Smoothed Analysis of Dynamic Networks

Michael Dinitz[1], Jeremy Fineman[2], Seth Gilbert[3], and Calvin Newport[4](✉)

[1] Johns Hopkins University, Baltimore, USA
mdinitz@cs.jhu.edu
[2] Georgetown University, Washington, DC, USA
jfineman@cs.georgetown.edu
[3] National University of Singapore, Singapore, Singapore
seth.gilbert@comp.nus.edu.sg
[4] Georgetown University, Washington, DC, USA
cnewport@cs.georgetown.edu

Abstract. We generalize the technique of *smoothed analysis* to distributed algorithms in dynamic networks. Whereas standard smoothed analysis studies the impact of small random perturbations of input values on algorithm performance metrics, dynamic graph smoothed analysis studies the impact of random perturbations of the underlying changing network graph topologies. Similar to the original application of smoothed analysis, our goal is to study whether known strong lower bounds in dynamic network models are *robust* or *fragile*: do they withstand small (random) perturbations, or do such deviations push the graphs far enough from a precise pathological instance to enable much better performance? Fragile lower bounds are likely not relevant for real-world deployment, while robust lower bounds represent a true difficulty caused by dynamic behavior. We apply this technique to three standard dynamic network problems with known strong worst-case lower bounds: random walks, flooding, and aggregation. We prove that these bounds provide a spectrum of robustness when subjected to smoothing—some are extremely fragile (random walks), some are moderately fragile / robust (flooding), and some are extremely robust (aggregation).

1 Introduction

Dynamic network models describe networks with topologies that change over time (c.f., [10]). They are used to capture the unpredictable link behavior that characterize challenging networking scenarios; e.g., connecting and coordinating moving vehicles, nearby smartphones, or nodes in a widespread and fragile overlay. Because fine-grained descriptions of link behavior in such networks are hard to specify, most analyses of dynamic networks rely instead on a worst-case

M. Dinitz—Supported in part by NSF grant #1464239.

J. Fineman—Supported in part by NSF grants CCF-1218188 and CCF-1314633.

S. Gilbert—Supported in part by Singapore MOE Tier 2 ARC project 2014-T2-1-157.

C. Newport—Supported in part by NSF grant CCF 1320279.

Y. Moses (Ed.): DISC 2015, LNCS 9363, pp. 513–527, 2015.
DOI: 10.1007/978-3-662-48653-5_34

selection of graph changes. This property is crucial to the usefulness of these analyses, as it helps ensure the results persist in real deployment.

A problem with this worst case perspective is that it often leads to extremely strong lower bounds. These strong results motivate a key question: *Is this bound* robust *in the sense that it captures a fundamental difficulty introduced by dynamism, or is the bound* fragile *in the sense that the poor performance it describes depends on an exact sequence of adversarial changes?* Fragile lower bounds leave open the possibility of algorithms that might still perform well in practice. By separating fragile from robust results, we can expand the algorithmic tools available to those seeking useful guarantees in these challenging environments.

In the study of traditional algorithms, an important technique for explaining why algorithms work well in practice, despite disappointing worst case performance, is *smoothed analysis* [16,17]. This approach studies the expected performance of an algorithm when the inputs are slightly perturbed. If a strong lower bound dissipates after a small amount of smoothing, it is considered fragile—as it depends on a carefully constructed degenerate case. Note that this is different from an "average-case" analysis, which looks at instances drawn from some distribution. In a smoothed analysis, you still begin with an adversarially chosen input, but then slightly perturb this choice. Of course, as the perturbation grows larger, the input converges to something entirely random. (Indeed, in the original smoothed analysis papers [16,17], the technique is described as interpolating between worst and average case analysis.)

In this paper, we take the natural step of adapting smoothed analysis to the study of distributed algorithms in dynamic networks. Whereas in the traditional setting smoothing typically perturbs numerical input values, in our setting we define smoothing to perturb the network graph through the random addition and deletion of edges. We claim that a lower bound for a dynamic network model that improves with just a small amount of graph smoothing of this type is fragile, as it depends on the topology evolving in an exact manner. On the other hand, a lower bound that persists even after substantial smoothing is robust, as this reveals a large number of similar graphs for which the bound holds.

Results. We begin by providing a general definition of a dynamic network model that captures many of the existing models already studied in the distributed algorithms literature. At the core of a dynamic network model is a dynamic graph that describes the evolving network topology. We provide a natural definition of *smoothing* for a dynamic graph that is parameterized with a *smoothing factor* $k \in \{0, 1, ..., \binom{n}{2}\}$. In more detail, to k-smooth a dynamic graph $\mathcal{H}$ is to replace each static graph G in $\mathcal{H}$ with a smoothed graph G' sampled uniformly from the space of graphs that are: (1) within edit distance[1] k of G, and (2) are allowed by the relevant dynamic network model. (E.g., if the model requires the graph to be connected in every round, smoothing cannot generate a disconnected graph.)

[1] Edit distance, in this paper, is the number of edge additions/deletions needed to transform one graph to another, assuming they share the same node set.

Table 1. A summary of our main results. The columns labelled "k-smoothed" assume $k > 0$. Different results assume different upper bounds on k.

