UNIVERSITY OF CALIFORNIA
Los Angeles

On Explicit Depth Robust Graphs

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Science in Computer Science

by

Aoxuan Li

2019

ABSTRACT OF THE THESIS

On Explicit Depth Robust Graphs

by

Aoxuan Li
Master of Science in Computer Science
University of California, Los Angeles, 2019
Professor Rafail Ostrovsky, Chair

We study the problem defined by Erdős and Szemerédi in 1975 of constructing sparse depth-robust graphs. Recall that a directed acyclic graph G is (e, d)-depth-robust if it is guaranteed to contain a path of length d even after the deletion of any e nodes and all of their incident edges. The original construction (of nearly optimal depth-robust graphs) of Erdős and Szemerédi required logarithmic in-degree and subsequent work by Mahmoody, Moran, and Vadhan made that construction explicit. One of the major open questions left since that 1975 seminal work was to construct depth-robust graphs of constant degree. Our contribution is the first explicit construction of constant-degree depth-robust graphs. Our construction too enjoys nearly optimal depth-robustness. We accomplish this via a novel *expanding graph product* operator $\mathcal{X}$ that takes three input graphs (G, H, X) with special properties and outputs a new graph. Informally, we show that our operator provides the following guarantee: if G and H are depth-robust graphs and H is a constant-degree *expanding* graph [RVW00], then $G^* = \mathcal{X}(G, H, X)$ is a depth-robust graph of size $|G| \cdot |H|$ whose degree depends only additively on degrees of H and X. We then show that the recursive application of the expanding graph product operator yields a simple and explicit iterative construction for constant-degree depth-robust graphs of arbitrary size. In particular, we show that a graph of size n will enjoy $(\Omega(n^{1-\epsilon}), \Omega(n^{1-\epsilon}))$-depth-robustness for any $\epsilon > 0$ and give an algorithm for computing labels of all nodes that have a direct edge to/from a given node labeled i in time poly$(\log i)$. Ours is the first explicitly constructed constant-degree depth robust graph with guaranteed lower bounds on its depth-robustness (in contrast to only probabilistic

guarantees). Previous explicit constructions were of logarithmic degree or worked only with probability < 1. Beyond theoretical relevance, our construction has practical implications including a new data-independent memory-hard function (iMHF), a desirable cryptographic primitive for crypto-currencies, with guaranteed lower bounds on its memory complexity.

The thesis of Aoxuan Li is approved.

Eliezer M Gafni

Alexander Sherstov

Rafail Ostrovsky, Committee Chair

University of California, Los Angeles

2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis is the result of a co-authored work with Eli Jaffe and Prof. Rafail Ostrovsky. I would like to express my gratitute to my supervisor Prof. Ostrovsky for his guidance trhougout the research and writing of this thesis. I would also like to thank Eli Jaffe for his collaboration on this exciting research.

CHAPTER 1

Introduction

1.1 Summary of Known Results

Depth-robust graphs of low degree were first studied by Erdös and Szemerédi in [EGS75] under the name of "sparse graphs with dense long paths." Their original motivation involved bounding the complexity of Boolean functions, but since then the concept of depth-robust graphs has arisen in various areas of mathematics and computer science. As the name implies, such graphs are intuitively characterized by having a large number of distinct long paths. Formally, we say a graph is (e, d)-depth-robust if after removing any subset of e vertices, a path through at least d vertices still remains. The trivial and optimal example of such a graph is the complete graph on n vertices K_n. Since every pair of vertices share an edge, after removing any set of e vertices, a path will still exist through the remaining $n - e$ vertices. Thus, for every $e < n$, K_n is $(e, n - e)$-depth-robust. The question of interest for Erdös and Szemerédi is how sparse can such a graph be? That is, can we hope to construct a graph whose depth-robustness is nearly as good as that of K_n but which enjoys a lower indegree?

Erdös and Szemerédi constructed in [EGS75] a family of graphs $\{G_n^\epsilon\}$ on n vertices which are (an, bn)-depth-robust for any $a + b < 1 - \epsilon$. The complete graph K_n is (an, bn) depth robust for any $a + b = 1$, so this result is only ϵ short of optimal. The indegree of Erdös and Szemerédi construction, however, is $O(\log n)$, an exponential upgrade over K_n. Their construction is centered around the bipartite *expanding* graph. In such a graph, any two large enough sets from opposite parts of the graph are guaranteed to have an edge between them. By overlaying such expanding graphs between various sets of vertices, Erdös and

Szemerédi guaranteed the existence of particular edges unless many nodes are removed. This is the basis of their proof of depth-robustness. Their result, however, provides only a probabilistic argument showing the existence of such expanding graphs. If one hopes to use such a graph in any practical way, an explicit construction of such an expanding graph is necessary. Here, by explicit, we mean the neighbors of a vertex with label i can be computed in time poly($\log i$). This definition allows one to compute neighbors in an exponentially-large graph in polynomial-time, a particularly desirable property in applications.

In 2000, Reingold, Vadhan and Wigderson [RVW00] published seminal paper in which they define and utilize the novel zig-zag graph product to iteratively and explicitly construct expander graphs of constant degree. Expanders, which are distinct but similar to expanding graphs, enjoy guaranteed connectivity properties under certain conditions. The zig-zag product defined there is one type of "replacement" product operator, in which each vertex of a graph is replaced by a full copy (or cloud) of another graph. Their result demonstrates that replacement-style products are useful in maintaining connectivity properties and increasing graph size exponentially fast without increasing the degree of the graph, an important insight for future results. In the full version of their [RVW00] paper, they extend their result using the theory of superconcentrators to explicitly construct expanding graphs in addition to expanders. After another 10 years, these explicit expanding graphs are applied by Mahmoody, Moran, and Vadhan [MMV13] to make explicit the family of graphs $\{G_n^\epsilon\}$ from [EGS75]. Their work utilizes the new explicit depth-robust graphs to build a specific application, namely a publicly verifiable proof of sequential work (PoSW). More relevant to our interests, however, is their explicit construction of a family of (an, bn)-depth-robust graphs of indegree $O(\log^2(n))$ for any $a + b < 1 - \epsilon$.

The natural question is to achieve the same results for a graph of constant indegree? If not, what is the optimal depth-robustness for a constant-indegree graph? Not only is this a relevant and interesting theoretical question, explicitly constructed depth-robust graphs of constant indegree have practical usage in areas like cryptography and network design. This line of research has been heavily pursued by [AS14, AS15, ABP17, ABH17, ABP18] in the hopes of constructing optimal data-independent memory-hard functions (iMHFs), an

important cryptographic primitive. Of special interest to us is the work of Alwen, Blocki, and Pietrzak [ABP17], where the authors demonstrate an indegree-reduction method which transforms an (e, d)-depth-robust graph of arbitrary indegree δ into an $(e, d\gamma)$-depth-robust graph of constant degree 2, where $\gamma \in \mathbf{Z}^+$. Their proof also utilizes a type of replacement product, this time replacing each vertex in the graph with a chain of length $\delta + \gamma$. The incoming edges are then spread across the chain to decrease the indegree of any one vertex to 2. Applying this indegree-reduction to the family $\{G_n^\epsilon\}$ from [EGS75] gives a family of $(\frac{c_1 n}{\log n}, c_2 n)$-depth-robust graphs of constant indegree 2. The authors state that their result is only existential and not explicit and leave is as an open question to make an explicit construction. Moreover, if we wish to create an exponentially-large depth-robust graph with indegree 2 using their method, the graph we must start with must also be exponentially large. It would be preferable to start with a constant-size depth-robust graph and enlarge it to exponential-size without changing the degree or sacrificing depth-robustness.

Other constructions of various iMHFs also claim some type of depth-robustness for their underlying graph. Examples include Argon2i, winner of the password-hashing competition, and the DRSample algorithm from [ABH17]. As summarized in Table 1, the depth-robustness parameters of DRSample are as good as the existential result from [ABP17]. These constructions, however, are probabilistic, and may fail to generate a depth-robust graph with some small probability. Such an algorithm is useful practically, but fails to rigorously answer the more theoretical problem. That is, the problem of explicitly constructing a depth-robust graph of constant indegree with optimal parameters has remained open.

In this work, we explicitly construct a family of constant-degree $(\Omega(N^{1-\epsilon}), (\Omega(N^{1-\epsilon}))$-depth-robust graphs, where N is the number of nodes in the graph. Our construction is iterative and uses a novel graph product following the inspiration of [RVW00] in their construction of explicit constant-degree expanders. Generally speaking, the product takes three graphs G, H, and X, and replaces each vertex of G by a copy of H and each edge of G by a copy of X. This product is then applied repeatedly to increase the size of the graph exponentially fast without sacrificing depth-robustness nor increasing the degree. We give a high-level overview of the expanding graph product as well as our iterative construction.

1.2 Our Contributions

The Expanding Graph Product: We provide a novel graph product on *three* input graphs. The expanding graph product (or expanding product) $\mathcal{X}(G, H, X)$ replaces each vertex of G with a copy of H as in any replacement product. Each copy of H, or "cloud," maintains all internal edges. The connections between clouds (or edges of G) are replaced by expanding graphs. Each cloud is partitioned according to the labels of the vertices so that one expanding graph touches each part of the cloud. Ours is the first replacement product which incorporates the expanding graphs underlying Erdös' original construction. If G and H are depth-robust, regular graphs and X is a regular expanding graph, the product $G' = \mathcal{X}(G, H, X)$ is a depth-robust, regular graph of size $|H| \cdot |G|$. The degree of G' depends only on the degree of H and the degree of X, which enables a successful iterative process.

Explicit Constant-Degree Depth-Robust Graphs: Similarly to the explicit construction of expanders in [RVW00], we utilize our graph product to iteratively construct a family of explicit depth-robust graphs. Since $G' := \mathcal{X}(G, H, X)$ enjoys virtually every property that G does, we can iteratively produce larger and larger depth-robust graphs by replacing G with G' in the next iteration. That is, we set $G_0 = G$ and compute $G_k = \mathcal{X}(G_{k-1}, H, X)$ for $k > 0$. As we will show, this produces a family of graphs $\{G_n\}$ where G_n (consisting of N vertices) is $(\Omega(N^{1-\epsilon}), \Omega(N^{1-\epsilon}))$-depth-robust. Since the degree of G_k depends only on H, and X, which are held constant in each step, we achieve constant degree (and hence constant indegree). We prove explicitness of our construction by providing an efficient $(\text{poly}(\log i))$ algorithm for computing the neighbors of vertex i in G_n given (n, i).

1.3 Comparison to Existing Work

As mentioned above, all previous constructions of depth-robust graphs are either non-explicit, probabilistically generated, or have non-constant indegree. A summary of these results can be found in Table 1.1. With the proper choice of base graphs, our construction achieves higher depth-robustness than all other constant-degree constructions except for DRSample.

Our construction guarantees a path of length $\Omega(n^{1-\epsilon})$ even after the deletion of $n^{1-\epsilon}$ nodes, whereas the Argon2i-A and Argon2i-B graphs (except with negligible probability) guarantee paths of only length $\widetilde{\Omega}(n^{2\epsilon})$ and $\widetilde{\Omega}(n^{3\epsilon})$ respectively.

