Local Access to Random Walks

Amartya Shankha Biswas
CSAIL, MIT, Cambridge, MA, USA

Edward Pyne
Harvard University, Cambridge, MA, USA

Ronitt Rubinfeld
CSAIL, MIT, Cambridge, MA, USA

Abstract

For a graph G on n vertices, naively sampling the position of a random walk of at time t requires work $\Omega(t)$. We desire *local access* algorithms supporting $\textsc{position}_G(t)$ queries, which return the position of a random walk from some fixed start vertex s at time t, where the joint distribution of returned positions is $1/\operatorname{poly}(n)$ close to those of a uniformly random walk in ℓ_1 distance.

We first give an algorithm for local access to random walks on a given undirected d-regular graph with $\widetilde{O}(\frac{1}{1-\lambda}\sqrt{n})$ runtime per query, where λ is the second-largest eigenvalue of the random walk matrix of the graph in absolute value. Since random d-regular graphs $G(n,d)$ are expanders with high probability, this gives an $\widetilde{O}(\sqrt{n})$ algorithm for a graph drawn from $G(n,d)$ whp, which improves on the naive method for small numbers of queries.

We then prove that no algorithm with subconstant error given probe access to an input d-regular graph can have runtime better than $\Omega(\sqrt{n}/\log(n))$ per query in expectation when the input graph is drawn from $G(n,d)$, obtaining a nearly matching lower bound. We further show an $\Omega(n^{1/4})$ runtime per query lower bound even with an oblivious adversary (i.e. when the query sequence is fixed in advance).

We then show that for families of graphs with additional group theoretic structure, dramatically better results can be achieved. We give local access to walks on small-degree abelian Cayley graphs, including cycles and hypercubes, with runtime $\operatorname{polylog}(n)$ per query. This also allows for efficient local access to walks on polylog degree expanders. We show that our techniques apply to graphs with high degree by extending or results to graphs constructed using the tensor product (giving fast local access to walks on degree n^ϵ graphs for any $\epsilon \in (0,1]$) and Cartesian product.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms

Keywords and phrases sublinear time algorithms, random generation, local computation

Digital Object Identifier 10.4230/LIPIcs.ITCS.2022.24

Funding *Amartya Shankha Biswas*: Big George Ventures Fund, MIT-IBM Watson AI Lab and Research Collaboration Agreement No. W1771646, NSF awards CCF-1733808 and IIS-1741137
Ronitt Rubinfeld: NSF awards CCF-2006664, CCF-1740751, IIS-1741137, Fintech@CSAIL. Part of this work was done while the author was participating in the program on Probability, Geometry, and Computation in High Dimensions at the Simons Institute for the Theory of Computing.

Acknowledgements We thank the anonymous reviewers for comments that improved our presentation, and for comments that led us to strengthen our results. The second author thanks Andrew Lu for valuable comments on a draft of this paper.

1 Introduction

Given some huge random object that an algorithm would like to query, is it always necessary to generate the entire object up front? For sublinear time algorithms, generating such a large object would dominate the runtime. Recent works [4, 12, 18, 7] demonstrate that this is not always necessary, giving incremental query access to random objects such as random graphs, Dyck paths and graph colorings. These *local access* algorithms answer queries in a manner consistent with an instance of the random object sampled from the true distribution (or close to it). A major challenge in developing local access algorithms is to maintain consistency of the joint distribution between query answers.

In this work, we explore the question of implementing local access to random walks. Random walks are a critical primitive in many algorithms [16, 14, 9], including sublinear ones [13, 21, 3]. Given a graph G on n vertices, naively generating a random walk of length t requires time $\Omega(t)$. As the size of real-world graphs, and the length of the walks taken on them, have increased tremendously, breaking this $\Omega(t)$ barrier has become crucial. Thus, algorithms for generating random walks have been considered in both distributed and parallel models: specifically, the works of [22] and [15] have developed algorithms for generating random walks in the CONGEST distributed and Massive Parallel Computation models respectively.

In this work we explore the question of implementing local access to random walks. Since t can be large, one may want to generate only the segments of the walk that are needed at the present time, while ensuring the *joint distribution* of the returned segments is close to the true distribution of random walks.[1]

As is common in the setting of sublinear and local algorithms, we assume the local access algorithm is given access to a graph G on n vertices through probe oracles. This allows us to work with graphs that are too large to fit in main memory, and also results in running times that are not dominated by the size of the input. Our goal is to provide implementations of the queries needed by the user – in particular, we focus on $\textsc{position}_G$ queries, which return the position of a random walk starting from a fixed vertex s at time t, such that, given any sequence of user queries, the joint distribution of returned positions is $1/\operatorname{poly}(n)$-close to the true uniform distribution of those positions over random walks (in ℓ_1 distance). We desire per query runtime that is sublinear in n and t, and preferably polylogarithmic in both. In the latter case, locally generating *all* vertices in a walk of length t (in an arbitrary order) has total work within a polylog factor of the naive sequential runtime.

Our algorithms can be used even when the algorithm wipes its memory after each answer and must answer subsequent queries only retaining access to a (public) random string. This fits within the framework of the LCA models in [20, 2]. This feature allows independent copies of the algorithm that share a random string but do not communicate, yet still give consistent responses. We assume the random string is of polynomial size, which is necessary, as a family of objects with description size t requires $\Omega(t)$ random bits to approximate in ℓ_1 distance. Due to the lack of persistent memory between queries, this is a stronger model than used in [12, 18, 7, 4][2] for locally accessing huge random objects.

Obtaining local access for walks on arbitrary graphs without knowing the entire structure seems to be a very difficult problem, and therefore in this paper we restrict our attention to regular graphs. However, regular graphs include widely studied families such as random regular and Cayley graphs, both of which we analyze.

[1] We note that one can use fast distributed algorithms in the LOCAL distributed model in order to construct fast sequential algorithms via the known Parnas-Ron reduction, but the aforementioned distributed algorithms are not fast enough to be useful. We will elaborate more on this in Section 1.2.

[2] Note that the result in [4] for generating random colorings in degree bounded graphs also works within the model used here.

1.1 Our Results and Techniques

We begin by presenting a $\tilde{\mathcal{O}}(\frac{1}{1-\lambda}\sqrt{n})$ algorithm that provides local access to undirected d-regular graphs with spectral expansion λ. We assume that the algorithm is given access to a tape of random bits $\mathbf{R}$. For a graph G, the local access algorithm divides the random walk into epochs of length t_{mix}, where t_{mix} is on the order of the mixing time of G. At the beginning of each epoch (times that are multiples of t_{mix}), the local access algorithm uses appropriate bits from the random tape to determine the location of the walk, which are distributed according to the uniform distribution. When given a query to time t, the algorithm first finds the locations u, v of the walk at the beginning of epochs t and $t+1$. It then samples $\tilde{\mathcal{O}}(\sqrt{n})$ random walks from both endpoints u, v, each of length $t_{mix}/2$, until we find a forward walk from u and a backward walk from v that terminate at the same vertex. Since walks of this length are well mixed, a collision occurs with high probability. When a collision is found, we can stitch together the forward and the backward walk and thus interpolate between both endpoints. Since we use the same section of random tape for any query in this epoch, we consistently find the same collision, and thus answer consistently despite no persistent memory.

Moving forwards, we demonstrate that such a runtime is optimal in general, even for local access algorithms that are allowed to remember their prior answers and graph probes. Specifically, our lower bound holds for the case of *random* d-regular graphs, which provides some evidence that obtaining fast query algorithms for "large" classes is challenging.[3] Our lower bounds present adaptively chosen query sequences, and demonstrate that for the vast majority of these random graphs, any algorithm making $\tilde{\mathcal{O}}(\sqrt{n}/\log n)$ NEIGHBOR probes to the underlying graph G will fail to answer the queries in a consistent manner. The main structural result used here is Lemma 4.8, which states that as long as the algorithm makes fewer than $\Theta(\sqrt{n})$ probes, the revealed edges and vertices of the graph will form a forest, and additionally, no trees will ever be merged, with probability at least 0.995. This allows us to define a distance metric $d(\cdot,\cdot)$ where $d(u,v)$ is the distance between vertices u and v *using only the edges known to the algorithm*, and is defined to be ∞ if no such path has been revealed. The high level strategy in the lower bound is to first query the positions v_0, v_e of the walk at time $t = 0$ and $t = \sqrt{n}$ respectively, and then adaptively query $\mathcal{O}(\log n)$ intermediate positions (where the query times may depend on the internal state of the algorithm), until an inconsistency is found. The hypothesis at this point is that the algorithm does not actually know of a path of the correct length between the two returned vertices. Specifically, we show that either the revealed edges fail to connect the vertices in the limited number of available probes, or the known path between them is shorter than $\sqrt{n}/20$.

In the first case, we can perform binary search for a location such that we end up with two reported positions which are adjacent in time, but do not have an edge between them whp, thus yielding the inconsistency. The latter case is more complicated, and requires some case analysis, but we are able to query adaptively and always find two positions v_i and v_j (revealed at times t_i and t_j), such that one of the following two outcomes hold: either the distance is too large $d(v_i, v_j) > |t_i - t_j|$ or the distance is too small $d(v_i, v_j) < |t_i - t_j|/8$. In the first outcome, if the distance is greater, we can again perform binary search to find adjacent positions in the walk that are not connected by an edge. For the second outcome, we again perform binary search to find a short segment with unusually short distance in the subgraph

[3] There are simple constructions of artificial classes, such as cycles on a random subset of $n/2$ vertices, where $\Omega(n)$ probes are required to answer a single query.

induced by the edges known to the algorithm, and then query all intermediate locations to find a segment of the walk $\sigma_1, \sigma_2, \cdots, \sigma_l$ of length $\Theta(\log n)$, such that $d(\sigma_1, \sigma_l) < l/8$. Note that we are then able to query all the locations in this segment because its length was reduced to $\mathcal{O}(\log n)$. Intuitively, such a segment is unlikely to occur in a truly random walk because the edges known to the algorithm define a tree, and any walk of length $> 8l$ between vertices of distance l (with respect to the tree metric) in a tree must involve a significant amount of backtracking. We demonstrate that the probability of significant backtracking over a truly random walk on a random regular graph is $o(1)$.