	Graph	k-Smoothed Algorithm	k-Smoothed Lower Bound	0-Smoothed Lower Bound
Flooding	Connected	$O(n^{2/3} \log n / k^{1/3})$	$\Omega(n^{2/3}/k^{1/3})$	$\Omega(n)$
Hitting Time	Connected	$O(n^3/k)$	$\Omega(n^{5/2}/(\sqrt{k} \log n)$	$\Omega(2^n)$
Aggregation	Paired	$O(n)$-competitive	$\Omega(n)$-competitive	$\Omega(n)$-competitive

We next argue that these definitions allow for useful discernment between different dynamic network lower bounds. To this end, we use as case studies three well known problems with strong lower bounds in dynamic network models: flooding, random walks, and aggregation. For each problem, we explore the robustness/fragility of the existing bound by studying how it improves under increasing amounts of smoothing. Our results are summarized in Table 1. We emphasize the surprising variety in outcomes: these results capture a wide spectrum of possible responses to smoothing, from quite fragile to quite robust.

For the minimal amount of smoothing ($k = 1$), for example, the $\Omega(2^n)$ lower bound for the hitting time of a random walk in connected dynamic networks (established in [2]) decreases by an *exponential* factor to $O(n^3)$; the $\Omega(n)$ lower bound for flooding time in these same networks (well-known in folklore) decreases by a *polynomial* factor to $O(n^{2/3} \log n)$; and the $\Omega(n)$ lower bound on the achievable competitive ratio for token aggregation in pairing dynamic graphs (established in [4]) decreases by only a *constant* factor.

As we increase the smoothing factor k, our upper bound on random walk hitting time decreases as $O(n^3/k)$, while our flooding upper bound reduces more slowly as $O(n^{2/3} \log n/k^{1/3})$, and our aggregation bound remains in $\Omega(n)$ for k values as large as $\Theta(n/\log^2 n)$. In all three cases we also prove tight or near tight lower bounds for all studied values of k.

Among other insights, these results indicate that the exponential hitting time lower bound for dynamic walks is extremely fragile, while the impossibility of obtaining a good competitive ratio for dynamic aggregation is quite robust. Flooding provides an interesting intermediate case. While it is clear that an $\Omega(n)$ bound is fragile, the claim that flooding can take a polynomial amount of time (say, in the range $n^{1/3}$ to $n^{2/3}$) seems well-supported.

Full Version. Due to space constraints, we omit proofs from this extended abstract. Full details for our results can be found in the full version [6].

Next Steps. The definitions and results that follow represent a first (but far from final) step toward the goal of adapting smoothed analysis to dynamic networks. There are many additional interesting dynamic network bounds that could be subjected to a smoothed analysis. Moreover, there are many other reasonable definitions of smoothing beyond the ones herein. While our definition is natural and our results suggestive, for other problems or models other definitions might be more appropriate. Rather than claiming that our approach here is the "right" way to study the fragility of dynamic network lower bounds, we instead claim that smoothed analysis generally speaking (in all its various possible formulations) is an important and promising tool when trying to understand the fundamental limits of distributed behavior in dynamic network settings.

Related Work. Smoothed analysis was introduced by Spielman and Teng [16,17], who used the technique to explain why the simplex algorithm works well in practice despite strong worst-case lower bounds. It has been widely applied to traditional algorithm problems (see [18] for a good introduction and survey). Recent interest in studying distributed algorithms in dynamic networks was sparked by Kuhn et al. [11]. Many different problems and dynamic network models have since been proposed; e.g., [1,3,5,7–9,12,14] (see [10] for a survey). The dynamic random walk lower bound we study was first proposed by Avin et al. [2], while the dynamic aggregation lower bound we study was first proposed by Cornejo et al. [4]. We note other techniques have been proposed for exploring the fragility of dynamic network lower bounds. In recent work, for example, Denysyuk et al. [5] thwart the exponential random walk lower bound due to [2] by requiring the dynamic graph to include a certain number of static graphs from a well-defined set, while work by Ghaffari et al. [8] studies the impact of adversary strength, and Newport [14] studies the impact of graph properties, on lower bounds in the dynamic radio network model.

2 Dynamic Graphs, Networks, and Types

There is no single dynamic network model. There are, instead, many different models that share the same basic behavior: nodes executing synchronous algorithms are connected by a network graph that can change from round to round. Details on how the graphs can change and how communication behaves given a graph differ between model types.

In this section we provide a general definition for a dynamic network models that captures many existing models in the relevant literature. This approach allows us in the next section to define smoothing with sufficient generality that it can apply to these existing models. We note that in this paper we constrain our attention to *oblivious* graph behavior (i.e., the changing graph is fixed at the beginning of the execution), but that the definitions that follow generalize in a straightforward manner to capture adaptive models (i.e., the changing graph can adapt to behavior of the algorithm).

Dynamic Graphs and Networks. Fix some node set V, where $n = |V|$. A *dynamic graph* $\mathcal{H}$, defined with respect to V, is a sequence $G_1, G_2, ...$, where each $G_i = (V, E_i)$ is a graph defined over nodes V. If this is not an infinite sequence, then the *length* of $\mathcal{H}$ is $|\mathcal{H}|$, the number of graphs in the sequence. A *dynamic network*, defined with respect to V, is a pair, $(\mathcal{H}, C)$, where $\mathcal{H}$ is a dynamic graph, and C is a *communication rules* function that maps *transmission patterns* to *receive patterns*. That is, the function takes as input a static graph and an assignment of messages to nodes, and returns an assignment of received messages to nodes. For example, in the classical radio network model C would specify that nodes receive a message only if exactly one of their neighbors transmits, while in the $\mathcal{LOCAL}$ model C would specify that all nodes receive all messages sent by their neighbors. Finally, an *algorithm* maps process definitions to nodes in V.