Construction	Depth-Robustness	Reference	Indegree	Complexity	Pr[$fail$]	Expl.?
Graph by Erdös et al.	(an, bn)	[EGS75]	$O(\log n)$	N/A	0	No
Erdös & Mahmoody et al.	(an, bn)	[MMV13]	$O(\log^2 n)$	N/A	0	Yes
Argon2i-A	$(e, \widetilde{\Omega}(n^2/e^2))$	[ABP17]	$O(1)$	$\Omega(n^{1.6})$ $O(n^{1.708})$	negl(n)	Yes
Argon2i-B	$(e, \widetilde{\Omega}(n^3/e^3))$	[BZ17]	$O(1)$	$\Omega(n^{1.75})$ $O(n^{1.767})$	negl(n)	Yes
Indegree Reduction	$(\Omega(\frac{n}{\log n}), \Omega(n))$	[ABP17]	$O(1)$	$\Omega(\frac{n^2}{\log n})$	0	No
DRSample	$(\Omega(\frac{n}{\log n}), \Omega(n))$	[ABH17]	$O(1)$	$\Omega(\frac{n^2}{\log n})$	negl(n)	Yes
Our Result	$(\Omega(N^{1-\epsilon}))$, $(\Omega(N^{1-\epsilon}))$	Here	$O(1)$	$\Omega(N^{2-\epsilon})$	0	Yes

Table 1.1: Existing constructions of depth-robust graphs and their indegree, associated iMHF complexity, failure probability, and explicitness.

1.4 Overview of the Expanding Product

Here we provide a high-level overview of our new graph product which we call the *expanding graph product*. Ours is a variant on replacement product as established in [RVW00]. The

defining property of such a product is the replacement of each node in one graph with a whole copy (or "cloud") of another graph. Such products are designed to maintain various connectivity properties of the individual graphs while shedding some less-desirable properties such as large degree. The expanding graph product is the first to extend this concept to the replacement of *edges* by a third graph.

Our product is particularly novel in its utilization of the constant-degree expanding graph to ensure strong connectivity between clouds. As mentioned, the expanding graph is a key component in Erdös' original construction of logarithmic-degree depth-robust graphs. We combine the concepts from [EGS75] and [RVW00] in hopes of achieving depth-robustness (as in [EGS75]) as well as explicitness and constant degree (as in [RVW00]).

The product $\mathcal{X}(G, H, X)$ is abstractly constructed as follows. A figure depicting this process can be found in Section 3. We assume that G is a D-regular graph on M vertices, H is a d-regular graph on m vertices, and m is a multiple of D. X is a bipartite graph with parts of size m/D.

1. Every vertex v in G is replaced by a copy of H (edges included) which we call a "cloud" and denote by C_v.

2. We partition each cloud C_v (according to the order of the labels) into D disjoint equal parts denoted $\{L(v,k)\}_{k=1}^{D}$, each of size m/D.

3. For each edge e going from u toward v in G, we overlay a copy of X from C_u to C_v. In particular, if e is the i^{th} edge of u and the j^{th} edge of v, a copy of X is overlaid from $L(u,i)$ to $L(v,j)$.

The resulting graph $\mathcal{X}(G, H, X)$ is of size $|G| \cdot |H| = M \cdot m$.

1.5 Overview of the Iterative Construction

The construction is an iteration based on some base graphs which we presume to have some initial properties. Particularly, for some $\epsilon > 0$, we begin with graphs G, H, and X. G is a D-

regular (AM, BM)-depth-robust graph on M vertices for any A, B satisfying $A + B < 1 - \epsilon$. Similarly, H is a d-regular (am, bm)-depth-robust graph on m vertices for any $a + b < 1 - \epsilon$. X is a c-regular bipartite $(1/\epsilon)$-expanding graph with parts of size m/D. For our expanding product to work, we also require that D divides m. It is necessary that every vertex of G has some pre-defined ordering on its D neighbors. This can easily be established by lexicographically ordering the labels. We assume a similar ordering on the neighbors of vertices in H and X. For simplicity, we will use labels from $[M]$ and $[m]$ for vertices in G and H respectively. Precise definitions for these graphs are given in Section 2 and methods for constructing such graphs appear in Section 5.

The construction can be defined simply in terms of the expanding product and the base graphs. The family of graphs $\{G_n\}$ is defined as follows.

- $G_0 = G$
- $G_k = \mathcal{X}(G_{k-1}, H, X)$

That is, we repeatedly apply the expanding product using the same graphs H and X for each iteration. The result of this construction is the family of graphs $\{G_n\}$ where G_n has size $N = Mm^n$. The main result of this paper proves that this family is $(\Omega(N^{1-\epsilon}, (\Omega(N^{1-\epsilon})$-depth-robust. We also show that it is explicit and has indegree at most $d + c = O(1)$. Since this is an iterative process, the crux of the argument boils down to proving that the depth-robustness of G and H is carried over after being composed with X. The remainder of the argument consists of counting arguments and basic induction. Next, we provide some light intuition as to why the expanding product is well-fitted for maintaining depth-robustness.

1.6 Intuition

Here we provide some basic intuition into the efficacy of the expanding product. The hope is that if G and H have some depth-robustness guarantees and X is a proper expanding graph then the product $\mathcal{X}(G, H, X)$ will itself be depth-robust. Recall the definition of (e, d)-depth-

robustness: for any subset of vertices $S \subseteq V$ of size $|S| \leq e$, a path remains in G/S of length at least d. Why should we expect $G' := \mathcal{X}(G, H, X)$ to achieve this property?

We can argue this by using the depth-robustness of G and H. The product G' has a very particular structure; it is composed of M identical "clouds" (copies of H) and the connections between these clouds mirror the structure of G. If G is depth-robust, unless we destroy a lot of clouds, a long "cloud path" of clouds connected by expanding graphs will remain. Each cloud is made up of D equal parts which (as we shall prove) are independently depth-robust. Since these parts are where expanding graphs attach, one can "destroy" a cloud by destroying one of these D parts. To destroy a lot of clouds, one must destroy a lot of these small parts by deleting a lot of vertices. If one does not destroy a lot of clouds, we can show that this long intact "cloud path" (along with properties of the expanding graph connecting each pair of clouds) implies the existence of a long path through these clouds. A more precise version of this argument gives rise to the depth-robustness bounds claimed in the previous section.

1.7 Applications

The main application of this work is in the construction of more complex data-independent memory-hard functions (iMHFs). Such functions are used in various proof-of-work protocols and provide many desirable security guarantees. In particular, such functions enjoy a high "cumulative memory complexity." This is a measure of the total memory required to compute a function, computed by summing memory at each time step over all time steps. Many recent works ([AS14, AS15, ABP17, ABH17, ABP18]) have aimed at constructing iMHFs with cumulative memory complexity (cmc) as close to optimal as possible. It turns out that such a problem can be reduced to a problem of constructing constant-indegree graphs of optimal depth-robustness.

The memory access pattern of an iMHF can be represented by a directed acyclic graph (DAG). The label l_v of each node v represents a value stored in memory. If values $l_1, \ldots, l_k$ are necessary to retrieve value l_v, the corresponding graph would have $\{v_1, \ldots, v_k\} \subseteq \mathsf{parents}(v)$.

A unifying result from [ABP17] shows that if G is (e, d)-depth-robust, the iMHF corresponding to G has cumulative memory complexity at least $e \cdot d$. Thus, constructions of new explicit depth-robust graphs imply constructions of new iMHFs. As mentioned earlier, all constructions of constant-indegree are probabilistic in nature and feature a small error probability. Our construction implies the existence of an iMHF with a cumulative memory complexity of at least $\Omega(n^{2-\epsilon})$ with zero error probability for any $\epsilon > 0$. This beats the complexity of the Argon-2i function but falls short of that achieved by DRSample.

Other applications of improved depth-robust graphs include proofs of sequential work (PoSW) and fault-tolerant distributed storage networks (DSNs). Further details are discussed in Section 6.

1.8 Open questions that remain

The following are open problems of interest relating to this work.

1. Ideally we wish to close the gap in depth-robustness between this result and that of DRSample in [ABH17]. That is, can we construct a $(\Omega(\frac{n}{\log n}), \Omega(n))$-depth-robust graph with zero probability of failure?

2. Further analysis is necessary to determine what base graphs can be chosen to ideally tune the constants in front of our depth-robustness parameters. Specifically, what values of a, b, A, B, D give optimal depth-robustness?

3. Though m is constant, it must be quite high in order to achieve strong depth-robustness. Does a modification to the construction exist which reduces the necessary size of m?

1.9 Organization of the Paper

In Chapter 2, we establish preliminary definitions necessary to describe our graph product and the construction, including the definitions of depth-robust and expanding graphs. We also define the notion of explicitness. In Chapter 3, we define formalize the expanding graph

product and the iterative construction of the family $\{G_n\}$. Chapter 4 includes the majority of our main results and establishes both the explicitness and depth-robustness of $\{G_n\}$. In Chapter 5, we prove the existence of the depth-robust graphs and the expanding graph which form the base of the iterative construction. Finally, Chapter 6 discusses potential applications of this work in more detail.

CHAPTER 2

Preliminaries

This chapter defines the graph-theoretic notions underlying our construction. We first discuss basic graph notions and the definition of explicitness. Following that we define expanding graphs and depth-robust graphs.

2.1 Graph Notions

A directed graph $G = (V, E)$ is a set of vertices and a set of directed edges. We say $(u, v) \in E$ if an edge points from vertex $u \in V$ to vertex $v \in V$. The object of interest in this paper is a directed acyclic graph (DAG). Although one may consider depth-robustness in the more general case where cycles are allowed, it is easy enough to avoid cycles in our construction which enables many more applications for our result. We will often use the letters m and M to refer to the number of vertices in particular graphs; do not mistake m or M to mean the number of edges in a graph. We shall not reference the number of edges in the graph.

The indegree of a vertex v is the number of incoming edges incident to v. Similarly, the outdegree is the number of outgoing edges incident to v. The indegree and outdegree of a vertex v are denoted by $\deg^-(v)$ and $\deg^+(v)$ respectively. The degree of a vertex is simply the sum of the indegree and outdegree, $\deg(v) = \deg^-(v) + \deg^+(v)$. A directed graph G is D-regular if for every vertex $v \in G$, $\deg(v) = D$.

2.2 Explicitness

Intuitively, we wish to say that a representation of a graph is *explicit* if one can compute the neighbors of a particular vertex in a reasonable amount of time. The necessary question to ask is then what is a reasonable amount of time? Suppose given a label i we are able to compute its neighbors in time poly(i). As long as i is bounded by some polynomial, our algorithm will run in polynomial-time. However, if the number of labels is exponentially-large, the time necessary to compute neighbors becomes implausible. For practical applications, access to objects of exponential-size is ideal, so this definition of explicitness will not work. Instead, we require that the neighbors of i be computable in time poly($\log i$). This is the well-accepted definition of explicitness used in works such as [RVW00]. We give a formal definition below.

Definition 2.1. Let $\{G_n\}$ be a family of graphs of size $|G_n| = N$ vertices. Suppose the vertices of G_n are labeled $\{1, \ldots, N\}$. We say that $\{G_n\}$ is **explicit** if there exists an algorithm neighbors satisfying the following:

1. neighbors takes as input (n, i). n is the index (and any necessary additional info) which completely specifies a particular $G_n \in \{G_n\}$. $i \in \{0,1\}^{\log n}$ is the label of some vertex in G_n.

2. For $k \leq n$, if vertex i has k total neighbors labeled $l_1, \ldots, l_k$, then neighbors(G_n, i) returns the list $(l_1, \ldots, l_k)$.

3. neighbors(n, i) runs in time at most poly($\log i$).

We will also say that a particular graph G is explicit as a shorthand meaning that G comes from some explicit family. We will eventually show that our construction satisfies this definition of explicitness.

2.3 Expanding Graphs

As mentioned in the introduction, [RVW00] provided the first explicit construction of constant-degree expander graphs. Though expanders are interesting in their own right, we will be

applying a supplementary result from their paper which constructs the slightly different expanding graph. Intuitively, a graph is an expanding graph if any two sufficiently large vertex subsets admit at least one edge between the two sets. Expanding graphs form the basis of various switching and sorting networks and can be used to establish time-space tradeoffs for many computational problems. As discussed in [Kla84], the existence of constant-degree expanding graphs was proven probabilistically in many independent cases but no efficient explicit construction was known until [RVW00]. Many works such as [Pip87, EGS75] can be and have been made explicit by the construction of explicit expanding graphs. The formal definition we shall use (for bipartite expanding graphs) from [MMV13] can be found below.

