edges so that the graph $G = (V, E)$ has $n/\ell$ disjoint subgrids of size $\ell$.

If $d_h(E) = \Omega(\ell n^2)$, then there exists an edge $(u, v) \in E$ such that $d_h(u, v) = \Omega(n/\ell)$.

*Proof.* There are $n/\ell \cdot \binom{\ell}{2} = O(\ell n)$ distinct connected pairs of vertices in $G$, meaning that the average distortion of the shortest path between each pair is $\frac{\Omega(\ell n^2)}{O(\ell n)} = \Omega(n)$. The highest distortion is at least the average, and every path in $G$ has length at most $\ell$. On this path, there must exist an edge with distortion $\Omega(n/\ell)$, since if all edges have a distortion of $o(n/\ell)$, the total would be $o(n)$. □

Combined, Lemma 2 and Lemma 3 imply that the adversary can either request an edge at cost $\Omega(n/\ell)$, or increase the distortion to $\Omega(\ell n^2)$ by revealing a new batch of edges. The final ingredient is a lower bound on how much cost the adversary can impose on *ON* in between these batches.

**Lemma 4.** Let $N$ be a $d$-grid, and $E \subset V \times V$ a set of communication edges forming disjoint subgrids. If $h, h' \in C_{V \hookrightarrow N}$ are two embeddings that differ only in the order of two adjacent elements $u$ and $v$, then $d_h(E) \leq d_{h'}(E) + 2\ell$, where $\ell$ is the size of the largest sublist in $E$.

*Proof.* Consider all shortest paths in $E$ that end in $u$. At most $\ell$ paths ending in $u$ (or $v$) are reduced by 1, and therefore $d_h(E)) - d_{h'}(E) \leq 2\ell$. □

Combining the previous lemmata, we can prove the main technical result.

**Lemma 5.** For every online algorithm A, there is a sequence $\sigma_{ON}$ of length $\Theta(n^{1+\varepsilon})$ such that $\text{cost}(ON(\sigma_{ON})) = \Omega(\varepsilon n^{1+\frac{1}{d}} \log n)$, for $0 < \varepsilon \leq 1$. Furthermore, the resulting request graph $R(\sigma_{ON})$ is a subgrid.

*Proof.* W.l.o.g. assume that $n = 2^{dp}$ for some integer $p$ so that the hypercube concatenation scheme described earlier can be employed.

Consider the situation right after a batch of edges is revealed, where all subgrids have cardinality $\ell$. By Lemma 2 this implies that the distortion is $\Omega(\ell n^2)$. Let $\sigma = \sigma_i, \sigma_{i+1}, ..., \sigma_{i+\ell n}$ be the requests obtained by repeatedly requesting the edge in $E_i$ with largest distortion. There are two situations:

- Throughout serving $\sigma$, the distortion is always at least $\Omega(\ell n^2)$. Then by Lemma 3 each $\sigma_j$, $i \leq j \leq i + \ell n$ incurred a cost of $\Omega(n/\ell)$, at total cost $\Omega(n^2)$.
- By serving $\sigma$, *ON* halves the distortion, thus reducing it by at least $\Omega(\ell n^2)$. Then, since by Lemma 4 every swap reduces the distortion by at most $2\ell$, *ON* must have used at least $\Omega(n^2)$ swaps.

This argument holds for each batch of edges revealed. The adversary stops when the subgrids have cardinality $2^{\varepsilon \log n}$, yielding a sequence $\sigma_{ON}$ with

$$|\sigma_{ON}| = \sum_{\ell \in \{2^0, ..., 2^{\varepsilon \log n}\}} \ell n = \Theta(n^{1+\varepsilon})$$

and $cost(\sigma_{ON}) = \Omega(\varepsilon n^{1+\varepsilon} \log n)$. By Lemma 3, the adversary only requests edges that are introduced using the matching from Lemma 2. Any set of edges introduced by the latter Lemma concatenates two already existing subgrids, hence $R(\sigma_{ON})$ is a subgrid. □

To wrap up the proof for Theorem 3, we conclude by showing that for any online algorithm *ON*, the sequence $\sigma_{ON}$ can be solved in $\Theta(n^{1+\frac{1}{d}})$ by an optimal offline algorithm.