Given a dynamic network $(\mathcal{H}, C)$ and an algorithm $\mathcal{A}$, an execution of $\mathcal{A}$ in $(\mathcal{H}, C)$ proceeds as follows: for each round r, nodes use their process definition according to $\mathcal{A}$ to determine their transmission behavior, and the resulting receive behavior is determined by applying C to $\mathcal{H}[r]$ and this transmission pattern.

Dynamic Network Types. When we think of a dynamic network model suitable for running executions of distributed algorithms, what we really mean is a combination of a description of how communication works, and a set of the different dynamic graphs we might encounter. We formalize this notion with the concept of the *dynamic network type*, which we define as a pair $(\mathcal{G}, C)$, where $\mathcal{G}$ is a set of dynamic graphs and C is a communication rules function. For each $\mathcal{H} \in \mathcal{G}$, we say dynamic network type $(\mathcal{G}, C)$ contains the dynamic network $(\mathcal{H}, C)$.

When proving an upper bound result, we will typically show that the result holds when our algorithm is executed in any dynamic network contained within a given type. When proving a lower bound result, we will typically show that there exists a dynamic network contained within the relevant type for which the result holds. In this paper, we will define and analyze two existing dynamic network types: *(1-interval) connected networks* [7,9,11,12], in which the graph in each round is connected and C describes reliable broadcast to neighbors in the graph, and *pairing networks* [4], in which the graph in each round is a matching and C describes reliable message passing with each node's neighbor (if any).

3 Smoothing Dynamic Graphs

We now define a version of smoothed analysis that is relevant to dynamic graphs. To begin, we define the *edit distance* between two static graphs $G = (V, E)$ and $G' = (V, E')$ to be the minimum number of edge additions and removals needed to transform G to G'. With this in mind, for a given G and $k \in \{0, 1, ..., \binom{n}{2}\}$, we define the set:

$$\mathit{editdist}(G, k) = \{G' \mid \text{the edit distance between } G \text{ and } G' \text{ is no more than } k\}.$$

Finally, for a given set of dynamic graphs $\mathcal{G}$, we define the set:

$$\mathit{allowed}(\mathcal{G}) = \{G \mid \exists \mathcal{H} \in \mathcal{G} \text{ such that } G \in \mathcal{H}\}.$$

In other words, *allowed* describes all graphs that show up in the dynamic graphs contained in the set $\mathcal{G}$. Our notion of smoothing is always defined with respect to a dynamic graph set $\mathcal{G}$. Formally:

Definition 1. *Fix a set of dynamic graphs $\mathcal{G}$, a dynamic graph $\mathcal{H} \in \mathcal{G}$, and* smoothing factor $k \in \{0, 1, ..., \binom{n}{2}\}$*. To* k-smooth *a static graph $G \in \mathcal{H}$ (with* *respect to $\mathcal{G}$) is to replace G with a graph G' sampled uniformly from the set $editdist(G,k) \cap allowed(\mathcal{G})$. To k-*smooth *the entire dynamic graph $\mathcal{H}$ (with respect to $\mathcal{G}$), is to replace $\mathcal{H}$ with the dynamic graph $\mathcal{H}'$ that results when we k-smooth each of its static graphs.*

We will also sometimes say that G' (resp. $\mathcal{H}'$) is a *k-smoothed* version of G (resp. $\mathcal{H}$), or simply a *k-smoothed* G (resp. $\mathcal{H}$). We often omit the dynamic graph set $\mathcal{G}$ when it is clear in context. (Typically, $\mathcal{G}$ will be the set contained in a dynamic network type under consideration.)

Discussion. Our notion of k-smoothing transforms a graph by randomly adding or deleting k edges. A key piece of our definition is that smoothing a graph with respect to a dynamic graph set cannot produce a graph not found in any members of that set. This restriction is particularly important for proving lower bounds on smoothed graphs, as we want to make sure that the lower bound results does not rely on a dynamic graph that could not otherwise appear. For example, if studying a process in a dynamic graph that is always connected, we do not want smoothing to disconnect the graph—an event that might trivialize some bounds.

4 Connected and Pairing Dynamic Network Types

We now define two dynamic network types: the *connected network type* [7,9,11,12], and the *pairing network type* [4]. We study random walks (Section 6) and flooding (Section 5) in the context of the connected network type, whereas we study token aggregation (Section 7) in the context of the pairing type.

4.1 Connected Network

The *connected network type* [7,9,11,12] is defined as $(\mathcal{G}_{conn}, C_{conn})$, where $\mathcal{G}_{conn}$ contains every dynamic graph (defined with respect to our fixed node set V) in which every individual graph is connected, and where C_{conn} describes reliable broadcast (i.e., a message sent by u in rounds r in an execution in graph $\mathcal{H}$ is received by every neighbor of u in $\mathcal{H}[r]$).

Properties of Smoothed Connected Networks. For our upper bounds, we show that if certain edges are added to the graph through smoothing, then the algorithm makes enough progress on the smoothed graph. For our lower bounds, we show that if certain edges are not added to the graph, then the algorithm does not make much progress. The following lemmas bound the probabilities that these edges are added. The proofs roughly amount to showing that sampling uniformly from $editdist(G,k) \cap allowed(\mathcal{G}_{conn})$ is similar to sampling from $editdist(G,k)$.