Definition 2.2 (Expanding Graphs [MMV13])**.** Let $G = (V_1, V_2, E)$ be a bipartite graph with parts of size $|V_1| = |V_2| = M$. G is A-expanding if for every pair of subsets $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$ satisfying $|S_1| = |S_2| = \lceil M/A \rceil$ there exists an edge between S_1 and S_2.

As we will discuss later, these can be constructed efficiently for any constant degree by the methods in [RVW00]. As suggested in the introduction, expanding graphs will be a crucial component in our construction.

2.4 Depth-Robustness

Depth-robustness measures the density of long paths within a graph. We first give the formal definition and then discuss its intuition and relevance.

Definition 2.3 (Depth-Robust Graph)**.** Let $G = (V, E)$ be a graph on n vertices ($|V| = n$). For $e, d \in \{1, \ldots, n\}$, G is (e, d)-depth-robust if for every subset of vertices $S \subseteq V$ satisfying $|S| \leq e$, $\text{depth}(G - S) \geq d$. That is, a path through at least d vertices exists in G/S.

It is intuitively helpful to contextualize the definition in terms of an adversary whose goal is to reduce the depth of the graph to some value below d. This adversary is only allowed a budget of at most e node deletions to accomplish this. If no such adversary can remove all paths of length d or more using a budget of at most e deletions, then we say the graph

is (e, d)-depth-robust. Essentially, depth-robustness is a measure of how well a graph resists such depth-reducing attacks.

As mentioned in the introduction, the (directed) complete graph K_n is a trivial example of a $(e, n-e)$-depth-robust graph for every $e \leq n$ but has a prohibitively high indegree $O(n)$. Erdös achieved $(e, n-e-\epsilon)$-depth-robustness for every $e \leq n$ and reduced the indegree to $O(\log n)$. This eventually increased to $O(\log^2(n))$ with the inclusion of explicit expanding graphs from [RVW00].

The following result from [ABP18] establishes that with one slight modification, the family of graphs from [EGS75] achieves a better form of depth-robustness which is desirable for our purposes. Their additional analysis establishes that one can characterize the depth-robustness of the (modified) family $\{G_n^\epsilon\}$ from [EGS75] by a single parameter ϵ rather than by two parameters e and d. The graph G_n^ϵ is, as it turns out, $(\alpha n, \beta n)$-depth-robust for *any* α, β whose sum is at most $1-\epsilon$ (in this case G_n^ϵ has n vertices). We will informally refer to this property as "scaling depth-robustness": as the adversary's budget for corruption increases, the length of the guaranteed path decreases. We state this formally below.

[[ABP18] Theorem 3] Fix $\epsilon > 0$. Then there exists a family of DAGs $\{G_n^\epsilon\}_{n=0}^\infty$ with indeg $G_n^\epsilon = O(\log n)$ that is $(\alpha n, \beta n)$-depth-robust for any constants α, β such that $\alpha + \beta < 1 - \epsilon$.

We will informally refer to such a graph for fixed $\epsilon > 0$ as ϵ-scaling-depth-robust. As we will show in Chapter 5, any such graph can be made regular (as is necessary for our construction) without sacrificing depth-robustness.

For constant-indegree graphs, no provably depth-robust explicit construction has yet been given, but an existential result exists via the indegree-reduction method from [ABP17]. In the next chapter, we describe our construction of explicit constant-degree depth-robust graphs by introducing the novel expanding graph product.

CHAPTER 3

The Expanding Product & The Iterative Construction

In this chapter, we first define a novel graph product designed to maintain depth-robustness and increasing the size of the graph while holding the degree to a constant. Afterward, we use the product to define an iterative process which constructs an explicit family of constant-degree $(\Omega(N^{1-\epsilon}), \Omega(N^{1-\epsilon}))$-depth-robust graphs.

3.1 The Expanding Product

The product utilizes the explicit constant-degree expanding graph which was successfully constructed in [RVW00]. The product is designed to take as input two regular scaling-depth-robust graphs G, H (that is, from the family in Theorem 2.4) as well as a regular constant-degree expanding graph X. Intuitively, to compute the product of G, H, and X, we first replace each node in G with a copy of H which we call a "cloud". Then, instead of connecting clouds by a single edge, we replace each edge with a copy of X. That is, for each edge (u, v) in G, we use a copy of X to connect clouds C_u and C_v in the new graph. Each cloud is partitioned so that the expanders do not overlap one another. The formal definition follows.

Definition 3.1 (Expanding Graph Product)**.** Let G be a D-regular graph on vertex set $[M]$ and let H be a d-regular graph on vertex set $[m]$. Assume some pre-defined ordering on the neighbors of each vertex in G and H (for example, lexicographically by label). Assume further that $D|m$ so that the vertices of H can be split evenly into D groups. Partition the vertices of H into D sets of equal size via the labeling. That is, the first part consists of the m/D vertices of smallest label, and so on. Let X be a bipartite graph with m/D vertices

in each part. The expanding graph product $\mathcal{X}(G, H, X)$ is a $(d + O(1))$-regular graph on vertex set $[M] \times [m]$ constructed as follows:

- Replace each vertex of $v \in G$ with the cloud $C_v \cong H$ consisting of m vertices with all edges included. Mathematically, a vertex $(v, u) \in \mathcal{X}(G, H, X)$ is the vertex in cloud C_v corresponding to vertex $u \in H$. The edge $((v, u_1), (v, u_2)) \in \mathcal{X}(G, H, X)$ if and only if the edge $(u_1, u_2) \in H$. C_v is partitioned exactly as H is; for $v \in G$ and index $i \in [D]$, let $P(v, i)$ denote the i^{th} part of C_v.
- For each edge $(u, v) \in G$, do the following: suppose v is the i^{th} neighbor of u and u is the j^{th} neighbor of v for $i, j \in [D]$. Overlay a copy of X going from $P(u, i)$ toward $P(v, j)$. Mathematically, let $u_1, u_2 \in G$ and $v_1, v_2 \in H$. For L and R ordered sets of size m/D, let $X(L, R)$ be a copy of X with its left part relabeled as L and its right part relabeled as R. Then $((u_1, v_1), (u_2, v_2)) \in \mathcal{X}(G, H, X)$ if and only if the following hold:
  - $(u_1, u_2) \in G$
  - For $i, j \in [D]$, u_1 is the neighbor of u_2 with i^{th} largest label and u_2 is the neighbor of u_1 with j^{th} largest label.
  - $v_1 \in L(u_1, i)$ and $v_2 \in L(u_2, j)$.
  - The edge $(v_1, v_2) \in X(P(u_1, i), P(u_2, j))$.

3.2 The Iteration

We state our simple iterative process for constructing the family of depth-robust graphs $\{G_n\}$ below. In the next chapter we will analyze this family and prove our claims of depth-robustness.

Definition 3.2. Fix $\epsilon > 0$. Let G be a D-regular ϵ-scaling-depth-robust graph on vertex set $[M]$ and let H be a d-regular ϵ-scaling-depth-robust graph on vertex set $[m]$ where $D|m$.

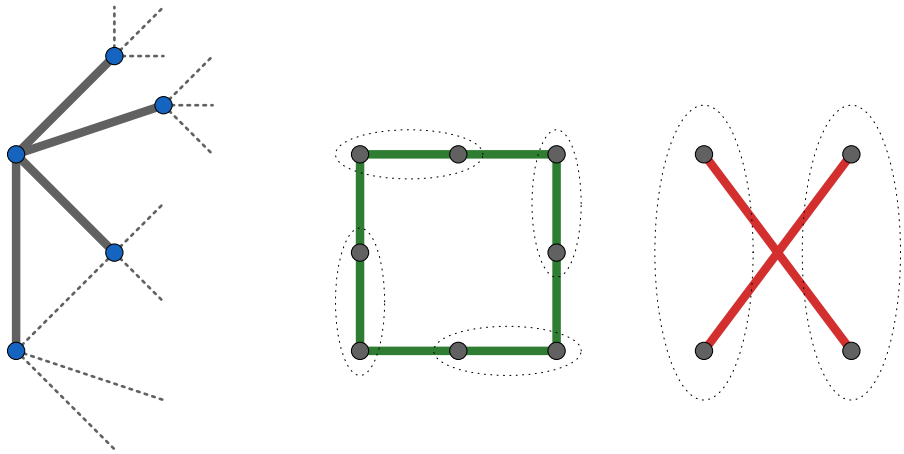

Figure 3.1: The base graphs G, H, X.

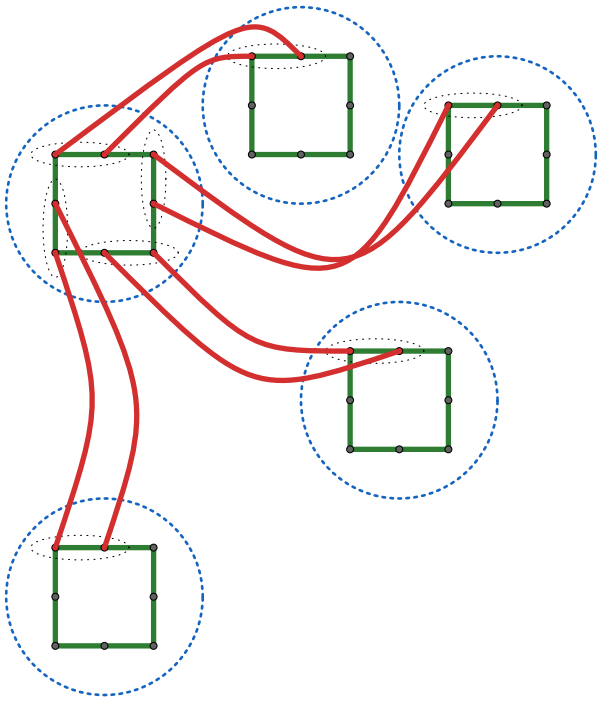

Figure 3.2: The expanding product $\mathcal{X}(G, H, X)$ of G, H, X

Assume some ordering on the neighbors of every vertex. Let X be a bipartite $(1/\epsilon)$-expanding graph of constant degree c with each part of size m/D. We define G_n recursively as follows:

$$G_0 = G$$

$$G_{k+1} = \mathcal{X}(G_k, H, X)$$

Since the size of G_n is not n, we will denote the size of G_k by $N(k)$ or simply N when the index is clear by context. Note that in the definition of $\mathcal{X}(G, H, X)$, we require that G be D-regular for some $D|m$ so that H can be properly partitioned. This means that in order for $G_{k+1} = \mathcal{X}(G_k, H, X)$ to be defined, G_k must be qm-regular for some integer q. In the following chapter, we analyze this construction and establish that G_k is in fact a regular graph and that m and D can be chosen to satisfy this constraint.

CHAPTER 4

Analysis

Having defined our construction, we will now analyze the depth-robustness and explicitness of $\{G_n\}$. To accomplish this, we first provide some basic analysis on the size and degree of G_n. We then prove the main theorem of this paper which states that the expanding product of two depth-robust graphs and an expanding graph is a new graph with strong depth robustness. This will imply the depth-robustness of our iteratively constructed $\{G_n\}$. Finally, we prove that our construction is in fact explicit by giving an efficient algorithm for computing neighbors.

4.1 Basic Analysis

Our first theorem establishes that the expanding product $\mathcal{X}(G_k, H, X)$ outputs a constant-degree, regular graph whose degree depends only on the degrees of H and X. Since H and X are kept constant throughout the iteration, this implies that the degree of G_n is constant. Later, we show this degree is a multiple of m (with the proper base graphs), satisfying the necessary constraint for iteration mentioned in the previous chapter.