We also prove an oblivious lower bound of $\Omega(n^{1/4})$, for the case when the queries do not depend on the internal state of our algorithm. In this case, we present the sequence of query times $(n^{1/4}, 2, 3, \cdots, n^{1/4} - 1)$. If the algorithm makes $\mathcal{O}(n^{1/4})$ graph probes, then the total number of probes is bounded above by $\Theta(\sqrt{n})$, and therefore we use the same structural Lemma 4.8 mentioned above in order to derive a contradiction.

Finally, motivated by our lower bounds against local access to walks on general classes of graphs, we turn to algorithms for local access on families of graphs with additional algebraic structure. We give fast local access (i.e. runtime polylog(n) per query) to walks on small-degree abelian Cayley graphs (for instance, cycles and hypercubes). This also allows for fast local access to walks on a class of polylog degree expanders. We extend our results to graphs constructed using the tensor product, giving fast local access to walks on degree n^ϵ graphs for any $\epsilon \in (0, 1]$.

1.2 Related Work

The problem of providing local access to huge random objects was first proposed in [12, 11]. Subsequent work in [18] presented algorithms that provide access to sparse Erdos-Renyi $G(n, p)$ graphs through All-Neighbors queries, as long as the number of queries is small and $p = \mathcal{O}(\text{poly}(\log n))$. Many of the results in these earlier works only guarantee that the generated random objects *appear* to look random, as long as the number of queries are bounded, usually by $\mathcal{O}(\text{poly}(\log n)/n)$. More recently, in [7], an implementation of random recursive trees and BA preferential attachment graphs are presented. Further, local access is given for the Next-Neighbor query that returns the neighbors of a vertex in lexicographic order, which is useful for accessing graphs where the degree is not bounded. Subsequently, [4] presented implementations for random $G(n, p)$ graphs for any value of n, while supporting Next-Neighbor as well as the newly introduced Random-Neighbor queries. In [4], algorithms are provided for accessing random walks on the line, random Dyck paths, and random colorings of a graph. Implementing access to random walks on the line graph was motivated by the implementation of interval summable functions in [12, 10].

Related to generation of random walks, [22] give an algorithm that approximately samples from random walks in $\widetilde{O}(\sqrt{tD})$ rounds in the CONGEST model, where t is the length of the walk and D is the diameter of the graph. The standard reduction from distributed algorithms to LCAs from [19] would result in an algorithm with query complexity that is exponential in $\mathcal{O}(\sqrt{tD})$, whereas we obtain bounds of the order $\mathcal{O}(\sqrt{t})$. Later, [15] gave an algorithm for generating walks in $O(\log t)$ rounds in the Massively Parallel Computation (MPC) model. However, their techniques do not seem to be amenable to constructing fast LCAs.

1.3 Organization

In Section 2 we introduce notation and basic sampling tools. In Section 3 give a memoryless local access oracle for undirected regular graphs with runtime in terms of expansion. In Section 4, we first apply the previous oracle to random regular graphs. We then prove a nearly matching lower bound even for local access algorithms with persistent storage, with respect to an adaptive adversary (one who has access to the persistent storage of the algorithm), and a weaker bound with respect to an oblivious adversary. In Section 5 we give memoryless local access oracles for small degree abelian Cayley graphs, such as hypercubes and cycles. In Appendix B, we prove claims in Section 5 and give memoryless local access oracles for the tensor product.

2 Preliminaries

We first define terminology and introduce basic tools for sampling. We characterize the closeness of query responses to true random walks via ℓ_1 distance, and use ℓ_2 distance for spectral arguments.

▶ **Notation 2.1.**

- *Given distributions A, B over a set $[S]$, the ℓ_1 distance between A and B is defined as $||A - B||_1 = \sum_{i=1}^{S} |A_i - B_i|$. The ℓ_2 distance is defined as $||A - B||_2 = \sqrt{\sum_{i=1}^{S}(A_i - B_i)^2}$.*
- *For some set S, let U_S denote the uniform distribution over S. Let $s \leftarrow U_S$ be an element drawn from this distribution.*

Next, we define notation for the distribution of random walks on fixed graphs.

▶ **Notation 2.2.** *Given a regular graph $G = (V, E)$ where $V = [n]$, $v_1, v_2 \in V$ and $t \in \mathbb{N}$:*

- *Let $\lambda(G) = \max_{x \in \mathbb{R}^n : x \perp \bar{1}} ||xW||_2/||x||_2 = \max(|\lambda_2(G)|, |\lambda_n(G)|)$ where W is the random walk matrix of G.*
- *Let $D_C(G, v_1, v_2, t)$ be the distribution over random walks of length t from v_1 that end at v_2. As G is regular, this is the uniform distribution over all satisfying walks.*
- *Let U_G^L be the uniform distribution of random walks from vertex 1 of length L.*
- *For any finite set of times $S \in \mathbb{N}^k$, let the **walk projection** $P_S(U_G)$ be the joint distribution over $V^{|S|}$ of the positions at times S of random walks from vertex 1. We will measure the accuracy of a local access oracle given time queries S by bounding the ℓ_1 distance of (the distribution of) its responses to $P_S(U_G)$. For notational convenience, let $P_i = P_{\{i\}}$.*

Note that $P_T(U_G)$ captures the *joint* distribution of the location of a walk at times T. That is, the position of a walk at time t is correlated with that at time t'.

We then define the generalization of Local Computation Algorithms with which we give our upper bounds. The model is similar to the LCA model in [20, 2], which notably means there is no persistent storage between queries. Thus, for queries to the memoryless local access oracle the user can expect to see the same results whether making queries to the same copy or to independent copies of the oracle sharing the same random string.

▶ **Definition 2.3.** *Given a graph G and a maximum query time L, a **memoryless local access oracle** implementation of a family of query functions $\langle \text{POSITION}_G(1), \ldots, \text{POSITION}_G(L) \rangle$, provides an oracle $\mathcal{A}$ with the following properties. $\mathcal{A}$ has probe access to the input description G, a tape of public random bits $\mathbf{R}$, and the maximum possible query time L. Upon being queried with G, L and t, the oracle uses sub-linear resources to return the value $\mathcal{A}(G, \mathbf{R}, \text{POSITION}_G(t))$, which must equal the position of the walk X at time t for a specific $X \in U_G^L$ where the choice of X depends only on $\mathbf{R}$, and the distribution of X (over $\mathbf{R}$) is $1/n^c$-close to U_G^L in ℓ_1 distance, for any given constant c. Between consecutive queries, $\mathcal{A}$'s memory (but not the public random tape or input) is erased. Thus, different instances of $\mathcal{A}$ with the same graph G and the same random bits $\mathbf{R}$, must agree on the choice of X that is consistent with all answered queries regardless of the order and content of queries that were actually asked.*

We prove our lower bounds against a stronger model, where the algorithm is allowed to remember its prior answers and graph probes. Local computation algorithms with persistent memory have been investigated before [6, 12, 18, 7, 4], and can be thought of as a single algorithm answering queries in series, rather than a swarm of algorithms working independently.

▶ **Definition 2.4.** *Given a graph G and a maximum query time L, a* ***(persistent) local access oracle*** *implementation of a family of query functions $\langle \textsc{position}_G(1), \ldots, \textsc{position}_G(L)\rangle$, provides an oracle $\mathcal{A}$ with the following properties. $\mathcal{A}$ has probe access to the input description G, a tape of public random bits $\mathbf{R}$, and the maximum possible query time L. Upon being queried with G, L and t, the oracle uses sub-linear resources to return the value $\mathcal{A}(G, \mathbf{R}, \textsc{position}_G(t))$, which must equal the position of the walk X at time t for a specific $X \in U_G^L$ where the choice of X depends only on $\mathbf{R}$, and the distribution of X (over $\mathbf{R}$) is $1/n^c$-close to U_G^L in ℓ_1 distance, for any given constant c. Between consecutive queries, $\mathcal{A}$'s memory is* ***not*** *erased, and it is allowed arbitrary persistent local storage.*

We next define our probe model, which we use for our upper and lower bounds.

▶ **Definition 2.5.** *Given an undirected d-regular graph $G = (V, E)$ on n vertices, $\textsc{neighbor}(G, v, i)$ returns the ith neighbor of vertex v for every $(v, i) \in [n] \times [d]$, where the outgoing edges are assigned a random permutation of $[d]$ for every v.*

Here we place a random labeling on the out-edges of each vertex since we are dealing with unlabeled graphs. The random labeling is fixed as part of G and does not change for subsequent probes or instantiations of the algorithm.

Finally, we state a basic result on partial sampling.

▶ **Proposition 2.6.** *Let G be a graph and T an ordered list of determined times in a walk on G. Let V_T be the associated set of determined positions. Suppose V_T has been sampled to within ϵ of the true distribution in ℓ_1 distance. For any new query t, let $t_- < t < t_+$ be the closest low and high previously determined times. These are denoted the* ***bracketing queries****. Then:*

1. *The distribution of v_t conditioned on v_{t_-}, v_{t_+} is equal to the distribution conditioned on all previously determined vertices.*
2. *If v_t is sampled from a distribution D where $||D - P_{t-t_-}(D_C(v_{t_-}, v_{t_+}, t_+ - t_-))||_1 \leq \delta$, then (V_T, v_t) is $\epsilon + \delta$ close to the true distribution. Furthermore, if the true distribution of v_t is some deterministic function of k distributions, an equivalent result holds for sampling each distribution to within δ/k and returning the deterministic function applied to these samples.*

In effect, this gives us the ability to only focus on the distribution of walks conditioned on the nearest determined positions when analyzing the distance of a local access oracle to uniform.