The first two lemmas are applicable when upper-bounding the performance of an algorithm on a smoothed dynamic graph. The first lemma states that the k-smoothed version of graph G is fairly likely to include at least one edge from the set S of helpful edges. The second lemma, conversely, says that certain critical edges that already exist in G are very unlikely to be removed in the smoothed version.

Lemma 1. *There exists constant $c_1 > 0$ such that the following holds. Consider any graph $G \in allowed(\mathcal{G}_{conn})$. Consider also any nonempty set S of potential edges and smoothing value $k \leq n/16$ with $k\,|S| \leq n^2/2$. Then with probability at least $c_1 k\,|S|\,/n^2$, the k-smoothed graph G' of G contains at least one edge from S.*

Lemma 2. *There exists constant $c_2 > 0$ such that the following holds. Consider any graph $G = (V, E) \in allowed(\mathcal{G}_{conn})$. Consider also any nonempty set $S \subseteq E$ of edges in the graph and smoothing value $k \leq n/16$. Then with probability at most $c_2 k\,|S|\,/n^2$, the k-smoothed graph G' removes an edge from S.*

Our next lemma is applicable when lower-bounding an algorithm's performance on a dynamic graph. It says essentially that Lemma 1 is tight—it is not too likely to add any of the helpful edges from S.

Lemma 3. *There exists constant $c_3 > 0$ such that the following holds. Consider any graph $G = (V, E) \in allowed(\mathcal{G}_{conn})$. Consider also any set S of edges and smoothing value $k \leq n/16$ such that $S \cap E = \emptyset$. Then with probability at most $c_3 k\,|S|\,/n^2$, the k-smoothed graph G' of G contains an edge from S.*

4.2 Pairing Network

The second type we study is the *pairing network type* [4]. This type is defined as $(\mathcal{G}_{pair}, C_{pair})$, where $\mathcal{G}_{pair}$ contains every dynamic graph (defined with respect to our fixed node set V) in which every individual graph is a (not necessarily complete) matching, and C_{pair} reliable communicates messages between pairs of nodes connected in the given round. This network type is motivated by the current peer-to-peer network technologies implemented in smart devices. These low-level protocols usually depend on discovering nearby nodes and initiating one-on-one local interaction.

Properties of Smoothed Pairing Networks. In the following, when discussing a matching G, we partition nodes into one of two *types*: a node is *matched* if it is connected to another node by an edge in G, and it is otherwise *unmatched*. The following property concerns the probability that smoothing affects (i.e., adds or deletes at least one adjacent edge) a given node u from a set S of nodes of the same type. It notes that as the set S containing u grows, the *upper bound* on the probability that u is affected decreases. The key insight behind this not necessarily intuitive statement is that this probability must be the same for *all* nodes in S (due to their symmetry in the graph). Therefore, a given probability will generate more expected changes as S grows, and therefore, to keep the expected changes below the k threshold, this bound on this probability must decrease as S grows.

Lemma 4. *Consider any graph $G = (V, E) \in allowed(\mathcal{G}_{pair})$ and constant $\delta > 1$. Let $S \subseteq V$ be a set of nodes in G such that: (1) all nodes in S are of the same type (matched or unmatched), and (2) $|S| \geq n/\delta$. Consider any node $u \in S$ and smoothing factor $k < n/(2 \cdot \delta)$. Let G' be the result of k-smoothing G. The probability that u's adjacency list is different in G' as compared to G is no more than $(2 \cdot \delta \cdot k)/n$.*

5 Flooding

Here we consider the performance of a basic flooding process in a connected dynamic network. In more detail, we assume a single *source* node starts with a message. In every round, every node that knows the message broadcasts the message to its neighbors. (Flooding can be trivially implemented in a connected network type due to reliable communication.) We consider the flooding process *complete* in the first round that every node has the message. Without smoothing, this problem clearly takes $\Omega(n)$ rounds in large diameter static graphs, so a natural alternative is to state bounds in terms of diameter. Unfortunately, there exist dynamic graphs (e.g., the spooling graph defined below) where the graph in each round is constant diameter, but flooding still requires $\Omega(n)$ rounds.

We show that this $\Omega(n)$ lower bound is somewhat fragile by proving a polynomial improvement with any smoothing. Specifically, we show an upper bound of $O(n^{2/3}\log(n)/k^{1/3})$ rounds, with high probability, with k-smoothing. We also exhibit a nearly matching lower bound by showing that the dynamic spooling graph requires $\Omega(n^{2/3}/k^{1/3})$ rounds with constant probability.

5.1 Lower Bound

We build our lower bound around the dynamic *spooling graph*, defined as follows. Label the nodes from 1 to n, where node 1 is the source. The spooling graph is a dynamic graph where in each round r, the network is the $\min\{r, n-1\}$-spool graph. We define the *i-spool graph*, for $i \in [n-1]$ to be the graph consisting of: a star on nodes $\{1, \ldots, i\}$ centered at i called the *left spool*, a star on nodes $\{i+1, \ldots, n\}$ centered on $i+1$ called the *right spool*, and an edge between the two centers i and $i+1$. We call $i+1$ the *head* node.

With node 1 as the source node, it is straightforward to see that, in the absence of smoothing, flooding requires $n-1$ rounds to complete on the spooling network. (Every node in the left spool has the message but every node in the right spool does not. In each round, the head node receives the message then moves to the left spool.) We generalize this lower bound to smoothing. The main idea is that in order for every node to receive the message early, one of the early heads must be adjacent to a smoothed edge.