Theorem 4.1. *Let $\{G_n\}$ be the family of graphs defined in Definition 3.2. Let d denote the degree of H and c denote the degree of X. Then for $k > 0$, G_k is a $(d + c)$-regular graph on Mm^k vertices.*

Proof. We prove the theorem by induction. In the base case, G_1 has Mm vertices by the definition of the expanding product. To see G_1 is $(d+c)$-regular, let v be an arbitrary vertex in G_1. Since v is part of some d-regular cloud (copy of H), v has d internal neighbors inside the same cloud. Since each part of H is overlaid with a single copy of X, v also has precisely

c external neighbors in some other cloud due to the degree of X. Thus, every $v \in G_1$ has precisely $d+c$ neighbors, as desired. For the inductive step, fix $k > 0$, and assume that G_{k-1} is a $(d+c)$ regular graph on Mm^{k-1} vertices. G_k has $m \cdot Mm^{k-1} = Mm^k$ vertices since each of the Mm^{k-1} vertices in G_{k-1} corresponds to m vertices in G_k. Our previous argument for the degree of G_1 depends only on H and X, so an identical argument establishes that G_k is $(d+c)$-regular. In particular, each vertex in G_k has precisely d internal neighbors and c external neighbors by the same logic as above. □

We have now established that the degree of G_k remains constant throughout the iteration. We have also found an explicit formula for $N(k) = Mm^k$. It remains to show that the degree $d+c$ is in fact a multiple of m, our necessary constraint for iteration. It is intuitive, however, that satisfying this constraint is possible because the parameters m and c are up to our choosing. The parameter d depends on m and in our case we will have $d = O(\log m)$. A detailed discussion of this constraint satisfaction can be found in Chapter 5.

4.2 Proving Depth-Robustness

The primary goal of this section is to prove the following theorem.

Theorem 4.2. *Fix $\epsilon > 0$. There exist base graphs G, H, X such that $G_n \in \{G_n\}$ defined according to Definition 3.2 is $(\Omega(N^{1-\epsilon}), \Omega(N^{1-\epsilon}))$-depth-robust.*

Technically our proof will not be complete until Chapter 5 where we establish the existence of regular ϵ-scaling depth-robust graphs of the correct parameters. What we can accomplish here is to prove that such parameters exist which give the desired depth-robustness of $\{G_n\}$. Then, in the next chapter, we show how to find base graphs matching these parameters.

Theorem 4.2 follows essentially by arithmetic from the following theorem.

Theorem 4.3. *Fix $\epsilon > 0$ and let the family $\{G_n\}$ be defined as in Definition 3.2. Then for any $a+b < 1-\epsilon$ and $A+B < 1-\epsilon$, $G_n \in \{G_n\}$ is $\left((\frac{am}{D})^n AM, (\frac{((b-D\epsilon)m)}{D})^n BM\right)$-depth-robust.*

With the proper constants a, b, m, M, D, ϵ, which are up to our choosing, the bases $\frac{am}{D}$ and $\frac{(b-D\epsilon)m}{D}$ in the above parameters exceed 1. This forces both terms to be almost linear, $\Omega(N^{1-\epsilon})$ for arbitrarily small $\epsilon > 0$. This gives the desired conclusion of Theorem 4.2.

Proof of 4.3 $\Longrightarrow$ 4.2. Assuming that Theorem 4.3 holds, proving the corollary is a matter of arithmetic and algebra. Let $G = G_i \in \{G_n\}$. We will use $|G| = N$ for the size of G. By Theorem 4.3, G is $((\frac{am}{D})^i AM, (\frac{((b-D\epsilon)m)}{D})^i BM)$-depth-robust. For ease of notation, let $C_a = \frac{a}{D}$, $C_A = AM$, $C_b = \frac{b-D\epsilon}{D}$, and $C_B = BM$. Since $N = m^i M$, solving for i we get $i = \log_m(\frac{N}{M})$. We will substitute this into the depth-robustness parameters from above in order to write them in terms of N, the number of vertices in G. We get the following:

$$
\begin{aligned}
&\Big(\Big(\frac{am}{D}\Big)^i AM, \Big(\frac{((b-D\epsilon)m)}{D}\Big)^i BM\Big) \\
=&\Big((C_a m)^{log_m(\frac{N}{M})} C_A, (C_b m)^{log_m(\frac{N}{M})} C_B\Big) \\
=&\Big(\Big(\frac{N}{M}\Big)^{log_m(C_a m)} C_A, \Big(\frac{N}{M}\Big)^{log_m(C_b m)} C_B\Big) \\
=&\Big(\Big(\frac{N}{M}\Big)^{\log_m(C_a)+\log_m(m)} C_A, \Big(\frac{N}{M}\Big)^{\log_m(C_b)+\log_m(m)} C_B\Big) \\
=&\Big(\Big(\frac{N}{M}\Big)^{1+\log_m(C_a)} C_A, \Big(\frac{N}{M}\Big)^{1+\log_m(C_b)} C_B\Big) \\
=&\Big(\big(C_A M^{-(1+\log_m(C_a))}\big) N^{1+\log_m(C_a)}, \big(C_B M^{-(1+\log_m(C_b))}\big) N^{1+\log_m(C_b)}\Big)
\end{aligned}
$$

The third equality comes from the fact that $x^{\log y} = y^{\log x}$ in general. This fact is easily seen by taking the logarithm of both sides of the equation. The leading coefficient of N is constant in both parameters, and the values $-\log_m(C_a)$ and $-\log_m(C_b)$ can be forced below any $\epsilon > 0$ with the appropriate choices of m, a, b, and D. This completes the proof. □

We now know that Theorem 4.3 (and an argument for the existence of proper base graphs) implies the desired depth-robustness bounds of Theorem 4.2. We will prove Theorem 4.3 via a simple induction on the result of the following, more substantial theorem.

Theorem 4.4. *Fix $\epsilon > 0$. Let G be a D-regular ϵ-scaling-depth-robust graph on M vertices and H be a d-regular ϵ-scaling-depth-robust graph on m vertices. Assume some ordering on*

the neighbors of each vertex. Let X be a $(1/\epsilon)$-expanding graph, and define $G' = \mathcal{X}(G, H, X)$. Then, for any $a + b < 1 - \epsilon$ and any $A + B < 1 - \epsilon$, G' is $(\frac{amAM}{D}, \frac{(b-D\epsilon)mBM}{D})$ depth-robust.

Once these theorems are proven, all that remains in our proof of existence is to show that there is some choice of base graphs G, H, X such that the recurrence constraint $(d + c)|m$ is satisfied and such that the depth-robustness of $\{G_n\}$ exceeds $(N^{1-\epsilon}, N^{1-\epsilon})$. Existence of these base graphs, along with the above results, complete the proof that $\{G_n\}$ exists and is indeed depth-robust. Proving its explicitness requires an additional argument which we provide in the following section.

The strategy for proving Theorem 4.4 will be as follows: we first prove in Lemma 4.5 that two "clouds", partitioned and overlaid with an expanding graph (as in the expanding graph product), maintain some form of depth-robustness. More precisely, as long as neither of the two clouds is "corrupted" by a large number of deleted nodes, a long path remains whose intersection with each cloud is large. This means a long path will remain through two clouds unless the adversary commits a large number of deletions to corrupting one of them. This lemma is proven using 4.6 and 4.7 as auxiliary lemmas. After this, we use Lemmas 4.8 and 4.9 to establish a correspondence between long paths in G and long paths in $\mathcal{X}(G, H, X)$. Since the structure of clouds mirrors the structure of G, the depth-robustness of G tells us exactly how many clouds must be corrupted in order to destroy long paths in $\mathcal{X}(G, H, X)$. Similarly, since each cloud has the structure of H, the depth-robustness of H tells us exactly how many nodes must be deleted to corrupt a particular cloud. Thus, an adversary must commit a large number of deletions to corrupt many clouds if they wish to destroy all of the long paths in $\mathcal{X}(G, H, X)$. More precise calculations yield the desired depth-robustness bounds.

As outlined above, we begin our proof of 4.4 with the following lemma. Intuitively, this says that overlaying an expanding graph between depth-robust graphs produces a graph with "balanced" depth-robustness; if one deletes sufficiently few vertices from each side of the resulting graph, a path remains which has a long intersection with each part of the graph.

Lemma 4.5. *Let $\epsilon > 0$ be arbitrarily small, and consider G_1 and G_2 be ϵ-scaling-depth-*

robust graphs on m vertices with some pre-defined ordering on their vertex labels. Let a, b be any constants satisfying $a + b < 1 - \epsilon$. For some D dividing m, partition each graph into D subgraphs of equal size according to the order of labels. Denote the i^{th} subgraph in G_1 as L_i^1 and the j^{th} subgraph in G_2 as L_j^2. For arbitrary $i, j \in [D]$, overlay a $|L_i^1| \times |L_j^2|$ bipartite $1/\epsilon$-expanding graph from L_i^1 to L_j^2 and call the resulting graph G. Then:

> *For any subset of vertices $S \subseteq G$ whose intersection with G_1 is at most $|S \cap G_1| < \frac{am}{D}$ and whose intersection with G_2 is at most $|S \cap G_2| < \frac{am}{D}$, there exists some path $P \subset (G/S)$ such that $|P \cap G_1| > \frac{(b-D\epsilon)m}{D}$ and $|P \cap G_2| > \frac{(b-D\epsilon)m}{D}$.*

We prove Lemma 4.5 using two auxiliary lemmas. Lemma 4.6 establishes that for any depth-robust graph G, such as those from [EGS75, ABP18], sub-graphs of G are also depth-robust. Lemma 4.7 will show that overlaying a bipartite expanding graph between two depth-robust graphs results in a new graph with the "balanced" depth-robustness property of Lemma 4.5. Together, these show that given two depth-robust graphs H and H', partitioning each graph and overlaying an expanding graph from a part in H to a part in H' produces a graph satisfying the balanced depth-robustness we seek in Lemma 4.5.

Lemma 4.6. *Fix $\epsilon > 0$. Let $H \in \{G_n^\epsilon\}$ be some ϵ-scaling-depth-robust graph on vertex set $[m]$. Partition H into D equal subgraphs denoted $\{L_i\}_{i=1}^D$, were L_k consists of vertices $\{\frac{(k-1)m}{D}, \ldots, \frac{km}{D} - 1\}$. Then for any $a + b < 1 - \epsilon$, each L_i is $(\frac{am}{D}, \frac{(b-(D-1)\epsilon)m}{D})$-depth-robust.*

Proof. Fix $\epsilon > 0$ and a and b such that $a + b < 1 - \epsilon$, and let H be as above. For arbitrary L_i, select a subset $S_{bad} \subset L_i$ of at most $\frac{am}{D}$ nodes from L_i.