*Proof of Theorem 3.* Let *ON* be any online algorithm solving Pairwise Grid Update. Apply Lemma 5 with

$\varepsilon = 1/d$, yielding $\text{cost}(ON(\sigma_{ON})) = \Omega(n^{1+\frac{1}{d}} \log n)$. Since $\sigma_{ON}$ is a $d$-grid, an offline algorithm can embed this graph at (worst case optimal) cost $\Theta(n^{1+\frac{1}{d}})$, and serve every request at optimal cost $O(1)$. This yields $\text{cost}(OFF(\sigma_{ON})) = \Theta(n^{1+\frac{1}{d}})$, and thus

$$\rho = \frac{\text{cost}(ON(\sigma))}{\text{cost}(OFF(\sigma))} = \Omega(\log n)$$

In order to make this bound hold for arbitrary long sequences, we slightly modify the adversary. After every $\Theta(n^{1+\frac{1}{d}})$ requests it serves, it can reconfigure to a new grid at cost $O(n^{1+\frac{1}{d}})$, and repeat the argument to force cost of $\Omega(n^{1+\frac{1}{d}} \log n)$ to $ON$ for the subsequent $\Theta(n^{1+\frac{1}{d}})$ requests. □

## 4. Online Algorithms and Complexity

While our main contribution is the derived lower bound, we in this section also initiate the study of upper bounds, shedding light on the underlying algorithmic challenges. We present a $O(\log n)$-competitive online algorithm Gread for requests issued according to a 1-dimensional grid (i.e., a linear order), and discuss its distributed implementation. We also prove that the offline variant of Pairwise List Update, the 1-dimensional grid update problem, is already NP-complete.

### 4.1. An Upper Bound

This section presents a $O(\log n)$-competitive online algorithm for Pairwise List Update. Our main technical lemma shows that the total cost spent on learning the optimal embedding never exceeds $O(n^2 \log n)$. We propose a simple greedy algorithm that identifies a *locally optimal* embedding, and always moves towards this embedding. Observe that a set of $k$ sublists can be embedded perfectly on a line graph in at most $2^k k!$ ways (they are permuted in some order, and every list has at most two orientations). Given a configuration $h \in C_{V \hookrightarrow N}$ of the lists, we define the locally optimal embedding to be an optimal embedding one that takes the fewest number of reconfigurations to reach, starting at $h$. Formally, if $opt(E)$ is the set of optimal embeddings of a set edges, then the $h$-optimal embedding of $E$ is

$$h[E] = \arg\min_{h' \in opt(E)} \sum_{v \in V} |h(v) - h'(v)|$$

With such a configuration we associate the cost:

$$\Phi_h[E] = \sum_{v \in V} |h(v) - h[E](v)|$$

Let Gread be the algorithm (it GREedily ADjoins sublists), that upon seeing a new edge $\sigma_i$, *immediately* moves to the embedding $h[E_i \cup \{\sigma_{i+1}\}]$.

For each $E_i$, let $\mathcal{V}(E_i)$ be the connected components of $(V, E_i)$, so that $\mathcal{V}_\sigma = \cup_{1 \le i \le m} \mathcal{V}(E_i)$ is the set of all sublists induced by $\sigma$. This naturally defines a binary tree $T_\sigma = (\mathcal{V}_\sigma, E_\sigma)$: for every first occurrence $\sigma_i$ of $(u, w) \in E_m$ connecting two sublists $U, W$ in $R(E_i)$, there are two corresponding edges $(U, U \cup W)$ and $(W, U \cup W)$ in $E_\sigma$.For every $\sigma_i \in E_m$, Gread incurs some cost for reconfiguring, and the following lemma bounds this cost.

**Lemma 6.** Let $h$ be an optimal embedding of $E_i$, and let $\sigma_{i+1}$ be an edge connecting two sublists $U$ and $W$ of $E_i$. It holds that

$$\Phi_h[E_i \cup \{\sigma_{i+1}\}] \le n \cdot \min(|U|, |W|)$$

*Proof.* Since $E_i$ is optimally embedded by $h$, we simply need to move the smaller of $U$ and $W$ into its correct location so that $E_i \cup \{\sigma_{i+1}\}$ is optimally embedded. This requires every element in the smaller list to be

moved at most $n$ locations, therefore $\Phi_h[E_i \cup \{\sigma_{i+1}\}] \leq n \min(|U|, |W|)$. □