Theorem 1. *Consider the flooding process on a k-smoothed n-vertex spooling graph, with $k \leq \sqrt{n}$ and sufficiently large n. With probability at least $1/2$, the flooding process does not complete before the $\Omega(n^{2/3}/k^{1/3})$-th round.*

5.2 An $O(n^{2/3}\log n/k^{1/3})$ Upper Bound for General Networks

Next, we show that flooding in *every* k-smoothed network will complete in $O(n^{2/3}\log n/k^{1/3})$ time, with high probability. When this result is combined with the $\Omega(n^{2/3}/k^{1/3})$ lower bound from above, this shows this analysis to be essentially tight for this problem under smoothing.

Support Sequences. The core idea is to show that every node in every network is supported by a structure in the dynamic graph such that if the message can be delivered to *anywhere* in this structure in time, it will subsequently propagate to the target. In the spooling network, this structure for a given target node u consists simply of the nodes that will become the head in the rounds leading up to the relevant complexity deadline. The *support sequence* object defined below generalizes a similar notion to all graphs. It provides, in some sense, a fat target for the smoothed edges to hit in their quest to accelerate flooding.

Definition 2. *Fix two integers t and ℓ, $1 \leq \ell < t$, a dynamic graph $\mathcal{H} = G_1, \ldots, G_t$ with $G_i = (V, E_i)$ for all i, and a node $u \in V$. A (t, ℓ)-*support sequence *for u in G is a sequence $S_0, S_1, S_2, ..., S_\ell$, such that the following properties hold: (1) For every $i \in [0, \ell]$: $S_i \subseteq V$. (2) $S_0 = \{u\}$. (3) For every $i \in [1, \ell]$: $S_{i-1} \subset S_i$* and *$S_i \setminus S_{i-1} = \{v\}$, for some $v \in V$ such that v is adjacent to at least one node of S_{i-1} in G_{t-i}.*

Notice that the support structure is defined "backwards" with S_0 containing the target node u, and each subsequent step going one round back in time. We prove that every connected dynamic graph has such a support structure, because the graph is connected in every round.

Lemma 5. *Fix some dynamic graph $\mathcal{H} \in \mathcal{G}_{conn}$ on vertex set V, some node $u \in V$, and some rounds t and ℓ, where $1 \leq \ell < t$. There exists a (t, ℓ)-support sequence for u in $\mathcal{H}$.*

The following key lemma shows that over every period of $\Theta(n^{2/3}/k^{1/3})$ rounds of k-smoothed flooding, every node has a constant probability of receiving the message. Applying this lemma over $\Theta(\log n)$ consecutive time intervals with a Chernoff bound, we get our main theorem.

Lemma 6. *There exists constant $\alpha \geq 3$ such that the following holds. Fix a dynamic graph $\mathcal{H} \in \mathcal{G}_{conn}$ on vertex set V, any node $u \in V$, and a consecutive interval of $\alpha n^{2/3}/k^{1/3}$ rounds. For smoothing value $k \leq n/16$, node u receives the flooded message in the k-smoothed version of $\mathcal{H}$ with probability at least $1/2$.*

Theorem 2. *For any dynamic graph $\mathcal{H} \in \mathcal{G}_{conn}$ and smoothing value $k \leq n/16$, flooding completes in $O(n^{2/3} \log n / k^{1/3})$ rounds on the k-smoothed version of $\mathcal{H}$ with high probability.*

6 Random Walks

As discussed in Section 1, random walks in dynamic graphs exhibit fundamentally different behavior from random walks in static graphs. Most notably, in dynamic graphs there can be pairs of nodes whose hitting time is exponential [2], even though in static (connected) graphs it is well-known that the maximum hitting time is at most $O(n^3)$ [13]. This is true even under obvious technical restrictions necessary to prevent infinite hitting times, such as requiring the graph to be connected at all times and to have self-loops at all nodes.

We show that this lower bound is extremely fragile. A straightforward argument shows that a small perturbation (1-smoothing) is enough to guarantee that in *any* dynamic graph, all hitting times are at most $O(n^3)$. Larger perturbations (k-smoothing) lead to $O(n^3/k)$ hitting times. We also prove a lower bound of $\Omega(n^{5/2}/\sqrt{k})$, using an example which is in fact a static graph (made dynamic by simply repeating it). In some sense, it is not surprising that the lower bound on random walks is fragile, as there exist algorithms for accomplishing the same goals (e.g., identifying a random sample) in dynamic graphs in polynomial time [2,15].

6.1 Preliminaries

We begin with some technical preliminaries. In a static graph, a random walk starting at $u \in V$ is a walk on G where the next node is chosen uniformly at random from the set of neighbors on the current node (possibly including the current node itself if there is a self-loop). The *hitting time* $H(u, v)$ for $u, v \in V$ is the expected number of steps taken by a random walk starting at u until it hits v for the first time. Random walks are defined similarly in a dynamic graph $\mathcal{H} = G_1, G_2, \ldots$: at first the random walk starts at u, and if at the beginning of time step t it is at a node v_t then in step t it moves to a neighbor of v_t in G_t chosen uniformly at random. Hitting times are defined in the same way as in the static case.