Delete $S_{bad} \cup (H/L_i)$ (everything but the good vertices in L_i) from H, and call what remains S_{good}. $|S_{bad} \cup (H/L_i)| < \frac{am}{D} + \frac{(D-1)m}{D} = \frac{a+D-1}{D}m$. We now use the ϵ-scaling-depth-robustness of H. That is, since $H \in \{G_n^\epsilon\}$, 2.4 tells us that the number of nodes deleted is inversely related to the length of the guaranteed path (as long as the deletions do not exceed $(1 - \epsilon)m$). That is, for any $p < 1 - \epsilon$, if at most pm nodes are deleted from H, a path of length at least $(1 - p - \epsilon)m$ will remain. Thus, since $\frac{a+D-1}{D}m < (1 - \epsilon)m$ nodes were deleted

from H, we can find a path $P \subset S_{good}$ of length

$$|P| > (1 - \frac{a + D - 1}{D} - \epsilon)m = (\frac{1 - a}{D} - \epsilon)m > \frac{(b - (D - 1)\epsilon)m}{D}$$

as desired. The last inequality comes from the fact that $a+b < 1-\epsilon$ and thus $b+\epsilon < 1-a$. $\square$

Lemma 4.7. *Fix $\epsilon > 0$ and choose arbitrary $a + b < 1 - \epsilon$. Let L and R each be ϵ-scaling depth-robust graphs from $\{G_n^\epsilon\}$ each on m vertices (they need not be distinct). Let G be the result of overlaying a $(1/\epsilon)$-expanding graph from L to R. Then,*

> *For any subset of vertices $S \subset G$ whose intersection with L is at most $|S \cap L| < am$ and whose intersection with R is at most $|S \cap R| < am$, there remains a path $P \subseteq (G/S)$ of length at least $|P| > 2(b-\epsilon)m$ such that $|P \cap L| > (b-\epsilon)m$, and $|P \cap R| > (b-\epsilon)m$.*

Proof. Select a subset S of at most $2am$ vertexes from G such that $|S \cap R|$ and $|S \cap L|$ are each at most am and call nodes in this set "bad". Let $S_L = S \cap L$ and $S_R = S \cap R$. Let $\sigma m = |S_L|$ be the number of bad nodes in S_L (where σ is the fraction of bad nodes). Then, since $|S| < 2am$, we must have $|S_R| < (2a - \sigma)m$. Since both subsets are of size at most am by assumption, we can invoke the depth-robustness of L and R. We again utilize the fact that L and R are ϵ-scaling depth-robust by Theorem 2.4 to conclude that there exists a path P_L in L of length $|P_L| > (1 - \sigma - \epsilon)m$. Similarly, there exists a path P_R in R of length $|P_R| > (1 - (2a - \sigma) - \epsilon)m$. We wish to find a path P in G of length $|P| > 2(1 - a - 2\epsilon)m = 2(b - \epsilon)m$. We also want this path to have an intersection of size at least $(b - \epsilon)m$ with both P_L and P_R.

Let $\hat{P_L} \subseteq P_L$ be the ϵm-tail of P_L consisting of the last $[\epsilon m]$ nodes and let $\hat{P_R} \subseteq P_R$ be the ϵm-head of P_R consisting of the first $[\epsilon m]$ nodes (as determined by the direction of edges in the path). Then, since L and R are connected by a $(1/\epsilon)$-expanding graph, and $\hat{P_L}$ and $\hat{P_R}$ are of size ϵm, we are guaranteed the existence of an edge from $\hat{P_L}$ to $\hat{P_R}$ by Definition 2.2. We can create the path P by connecting P_L and P_R with this extra edge, potentially

losing the ϵm head and tail in the process. The length of the path is at least

$$\begin{aligned}
|P| &= |P_R| - |\hat{P}_R| + |P_L| - |\hat{P}_L| \\
&\geq |P_R| - (\lceil \epsilon m \rceil - 1) + |P_L| - (\lceil \epsilon m \rceil - 1) \\
&> (1 - \sigma - \epsilon)m + (1 - (2a - \sigma) - \epsilon)m - 2\epsilon m \\
&= 2 - 2a - 4\epsilon m \\
&= 2(1 - a - 2\epsilon)m \\
&> 2(b - \epsilon)m
\end{aligned}$$

The last inequality comes from the fact that $a+b < 1-\epsilon$. It remains to show that $|P \cap P_R|$ and $|P \cap P_L|$ are both at least $(b-\epsilon)m$. This can be seen by observing that $P \cap P_R = P_R/\hat{P}_R$ and thus

$$|P \cap P_R| = |P_R| - |\hat{P}_R| = (1 - \sigma - \epsilon)m - \epsilon m$$

Since $\sigma < a$, we have that

$$(1 - \sigma - \epsilon)m - \epsilon m > ((1 - a - \epsilon) - \epsilon)m > (b - \epsilon)m$$

using the fact that $1 - a - \epsilon > b$. An identical argument shows that $|P \cap P_L| > (b - \epsilon)m$, and this completes the proof. □

We are now in position to prove Lemma 4.5 using Lemmas 4.6 and 4.7 as building blocks. The proof is a relatively straightforward application of Lemma 4.7 to the partitions of H in Lemma 4.6.

Proof of Lemma 4.5. Let G be constructed as in the hypotheses of the lemma. By Lemma 4.6, G_1's subgraph L_i^1 and G_2's subgraph L_j^2 are each $(\frac{am}{D}, \frac{(b-(D-1)\epsilon)m}{D})$-depth-robust. Applying Lemma 4.7 with the same value of ϵ, we conclude that if a "balanced" set S is deleted whose intersections with G_1 and with G_2 are each at most $\frac{am}{D}$, a path remains in G/S with length $|P \cap G_1| > \frac{(b-D\epsilon)m}{D}$ in G_1 and with length $|P \cap G_2| > \frac{(b-D\epsilon)m}{D}$ in G_2. Thus, the lemma is proven. □

We have now established that two partitioned depth-robust graphs overlaid with an expanding graph will contain a long, balanced path so long as neither graph corrupted by removing too many nodes. Our next lemma applies this to the clouds in our expanding product $\mathcal{X}(G, H, X)$. The conclusion is that two adjacent clouds which are not corrupted contain a long, balanced path. The following lemma extends this inductively to say that a sequence of adjacent clouds, none of which are corrupted, implies the existence of a long path throughout these clouds. Once these two lemmas are proven, we are in position to prove Theorem 4.4, the basis of our main Theorem 4.3. Once we have established the direct correspondence between paths in G and paths in $\mathcal{X}(G, H, X)$, the depth-robustness of G tells us how many clouds one must corrupt to destroy long paths, and the depth-robustness of H tells us the price (in deletions) of corrupting a single cloud.

Before we state and prove our lemmas, we must formalize what it means to "corrupt" a cloud. Suppose we select a subset $S \subset V(\mathcal{X}(G, H, X))$ and call them bad nodes. If a particular cloud contains more than $\frac{am}{D}$ bad nodes, we call it a *bad* cloud; otherwise, we call it a *good* cloud.

Lemma 4.8. *Fix $\epsilon > 0$. Let G (and H) be D-regular (d-regular) ϵ-scaling depth-robust graphs as in Theorem 4.4. Suppose some subset of vertices in $\mathcal{X}(G, H, X)$ is selected as bad nodes. For any vertices $u, v \in G$, if C_u, C_v are good clouds and edge $(u, v) \in G$, then there exists a path P in $\mathcal{X}(G, H, X)$ of length $|P| > \frac{2(b-D\epsilon)m}{D}$ such that $|P \cap C_u| > \frac{(b-D\epsilon)m}{D}$ and $|P \cap C_v| > \frac{(b-D\epsilon)m}{D}$*

Proof. By the construction of $\mathcal{X}(G, H, X)$, C_i and C_j satisfy the hypotheses of Lemma 4.5. Since C_i and C_j are good clouds by assumption, each has had at most $\frac{am}{D}$ nodes deleted. Thus, by the conclusion of Lemma 4.5, a path exists in $\mathcal{X}(G, H, X)$ whose intersection with each cloud exceeds $\frac{(b-D\epsilon)m}{D}$, as desired. The total length comes from summing the lengths of these two disjoint halves. □

Lemma 4.9. *Fix $\epsilon > 0$ and suppose G and H are regular and ϵ-scaling depth-robust as before. Suppose some subset of vertices in $\mathcal{X}(G, H, X)$ is bad. Suppose clouds C_{v_1}, C_{v_2},...,C_{v_k} are all good clouds, where C_{v_i} is the cloud corresponding to vertex $v_i \in G$. If there exists a*

path $(v_1, v_2, \ldots, v_k)$ in G, then there exists a corresponding path P in $\mathcal{X}(G, H, X)$ of length $|P| > \frac{k(b-D\epsilon)n}{D}$ such that for each $i \in [k]$, $|P \cap C_{v_1}| > \frac{(b-D\epsilon)n}{D}$.

Proof. We prove by induction. When $k = 2$, Lemma 4.8 applied to C_{v_1} and C_{v_2} proves the statement. For the inductive step, fix $k > 2$. Suppose $C_{v_1}, \ldots, C_{v_k}$ are good clouds in $\mathcal{X}(G, H, X)$ and further suppose that there exists a path $(v_1, \ldots, v_k)$ in G. For our inductive hypothesis, we assume there exists a path P_{k-1} in $\mathcal{X}(G, H, X)$ of length $|P_{k-1}| > \frac{(k-1)(b-D\epsilon)m}{D}$ which satisfies $|P_{k-1} \cap C_{v_i}| > \frac{(b-D\epsilon)m}{D}$ for every $i \in [k-1]$. Now, we apply Lemma 4.8 to clouds $C_{v_{k-1}}$ and C_{v_k}. Since both are good clouds, there exists a path P_k in $\mathcal{X}(G, H, X)$ of length $|P_k| > \frac{2(b-D\epsilon)m}{D}$ such that $|P_k \cap C_{v_{k-1}}| > \frac{(b-D\epsilon)m}{D}$ and $|P_k \cap C_{v_k}| > \frac{(b-D\epsilon)m}{D}$. Now we will join P_{k-1} with P_k inside $C_{v_{k-1}}$ to create the path P that we want. Let $\hat{P}_{k-1}$ denote the part of P_{k-1} intersecting $C_{v_{k-1}}$ (of size $|\hat{P}_{k-1}| > \frac{(b-D\epsilon)m}{D}$), and let $\hat{P}_k$ denote the part of P_k intersecting $C_{v_{k-1}}$ (of size $|\hat{P}_k| > \frac{(b-D\epsilon)m}{D}$). We wish to claim that there exists a path $\hat{P}$ in $C_{v_{k-1}}$ connecting $\hat{P}_{k-1}$ to $\hat{P}_k$. Since $C_{v_{k-1}}$ is a copy of H, it is ϵ-scaling depth-robust. Only $\frac{am}{D}$ nodes have been deleted from C_{v_k}, so by Theorem 2.4, there exists a path $\hat{P}$ in $C_{v_{k-1}}$ of length at least $|\hat{P}| > (1 - \frac{a}{D} - \epsilon)m$. Since $\frac{am}{D}$ nodes are bad, this is equivalent to saying that there are at most ϵm good nodes in $C_{v_{k-1}}$ which do not lie on the path. Since both partial paths $\hat{P}_k$ and $\hat{P}_{k-1}$ inside $C_{v_{k-1}}$ are of size greater than ϵm, the pigeonhole principle tells us that both partial paths intersect $\hat{P}$. We then construct our desired path by connecting $\hat{P}_{k-1}$ to $\hat{P}_k$ via $\hat{P}$. In the worst case, all ϵm good nodes in $C_{v_{k-1}}/\hat{P}$ lie in either $\hat{P}_{k-1}$ or $\hat{P}_k$ and those nodes are lost from the total length. This connects P_{k-1} to P_k, and we call the resulting path P. As a rough lower bound on the length, we have

$$\begin{aligned}
|P| &> |P_k| + |P_{k-1}| + |\hat{P}/(\hat{P}_{k-1} \cup \hat{P}_k)| - \epsilon m \\
&= \frac{k(b-D\epsilon)m}{D} + (b-D\epsilon)m - \frac{2(b-D\epsilon)m}{D} - \epsilon m \\
&= \frac{k(b-D\epsilon)m}{D} + \frac{((D+2)(b-D\epsilon) - D\epsilon)m}{D} \\
&> \frac{k(b-D\epsilon)m}{D}
\end{aligned}$$

The last inequality comes from the fact that $D > 0$, $b > 0$, and ϵ is arbitrarily small. This

completes the proof by induction. □

We are now in position to prove our main theorems. We restate them here for convenience. Recall, we have already established that Theorem 4.3 $\implies$ Theorem4.2.