For a node $U \in \mathcal{V}_\sigma$, let left($U$) and right($U$) denote $U$'s left and right child respectively. Further, let $w(U)$ denote the number of nodes in the subtree rooted at $U$. Observe that for any binary tree with nodes $N$, it holds that

$$\sum_{v \in N} \min(w(\text{left}(v)), w(\text{right}(v))) \leq |N| \log |N|$$

**Theorem 4.** For any $\sigma$, with $|\sigma| = m$, such that $|E_m| = k$ and $R(\sigma)$ is a list,

$$\text{cost}(\textsc{Gread}(\sigma)) = O(m + nk \log k)$$

*Proof.* Let $h_i$ denote the configuration after request $\sigma_1$, and let $h_0$ denote the trivial optimal initial embedding. Then the total cost of Gread is the sum of reconfiguring after every $\sigma_i$ plus accessing every request at cost 1:

$$\begin{aligned} \text{cost}(\textsc{Gread}(\sigma)) - m &= \sum_{i=0}^{m} \Phi_{h_i}[E_i \cup \{\sigma_{i+1}\}] \\ &\leq \sum_{U \in \mathcal{V}_\sigma} n \min(w(\text{left}(U)), w(\text{right}(U))) \\ &\leq nk \log k \end{aligned}$$

□

To give an example of Gread, we can consider the sequence derived for our lower bound in Section 3. When applied to this sequence, after each batch of revealed edges, Gread spends $O(n^2)$ reconfiguring to perfectly embed the current demand, yielding a total $O(n^2 \log n)$. This also means that for this sequence, Gread is $\log n$ competitive.

### 4.2. Distributed Implementation of Gread

To execute Gread in a distributed environment, we have to address several aspects: i) how to perform (local) routing, ii) how to ensure nodes know to which (temporary) sublist they belong, together with the size of the sublist, and iii) how to perform the reconfiguration and merging of two sublists. We address these issues one by one.

**Routing**: The basic problem with routing is that the source nodes do not know the location of the destination, since initially there is no *sense of direction*. To overcome this problem each source initiates an exponential search on *both* sides of the line network when it first needs to communicate with a destination. This will guarantee that the cost of the *first* route request will be $O(i)$ for a destination that is $i$ hops away on the line network. Note that this is proportional to the cost of any algorithm. According to Gread the cost of all future requests will be 1 since after the first communication request the source and destination are reconfigured to be neighbors.

**Sublist**: During the execution each node maintains the following information: A bit that indicates if it is at the *end* of a sublist (a node is at the end of a sublist if it has less than two neighbors from that sublist). If it is at the end of the list then the node maintains the size of the list (up to $\log n$ bits).

**Reconfiguration**: Basically Gread merges two sublists by swapping the shorter list toward the longer sublist. Note that this happens only on the *first* routing request from a source to destination. This can be done in a distributed manner in the following way. On the first routing request, the source (which must be an *end* node)

attaches the size of its sublist to the message. The destination (which also must be an *end* node), upon receiving the request, answers to the source with the size of its own sublist (initially set to one). It is then clear to both the source and destination which sublist needs to move toward which sublist and what will be the size of the merged sublist. Then, both source and destination send messages within their sublist informing the other *ends* of the sublist of the size of the merged list. Now, w.l.o.g., assume the destination needs to move toward the source. The destination then starts performing swaps (with its neighbor that is not on its current list) toward the source. This process ends when both the destination is a neighbor of the source and the source is a neighbor of its previous neighbor on its list. Before starting the swaps the destination informs its neighbor (which in turn informs its neighbor and so on) to *follow up* after it with similar swaps. It can be observed that after this process the two list will be merged into a larger list and both *ends* will know the sizes of the new sublist. The cost of the reconfiguration is $O(n \min(|U|, |W|))$ where $U$ and $W$ are the two sublists involved in the merging.

### *4.3. NP-Completeness of the Offline Variant*

We present a reduction from the classic Minimum Linear Arrangement (MLA) problem to the offline variant of Pairwise List Update. Throughout this section, we refer to the offline Pairwise List Update as DLU. The MLA problem can be seen as a static variant of the DLU problem, where we choose a static network configuration at the beginning for free, and we serve all requests without further reconfiguration.

In (17), the authors considered the Itinerant List Update problem, which is closely related to DLU. They suggested that MLA is equivalent to Itinerant List Update, where many identical copies of the MLA graph arrive. In this section, we highlight the technical challenge for showing the equivalency to MLA (that is valid for both DLU and Itinerant List Update). Then, we introduce necessary modifications to this idea to show the NP-completeness of DLU.