The definition of the hitting time in a smoothed dynamic graph is intuitive but slightly subtle. Given a dynamic graph $\mathcal{H}$ and vertices u, v, the *hitting time from u to v under k-smoothing*, denoted by $H_k(u, v)$, is the expected number of steps taken by a random walk starting at u until first reaching v in the (random) k-smoothed version $\mathcal{H}'$ of $\mathcal{H}$ (either with respect to $\mathcal{G}_{conn}$ or with respect to the set $\mathcal{G}_{all}$ of all dynamic graphs). Note that this expectation is now taken over two independent sources of randomness: the randomness of the random walk, and also the randomness of the smoothing (as defined in Section 3).

6.2 Upper Bounds

We first prove that even a tiny amount amount of smoothing is sufficient to guarantee polynomial hitting times even though without smoothing there is an exponential lower bound. Intuitively, this is because if we add a random edge at every time point, there is always some inverse polynomial probability of directly jumping to the target node. We also show that more smoothing decreases this bound linearly.

Theorem 3. *In any dynamic graph $\mathcal{H}$, for all vertices u, v and value $k \leq n/16$, the hitting time $H_k(u, v)$ under k-smoothing (with respect to $\mathcal{G}_{all}$) is at most $O(n^3/k)$. This is also true for smoothing with respect to $\mathcal{G}_{conn}$ if $\mathcal{H} \in \mathcal{G}_{conn}$.*

A particularly interesting example is the *dynamic star*, which was used by Avin et al. [2] to prove an exponential lower bound. The dynamic star consists of

n vertices $\{0, 1, \ldots, n-1\}$, where the center of the start at time t is $t \mod (n-1)$ (note that node $n-1$ is never the center). Every node also has a self loop. Avin et al. [2] proved that the hitting time from node $n-2$ to node $n-1$ is at least 2^{n-2}. It turns out that this lower bound is particularly fragile – not only does Theorem 3 imply that the hitting time is polynomial, it is actually a factor of n better than the global upper bound due to the small degrees at the leaves.

Theorem 4. *$H_k(u, v)$ is at most $O(n^2/k)$ in the dynamic star for all $k \leq n/16$ and for all vertices u, v (where smoothing is with respect to $\mathcal{G}_{conn}$).*

6.3 Lower Bounds

Since the dynamic star was the worst example for random walks in dynamic graphs without smoothing, Theorem 4 naturally leads to the question of whether the bound of $O(n^2/k)$ holds for all dynamic graphs in $\mathcal{G}_{conn}$, or whether the weaker bound of $O(n^3/k)$ from Theorem 3 is tight. We show that under smoothing, the dynamic star is in fact *not* the worst case: a lower bound of $\Omega(n^{5/2}/\sqrt{k})$ holds for the *lollipop graph*. The lollipop is a famous example of graph in which the hitting time is large: there are nodes u and v such that $H(u, v) = \Theta(n^3)$ (see, e.g., [13]). Here we will use it to prove a lower bound on the hitting time of dynamic graphs under smoothing:

Theorem 5. *There is a dynamic graph $\mathcal{H} \in \mathcal{G}_{conn}$ and nodes u, v such that $H_k(u, v) \geq \Omega(n^{5/2}/(\sqrt{k} \ln n))$ for all $k \leq n/16$ (where smoothing is with respect to $\mathcal{G}_{conn}$).*

In the lollipop graph $L_n = (V, E)$ the vertex set is partitioned into two pieces V_1 and V_2 with $|V_1| = |V_2| = n/2$. The nodes in V_1 form a clique (i.e. there is an edge between every two nodes in V_1), while the nodes in V_2 form a path (i.e., there is a bijection $\pi : [n/2] \to V_2$ such that there is an edge between $\pi(i)$ and $\pi(i+1)$ for all $i \in [(n/2) - 1]$). There is also a single special node $v^* \in V_1$ which has an edge to the beginning of the V_2 path, i.e., there is also an edge $\{v^*, \pi(1)\}$. The dynamic graph $\mathcal{H}$ in Theorem 5 is the dynamic lollipop: $G_i = L_n$ for all $i \geq 1$. The starting point of the random walk u is an arbitrary node in V_1, and the target node $v = \pi(n/2)$ is the last node on the path.

The intuition for the 1-smoothing case is relatively straightforward: if the random walk is on the path then every $\Theta(n)$ rounds it will follow one of the randomly added smoothed edges, which will (with probability 1/2) lead it back to the clique. So in order to hit v, it has to spend less than n time in the path. A standard analysis of the one-dimensional random walk then implies that it will only move $O(\sqrt{n})$ positions from where it started, so it needs to start in the final $O(\sqrt{n})$ nodes of the path. In each round, it will only see an edge from its current location to this final set of nodes with probability $O(1/n^{3/2})$, and will follow it with probability only $O(1/n)$ (since it will likely be in the clique). Hence the total hitting time is $\Omega(n^{5/2})$. This idea can be formalized and extended to k-smoothing.

If we do not insist on the dynamic graph being connected at all times, then in fact Theorem 3 is tight via a very simple example: a clique with a single disconnected node.

Theorem 6. *There is a dynamic graph $\mathcal{H}$ and vertices u, v such that $H_k(u,v) \geq \Omega(n^3/k)$ for all $k \leq n$ where smoothing is with respect to $\mathcal{G}_{all}$.*

7 Aggregation

Here we consider the aggregation problem in the pairing dynamic network type. Notice, in our study of flooding and random walks we were analyzing the behavior of a specific, well-known distributed process. In this section, by contrast, we consider the behavior of arbitrary algorithms. In particular, we will show the pessimistic lower bound for the aggregation problem for 0-smoothed pairing graphs from [4], holds (within constant factors), even for relatively large amounts of smoothing. This problem, therefore, provides an example of where smoothing does not help much.