3 Local Access Via Spectral Expansion

We first present an an oracle for undirected regular graphs that uses $\widetilde{O}\left(\frac{1}{1-\lambda}\sqrt{n}\right)$ work per query. This is sublinear for small numbers of queries on graphs with good expansion, but is far from polylog work per query.

▶ **Theorem 3.1.** *Given* NEIGHBOR *probe access to an undirected d-regular graph G on n vertices with $\lambda(G) \leq \lambda$, there is a memoryless local access oracle which uses $O(\sqrt{n}\frac{1}{1-\lambda}\operatorname{polylog}(nL))$ time and working space per query, where L is the maximum query time.*

We first give a procedure for sampling walks conditioned on their start *or* endpoint.

▶ **Lemma 3.2.** *Given a section of random tape $\mathbf{R}_s$ and* NEIGHBOR *probe access to an undirected d-regular graph G on n vertices, there are subroutines* RAND_PATH(G, v, l) *and* RAND_REV_PATH(G, v, l) *that run in time $O(l \log n)$, and return a uniformly random walk of length l starting and ending at v respectively.*

Proof. For RAND_PATH(G, v, l), let $v_1 = v$ and for l iterations let $v_{i+1} =$ NEIGHBOR$(G, v, U_{[d]})$, where we use the segment of tape $\mathbf{R}_s$ to generate the random bits. Then return $(v_1, \ldots, v_l)$. Correctness is direct from the definition of a random walk. For RAND_REV_PATH(G, v, l), let $(v_1, \ldots, v_l) \leftarrow$ RAND_PATH(G, v, l) be a random walk of length l from v, and return $(v_l, v_{l-1}, \ldots, v_1)$. The runtime of this procedure is direct.

To see that RAND_REV_PATH samples from the correct distribution, note that as G is undirected and regular, the probability of taking any fixed walk $v = v_1, v_2, \ldots, v_l$ is equal when taking the walk in either direction. ◀

We can then prove the theorem.

Proof of Theorem 3.1. Recall that we are given probe access to G, a random tape $\mathbf{R}$, and a maximum query time L. Let $c > 0$ be an arbitrary constant, where we achieve n^{-c} approximation. WLOG assume we assign each time t' a disjoint set of random bits $\mathbf{R}_{t'}$. Let $k = O(\frac{1}{1-\lambda} c \log(nL))$ be the smallest integer such that $\lambda(G^k) \leq 1/n^{3+c}L$.

Now suppose we are given a query at time t:

1. If $t \mod 2k = 0$, use $\mathbf{R}_t$ to generate a uniformly random vertex $v \in V$ and return v.
2. Otherwise, let t_- and t_+ be the bracketing multiples of $2k$. Use Case (1) to determine the positions of the walk at these times, which we denote v_{t_-} and v_{t_+}. Furthermore, use the random bits of $\mathbf{R}_{t_-}$ for the following procedure: Let S_L, S_R be empty sets of walks of length k from v_{t_-} and v_{t_+} respectively. Let COL be the event a path from S_L and S_R share an endpoint.
   a. Let $S_L \leftarrow S_L \cup$ RAND_PATH(G, v_{t_-}, k).
   b. Let $S_R \leftarrow S_R \cup$ RAND_REV_PATH(G, v_{t_+}, k).
   c. If COL, go to Phase II.
   d. After $O(\sqrt{n}\log(nL))$ iterations, declare fail (or return an arbitrary path), and otherwise repeat.

   In Phase II we have paths p_l, p_r sharing an endpoint. If there are multiple colliding paths, choose the first to occur. Let the determined path from $[t_-, t_+]$ be $p_l p_r$ and return v_t, the element of this path corresponding to time t.

From our description of the algorithm we have that, fixing the contents of the random tape $\mathbf{R}$, the returned vertex is deterministic and so consistent between independent copies, and moreover the position of the walk at multiples of $2k$ and the colliding walks are consistent between different queried times.

To show the distribution of generated walks (over $\mathbf{R}$) is $1/n^c$ close to U_G^L in ℓ_1 distance, we show that $\mathcal{A}$ determining the position of the walk at every time in $\{1, \ldots, L\}$ produces a walk with the desired properties. Without loss of generality let $\mathcal{A}$ determine the positions at multiples of $2k$ first, sequentially from time $2k$. For a single such position at time t, the position determined by $\mathcal{A}$ is uniform over $[n]$. Let W be the random walk matrix of G. We have from our choice of k that $\|W^k - J\|_2 \leq 1/n^{3+c}L$, and so the distribution of the position at time $l + 2k$ of a random walk conditioned on the position v_l at time l satisfies $\|W^{2k}_{v_l,\cdot} - U_{[n]}\|_1 \leq 1/n^{2+c}L$, so the distribution of $\mathcal{A}$s answer over $\mathbf{R}$ is $1/n^{2+c}L$ close in ℓ_1 distance by Proposition 2.6. Determining all multiples of $2k$ in this way thus results in a set of determined positions that are $1/n^{2+c}$ close to uniform.

Now fix the positions determined by $\mathcal{A}$ at time l and $l+2k$ and consider $D_C(G, v_l, v_{l+2k}, 2k)$, the uniform distribution over random walks of length $2k$ that start at v_l and end at v_{l+2k}. We have that the distribution of the *midpoint* of these walks $P_k(D_C(G, v_l, v_{l+2k}, 2k))$ satisfies $\|P_k(D_C(G, v_l, v_{l+2k}, 2k)) - U_{[n]}\|_1 \leq 1/n^{1+c}L$ by essentially the prior argument. Furthermore, $\mathcal{A}$ determines the position v_{l+k} as the first collision between distributions that are $1/n^{3+c}L$ close to uniform in ℓ_1 distance, so the distribution of positions returned by the oracle is $1/2n^{1+c}L$ close to the true distribution. Determining all midpoints T_m in this way thus results in a distribution $1/n^{1+c}$ close to $P_{T_m}(U^L_G)$. Finally, $\mathcal{A}$ exactly samples from the uniform distribution on walks from v_l to v_{l+k} and v_{l+k} to v_{l+2k}, since the returned paths were sampled via unconstrained random walks, so $\mathcal{A}$ loses nothing in ℓ_1 distance determining the remaining positions and are done. ◀

4 Random Regular Graphs

Next, we study the question of implementing access to *random regular degree graphs*, which have the property that for all $d \geq 3$, the probability a random d-regular graph is an expander tends to 1. This implies that Theorem 3.1 composed with the set of random regular graphs achieves runtime $\widetilde{O}(\sqrt{n})$ per query. In fact, this is nearly the best possible runtime, as we prove no local access oracle (i.e. one that is allowed to remember previous answers) given probe access to random regular graphs making $o(\sqrt{n}/\log(n))$ probes per query achieves achieves subconstant error on *adaptive* query sequences. Furthermore, no local access oracle making $o(n^{1/4})$ probes per query achieves subconstant error on *non-adaptive* (in fact fixed in advance) query sequences.

▶ **Definition 4.1.** *Let $\mathbf{G}(n, d)$ be the uniform distribution over d-regular graphs on n vertices.*

- *For d odd, we implicitly restrict to even n when taking limits.*
- *For a set of edges $S = \{(v_1, w_1), \dots, (v_k, w_k)\}$, let $\mathbf{G}(n, d) \cap S$ be the uniform distribution over d-regular graphs on n vertices containing all edges in S. Note that for certain S (for instance, any containing a self-loop), this set is empty.*

For the remainder of the section we treat d as a constant while n trends to infinity, so O notation sometimes hides factors dependent on d. Furthermore we assume $d \geq 3$, since the other two cases are degenerate. We now state informal versions of the main results. First, we note that composing the oracle of Theorem 3.1 with the set of random regular graphs produces an algorithm with vanishing error.

▶ **Corollary 4.2.** *Given* NEIGHBOR *probe access to a d-regular graph, there exists an oracle $\mathcal{A}$ with time per query $O(\sqrt{n}\,\mathrm{polylog}(nL))$, such that over a $1 - o_n(1)$ fraction of graphs $G \leftarrow \mathbf{G}(n, d)$ then $\mathcal{A}$ is a memoryless local access oracle for G.*

Next, a nearly matching $\widetilde{\Omega}(\sqrt{n})$ lower bound against local access algorithms.

▶ **Theorem 4.3** (Informal Statement of Theorem 4.16). *There is a constant n_0 and an (adaptively chosen) query sequence Q such that any local access oracle $\mathcal{A}$ given* NEIGHBOR *probe access to random d-regular graphs for $n \geq n_0$ with $L = O(\sqrt{n})$ makes $\Omega(\sqrt{n}/\log(n))$ graph probes per time query of Q in expectation.*

Finally, an $\Omega(n^{1/4})$ for a query sequence fixed in advance.

▶ **Theorem 4.4** (Informal Statement of Theorem 4.20)**.** *There is a constant n_0 and a fixed query sequence Q such that any local access oracle oracle $\mathcal{A}$ given* NEIGHBOR *probe access to random d-regular graphs for $n \geq n_0$ with $L = O(\sqrt{n})$ makes $\Omega(n^{1/4})$ graph probes per time query of Q in expectation.*

It is impossible to prove lower bounds for *all* subfamilies in $\mathbf{G}(n, d)$ (in fact we give very fast local access algorithms for some later), but any possible algorithm being $\Omega(1)$ from uniform on at least 99% of random regular graphs effectively rules out a unified approach.