Theorem 4.4. *Fix $\epsilon > 0$. Let G be a D-regular ϵ-scaling-depth-robust graph on M vertices and H be a d-regular ϵ-scaling-depth-robust graph on m vertices. Assume some ordering on the neighbors of each vertex. Let X be a $(1/\epsilon)$-expanding graph, and define $G' = \mathcal{X}(G, H, X)$. Then, for any $a + b < 1 - \epsilon$ and any $A + B < 1 - \epsilon$, G' is $(\frac{amAM}{D}, \frac{(b-D\epsilon)mBM}{D})$ depth-robust.*

Theorem 4.3. *Fix $\epsilon > 0$ and let the family $\{G_n\}$ be defined as in Definition 3.2. Then for any $a+b < 1-\epsilon$ and $A+B < 1-\epsilon$, $G_n \in \{G_n\}$ is $\left((\frac{am}{D})^n AM, (\frac{((b-D\epsilon)m)}{D})^n BM\right)$-depth-robust.*

Theorem 4.2. *Fix $\epsilon > 0$. There exist base graphs G, H, X such that $G_n \in \{G_n\}$ defined according to Definition 3.2 is $(\Omega(N^{1-\epsilon}), \Omega(N^{1-\epsilon}))$-depth-robust.*

Proof of Theorem 4.4. Fix $\epsilon > 0$. Let $G \in \{G_n^\epsilon\}$ be a D-regular graph on M vertices and $H \in \{G_n^\epsilon\}$ be a d-regular graph on m vertices. Fix arbitrary a, b, A, B such that $a+b < 1-\epsilon$ and $A + B < 1 - \epsilon$. Select a set S of bad vertices of size at most $\frac{amAM}{D}$. Since it requires at least $\frac{am}{D}$ bad vertices to corrupt any single cloud, there are at most AM bad clouds. Call this set of bad clouds C, where $|C| < AM$. Recall that clouds in $\mathcal{X}(G, H, X)$ correspond to vertices in G and consider the set of vertices $V_C \subseteq G$ corresponding to the set C. Note, we must have $|V_C| < AM$ since we have a bijection between the sets. G is (AM, BM)-depth-robust (since $A + B < 1 - \epsilon$), which implies that there exists a path P_G of length at least BM in G/V_C. Observe that every vertex in P_G corresponds to a good cloud in $\mathcal{X}(G, H, X)$. Lemma 4.9 then tells us that there exists a path P in $\mathcal{X}(G, H, X)$ of length $|P| > \frac{(b-D\epsilon)mBM}{D}$. Thus, $\mathcal{X}(G, H, X)$ is $(\frac{amAM}{D}, \frac{(b-D\epsilon)mBM}{D})$-depth-robust, as desired. □

Note that this bound holds for any a, b, A, B satisfying $a + b < 1 - \epsilon$ and $A + B < 1 - \epsilon$, justifying our freedom of choice over these parameters in earlier proofs. We are now in position to prove Theorem 4.3 by induction.

Proof of Theorem 4.3. We can see from Theorem 4.4 that if G_{k-1} is (e, d)-depth-robust, then G_k is $(\frac{am}{D} \cdot e, \frac{(b-D\epsilon)m}{D} \cdot d)$-depth-robust. With this observation, we prove the theorem by induction. In the base case, when $k = 0$, we already have that $G_0 = G$ is (AM, BM)-depth-robust by assumption. In the inductive step, suppose G_{k-1} is $((\frac{am}{D})^{k-1}AM, (\frac{(b-D\epsilon)m}{D})^{k-1}BM)$-depth-robust. Then by our observation, G_k is $((\frac{am}{D})^{k}AM, (\frac{(b-D\epsilon)m}{D})^{k}BM)$-depth-robust, as desired. □

Once we prove the existence of regular, ϵ-scaling depth-robust graphs satisfying the recurrence constraint, we will have established explicit lower bounds on the depth-robustness of our family of graphs $\{G_n\}_{n=0}^{\infty}$. We establish this in Chapter 5. Before we do this, we establish the explicitness of $\{G_n\}$, as this distinguishes our result from other existential results like that of [ABP17].

4.3 Explicitness

In order to show explicitness, we must give some efficient algorithm for computing the neighbors of vertex i in the graph G_n given only the label i and a description of G_n (in this case, simply the index n). In particular, the algorithm must run in time $\text{poly}(\log |i|)$. That is, polynomial in terms of the length of the label i. Since G_n is of size $|G_n| = M \cdot m^n$, the largest index we must encode is of size $i = m \cdot m^n$. Thus $\log i = \log M + n \log m$ bits suffice. Our algorithm must then run in time $\text{poly}(\log M + n \log m)$ which is precisely $\text{poly}(n)$. We begin by offering an intuitive description of our algorithm and then afterward provide details as well as proofs of correctness and efficiency.

We leverage the fact that our construction is iterative and construct the algorithm recursively. By the nature of our construction, vertices are naturally arranged in groups and subgroups. An analogy would be a universe composed of galaxies, each composed of solar systems, planets, continents, and so on. This structure arises due to the recursive usage of a replacement product, where at each stage every node is blown up into a larger group of nodes. If we already have an explicit description of G_{n-1}, this allows us to efficiently

compute the neighboring clouds to any particular cloud in G_n. If we can use the label i to determine which cloud we are in, we can use this information and the explicitness of the expanding graph X to determine any and all neighbors in other clouds. To find neighbors in the same cloud, we can simply use the explicitness of H, since every cloud is a copy of H. Thus, the explicitness of G, H, X, and G_{n-1} are sufficient for an explicit construction of G_n. This is the essence of our inductive argument. The depth of the recursive stack is precisely n, so to achieve a running time of poly(n) we must show that each execution on the stack takes time poly(n).

The task then is how to use the label i to determine which cloud contains vertex i. When explicitly defining other exponentially large objects (e.g. pseudorandom functions), a common visualization is to consider a tree with exponentially-many leaves. Instead of actually storing all of these values, which requires too many resources, one would rather create an efficient algorithm for computing any one particular value on the fly. Starting at the root, the algorithm navigates the tree to the correct index i using the bits of i essentially as turn-by-turn navigation. Once the leaf at index i is reached, the value stored there is returned by the algorithm. Our construction implies a tree with a branching factor of m (except the very first level, which has a branching factor of M). Thus, each chunk of $\log m$ bits (or $\log M$ in the first step) from our index i tells us which edge from this tree to follow. In our graph, this corresponds to deciding which one of the m subgroups (or M groups in the first step) contains vertex i and "zooming in" on that subgroup. With this view, level $n+1$ of the tree represents the nodes of G_n. To avoid dealing with any trees of infinite height, we use the initial value n to "chop" our tree at the $(n+1)^{\text{st}}$ level, since this is as much as we will need.

Recall that we assumed some ordering on the labels of the M nodes of G and the m nodes of H. For simplicity, assume these labels to be simply the integers $[M]$ and $[m]$ respectively, and assume without loss of generality that $M > m$. By intelligently labeling vertices, we can make navigation of our tree extraordinarily simple. When a node with label i is blown up into the cloud C_i, we simply use label i as the prefix for every node in the cloud. As a suffix, each node appends a single integer in $[m]$ representing their ordered labeling within

the cloud. Now, the first $\log M$ bits of i will represent the original vertex of $G = G_0$ that was blown up to eventually contain vertex i. The next $\log m$ bits will tell us the vertex of G_1 containing i, and so on. Each chunk of bits tells us exactly which edge in our tree to follow so that we end up at the leaf with label i. This reasoning also tells us that the label of a vertex in G_n will be a string i of precisely $|i| = \log M + n \log m$ bits.

We are now in position to prove the explicitness of our construction.

Theorem 4.10. *Consider the family of graphs $\{G_n\}$ from Theorem 4.3. There exists an algorithm for computing the neighbors of vertex i in G_n in time poly$(\log i)$ =poly(n).*

Proof. The base case of the induction is trivial since $G = G_0$ is assumed to be explicit. Fix $n > 0$. We will define our algorithm $\mathsf{neighbors}(n, i)$ as a sequence of 7 steps. We will show that each step takes time at most poly(n) =poly$(\log i)$. Since they are performed sequentially, taking the sum will show that a single level of the recursive stack takes time poly(n). Since the depth of the stack is n total calls, we will conclude that the total computation is poly(n). The high-level steps for computing $\mathsf{neighbors}(n, i)$ are as follows:

1. Parse i as an element of $[M] \times [m]^n$.
2. Compute internal neighbors of i (within the same cloud).
3. Determine which part of its own cloud i lies within.
4. Compute which clouds neighbor the cloud containing i.
5. Determine which neighboring cloud connects to the part containing i.
6. Determine which part of that neighboring cloud connects to the part containing i.
7. Compute external neighbors of i (which all lie within that part of that neighboring cloud).

We elaborate on each of the six steps and give upper bounds on the complexity of each task. We assume that each vertex in G is labeled by an integer in $[M]$, and each vertex in H by an integer in $[m]$.

1. *Parse i as a string in $[M]\times[m]^n$.* We perform a linear scan of the label i. After scanning the first $\log M$ bits, we interpret them as an binary integer $p_0 \in [M]$. Analogously, interpret the following n contiguous sequences of $\log m$ bits as binary integers $q_1, \ldots, q_n$ from the set $[m]$. Our "translated" label is the list $L = \{p_0, q_1 \ldots, q_n\}$. Since all conversions can be done in constant time via a lookup table of constant size, this requires $O(n)$ total work.

2. *Compute internal neighbors of i.* Our vertex i exists within some cloud C_i, and each cloud is a copy of H. Intuitively, we should be able to use the explicitness of H to determine internal neighbors. By construction, All translated labels of vertices in C_i share the same prefix $P = \{p_0, q_1 \ldots, q_{n-1}\}$. This is precisely the label of the vertex in G_{n-1} which was blown up into C_i. The suffix (final element) of each vertex's translated label determines the ordering of vertices within this cloud. The suffix of vertex i is the symbol $q_n \in [m]$. We use the explicit description of H to compute the neighbors of vertex $q_n \in H$ and simply append each result to the prefix P. This requires time at most $\text{poly}(\log m) + O(1) = O(1)$ computation by the definition of explicitness.

3. *Determine which part of its own cloud i lies within.* This requires only arithmetic. Again viewing i's cloud C_i as a copy of H, the suffix q_n determines relative ordering within the cloud and the cloud is partitioned according to this ordering. Thus, we need only look at the intervals $\{[\frac{km}{D}, \frac{(k+1)m}{D}) : k < D\}$ and choose the one which contains the integer q_n. This follows how H was initially partitioned: into subsets of m/D vertices by lexicographical order. This requires at most $D = O(1)$ computation.

4. *Compute which clouds neighbor the cloud containing i.* If we view each cloud as a vertex, then our graph G_n shrinks back into G_{n-1}. This means computing which the neighboring clouds of C_i reduces to computing which vertices in G_n neighbor the vertex in G_{n-1} which expands into C_i. The label of that vertex is precisely the prefix P. We can then compute the neighboring clouds via a single recursive call to $\mathsf{neighbors}(n-1, P^*)$, where P^* is the prefix P interpreted as a binary string. This takes at most $\text{poly}(n-1)$ by the inductive hypothesis.

5. *Determine which neighboring cloud connects to the part containing i.* For $n > 1$, G_n is $(d + c)$-regular where d and c are the degrees of H and X respectively. We partition each cloud into $d + c$ parts so that each part houses a single external edge (which will soon be replaced by an expanding graph). We know which part houses vertex i from (3) and we know all possible clouds it might connect to from (4). Now we must determine which one. Fortunately in our construction there is a pre-defined ordering on the neighbors (or equivalently, edges) of a vertex which is the ordering of the labels. We have free choice over which part to assign each external edge, so we use the same ordering to reduce computation. That is, the edge leading to the k^{th} largest external neighbor (according to label) is housed by part k of the cloud. In step (3), we already determined which part of C_i vertex i is in. We also have a list of the $d + c$ internal neighbors from (2) which we can sort in constant time. We can then retrieve the correct neighboring cloud C_j by taking the k^{th} largest label from the list of neighbors computed by (4). This takes $O(1)$ time.