The idea of repeating multiple MLA graphs faces the following technical challenge. If we are not diligent with the request order, the cost of DLU may decrease in comparison to MLA by interleaving requests with reconfigurations. To see that, consider the example of transforming of the MLA instance in form of a star with a central vertex $c$ and 5 other vertices to the instance of DLU. Two optimal MLA solutions of cost 9 exist: placing $c$ at any of two graph centers (vertices at positions 3 and 4). Assume $c$ is placed at position 3, and in the chosen edge order, the requests to the nodes at 1 and 2 precede the requests to the nodes at 5 and 6. Then, this ordering enables the optimal DLU solution to obtain a lower cost than the optimal MLA by performing a single reconfiguration. Moving $c$ to position 4 after serving requests to nodes at 1 and 2 decreases the distance to both nodes at 5 and 6 at the cost of a single reconfiguration. The optimal DLU cost is then 8, which is smaller than the optimal MLA cost 9.

We show a reduction from MLA to DLU that mitigates the problem of edge ordering entirely. To this end, we map any vertex of MLA to a group of DLU nodes that stay adjacent due to a large number of inter-group requests. We map the MLA edges to requests between the central nodes of the corresponding groups.

We repeat the MLA request pattern several times, called

rounds. We show that there exists a low-cost round where groups stay in a fixed order — and we reconstruct the solution to MLA from it. To determine the sufficient number of rounds, we introduce the following helper lemma.

**Lemma 7.** Fix any integers $A$ and $D$. Consider a sequence of non-negative integers with the average upper-bounded by $A$. We mark at most $D$ elements of the sequence as *defective*. If an element is defective, it equals 0. Then the following holds: if length of the sequence is at least $D(A + 1) + 1$, then a non-defective element no larger than $A$ exists.

*Proof.* Assume the opposite, i.e., that the sequence of length $n$, where $n \geq D \cdot (A + 1) + 1$ consists only of defective elements and elements larger than $A$. Then, the sum of the sequence is at least $(n - D) \cdot (A + 1)$ for $D \leq (n-1)/(A+1)$. This sum is at least $A \cdot n + 1$, thus the average is strictly larger than $A$, a contradiction. □

**Theorem 5.** Offline DLU is NP-complete.

*Proof.* We reduce MLA to (offline) DLU. Given any integer $k$ and any instance $I_{MLA}$ of MLA, we construct an instance $I_{DLU}$ of DLU. We show that there exists a solution to $I_{MLA}$ with cost at most $k$ if and only if there exists a solution to $I_{DLU}$ with cost at most $Thr$, where $Thr$ is bounded by a polynomial of $k$ and $|I_{MLA}|$. First, we describe the construction, and then we prove ($\Rightarrow$), followed by ($\Leftarrow$).

In the following, we describe the construction of $I_{DLU}$. Fix an integer $k$ and an instance $I_{MLA} = \langle V, E \rangle$. We denote $n = |V|$ and $m = |E|$. The construction consists of the following parts.

**Groups of vertices.** We order $V$ arbitrarily, and for each $v \in V$ we produce a group of adjacent $g$ nodes, where $g$ is the smallest odd number that exceeds $k^3$. We denote the sequence of nodes of the group by $G(v)$, and we distinguish its $(g/2 + 1)$-th (central) node by $c(v)$. In total we construct $n \cdot g$ nodes.

**Rounds of requests.** We produce the requests in $R := \lfloor (n^2 \cdot (k + 1) + 1)/(1 - 1/k) \rfloor$ rounds, each consisting of identical requests. We elaborate on the choice for $R$ later in this proof. In each round, for each edge of $MLA$ we produce a single inter-group request that we follow with a sequence of intra-group requests.

**Inter-group requests.** In each round, we order MLA edges arbitrarily, and for each edge $\{u, v\} \in E$ we produce a request $\langle c(u), c(v) \rangle$. After each such request, we issue a sequence of intra-group requests.

**Intra-group requests.** After each inter-group request in each round, we produce $S := Rk(g - 1) + n^2 g^2 + 1$ requests between each pair of adjacent vertices $\langle G(v)_{(i)}, G(v)_{(i+1)} \rangle$ in each group. These requests enforce the preservation of the structure of each group: the adjacency of consecutive nodes and the position of the central node. (The value $S$ is equal to $Thr_I + Thr_E + 1$ that we introduce in a moment.)