We begin by proving the $\widetilde{O}(\sqrt{n})$ upper bound using the algorithm from Section 3. To do so, we recall the famous result that almost all random regular graphs are good expanders.

▶ **Lemma 4.5** ([8])**.** *For all $d \geq 3$, $\Pr(\lambda(\mathbf{G}(n, d)) \leq .95) = 1 - o_n(1)$.*

Proof of Corollary 4.2. We compose the algorithm of Theorem 3.1 with $G \leftarrow \mathbf{G}(n, d)$, where we promise to $\mathcal{A}$ that $\lambda \leq .95$. In the case of poorly expanding graphs this will result in walks that are arbitrarily far from truly random (or the algorithm declaring fail an arbitrarily high fraction of the time), but the runtime per query will still be as claimed. Then for G such that $\lambda(G) \leq .95$ we have that $\mathcal{A}$ is an memoryless local access oracle given NEIGHBOR access to G, and this occurs with probability $1 - o_n(1)$ by Lemma 4.5 so we are done. ◀

4.1 Structure of Random Regular Graphs

To prove the lower bounds, we first give three structural results which establish any algorithm must succeed even when the first $\Omega(\sqrt{n})$ graph probes define disjoint forests, and give tests for closeness of walks to the uniform distribution supported on only a few queries.

Our first goal is to show no algorithm making NEIGHBOR probes to $G \leftarrow \mathbf{G}(n, d)$ can efficiently find cycles. This is essential, as the entire lower bound rests on the probes made by the algorithm defining a tree with $\Omega(1)$ probability. To do so, we first show conditioning on a small number of edges (e.g. those already known by the algorithm) does not increase the conditional probabilities of non-revealed edges by more than a constant factor.

▶ **Lemma 4.6.** *For all $d \in \mathbb{N}$ there is a constant c_d depending only on d such that for an arbitrary set of edges S with $|S| \leq \sqrt{n}$ and $v, w \in V$ arbitrary vertices where $(v, w) \notin S$, we have $\Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}[(v, w) \in G] \leq c_d/n$.*

We defer the proof to Appendix A. We use the configuration model of Bollobas [5] and a strengthening to handle degree sequences with small amounts of variation by [17].

Furthermore, probe access to $\mathbf{G}(n, d)$ is equivalent to successively generating edges uniformly at random over the set of regular graphs satisfying the existing constraint - in effect, we can only determine edges when required, and this is the perspective we will use for the proof.

▶ **Lemma 4.7.** *Let $\mathcal{A}$ be an algorithm having made k arbitrary* NEIGHBOR *probes to $\mathbf{G}(n, d)$ and let the returned edges be E. Then the conditional distribution over graphs given the probe responses is uniform over $\mathbf{G}(n, d) \cap E$.*

Proof. Let $v_1, \ldots, v_k$ be the origin vertices of the NEIGHBOR probes, $r_1, \ldots, r_k$ the probe indices and $w_1, \ldots, w_k$ the returned vertices. Fixing the set $\{r_{i_1}, \ldots, r_{i_l}\}$ of (WLOG distinct) probes from v_1, for any $H \in G(n, d) \cap E$ we have

$$\Pr[\forall_j \text{ NEIGHBOR}(H, v, r_{i_j}) = w_{i_j}] = \prod_{i=1}^{l} \frac{1}{d - i}$$

and for $H \notin G(n,d) \cap E$ the equivalent probability is zero. Then considering all distinct origin vertices, we obtain that the probability of the given probe responses is equal for any element of $G(n,d) \cap E$. ◀

Given these lemmas, we can now show the first $\Omega(\sqrt{n})$ probes made by any local access oracle will fail to find cycles or merge forests with constant probability.

▶ **Lemma 4.8.** *Let $\mathcal{A}$ be an algorithm, where at each step $\mathcal{A}$ makes a* NEIGHBOR *probe to $\mathbf{G}(n,d)$ or marks any vertex. Each vertex touched by a probe is marked. Then there is a constant k_d depending only on d such that for any $\mathcal{A}$ with at most $\sqrt{n}/k_d$ steps, with probability at least $.995$,*

- *the* NEIGHBOR *probes will define a forest,*
- *no* NEIGHBOR *probe will ever merge two marked trees.*

Proof. Let $V_{<i}$ be the set of vertices that are marked after probe $i-1$, and $E_{<i}$ the known edges. Let v_i be the vertex queried at probe i and r_i the edge index (WLOG assume $\mathcal{A}$ makes NEIGHBOR queries at every timestep). We have $|V_{<i}| \leq 2|E_{<i}| \leq 2i$. Define $k_d = 20\sqrt{c_d}$ where c_d is as in Lemma 4.6 and let $q = \sqrt{n}/k_d$. We obtain

$$\begin{aligned}
\Pr(\text{fail}) &\leq \sum_{i=1}^{q} \Pr_{\mathbf{G}(n,d)\cap E_{<i}}(\textsc{neighbor}(v_i, r_i) \in V_{<i}) && \text{(Lemma 4.7)}\\
&\leq \sum_{i=1}^{q} |V_{<i}| \frac{c_d}{n} && \text{(Lemma 4.6)}\\
&= \frac{2c_d}{n}\frac{q(q+1)}{2}\\
&\leq 1/200.
\end{aligned}$$

◀

▶ **Corollary 4.9.** *For any $\ell \leq \sqrt{n}/\log(n)$, we have that $\Pr_{G\leftarrow\mathbf{G}(n,d)} \Pr_{\sigma\leftarrow U_G^\ell}(\sigma \text{ defines a tree}) \geq 1 - O(1/\log^2(n))$.*

Proof. This directly follows from setting $q = \sqrt{n}/\log(n)$ in the above proof, as a random walk is simply a sequence where at each step we probe $\textsc{neighbor}(v, U_{[d]})$ at the current head. ◀

We now show random walks of length $\sqrt{n}/\log(n)$ over random regular graphs exhibit a distinguishing feature that can be checked on small segments. Intuitively, with high probability there will be no segment of length $r = \Omega(\log(n))$ where the simple path over the edges *traversed in the walk* between the endpoints of the segment is shorter than $r/8$. Since the edges traversed by the walk will define a tree with high probability, an unusually short induced simple path implies the biased random walk corresponding to the tree metric in that segment is much shorter than its expectation, which is vanishingly unlikely. To show this, we formally define the **path length** of the induced simple path.

▶ **Definition 4.10.** *For a partially determined vertex sequence $s = (s_1, \ldots, s_\ell) \in ([n], *)^\ell$, let path length $\mathrm{PL}(s)$ be the distance between s_1 and s_ℓ in the induced (undirected, unweighted) graph $G' = ([n], E')$, where $(u,v) \in E'$ if and only if there exists i such that $s_i = u, s_{i+1} = v$.*

We obtain that an unusually short simple path is vanishingly unlikely in any segment of a random walk.

▶ **Lemma 4.11.** *Let $\sigma \in [n]^\ell$ be a walk of length $\ell \leq \sqrt{n}/\log(n)$. Let $F(\sigma)$ be the event any segment $s = (\sigma_i, \dots, \sigma_j)$ of length $|s| \geq 40\log(n)$ has $\mathrm{PL}(s) < |s|/8$. Then*

$$\Pr_{G \leftarrow \mathbf{G}(n,d)} \Pr_{\sigma \leftarrow U_G^\ell} [F(\sigma)] = o_n(1).$$

Proof. Let $\Psi(\sigma)$ be the event σ defines a tree. Then $\Pr_{G \leftarrow \mathbf{G}(n,d)} \Pr_{\sigma \leftarrow U_G^\ell}(\Psi(\sigma)) \geq 1 - O(1/\log^2(n))$ by Corollary 4.9.

We now fix G and sequentially generate a random walk σ. For each vertex in the random walk, there is some edge that was the first traversed by σ (where we pick some edge for the first vertex arbitrarily). For each step of σ, at the current vertex v, label this first traversed edge a $-$ edge and all others $+$ edges. Then in a random walk in *any* d-regular graph the probability of step i being a $+$ step is exactly $(d-1)/d$ and these events are independent for all i. Furthermore, for all σ that define a tree the $+$ and $-$ labels exactly correspond to the distance metric on the tree induced by the random walk, with $-$ corresponding to backtracking towards the initial vertex.

Now let s be any segment of length at least $40\log(n)$. Let $F(s)$ be the event $F(\sigma)$ holds in this segment. Let $h(s)$ be the sum over $+$ and $-$ steps in s. Then by the definition of simple path and the correspondence between step labels and the tree metric:

$$\{h(s) \geq s/2\} \cap \Psi(\sigma) \implies \overline{F(s)}.$$

But then we can apply a basic Chernoff bound[4] to obtain $\Pr[h(s) < (1-\delta)\mu] \leq \exp(-\delta^2\mu/2)$. Choosing $\delta = 1/2$ and using that $\mu = \mathbb{E}[h(s)] \geq s/3$ we obtain $\Pr[h(s) < s/8] \leq \exp(-(s/3)/8) \leq n^{-1.2}$. Then taking a union bound over the at most ℓ^2 such segments:

$$\begin{aligned}\Pr(F(\sigma)) &\leq \Pr(\overline{\Psi(\sigma)}) + \sum_{s \subseteq \sigma : |s| \geq 40\log(n)} \Pr(\{h(s) < s/8\}) \\ &\leq O(1/\log^2(n)) + \ell^2 \cdot n^{-1.2} \\ &= o_n(1).\end{aligned}$$ ◀

4.2 Proof of Adaptive Lower Bound

We are now prepared to prove the lower bound. For the remainder of the section let $\mathcal{A}$ be a local access oracle with NEIGHBOR probe access to $\mathbf{G}(n,d)$.