6. *Determine which part of that neighboring cloud connects to the part containing i.* Now that we know the cloud C_j connecting to the part of C_i containing i, we must zoom in on C_j and determine which specific part connects with C_i. If we can get the neighboring clouds of C_j in order, we can determine where C_i lies in that ranking. Since the k^{th} neighbor is connected to the k^{th} part, that ranking and some arithmetic will tells us exactly which part of C_j connects with C_i at the part containing i. We can get that list by recursively calling $\mathsf{neighbors}(n-1, l)$ at a cost of $\text{poly}(n-1)$, where l is the label of C_j computed in (5). We can then sort the list in $O(1)$ time, since it is of constant size $d + c$. Finally, we can linearly scan for the string P^*, the label of C_i, and compute the index k of that label in the list. This takes at most $O(d + c) = O(1)$ time and we conclude that the correct part of C_j has suffixes in the set $[\frac{km}{d+c}, \frac{(k+1)m}{d+c})$. The labels of the part are obtained by appending these suffixes to the label l of C_j. The total work in this step is then at most $\text{poly}(n)$.

7. *Compute external neighbors of i.* We are now in position to compute all neighbors

outside of i's own cloud. We know for a fact that the only external edges are from the copy of X overlaid between the part containing i and the neighboring part from (6). This means that all possible external neighbors are in the list from (6). We can use the explicitness of X to compute exactly which of the vertices from (6) are adjacent to vertex i. This takes at most $\text{poly}(\log |X|) = O(1)$ by the definition of explicitness. Append this list to the list of internal neighbors from (2) and return the result.

The correctness of this algorithm is apparent by the construction of G_n. All neighbors that lie within the same cloud and must be included in the list of internal neighbors from (2), otherwise contradicting the explicitness of H. Similarly, any external neighbors being excluded from the list in (7) contradicts the explicitness of X.

To complete the proof, we simply show that our inductive hypothesis implies $\mathsf{neighbors}(\mathsf{n}, \mathsf{i})$ runs in time at most $\text{poly}(\log i)$ $=\text{poly}(\log i)$. Since all seven steps are performed sequentially, we need only sum the running times of each. This gives an upper bound of $O(n) + 4O(1) + 2\text{poly}(n-1) = \text{poly}(n)$ total work, as desired.

□

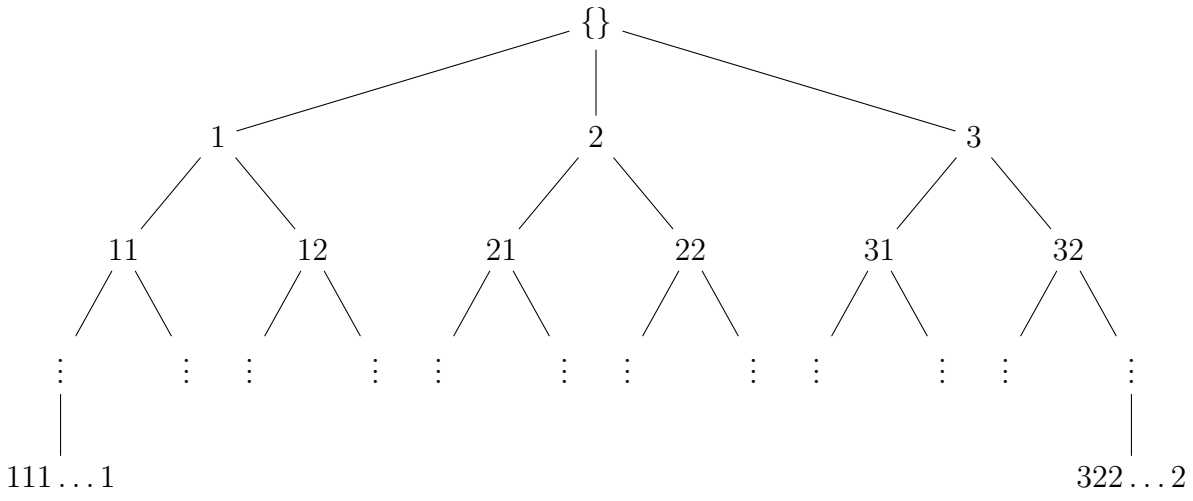

Figure 4.1: Tree with translated labels for $M = 3$, $m = 2$

Corollary 4.11. *The family of graphs $\{G_n\}$ is explicit.*

Proof. The conclusion comes from simply applying the definition of explicitness to the result of the previous theorem. □

CHAPTER 5

The Base Graphs

Our iterative construction is based on three graphs: G, H, and X. G and H are both regular, ϵ-scaling depth-robust graphs of some fixed constant sizes and X is a regular expanding graph of some fixed constant size. In this chapter we describe how to construct such graphs. We refer to previous work in this field, as such problems have been studied thoroughly. We then show that such graphs can be constructed to additionally satisfy the recurrence constraint necessary for our iteration. Namely, we will have that $d + c$ divides m, where d is the degree of each vertex in H, c is the degree of each vertex in X, and m is the number of vertices in H. This allows us to evenly divide H into $d + c$ copies (one for each outgoing edge in $\mathcal{X}(G, H, X)$) and continue the iteration.

To construct X, our constant degree expanding graph, we refer to work by Reingold, Vadhan, and Wigderson [RVW00]. As an addendum to their main result, they give an explicit construction for A-expanding graphs. To construct G and H, we could perform a brute-force search since it is only over a constant-size domain. However, since degree $\text{poly}(\log n)$ is sufficiently low for our purposes, an explicit construction exists due to Erdös, Graham, and Szemerédi. Improved analysis on this graph family and minor modifications by Alwen, Blocki, and Pietrzak gives the scaling property we require, and with minor modification one can achieve regularity also. All of these modifications can easily be included in any explicit construction, as they require only a constant number of neighbors to be added. We describe these constructions in detail below.

5.1 Constructing Expanding Graphs

As described in [RVW00], one can explicitly construct an A-expanding graph using extractors. We will restate their result without proof as we use expanding graphs in a black-box manner throughout this paper. For more details, see the proof of Corollary 7.6 in [RVW00].

Lemma 5.1 (Corollary 7.6 [RVW00])**.** *For every N and A there is a (regular) A-expanding graph $G_{N,A}$ of size N with degree $O(A \cdot \log^4 A)$ such that the rotation map of $G_{N,A}$ can be computed in time* $\text{poly}(\log N, 2^A)$.

For $A = 1/\epsilon$, the only relevant term is $\text{poly}(\log n)$. Thus, by definition, we have explicitly constructed A-expanding graphs of any size for any value of A.

5.2 Constructing Depth-Robust Graphs

This family of graphs was first constructed by Erdös, Graham, and Szemerédi in [EGS75]. In [ABP18], Alwen, Blocki, and Pietrzak improve on the analysis of depth robustness to establish a stronger form of depth-robustness which we informally call the scaling property. We describe the construction from [EGS75] below.

Let $n = 2^k$ for some large $k \in \mathbb{N}$ and let $G_n = (V, E)$ be a graph on vertex set $V = [n]$. For any positive integers v and m (not necessarily in $[n]$), let $D_v(m)$ denote the set $V \cap \{v, v+1, ..., v+m-1\}$. Fix an arbitrarily small constant $\epsilon_1 > 0$. We form the edge set E of G_n as follows:

1. For each $v \in V$, the edges $\{(v, x) | x \in D_{v+1}(4 \log n)\}$ are in E.

2. For each t dividing n in the range $(\log(n) - 1) \leq t < n$ and for each i as specified below, a $1/\epsilon_1$-expanding graph is formed between the vertex sets $A = D_{m \cdot t}(t)$ and $A' = D_{(m+i) \cdot t}(t)$, $0 \leq m < (n/t)$, where $i = 1, 2, ..., 10$ (or if i cannot assume value 10 because $(m + 10) \cdot t > n$, then it ranges from 1 to $n/t - m$). All edges are directed from x to y with $x < y$.

In this original construction, Claim 1 of [EGS75] establishes that for n sufficiently large, G_n is a so-called "local expander". That is, small subsets of G_n maintain the expanding property. Further analysis in [ABP18] establishes that with the addition of constantly many edges, this can be achieved for every value of $n > 0$. With this modification, for some fixed $\epsilon > 0$, the resulting G_n has indegree $O(\log n)$ and is (an, bn)-depth-robust for every $a + b < 1 - \epsilon$. Introducing explicit expanding graphs as in [MMV13] gives an explicit construction of ϵ-scaling depth-robust graphs with indegree $O(\log^2(n))$.

By minor modifications to this construction, which will not reduce depth-robustness nor increase the degree, we can obtain a regular graph. The irregular vertices in G_n are the last few; when the set $D_v(m)$ would end up overshooting the last vertex n, we decide not to include those edges. This is what makes the construction a DAG, as these last edges would create cycles and remove the sink. We simply choose to include these edges but in the reverse orientation. This pads the degree of the vertices to be equal to all others without creating any cycles. Let $D'_v(m)$ denote the set $\{v \mod n, (v+1) \mod n, ..., (v+m-1) \mod n\}$. Then we replace $D_v(m)$ with $D'_v(m)$ in the previous construction, change the range of i to be $i = 1, 2, ..., \min(10, n/t)$, and keep all other notions the same. All edges are still directed from x to y for $x < y$. Since we are adding edges to G_n, this cannot possibly decrease its depth robustness. After step (1) of this modified construction, each vertex touches precisely $2 \times 4 \log n = 8 \log n$ linked edges. During step (2), for each proper value of t, the number of expanding graphs overlaid on each vertex is $2 \times \min(10, n/t)$. As we will show in section 5.1, one can explicitly construct regular expanding graphs of degree c. Using such a graph in step (2), for each value of t, the degree of each vertex increases by $2 \times \min(10, n/t) \times c$, where c is the degree of a expanding graph. This proves that the modified G_n is regular. Let $\mathbb{T}$ denote the set of all possible value of t, and let $|\mathbb{T}|$ denote the size of $\mathbb{T}$. Notice that the number of iterations is $|\mathbb{T}|$. Since n is divisible by t and $n = 2^k$, $\mathbb{T} = \{2^{k-1}, 2^{k-2}, \ldots, 2^{k-a} | a \in \mathbb{N}, 2^{k-a} \geq \log n - 1\}$; therefore $|\mathbb{T}| \leq k = O(\log n)$. The number of iterations is also at most $O(\log n)$. Thus, after step (2), each vertex in G_n has degree at most $(8 + 20 \times c) \log n = O(\log n)$. We conclude that G_n is regular, has degree at most $O(\log n)$, and is (an, bn)-depth-robust for every $a + b < 1 - \epsilon$.

5.3 Satisfying the Recurrence Constraint

The final piece of the puzzle is to show that we can find H and X as described above which satisfy the following constraint: $d + c$ divides m, where d is the degree of H, c is the degree of X, and m is the size of H. To accomplish this, first fix G and H of the desired size with desired degree. According to our construction, if H is of size m it has degree $d = O(\log^2(m))$. Now all we need to do is find some value of c such that $d + c$ divides m and construct a regular X with that degree. Finding all possible values of c is simple arithmetic. Corollary 7.6 from [RVW00] allows us to construct an A-expanding graph of degree $O(A \cdot \log^4 A)$. We can then pad extra edges to each vertex to achieve the desired degree. This padding of edges cannot possibly reduce the expansion properties of the graph. A more formal statement of this follows.

Construct H using the modified construction from section 5.2. Suppose H has size m and degree $d = O(\log m)$. Consider the set of potential c values $S = \{c : d + c \text{ divides } m\}$. S is certainly nonempty, as the value $m - d \in S$. Now, construct X for the desired value of ϵ. X will have degree $c' = O(\frac{1}{\epsilon} \log^4(\frac{1}{\epsilon})$. For large enough m, this is certainly below $m - d$. Thus there exists some $c^* \in S$ such that $c^* \geq c'$. Consider the smallest such c^* and call it c. Evenly pad each vertex of X with $c - \deg(X)$ edges so that the new degree of each vertex is c and call the new graph X'. $c \in S$ so we can conclude that $d + c$ divides m as desired. X' still maintains all expansion properties of X as edges have only been added to any crossing set, none removed. Thus, graphs H and X can indeed be chosen to satisfy the recurrence constraint. This removes all assumptions from our construction and establishes our iterative construction indeed produces a family $\{G_n\}$ satisfying Theorem 4.3.