**The feasibility threshold.** Recall that we perform a reduction between two decision problems. The cost threshold $Thr$ for a solution to DLU consists of three parts: $Thr := Thr_I + Thr_E + Thr_S$.

1. The initial reconfiguration budget is $Thr_I := n^2 g^2$, and it allows for a full reconfiguration of all nodes.
2. The budget for serving inter-group requests is $Thr_E := Rk(g - 1)$, and it equals to $k$ times the distance between the centers of the groups, multiplied

by the number of rounds $R$.

3. The budget for intra-group requests is $Thr_S := RSn(g-1)$, and it allows for $S$ requests between adjacent nodes of each group at distance 1, multiplied by the number of rounds $R$.

First, we show the implication ($\Rightarrow$). We take any solution $S_{MLA}$ to MLA with cost at most $k$, and construct the DLU instance as follows. Before we serve any request, we reconfigure groups of nodes according to the vertex arrangement in $S_{MLA}$, and we serve all requests in this configuration. The reconfiguration cost is bounded by $Thr_I$, as we perform one reconfiguration of all nodes. The structure of groups is preserved (consecutive nodes of each group are adjacent), hence inter-group requests cost exactly $Thr_S$. If MLA incurs the cost $c_e$ for an edge $e$, then for it we incur the cost $c_e \cdot (g-1)$ in each of $R$ rounds for the corresponding request. As the cost of MLA edges sums to $k$, the cost of inter-group requests sum to $Thr_E$. In total, the constructed solution has cost $Thr$, and the claim holds.

Next, we show the implication ($\Leftarrow$). Fix a solution to $I$ of cost at most $Thr$. We say that a configuration is *well-aligned* if each pair of consecutive nodes of each group is adjacent. We say that a round is *defective*, if a group exchange happens during this round. We say that a round is *excessive*, if the cost of inter-cluster requests exceeds $k(g-1)$ during this round.

In the following, we express the necessary conditions for the existence of non-defective and non-excessive round. Then, we construct a solution to MLA from the group order in this round, and show that its cost is at most $k$.

We claim that the solution reaches a well-aligned configuration at least once during each sequence of inter-group requests. To see that, note that the minimum cost of serving inter-group requests throughout all rounds is $Thr_S$, and this is achievable by serving all requests over one hop. If the solution would not reach a well-aligned configuration, a pair of nodes requested $S$ times is not adjacent, and this additionally incurs the cost at least $S = Thr_I + Thr_E + 1$, and the solution exceeds the cost $Thr$ already.

This allows to reason about the cost of serving any intra-cluster request. Consider a round where no groups exchange positions. For this fixed order of the groups, by $p(u)$ we denote the index of the group $u$ in this order. Then, we claim that each request $\langle c(u), c(v)\rangle$ costs at least $|p(u)-p(v)|(g-1)$. If the cost would be lower, then the distance between $c(u)$ and $c(v)$ must be decreased by migrations. However, this means these must move back to guarantee the well-aligned position for the upcoming inter-group requests.

Some rounds may be defective, i.e., the solution may move to a configuration with a different group order during that round. This requires moving an entire group, and costs at least $g^2$. The defective rounds may be allowed by exchanges paid from budgets $Thr_I$ and $Thr_E$. Note that the budget $Thr_S$ is tight: the cost of inter-group requests is at least $Thr_S$.

We claim that at most $n^2 + R/k^2$ rounds may be defective (note that this is dependant on the number of rounds). At most $n^2$ of these can be paid from $Thr_I$. This can happen if the solution does not reconfigure groups at the beginning, but defers some of the exchanges to later rounds. Additionally, some exchanges may be paid from $Thr_E$: for each $k^2$ rounds, the budget $Thr_E$ allows for an addi-

tional exchange.

Now, we lower-bound the number of rounds $R$ sufficient to guarantee the existence of non-defective, non-excessive round. Note that on average, the cost of intra-group requests in each round is bounded by $k(g-1)$. We claim no cost for intra-group requests in defective rounds. Note that in a non-defective round, each intra-group request cost is divisible by $g-1$. Hence, to simplify calculations, we divide the cost of each round and the average cost by $g-1$.