We give a sequence of at most $c\log(n)$ time queries. By Lemma 4.8, any algorithm that makes fewer than $\sqrt{n}/k_d c\log(n)$ probes per query sees non-merging trees with probability .995 for the duration of the query sequence. Given this occurs, we force the algorithm to return a walk segment that appears with probability $o(1)$ over the true distribution of random walks on G.

We now begin to work with fixed instantiations of $\mathcal{A}$. We use the perspective of $\mathcal{A}$ successively determining the graph by making new NEIGHBOR probes.

▶ **Definition 4.12.** *For a fixed instantiation of $\mathcal{A}$ on $\mathbf{G}(n,d)$, let $T(Q) = (V_Q, S)$ be the **transcript** of the history of $\mathcal{A}$ after a sequence of queries Q. V_Q holds the vertices returned at the times in Q, and S holds the set of edges revealed by NEIGHBOR probes. Note the distribution over possible graphs at this time is $\mathbf{G}(n,d) \cap S$.*

[4] Let $X_1, \dots, X_n$ be independent random variables taking values in $\{0,1\}$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$. Then for any $\delta \in [0,1]$, $\Pr[X \leq (1-\delta)\mu] \leq \exp(-\delta^2\mu/2)$.

An adaptive query sequence is simply a function $f : T(Q) \to \mathbb{N}$, where the next query is a (in our case deterministic) function of the existing transcript. A non-adaptive query sequence is a function $g : V_Q \to \mathbb{N}$, where the next query can only depend on the vertices returned by $\mathcal{A}$ (the non-adaptive sequence used in our lower bound does not even depend on this), but not on the internal state of the algorithm.

▶ **Notation 4.13.**

- *Given a queried time t, denote by $v_t \in V_Q$ the vertex returned by $\mathcal{A}$ for this time.*
- *Given a transcript $T(Q) = (V_Q, S)$, for vertices $v, w \in V$, let $d(v, w)$ be the length of the simple path between the vertices v, w in the graph induced by the edges in S, where $d(v, w) = \infty$ if no path exists. Denote the simple path itself (if one exists) as $SP(v, w)$.*

In the case where probes define non-merging trees, for all $v, w \in V$ once $\mathrm{d}(v, w) < \infty$ it is fixed for the duration of the query sequence, and there are never multiple simple paths between vertices. This is a central component of the proof, as it implies $\mathcal{A}$ cannot "extend" paths without guessing.

We first give a family of distinguishing functions that we will use to lower bound ℓ_1 distance, and show that truly random walks satisfy them with vanishing probability. The function F_G checks two conditions - if the "walk" traversed edges that do not actually exist, and if the path length of a sufficiently large segment of the walk is too short.

▶ **Definition 4.14.** *For an arbitrary graph $G = (V, E)$ let $F_G : \{V, *\}^e \to \{0, 1\}$ be defined as*

$$F_G(w_0, \dots, w_e) = \mathbb{I} \begin{cases} \exists i \text{ st. } w_i \neq *, w_{i+1} \neq * \text{ and } (w_i, w_{i+1}) \notin E & OR \\ \exists i < j - 40\log(n) \text{ st. } \mathrm{PL}(w_i, \dots, w_j) < (j - i)/8. \end{cases}$$

Furthermore PL *is nonincreasing (and thus F is nondecreasing) with respect to revealing new vertices.*

Interestingly, the only reason we require knowing G to define F_G is to rule out edges that are not actually in the graph.

▶ Remark 4.15. For $\ell \leq \sqrt{n}/\log(n)$ we have $\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} F_G(U_G^\ell) = o_n(1)$ as a simple consequence of Lemma 4.11. Furthermore, as F_G is nondecreasing with regard to additional queries, for any set of timesteps $W \subseteq [\ell]$ and associated projection P_W we obtain $\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} F_G(P_W(U_G^\ell)) = o_n(1)$

For our first lower bound, as we chose the next query time based on the transcript of $\mathcal{A}$ after the previous query, we obtain that, for all local access oracles, there *exists* a sequence of bad queries. Note that every memoryless local access oracle must succeed asymptotically almost surely even on such adaptive sequences.

▶ **Theorem 4.16.** *There exist constants q_d, n_0 depending only on d, a family of distinguishing functions $\{F_G : G \in G(n, d)\}$, and an (adaptively determined) query sequence Q of at most $O(\log(n))$ queries such that any local access oracle $\mathcal{A}$, given* NEIGHBOR *probe access to $\mathbf{G}(n, d)$ that makes fewer than $\sqrt{n}/q_d \log(n)$ probes per query satisfies for all $n \geq n_0$:*

$$\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} |F_G(P_Q(U_G)) - F_G(D_{G,\mathcal{A},Q})| \geq .99$$

where $D_{G,\mathcal{A},Q}$ is the distribution of $\mathcal{A}$'s responses given probe access to G over sequence Q.

Proof. Let $q_d = k_d \cdot 1000$ where k_d is from Lemma 4.8. Our procedure generates a sequence of at most $1000 \log(n)$ queries, so by assumption $\mathcal{A}$ makes at most $\sqrt{n}/k_d$ probes. Thus by the lemma there is n_1 such that for $n > n_1$ with probability .995 the algorithm never finds cycles or merges trees. Note that we treat returned vertices as marked. Denote this event by Ξ, and for the remainder of the proof we assume it holds (and otherwise we can terminate the sequence).

Our first query is at time $e = \sqrt{n}/\log(n)$ (and there is an implicit query at time 0). We claim either $\mathrm{d}(v_0, v_e) = \infty$ or $\mathrm{d}(v_0, v_e) < e/20$. Otherwise we would have

$$\infty > \mathrm{d}(v_0, v_e) \geq e/20 = \sqrt{n}/20\log(n),$$

so the algorithm made at least $\sqrt{n}/20\log(n)$ probes at the first query, violating our assumption on probe complexity.

If $\mathrm{d}(v_0, v_e) < e/20$, we apply Lemma 4.18. Thus we can extend the query sequence $Q \leftarrow (Q, Q')$ by at most $321\log(n)$ queries such that any returned transcript $T(Q)$ either satisfies $F_G(V_Q) = 1$ for all G (in which case we are done) or contains $v_t, v_{t'} \in V_Q$ such that $d(v_t, v_{t'}) > |t' - t|$.

Now we have $v_t, v_{t'} \in V_Q$ such that $d(v_t, v_{t'}) > |t' - t|$, so we apply Lemma 4.17. Thus we can extend the query sequence $Q \leftarrow (Q, Q')$ by at most $\log(n)$ queries such that any returned transcript $T(Q)$ contains $v_t, v_{t+1} \in V_Q$ such that $d(v_t, v_{t+1}) > 1$. Then let S be the edges in the transcript at the termination of the query sequence. We have $|S| \leq \sqrt{n}$ and so by Lemma 4.6, and the definition of F_G,

$$\Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}[F_G(V_Q) = 1] \geq \Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}[(v_t, v_{t+1}) \notin G] = 1 - o_n(1).$$

Then taking n_2 such that this term is at least .999, for $n > \max(n_1, n_2)$ we obtain $\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} F_G(D_{G,\mathcal{A},Q}) \geq .994$. Then by Remark 4.15 there exists n_3 such that for any projection P_Q, for all $n > n_3$

$$\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} F_G(P_Q(U_G^\ell)) \leq \mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} F_G(U_G^\ell) < .004,$$

and by taking $n_0 = \max(n_1, n_2, n_3)$ the result follows. ◀

To complete the proof, we must give short query sequences that when Ξ holds drive almost all distinguishing functions to 1. We first show an algorithm that does not know of a short enough path between returned vertices can be forced to return consecutive vertices *in the walk* that it does not know a connecting edge between.

▶ **Lemma 4.17** (No Viable Path Known). *Assuming Ξ holds, given a transcript $T(Q)$ suppose there are prior queries $v_x, v_y \in V_Q$ such that $d(v_x, v_y) > |y - x|$. Then there exists an adaptive extension of the sequence $Q' \leftarrow (Q, q)$ of at most $\log(n)$ queries such that for any returned transcript $T(Q')$ there are $v_t, v_{t+1} \in V_{Q'}$ such that $d(v_t, v_{t+1}) > 1$.*

Proof. We show this by binary searching on the "gap". WLOG assume $x < y$. At each step:

1. Query at time $m = \lfloor (x + y)/2 \rfloor$.
2. We have $\mathrm{d}(v_x, v_m) + \mathrm{d}(v_m, v_y) \geq \mathrm{d}(v_x, v_y)$ so by non-negativity either $\mathrm{d}(v_x, v_m) > m - x$ or $\mathrm{d}(v_m, v_y) > y - m$.
3. If the first holds, let $y \leftarrow m$ and recurse. Otherwise let $x \leftarrow m$ and recurse.

Since $y - x < \sqrt{n}$ at the start of the recursion after $\log(n)$ queries we drive $|x - y|$ to 1, and so obtain $v_t, v_{t+1} \in V_Q$ such that $\mathrm{d}(v_t, v_{t+1}) > 1$ as desired. ◀

We next show algorithms cannot "fake" the existence of longer paths. The key idea is that modifying $\mathrm{SP}(v_0, v_e)$ (or finding a second simple path) after returning v_e is impossible when the algorithm fails to find cycles. We force $\mathcal{A}$ to return vertices that either trigger Lemma 4.17 or feature excessive backtracking, which drives the distinguishing function to 1.