The Aggregation Problem. The aggregation problem, first defined in [4], assumes each node $u \in V$ begins with a unique token $\sigma[u]$. The execution proceeds for a fixed length determined by the length of the dynamic graph. At the end of the execution, each node u *uploads* a set (potentially empty) $\gamma[u]$ containing tokens. An *aggregation algorithm* must avoid both losses and duplications (as would be required if these tokens were actually aggregated in an accurate manner). Formally:

Definition 3. *An algorithm $\mathcal{A}$ is an* aggregation algorithm *if and only if at the end of every execution of $\mathcal{A}$ the following two properties hold:*
(1) No Loss: $\bigcup_{u\in V} \gamma[u] = \bigcup_{u \in V}\{\sigma[u]\}$. *(2)* No Duplication: $\forall u, v \in V, u \neq v$: $\gamma[u] \cap \gamma[v] = \emptyset$.

To evaluate the performance of an aggregation algorithm we introduce the notion of *aggregation factor.* At at the end of an execution, the aggregation factor of an algorithm is the number of nodes that upload at least one token (i.e., $|\{u \in V : \gamma[u] \neq \emptyset\}|$). Because some networks (e.g., a static cliques) are more suitable for small aggregation factors than others (e.g., no edges in any round) we evaluate the competitive ratio of an algorithm's aggregation factor as compared to the offline optimal performance for the given network.

The worst possible performance, therefore, is n, which implies that the algorithm uploaded from n times as many nodes as the offline optimal (note that n is the maximum possible value for an aggregation factor). This is only possible when the algorithm achieves no aggregation and yet an offline algorithm could have aggregated all tokens to a single node. The best possible performance is a competitive ratio of 1, which occurs when the algorithm matches the offline optimal performance.

Results Summary. In [4], the authors prove that no aggregation algorithm can guarantee better than a $\Omega(n)$ competitive ratio with a constant probability or better. In more detail:

Theorem 7 (Adapted from [4]). *For every aggregation algorithm $\mathcal{A}$, there exists a pairing graph $\mathcal{H}$ such that with probability at least $1/2$: $\mathcal{A}$'s aggregation factor is $\Omega(n)$ times worse than the offline optimal aggregation factor in $\mathcal{H}$.*

Our goal in the remainder of this section is to prove that this strong lower bound persists even after a significant amount of smoothing (i.e., $k = O(n/\log^2 n)$). We formalize this result below (note that the cited probability is with respect to the random bits of both the algorithm and the smoothing process):

Theorem 8. *For every aggregation algorithm $\mathcal{A}$ and smoothing factor $k \leq n/(32 \cdot \log^2 n)$, there exists a pairing graph $\mathcal{H}$ such that with probability at least $1/2$: $\mathcal{A}$'s aggregation factor is $\Omega(n)$ times worse than the offline optimal aggregation factor in a k-smoothed version of $\mathcal{H}$ (with respect to $\mathcal{G}_{pair}$).*

7.1 Lower Bound

Here we prove that for any smoothing factor $k \leq (cn)/\log^2 n$ (for some positive constant fraction c we fix in the analysis), k-smoothing does not help aggregation by more than a constant factor as compared to 0-smoothing. To do so, we begin by describing a probabilistic process for generating a hard pairing graph. We will later show that the graph produced by this process is likely to be hard for a given randomized algorithm. To prove our main theorem, we will conclude by applying the probabilistic method to show this result implies the existence of a hard graph for each algorithm.

The α-Stable Pairing Graph Process. We define a specific process for generating a pairing graph (i.e., a graph in $allowed(\mathcal{G}_{pair})$). The process is parameterized by some constant integer $\alpha \geq 1$. In the following, assume the network size $n = 2\ell$ for some integer $\ell \geq 1$ that is also a power of 2.[2] For the purposes of this construction, we label the 2ℓ nodes in the network as $a_1, b_1, a_2, b_2, ..., a_\ell, b_\ell$. For the first α rounds, our process generates graphs with the edge set: $\{(a_i, b_i) : 1 \leq i \leq \ell\}$. After these rounds, the process generates ℓ bits, $q_1, q_2, ..., q_\ell$, with uniform randomness. It then defines a set S of *selected nodes* by adding to S the node a_i for every i such that $q_i = 0$, and adding b_i for every i such that $q_i = 1$. That is, for each of our (a_i, b_i) pairs, the process randomly flips a coin to select a single element from the pair to add to S.

For all graphs that follow, the nodes *not in* S will be isolated in the graph (i.e., not be matched). We turn our attention to how the process adds edges between the nodes that are in S. To do so, it divides the graphs that follow into *phases*, each consisting of α consecutive rounds of the same graph. In the first phase, this graph is the one that results when the process pairs up the nodes in S by adding an edge between each such pair (these are the only edges).

[2] We can deal with odd n and/or ℓ not a power of 2 by suffering only a constant factor cost to our final performance. For simplicity of presentation, we maintain these assumptions for now.