CHAPTER 6

Applications

The depth-robust graph is a main component in many cryptographic primitives e.g. proofs of sequential work [MMV13] and memory-hard functions [AS14, AS15, ABP17, ABP18]. We now give an overview of the potential applications of our result in these areas.

6.1 Cumulative Pebbling Complexity & Memory-Hard Functions

Many new technologies, such as spam filters and Bitcoin, utilize proof-of-work style protocols. Such protocols utilize some kind of "hard" function (such as an inverse hash function) and require participants to evaluate this hard function in order to solve some kind of puzzle and gain a reward. In spam filtering, this reward is the ability to successfully send a message. In Bitcoin, it's the authority to certify the next block in the chain and to claim the newly mined currency. Regardless, all such protocols are based on the existence of some underlying function which is hard to compute. Unfortunately, depending on one's definition of hardness, most functions will be easier for some than others. In particular, functions which are only *computationally* hard (requiring many time steps) are more susceptible to attacks by specialized hardware. This has become particularly apparent in the realm of cryptocurrency. Some developers have tried to forcibly expel miners using ASICs from their chains using hard forks and software updates but have had little success, as most reports claim that mining on these networks is dominated by specialized hardware. The effect that this has is that the protocol is no longer equitable: those with access to specialized hardware reap most of the rewards, while "regular" users are left to sit on the sidelines.

One solution which has been explored in [AS14, AS15, BCS16, ABP17, ABP18] is to

define a new notion of hardness which resists the advantage of specialized circuits. Their key insight is that to compute a function which requires access to a large amounts of memory over long time periods require specialized chips with much larger size and many more outgoing wires. That is, functions with high "cumulative memory complexity," or "memory-hard functions," diminish the advantage of users with specialized hardware.

These works also establish how one can construct such a memory-hard function based on a hash function h and an underlying DAG G representing the structure of the function. Intuitively, each node v has a hidden label l_v which is computed by evaluating h on the labels of all of v's parent nodes. That is, if $\mathsf{parents}(v) = v_1, \ldots, v_D$, then $l_v = h(l_{v_1}, \ldots, l_{v_D})$. Each node must then have a bounded indegree so that one can choose a hash function h with the proper domain size. The hash function h, the graph G, and the label(s) of the source node(s) are the input to the function. The output is a valid label for every sink node. That is, a set of labels which can be produced via legally evaluating h on various nodes according to the specified rules. By constructing the function in this way, the problem of calculating how much memory must be stored over how much time can be reduced to a cleanly stated problem in the language of graph theory. Alwen and Serbinkenko show that the depth-robustness of the underlying graph G provides a lower bound on the complexity of the associated function. We restate their theorem here as a lemma.

Lemma 6.1 ([ABP17],Theorem 4)**.** *Let G be an (e, d)-depth-robust DAG. Then $\Pi^{||}_{cc}(G) > ed$.*

Because our construction achieves stronger depth-robustness guarantees than previous constructions, it consequently implies a family of memory-hard functions with higher memory-complexity. We will give this proof once we formally define the model and our notion of complexity.

We normally model this process of computing new labels from labels stored in memory as a type of "pebbling" game. Such games have been used in many areas of theoretical computer science, particularly in analyzing time/space tradeoffs as in [Coo74]. In this model, to pebble a node represents computing its label via the labels of its parents. To remove a pebble from a node represents deleting that label from memory. The game of interest for our purposes

is the parallel black pebbling game. We summarize here the description given in [ABP17]. The game is played on a graph $G = (V, E)$ in rounds. The goal is to pebble all sink nodes of G (not necessarily simultaneously). A particular round $i \geq 1$ is fully characterized by the set of currently pebbled nodes in that round. We call this the i^{th} pebbling configuration $P_i \subseteq V$. The initial pebbling configuration is empty, $P_0 = \varnothing$, to represent all labels being unknown. A configuration P_i can be legally derived from the previous configuration P_{i-1} via two rules:

(1) A node v may be pebbled (added to P_i) if all of its parents were pebbled in the previous configuration, i.e. $\mathsf{parents}(v) \subseteq P_{i-1}$.

(2) A pebble can always be removed from P_i.

This pebbling game is called parallel if rule (1) can be applied more than once per round. We are interested in this version because real-life adversaries will attempt to parallelize the work and compute multiple hashes at once. A sequence of configurations $P = (P_0, P_1, \dots)$ is a legal pebbling of G if it adheres to these rules and each sink node of G is contained in at least one configuration P_i.

We are now able to define the cumulative memory complexity of a graph G. As described above, this quantity should represent the memory over time required to compute the "labeling" function f_G associated with G. Let $\mathcal{P}_G^{||}$ denote the set of all valid parallel pebblings of G.

Definition 6.1. The parallel cumulative pebbling complexity of a DAG G is

$$\Pi_{cc}^{||}(G) = \min_{(P_1,\dots,P_t)\in\mathcal{P}_G^{||})} \sum_{i=1}^{t} |P_i|$$

Remembering a pebble for a single round represents a single "unit" of cumulative memory-complexity. In this sense, summing the size of each pebbling configuration over a particular strategy is a great representation of the total memory over time required for this strategy. By taking the minimum over all possible strategies, the notion of parallel cumulative memory complexity precisely matches the total memory over time required by the best possible strategy to compute f_G.

It is worth mentioning here that for any such function to be realizable, the family of graphs from which G is sampled must have indegree bounded by some constant. Otherwise, for large enough G, any hash function h would not have a sufficient domain size for the number of parents in the graph. Since our construction is $(d + c)$ regular for constants d, c, it meets this requirement.

We now state our claim about the cumulative memory complexity of our newly constructed family of graphs $\{G_n\}$. The proof comes directly from applying Lemma 6.1 to the conclusion of Theorem 4.2.

Theorem 6.2. *For any arbitrary small constant $\epsilon > 0$, there exist a family of DAGs $\{G_n\}$ with $\Pi^{||}_{cc}(G_n) = \Omega(N^{2-\epsilon})$, $N = |G_n| = m^n M$.*

Proof of Theorem 6.2. Fix $\epsilon > 0$, and let $\epsilon_0 > 0$ satisfy $\epsilon_0 < \epsilon/2$. By Theorem 4.2, the family $\{G_n\}$ with proper base graphs G, H, X is $(\Omega(N^{1-\epsilon_0}), \Omega(N^{1-\epsilon_0}))$-depth-robust. By Lemma 6.1, this implies $\Pi^{||}_{cc}(G_n) = \Omega(N^{2-2\epsilon_0}) = \Omega(N^{2-\epsilon})$ as desired. □

In particular, if we wish to beat the complexity of Argon2i, we can choose $a = b \approx 0.5$, $m > 2^{60}$, and $D = O(\log^2(m))$. That is, choose a base graph H with more than 2^{60} vertices. We can conclude that

$$\Pi^{||}_{cc}(G_n) = \Omega(N^{2+\log_m(C_a)+\log_m(C_b)}) = \Omega(N^{1.78})$$

which exceeds the upper bound on the complexity of Argon2i. Notice that if we choose a larger m, the complexity of the memory hardness could be arbitrary close to $\Omega(N^2)$. For example, if the base graph H is has more than 2^{200} vertex, $\Pi^{||}_{cc}(G_n) = \Omega(N^{1.9})$.

6.2 Other Applications

Proof of Sequential Work (PoSW): A PoSW is closely related to a proof-of-work (PoW), with an additional caveat. In a PoSW, a prover wishes to prove that he has computed a hash chain of length n in a way that does not require the verifier to recompute the chain. Mahmoody et al.[MMV13] use a depth-robust graph to construct a PoSW. Instead of just a

chain, a depth-robust graph is constructed by adding additional edges to the chain. Then, the prover sends a commitment to some labeling to the verifier via the root of a Merkle tree. Because of the guarantees of depth-robustness, any adversary who outputs a correct labeling of the last node in the chain cannot have avoided performing the necessary sequential work. Alwen et al.[ABP18] argue that the in-degree of underlying graph is crucial to the protocol's efficiency. This is because to open a Merkle tree commitment, one must perform work directly proportional to the indegree. The depth-robust graph in their improved result has indegree $O(\log n)$ while our construction has indegree $O(1)$. Thus by replacing their G_n with ours one can improve efficiency by a factor of $O(\log n)$.

Distributed Storage Network (DSN): In a distributed storage network, data is stored in a peer-to-peer system without central servers. Such a network is intuitively robust if any file F stored on the network has copies stored at many nodes and F is recoverable even if some such copies are corrupted. A common technical challenge for DSN is then proving that data is stored robustly and quantifying sucha notion. Cecchetti et al. [CFM18] constructed a public verifiable DSN in which the server can prove file replication efficiently. Their construction utilizes a new graph called the *Dagwood Sandwich Graph* which combines depth-robust graphs and superconcentrators. At a high level, a *Dagwood Sandwich Graph* is a graph consisting of multiple depth-robust layers in which layers are connected by superconcentrators. Their construction, however, is probabilistic and relies on unproven assumptions about the randomly produced graphs. One can replace such graphs with our new construction from Theorem 4.3 and reduce the failure probability of the produced DSN to zero.

REFERENCES

[ABH17] Joël Alwen, Jeremiah Blocki, and Ben Harsha. "Practical graphs for optimal side-channel resistant memory-hard functions." In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1001–1017. ACM, 2017.

[ABP17] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. "Depth-Robust Graphs and Their Cumulative Memory Complexity." In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pp. 3–32, Cham, 2017. Springer International Publishing.

[ABP18] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. "Sustained Space Complexity." In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018*, pp. 99–130, Cham, 2018. Springer International Publishing.

[AS14] Joël Alwen and Vladimir Serbinenko. "High Parallel Complexity Graphs and Memory-Hard Functions." Cryptology ePrint Archive, Report 2014/238, 2014. `https://eprint.iacr.org/2014/238`.

[AS15] Joël Alwen and Vladimir Serbinenko. "High Parallel Complexity Graphs and Memory-Hard Functions." In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pp. 595–603, New York, NY, USA, 2015. ACM.

[BCS16] Dan Boneh, Henry Corrigan-Gibbs, and Stuart Schechter. "Balloon Hashing: A Memory-Hard Function Providing Provable Protection Against Sequential Attacks." In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pp. 220–248, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BZ17] Jeremiah Blocki and Samson Zhou. "On the depth-robustness and cumulative pebbling cost of Argon2i." In *Theory of Cryptography Conference*, pp. 445–465. Springer, 2017.

[CFM18] Ethan Cecchetti, Ben Fisch, Ian Miers, and Ari Juels. "PIEs: Public Incompressible Encodings for Decentralized Storage." Cryptology ePrint Archive, Report 2018/684, 2018. `https://eprint.iacr.org/2018/684`.

[Coo74] Stephen A. Cook. "An observation on time-storage trade off." *Journal of Computer and System Sciences*, **9**(3):308 – 316, 1974.

[EGS75] Paul Erdoes, Ronald L Graham, and Endre Szemeredi. *On sparse graphs with dense long paths*. Stanford University. Computer Science Department, 1975.

[Kla84] Maria Klawe. "Limitations on explicit constructions of expanding graphs." *SIAM Journal on Computing*, **13**(1):156–166, 1984.

[MMV13] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. "Publicly Verifiable Proofs of Sequential Work." In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pp. 373–388, New York, NY, USA, 2013. ACM.

[Pip87] Nicholas Pippenger. "Sorting and selecting in rounds." *SIAM Journal on Computing*, **16**(6):1032–1038, 1987.

[RVW00] Omer Reingold, Salil P. Vadhan, and Avi Wigderson. "Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors." *Electronic Colloquium on Computational Complexity (ECCC)*, **8**, 2000.