▶ **Lemma 4.18** (Known Path Too Short). *Assuming Ξ holds, given a transcript $T(Q)$ with $v_e \in V_Q$ suppose $d(v_0, v_e) < e/20$. Then there exists an adaptive extension of the sequence $Q' \leftarrow (Q, q)$ of at most $321 \log(n)$ queries such that any returned transcript $T(Q')$ either contains $v_t, v_{t'} \in V_{Q'}$ where $d(v_t, v_{t'}) > |t - t'|$ or satisfies $F_G(V_{Q'}) = 1$ for all G.*

Proof. For the remainder of the analysis we implicitly assume that for all queries t, t', $\mathrm{d}(v_t, v_{t'}) \leq |t - t'|$ since otherwise the transcript satisfies the first condition and we are done. We give a recursive construction of q that "pushes down" the short path. Let $x \leftarrow 0, y \leftarrow e$.

At each step we maintain the invariants that $\mathrm{d}(v_x, v_y) < (y - x)/10 + 20 \log(n) + 2$ and $320 \log(n) \leq y - x$, which are initially satisfied by the lemma statement.

1. Query $\mathcal{A}$ at time $m = \lfloor (x + y)/2 \rfloor$.
2. Let $r_m = \min_{v \in V} \{d(v_m, v) : v \in \mathrm{SP}(v_x, v_y)\}$ be the length of the simple path from v_m to the simple path from v_x to v_y.
3. If $r_m \geq 20 \log(n)$, we apply Lemma 4.19 with (x, y, m) which uses at most $3 \log(n)$ additional queries and achieves the condition.
4. If $r_m < 20 \log(n)$, we can bound the path length from some endpoint to v_m. Either $\mathrm{d}(v_x, v_m) \leq \mathrm{d}(v_x, v_y)/2 + r_m$ or $\mathrm{d}(v_m, v_y) \leq \mathrm{d}(v_x, v_y)/2 + r_m$. In the first case,

$$\begin{aligned} \mathrm{d}(v_x, v_m) &\leq \mathrm{d}(v_x, v_y)/2 + r_m \\ &< ((y - x)/10 + 20 \log(n) + 2)/2 + 20 \log(n) \\ &\leq (m - x)/10 + 20 \log(n) + 2 \end{aligned}$$

   so letting $y \leftarrow m$ the requirements of the recursion are satisfied. In the other case we set $x \leftarrow m$ and achieve the same.
5. Then if $y - x < 320 \log(n)$, we have $d(v_x, v_y) < (y - x)/20 + 20 \log(n) + 2 < (y - x)/8$. In this case, we query $\mathcal{A}$ at times $\{x + 1, x + 2, \ldots, y - 2, y - 1\}$. Then any set of vertices $\{v_x, \ldots, v_y\} \subset V_{Q'}$ where $d(v_t, v_{t+1}) \leq 1$ for all t lies entirely inside S, and thus must contain $\mathrm{SP}(v_x, \ldots, v_y)$. Therefore we have a walk segment of length at least $40 \log(n)$ where $\mathrm{PL}(v_x, \ldots, v_y) = d(v_x, v_y) < (y - x)/8$ and thus $F_G(V_{Q'}) = 1$ for all G by the definition of F_G as desired.

Then the total number of queries is bounded above by $(1 + 320) \log(n)$ by inspection and Lemma 4.19, so we conclude. ◀

▶ **Lemma 4.19.** *Assuming Ξ holds, given a transcript $T(Q)$ suppose there are $v_x, v_m, v_y \in V_Q$ where $\min_{v \in V} \{d(v_m, v) : v \in SP(v_x, v_y)\} \geq 20 \log(n)$. Then there exists an adaptive extension of the sequence $Q' \leftarrow (Q, q)$ of at most $2 \log(n)$ queries such that any returned transcript $T(Q')$ either contains $v_t, v_{t'} \in V_{Q'}$ where $d(v_t, v_{t'}) > |t - t'|$ or satisfies $F_G(V_{Q'}) = 1$ for all G.*

Proof. As before, we assume that for all queries t, t', the returned vertices $v_t, v_{t'}$ satisfy $\mathrm{d}(v_t, v_{t'}) \leq |t - t'|$ since otherwise the transcript satisfies the first condition and we are done.

We have x, m, y with a tree structure where the distance from v_m to the simple path from v_x to v_y is at least $r_t \geq 20 \log(n)$. Let

$$w = \arg\min_{v \in V} \{d(v_m, v) : v \in \mathrm{SP}(v_x, v_y)\}$$

be the vertex (which has not necessarily been returned) at the point where the simple path to v_m branches from $\mathrm{SP}(v_x, v_y)$. With at most $\log(n)$ queries, we force $\mathcal{A}$ to output that the random walk visits w at times $t_1 \leq m - 20 \log(n)$ and $t_2 \geq m + 20 \log(n)$.

To do so, we apply the following recursion. Let $a \leq b$ be times and u a vertex where $u \in \mathrm{SP}(v_a, v_b)$.

- Query $\mathcal{A}$ at time $t = \lfloor (a+b)/2 \rfloor$. If $v_t = u$, halt.
- We have $\mathrm{d}(v_a, v_t) \leq t - a$ and $\mathrm{d}(v_t, v_b) \leq b - t$ by assumption.
- Either $u \in \mathrm{SP}(v_a, v_t)$ or $u \in \mathrm{SP}(v_t, v_b)$. If the first let $b \leftarrow t$ and otherwise $a \leftarrow t$.

After $\log(n)$ queries we drive $b-a$ to 1. By assumption $d(v_a, v_b) \leq b-a = 1$ and $u \in \mathrm{SP}(v_a, v_b)$, so $\mathcal{A}$ must have returned u at some timestep.

We use this subrecursion twice, with $(a,b,u) = (x,m,w)$ for the first call and $(a,b,u) = (m,y,w)$ for the second. Let t_1, t_2 be the times obtained from these applications where $v_{t_1} = v_{t_2} = w$. We claim $t_1 \leq m - 20\log(n)$ and $t_2 \geq m + 20\log(n)$. If $t_1 < m - 20\log(n)$, we have $d(v_{t_1}, v_m) = d(w, v_m) \geq 20\log(n)$ and thus $|t_1 - m| < d(v_{t_1}, v_m)$, violating our first assumption (and the other case is identical). But then if this does not occur, we have a segment $\{v_{t_1}, \ldots, v_{t_2}\} \subset V_{Q'}$ of length at least $40\log(n)$ where $v_{t_1} = v_{t_2} = w$, so $\mathrm{PL}(v_{t_1}, \ldots, v_{t_2}) = 0 < 40\log(n)/8$ which implies $F_G(V_{Q'}) = 1$ for all G as desired. ◀

This concludes the proof of our adaptive lower bound.

4.3 Proof of Non-Adaptive Lower Bound

The proof of Theorem 4.16 relies on looking at the edges known to $\mathcal{A}$ to choose the next query to the oracle. While any local access oracle is required to succeed for *every* query sequence, so this construction is still valid, we furthermore wish to rule out oracles that succeed only non-adaptive sequences. We give a weaker $\Omega(n^{1/4})$ lower bound that uses a global query sequence (not even depending on the returned vertices) that still suffices to rule out fast local access by an exponential margin.

▶ **Theorem 4.20.** *There exist constants k_d, n_0 depending only on d, a family of distinguishing functions $\{F_G : G \in G(n,d)\}$, and a fixed query sequence Q of $n^{1/4}$ queries such that any local access oracle $\mathcal{A}$ given* NEIGHBOR *probe access that makes fewer than $n^{1/4}/k_d$ probes per query satisfies for all $n \geq n_0$:*

$$\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} |F_G(P_Q(U_G)) - F_G(D_{G,\mathcal{A},Q})| \geq .99$$

where $D_{G,\mathcal{A},Q}$ is the distribution of $\mathcal{A}$'s responses given probe access to G over sequence Q.

Proof. Take k_d as in Lemma 4.8, and define the query sequence as $Q = (n^{1/4}, 2, 3, \ldots, n^{1/4} - 1)$. For convenience, define $e = n^{1/4}$. The distinguishing function is identical to before, so by Remark 4.15 there is n_1 such that for $n \geq n_1$ we have $\mathbb{E}_{G \leftarrow \mathbf{G}(n,d)} F_G(P_Q(U_G)) < .004$.

As $|Q| = n^{1/4}$, the number of probes made by $\mathcal{A}$ is bounded by $\sqrt{n}/k_d$, and so by Lemma 4.8 there is n_2 such that for all $n > n_2$, with probability .995 the algorithm never finds cycles or merges trees. Note that we treat returned vertices as marked. Denote this event by Ξ.

Given Ξ holds, we claim that at the completion of the query sequence either $d(v_0, v_e) = \infty$ or $d(v_0, v_e) < n^{1/4}/2$. If this was not the case, since $\mathcal{A}$ cannot alter $d(v_0, v_e)$ after the first query without finding cycles, $\mathcal{A}$ made at least $n^{1/4}/2 > n^{1/4}/k_d$ probes after the first query which violates our assumption on probe complexity.

Then let the transcript at the end of the sequence be $T(Q) = (V_Q, S)$, recalling S is the edges revealed via probes.

1. If there exist $v_t, v_{t+1} \in V_Q$ such that $\mathrm{d}(v_t, v_{t+1}) > 1$, we have $|S| \leq \sqrt{n}$ and so by Lemma 4.6 and the definition of F_G,

$$\Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}(F_G(V_Q) = 1) \geq \Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}[(v_t, v_{t+1}) \notin G] = 1 - o_n(1).$$

2. If this never occurred, the segment $\{v_0, \ldots, v_e\} \subseteq V_Q$ traverses only edges in S, so it must contain all edges in $\mathrm{SP}(v_0, v_e)$. Therefore $\mathrm{PL}(v_0, \ldots, v_e) = \mathrm{d}(v_0, v_e) < n^{1/4}/8$ and $F_G(V_Q) = 1$ for all G.