In the second phase, the process defines a set S_2 that contains exactly one node from each of the pairs from the first phase. It then pairs up the nodes in S_2 with edges as before. It also pairs up all nodes in $S \setminus S_2$ arbitrarily. Every graph in the second phase includes only these edges. In the third phase, the process defines a set S_3 containing exactly one node from each of the S_2 pairs from the previous pairs. It then once again pairs up the remaining nodes in S arbitrarily. The process repeats this procedure until phase $t = \log_2 |S|$ at which point only a single node is in S_t, and we are done.

The total length of this dynamic graph is $\alpha(\log_2 (|S|) + 1)$. It is easy to verify that it satisfies the definition of the pairing dynamic network type.

Performance of the Offline Optimal Aggregation Algorithm. We now show that the even with lots of smoothing, a graph generated by the stable pairing graph process, parameterized with a sufficiently large α, yields a good optimal solution (i.e., an aggregation factor of 1).

Lemma 7. *For any $k \leq n/32$, and any pairing graph $\mathcal{H}$ that might be generated by the $(\log n)$-stable pairing graph process, with high probability in n: the offline optimal aggregation algorithm achieves an aggregation factor of 1 in a k-smoothed version of $\mathcal{H}$.*

Performance of an Arbitrary Distributed Aggregation Algorithm. We now fix an arbitrary distributed aggregation algorithm and demonstrate that it cannot guarantee (with good probability) to achieve a non-trivial competitive ratio in all pairing graphs. In particular, we will show it has a constant probability of performing poorly in a graph generated by our above process.

Lemma 8. *Fix an online aggregation algorithm $\mathcal{A}$ and smoothing factor $k \leq n/(32 \cdot \log^2 n)$. Consider a k-smoothed version of a graph $\mathcal{H}$ generated by the $(\log n)$-stable pairing graph process. With probability greater than 1/2 (over the smoothing, adversary, and algorithm's independent random choices): $\mathcal{A}$ has an aggregation factor in $\Omega(n)$ when executed in this graph.*

A final union bound combines the results from Lemmas 7 and 8 to get our final corollary. Applying the probabilistic method to the corollary yields the main theorem—Theorem 8.

Corollary 1. *Fix an aggregation algorithm $\mathcal{A}$ and smoothing factor $k \leq n/(32 \cdot \log^2 n)$. There is a method for probabilistically constructing a pairing graph $\mathcal{H}$, such that with probability greater than 1/2 (over the smoothing, adversary, and algorithm's independent random choices): $\mathcal{A}$'s aggregation factor in a k-smoothed version of $\mathcal{H}$ is $\Omega(n)$ times larger than the offline optimal factor for this graph.*

References

1. Augustine, J., Pandurangan, G., Robinson, P., Upfal, E.: Towards robust and efficient computation in dynamic peer-to-peer networks. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (2012)

2. Avin, C., Koucký, M., Lotker, Z.: How to explore a fast-changing world (cover time of a simple random walk on evolving graphs). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 121–132. Springer, Heidelberg (2008)
3. Clementi, A., Silvestri, R., Trevisan, L.: Information spreading in dynamic graphs. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2012)
4. Cornejo, A., Gilbert, S., Newport, C.: Aggregation in dynamic networks. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2012)
5. Denysyuk, O., Rodrigues, L.: Random walks on evolving graphs with recurring topologies. In: Kuhn, F. (ed.) DISC 2014. LNCS, vol. 8784, pp. 333–345. Springer, Heidelberg (2014)
6. Dinitz, M., Fineman, J., Gilbert, S., Newport, C.: Smoothed analysis of dynamic networks. Full version http://people.cs.georgetown.edu/cnewport/pubs/SmoothingDynamicNetworks-Full.pdf
7. Dutta, C., Pandurangan, G., Rajaraman, R., Sun, Z., Viola, E.: On the complexity of information spreading in dynamic networks. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (2013)
8. Ghaffari, M., Lynch, N., Newport, C.: The cost of radio network broadcast for different models of unreliable links. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2013)
9. Haeupler, B., Karger, D.: Faster information dissemination in dynamic networks via network coding. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2011)
10. Kuhn, F., Oshman, R.: Dynamic networks: models and algorithms. ACM SIGACT News **42**(1), 82–96 (2011)
11. Kuhn, F., Lynch, N., Oshman, R.: Distributed computation in dynamic networks. In: Proceedings of the ACM Symposium on Theory of Computing (2010)
12. Kuhn, F., Oshman, R., Moses, Y.: Coordinated consensus in dynamic networks. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (2011)
13. Lovász, L.: Random walks on graphs: a survey. In: Miklós, D., Sós, V.T., Szőnyi, T. (eds.) Combinatorics, Paul Erdős is Eighty, vol. 2, pp. 1–46. János Bolyai Mathematical Society (1996)
14. Newport, C.: Lower bounds for structuring unreliable radio networks. In: Kuhn, F. (ed.) DISC 2014. LNCS, vol. 8784, pp. 318–332. Springer, Heidelberg (2014)
15. Das Sarma, A., Molla, A.R., Pandurangan, G.: Fast distributed computation in dynamic networks via random walks. In: Aguilera, M.K. (ed.) DISC 2012. LNCS, vol. 7611, pp. 136–150. Springer, Heidelberg (2012)
16. Spielman, D.A., Teng, S.: Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. J. ACM **51**(3), 385–463 (2004)
17. Spielman, D.A., Teng, S.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. Commun. ACM **52**(10), 76–84 (2009)
18. Spielman, D.A., Teng, S.H.: Smoothed analysis: an attempt to explain the behavior of algorithms in practice. Communications of the ACM **52**(10), 76–84 (2009)