To justify the choice of $R$, we use Lemma 4.3. We apply it to the sequence of (scaled) costs of rounds, and conclude that it is sufficient that the length of the round sequence ($R$) satisfies the inequality $R \geq (n^2 + R/k^2) \cdot (k+1) + 1$. We note that our choice of $R$ satisfies this criterion. This inequality also justifies the choice of $g = \Omega(k^3)$. Crucially, the rate of growth of the budget $Thr_E$ must allow for an extra reconfiguration no more frequently than every $k^2$ rounds. Otherwise, no positive $R$ would satisfy the inequality.

The solution to MLA reconstructed from the node order in a non-defective, non-excessive round has the cost at most $k$. The reconstruction is correct, as the round is non-defective, and has cost at most $k$ as the round is non-excessive.

The reduction is polynomial: the number of nodes and requests and the value $Thr$ are polynomials of $m, n$ and $k$. Finding the round without reconfigurations is linear in terms of the number of requests. □

## 5. Conclusion

We presented a lower bound of $\Omega(\log n)$ on the online competitiveness of Pairwise Grid Update, even under very restricted and seemingly simple communication patterns. We further initiated the discussion of online algorithms for such restricted scenarios and presented a deterministic $\Theta(\log n)$-competitive algorithm. We believe that our work opens several interesting directions for future research. In particular, it would be interesting to shed light on the competitive ratio achievable in more general network topologies and request patterns. It would further be interesting to study randomized algorithms and to generalize our cost model, for example also accounting for congestion aspects.

**Acknowledgments.** Research supported by the European Research Council (ERC), consolidator grant AdjustNet (grant agreement No. 864228).

## References

[1] S. Albers. A competitive analysis of the list update problem with lookahead. *Theoretical Computer Science*, 197(1-2):95–109, 1998.

[2] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.

[3] S. Albers, B. Von Stengel, and R. Werchner. A combined bit and timestamp algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995.

[4] S. Albers and J. Westbrook. Self-organizing data structures. In *Online algorithms*, pages 13–51. Springer, 1998.

[5] C. Ambühl. Offline list update is np-hard. In *European Symposium on Algorithms*, pages 42–51. Springer, 2000.

[6] C. Avin, M. Borokhovich, B. Haeupler, and Z. Lotker. Self-adjusting grid networks to minimize expected path length. In *International Colloquium on Structural Information and Communication Complexity*, pages 36–54. Springer, 2013.

[7] C. Avin, O. Dunay, and S. Schmid. Strategies for traffic-aware vm migration. In *Proc. 6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, December 2013.

[8] C. Avin, A. Hercules, A. Loukas, and S. Schmid. Towards communication-aware robust topologies. *ArXiv Technical Report*, 2017.

[9] C. Avin, A. Loukas, M. Pacut, and S. Schmid. Online balanced repartitioning. In *International Symposium on Distributed Computing*, pages 243–256. Springer, 2016.

[10] C. Avin, K. Mondal, and S. Schmid. Demand-aware network design with minimal congestion and route lengths. In *Proc. IEE INFOCOM*, 2019.

[11] C. Avin and S. Schmid. Toward demand-aware networking: A theory for self-adjusting networks. In *ACM SIGCOMM Computer Communication Review (CCR)*, 2018.

[12] C. Avin, I. van Duijn, and S. Schmid. Self-adjusting linear networks. In *Proc. 21st International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2019.

[13] S. Bubeck, N. Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[14] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys (CSUR)*, 34(3):313–356, 2002.

[15] S. Huq and S. Ghosh. Locally self-adjusting skip graphs. In *Proc. IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 805–815, 2017.

[16] M. Ghobadi et al. Projector: Agile reconfigurable data center interconnect. In *Proc. ACM SIGCOMM*, pages 216–229, 2016.

[17] N. Olver, K. Pruhs, K. Schewior, R. Sitters, and L. Stougie. The itinerant list update problem. In *13th Workshop on Models and Algorithms for Planning and Scheduling Problems*, page 163, 2017.

[18] B. Peres, O. Souza, O. Goussevskaia, S. Schmid, and C. Avin. Distributed self-adjusting tree networks. In *Proc. IEE INFOCOM*, 2019.

[19] N. Reingold and J. Westbrook. Off-line algorithms for the list update problem. *Information Processing Letters*, 60(2):75–80, 1996.

[20] N. Reingold, J. Westbrook, and D. D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[21] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking (ToN)*, 2016.

[22] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[23] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993.