Then taking $n_0 = \max(n_1, n_2, n_3)$ where n_3 is chosen such that the $1 - o_n(1)$ term is above .999, the result follows. ◀

5 Efficient Local Access for Abelian Cayley Graphs

We now turn to classes of graphs with algebraic structure. We achieve fast (i.e. runtime polylogarithmic in n) memoryless local access oracles for random walks on the hypercube, n-cycle, and classes of spectral expanders. In Appendix B, we achieve fast memoryless local access oracles for arbitrarily dense graphs via the tensor product. In both cases, we assume *a priori* knowledge of the structure of the graph, rather than accessing it through a neighborhood oracle. This is comparable to the work of [4, 12], which (among many other results) give local access oracles (with persistent storage) for random walks with fixed start and end vertices on the line segment graph.

▶ **Definition 5.1.** *For a group Γ of order n and $S \subseteq \Gamma$, the (directed) Cayley graph $G = \mathrm{Cay}(\Gamma, S)$ is the degree $|S|$ graph on n vertices where for all $g \in \Gamma, e \in S$ we add the edge (g, ge) with label e. We call S the **generators** of G. We say the Cayley graph is **abelian** if the subgroup generated by S is.*

More concretely, a Cayley graph is abelian if for all $e_i, e_j \in S$ we have $e_i e_j = e_j e_i$. We do not require S to be closed under inverses, and so must handle directed graphs.

▶ **Theorem 5.2.** *Let $G = \mathrm{Cay}(\Gamma, S)$ be an abelian Cayley graph on n elements with $d = |S|$, where for all $g \in G$, g^2 is computable in $\mathrm{polylog}(n)$ time. There is a memoryless local access oracle for random walks on G which uses $O(d \cdot \mathrm{polylog}(nL))$ time and working space per query, where L is the maximum query time.*

We defer the proof to Appendix B. In the parallel model, [23] gives an algorithm for efficient generation of random walks on all Cayley graphs. For a walk of length t, they sample $\sigma \in S^t$ and compute the t prefixes $\{s_i\}_{i \in [t]} = \{\prod_{j=1}^{i} \sigma_j\}_{i \in [t]}$ in parallel. Unfortunately, even computing a single prefix of a product of generators in sequential sublinear time is not obviously possible without further restrictions.

Although abelianness represents a strong algebraic assumption, Theorem 5.2 immediately provides memoryless local access oracles for several graph families of interest in computer science.

▶ **Corollary 5.3.** *There is a memoryless local access oracle with per query runtime $O(\mathrm{polylog}(nL))$ for random walks on the following classes of graphs:*

1. *By considering $\Gamma = (\mathbb{Z}/2\mathbb{Z})^d$ and taking $S = (e_1, \ldots, e_d)$ as the generating set, there is a memoryless local access oracle for random walks on the dimension d hypercube for all d.*
2. *By considering $\Gamma = \mathbb{Z}/n\mathbb{Z}$ and taking $S = (1, -1)$ as the generating set, there is a memoryless local access oracle for random walks on the n-cycle for all n.*

3. *By Proposition 5 of [1], for all $m \in \mathbb{N}$ there is an explicitly constructible set S_m where $|S_m| = O(m)$ such that* $\mathrm{Cay}(\mathbb{Z}_2^m, S_m)$ *has spectral gap* $1/3$*. Thus there is a memoryless local access oracle for random walks on a class of* polylog *degree expanders of size* 2^m *for all m.*

We remark that despite all constant-degree abelian Cayley graphs being poor expanders, efficient local access is easy to provide, while for well-expanding random-regular graphs we obtain a polynomial lower bound. This indicates fast mixing time is not a determinative property for efficient local access.

References

1 Noga Alon and Yuval Roichman. Random Cayley graphs and expanders. *Random Struct. Algorithms*, 5(2):271–285, 1994. `doi:10.1002/rsa.3240050203`.

2 Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139. SIAM, 2012. `doi:10.1137/1.9781611973099.89`.

3 Alexandr Andoni, Robert Krauthgamer, and Yosef Pogrow. On solving linear systems in sublinear time. In Avrim Blum, editor, *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, volume 124 of *LIPIcs*, pages 3:1–3:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ITCS.2019.3`.

4 Amartya Shankha Biswas, Ronitt Rubinfeld, and Anak Yodpinyanee. Local access to huge random objects through partial sampling. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

5 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1(4):311–316, 1980.

6 Artur Czumaj, Yishay Mansour, and Shai Vardi. Sublinear graph augmentation for fast query implementation. In Leah Epstein and Thomas Erlebach, editors, *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, volume 11312 of *Lecture Notes in Computer Science*, pages 181–203. Springer, 2018. `doi:10.1007/978-3-030-04693-4_12`.

7 Guy Even, Reut Levi, Moti Medina, and Adi Rosén. Sublinear random access generators for preferential attachment graphs. *arXiv preprint arXiv:1602.06159*, 2016.

8 Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *CoRR*, cs.DM/0405020, 2004. URL: `http://arxiv.org/abs/cs/0405020`.

9 Alan M. Frieze, Navin Goyal, Luis Rademacher, and Santosh S. Vempala. Expanders via random spanning trees. *SIAM J. Comput.*, 43(2):497–513, 2014. `doi:10.1137/120890971`.

10 Anna C Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, Sivaramakrishnan Muthukrishnan, and Martin J Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 389–398, 2002.

11 Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, pages 68–79. IEEE Computer Society, 2003. `doi:10.1109/SFCS.2003.1238182`.

12 Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM Journal on Computing*, 39(7):2761–2822, 2010.

13 Oded Goldreich and Dana Ron. On testing expansion in bounded-degree graphs. *Electron. Colloquium Comput. Complex.*, 7(20), 2000. URL: `http://eccc.hpi-web.de/eccc-reports/2000/TR00-020/index.html`.

14 Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 13–21. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.75.

15 Jakub Lacki, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. Walking randomly, massively, and efficiently. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 364–377. ACM, 2020. doi:10.1145/3357713.3384303.

16 Aleksander Madry, Damian Straszak, and Jakub Tarnawski. Fast generation of random spanning trees and the effective resistance metric. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 2019–2036. SIAM, 2015. doi:10.1137/1.9781611973730.134.

17 Brendan D McKay and Nicholas C Wormald. Asymptotic enumeration by degree sequence of graphs with degrees $o(n^{1/2})$. *Combinatorica*, 11(4):369–382, 1991.

18 Moni Naor and Asaf Nussboim. Implementing huge sparse random graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 596–608. Springer, 2007.

19 Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196, 2007.

20 Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 223–238. Tsinghua University Press, 2011. URL: http://conference.iiis.tsinghua.edu.cn/ICS2011/content/papers/36.html.

21 Ronitt Rubinfeld and Arsen Vasilyan. Approximating the noise sensitivity of a monotone boolean function. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, volume 145 of *LIPIcs*, pages 52:1–52:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.52.

22 Atish Das Sarma, Danupon Nanongkai, Gopal Pandurangan, and Prasad Tetali. Distributed random walks. *J. ACM*, 60(1):2:1–2:31, 2013. doi:10.1145/2432622.2432624.

23 Shang-Hua Teng. Independent sets versus perfect matchings. *Theor. Comput. Sci.*, 145(1&2):381–390, 1995. doi:10.1016/0304-3975(94)00289-U.

A Proof of Lemma 4.6

We apply the configuration model of [5], extended to sequences of degrees. In the configuration model, given a degree sequence $\mathbf{d} = (d_i)_{i\in[n]}$, we place d_i half-edges at vertex i and connect all half-edges with a random matching. In the case where $d_i = d$ for all i, if the graph induced by a random matching is simple, we produce a random draw from $\mathbf{G}(n, d)$. In our case, we "remove" half edges that are already occupied by S, place a random matching on the remaining half edges, and show that if the induced graph is simple and does not duplicate edges in S, we obtain a random draw from $\mathbf{G}(n, d) \cap S$. We can use then use this sampling procedure to analyze conditional edge probabilities.

We first recall a lower bound on the probability that such a random matching induces a simple graph. For a degree sequence $\mathbf{d}$, define $D = D(\mathbf{d}) = \sum_{i=1}^{n} d_i$, $D_2 = \sum_{i=1}^{n} d_i(d_i - 1)$ and $D_3 = \sum_{i=1}^{n} d_i(d_i - 1)(d_i - 2)$. Let $P(\mathbf{d})$ be the probability that a random matching on $\mathbf{d}$ has no loops or multiple edges. The forthcoming lemma assumes $\max_i d_i^3 = o(D)$ which clearly holds in our application.

▶ **Lemma A.1** ([17] Lemma 5.1).

$$P(\mathbf{d}) \geq \exp\left(-\frac{D_2}{2D} - \frac{D_2^2}{4D^2} - \frac{D_2^2 D_3}{2D^4}\right).$$

We can then apply this lemma to prove the main claim. We remark that the bound $|S| \leq \sqrt{n}$ be be improved to $|S| = o(n)$, with c_d depending on when $|S|/n$ falls below some constant threshold.

▶ **Lemma 4.6.** *For all $d \in \mathbb{N}$ there is a constant c_d depending only on d such that for an arbitrary set of edges S with $|S| \leq \sqrt{n}$ and $v, w \in V$ arbitrary vertices where $(v, w) \notin S$, we have $\Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}[(v, w) \in G] \leq c_d/n$.*

Proof. Let $\mathbf{d} = (d_i)_{i \in [n]}$ be the sequence where d_i is the remaining degree of vertex i given S. We place a random matching on this degree sequence. Given such a matching M, we contract it to a (multi) graph G_M by treating each bucket as a single vertex.

▷ Claim A.2. Given a randomly drawn matching M where G_M is simple and $G_M \cap S = \emptyset$, $G_M \cup S$ is a uniform draw from $\mathbf{G}(n, d) \cap S$.

Proof. All possible simple graphs G_M are induced by exactly $\prod_i (d_i!)$ matchings, so the conditional distribution over such graphs is uniform. Then multiplying by the indicator variable $\mathbb{I}[G_M \cap S = \emptyset]$, which corresponds to there being no duplicated edges between G_M and S, produces the uniform distribution over the desired subset of graphs. ◁

We next show G_M satisfies the conditions of Claim A.2 with probability depending only on d. First, we show the matching is simple not considering the edges of S with constant probability.

▷ Claim A.3. We have $\Pr(\mathbb{I}[G_M \text{ simple}]) = P(\mathbf{d}) \geq \exp(-d(d + 2))$.

Proof. We use the (crude) bounds $D \geq dn/2$, $D_2 \leq d^2 n$ and and $D_3 \leq d^3 n$. Then applying Lemma A.1,

$$P(\mathbf{d}) \geq \exp(-d^2 n/dn - d^4 n^2/d^2 n^2 - d^4 n^2 d^3 n/8d^4 n^4) = \exp(-d(d + 1 + d^2/8n))$$

and choosing $n \geq d^2$ gives the claimed bound. ◁

We then show G_M duplicates edges in S with vanishing probability, which suffices to establish a constant lower bound on the probability of a "good" draw.

▷ Claim A.4. $\Pr(\{G_M \text{ simple}\} \cap \{G_M \cap S = \emptyset\}) = \rho_d > 0$

Proof.

1. Taking n large enough $\Pr(G_M \text{ simple}) \geq \exp(-d(d + 2))$ by the previous claim.
2. The probability of an edge between any two vertices in G_M is at most $2d^2/dn$ by a union bound. There are at most $\sqrt{n}$ pairs of vertices with edges in S, so by a further union bound all such pairs are missing with probability at least $1 - 2d/\sqrt{n}$.

Then taking n large enough that the second term is at least $1 - \exp(-d(d + 2))/2$, we have $\Pr(\mathbb{I}[G_M \text{ simple}] \cap \mathbb{I}[G_M \cap S = \emptyset]) \geq \exp(-d(d + 2)/2)/2 = \rho_d$ as desired. ◁

Now we are almost done. We have $\Pr[(v,w) \in G_M] \le 2d/n$ and thus

$$\begin{aligned}\Pr_{G \leftarrow \mathbf{G}(n,d) \cap S}[(v,w) \in G] &= \Pr[(v,w) \in G_M | \{G_M \text{ simple}\} \cap \{G_M \cap S = \emptyset\}] \\ &\le \Pr[(v,w) \in G_M] / \Pr[\{G_M \text{ simple}\} \cap \{G_M \cap S = \emptyset\}] \\ &\le \frac{2d/\rho_d}{n}.\end{aligned}$$

So taking $c_d = 2d/\rho_d$ (and increasing as needed to handle the small n cases by making the bound greater than 1) we conclude. ◀

B Local Access With Algebraic Structure

We now detail the approach for memoryless local access oracles for random walks on abelian Cayley graphs and graph products. The methods we use are simple and similar to those of [23], who construct algorithms for efficient *parallel* generation of random walks on a variety of structured graphs. In each case, there is some element of algebraic structure that enables sampling the relevant feature of a walk (its position at a new timestep) via sampling lower-dimensional distributions.

We first recall the Multinomial (MNom) and Multivariate Hypergeometric (MHGeom) distributions, which we can sample from efficiently.

▶ **Proposition B.1** ([10],[4] Theorem 21). *Given $\epsilon > 0$, we can sample from the following distributions within ϵ in ℓ_1 distance:*

1. *given $t \in \mathbb{N}$, $(p_1, \dots, p_d) \in \mathbb{Q}^d$, we can generate $S \leftarrow \mathrm{MNom}(t, (p_1, \dots, p_d))$ in time $O(d \cdot \mathrm{polylog}(t, 1/\epsilon))$,*
2. *given $m \in \mathbb{N}$, $(c_1, \dots, c_d) \in \mathbb{N}^d$, we can generate $S \leftarrow \mathrm{MHGeom}(m, (c_1, \dots, c_d))$ in time $O(d \cdot \mathrm{polylog}(m, \sum_i c_i, 1/\epsilon))$.*

Note that sample time is linear in the dimension of the distribution but (poly)logarithmic in the number of elements.

B.1 Low-Degree Abelian Cayley Graphs

For all Cayley graphs, sampling a walk of length ℓ is equivalent to sampling a random product of elements in S of length ℓ. But in the abelian case, the value of a random product (and thus endpoint of a random walk) only depends on the *counts* of elements in the product. Thus we can sample the distribution of edge labels, and thus endpoints, in time linear in d but logarithmic in ℓ.

To do this, we first recall the distribution of edge labels in a random product.

▶ **Proposition B.2.** *Let $G = \mathrm{Cay}(\Gamma, (e_1, \dots, e_d))$ be an abelian Cayley graph where $|\Gamma| = n$.*

1. *The counts of edge labels in a random walk of length ℓ from any vertex are distributed $\mathrm{MNom}(\ell, (1/d, \dots, 1/d))$.*
2. *Let $D_C(c_1, \dots, c_d)$ be the set of random walks from any vertex of length $\ell = \sum_{i=1}^d c_i$ that traverse c_i edges with label i. Then the counts of edge labels along the first $t \le \ell$ steps of walks in $D_C(c_1, \dots, c_d)$ are distributed $\mathrm{MHGeom}(t, (c_1, \dots, c_d))$.*

We can then provide a memoryless local access oracle for abelian Cayley graphs. In the random regular graph case, the difficulty came from sampling conditional “products”, but here we take advantage of that fact that permuting the order of elements in a product preserves endpoints in order to sample counts of edge labels unconditionally.

▶ **Theorem 5.2.** *Let $G = \mathrm{Cay}(\Gamma, S)$ be an abelian Cayley graph on n elements with $d = |S|$, where for all $g \in G$, g^2 is computable in $\mathrm{polylog}(n)$ time. There is a memoryless local access oracle for random walks on G which uses $O(d \cdot \mathrm{polylog}(nL))$ time and working space per query, where L is the maximum query time.*

Proof. Suppose we receive a query at time t. Note that in all cases when we sample from a distribution to determine the labels starting at time t', we use the random bits $\mathbf{R}_{t'}$ associated with the time t'. This guarantees consistently between independent instantiations of the oracle sharing the same random tape.

We perform binary search on the distribution of edge label counts. We first sample $S = [s_1, \ldots, s_d] \leftarrow \mathrm{MNom}(L, (1/d, \ldots, 1/d), 1/n^c L)$, where s_i holds the number of steps along edges with label i from time 0 to time L. We then maintain endpoints $t_- = 0$, $t_+ = L$ with the invariant that $t_- \leq t \leq t_+$ and the position of the walk at time t_- has been determined (which is initially satisfied by the start vertex).

At each step, suppose the midpoint $m = (t_+ + t_-)/2$ satisfies $m \leq t$. Then let $S' = \mathrm{MHGeom}(m, S, 1/n^c L)$ be the counts of edge labels traversed in the interval $(t_-, m]$. Since the endpoint of a walk is a function purely of the counts of edge labels, given v_{t_-} and S' we can compute the position of the walk at time m, denoted v_m, in time $d\,\mathrm{polylog}(nL)$. Furthermore, let $S \leftarrow S - S'$ be the counts of edge labels in the interval $(m, t_+]$. We then recurse on this interval with $t_- = m$ and $t_+ = t_+$. A nearly identical procedure holds when $t < m$. Then after $\log L$ levels we will have determined the position v_t of the sampled walk at time t, which we can return.

The runtime is immediate from Proposition B.1 and the assumption that group products are computable in time $\mathrm{polylog}(n)$. Furthermore, the vertex reached by a random walk on an abelian Cayley graph is a deterministic function of the counts of the preceding *edge labels*. Therefore ensuring the counts in each new dictionary are sampled to within $1/n^c L$ of the true distribution is sufficient to establish the approximation by Proposition 2.6. Since in both cases we approximate the true distribution to within $1/n^c L$ in ℓ_1 distance by Proposition B.2, the result follows. ◀

B.2 Tensor Products

We can utilize the structure of common graph product operations to provide memoryless local access oracles , given oracles for their components. To do so, we give arguably the simplest possible memoryless local access oracle, one that is only efficient when the time queries are far larger than the size of the graph, to use as the basis for product constructions.

▶ **Lemma B.3.** *Given a graph $G = (V, E)$ on n vertices, there is a memoryless local access oracle with $O(n^\omega \,\mathrm{polylog}(nL))$ runtime and working space per query.*

Proof. Suppose we receive a query at time t. We again perform binary search on which times to determine. We first determine v_L, then determine $\log L$ positions bracketing t. For each position to be determined, let W be the transition matrix of G and suppose the closest previously determined times are v_{t_-} and v_{t_+}. Then for $v \in V$,

$$\Pr(P_m(D_C(G, v_{t_-}, v_{t_+}, l)) = v) = W^m_{v_{t_-},v} W^{l-m}_{v,v_{t_+}} \Big/ \sum_{u \in V} W^m_{v_{t_-},u} W^{l-m}_{u,v_{t_+}}.$$

We can then use $\log(L)$ repeated squares of the random walk matrix of the graph to compute the PDF, and then sample to the desired accuracy. The runtime is direct from the description. ◀

To make this algorithm concrete, for a graph G on n vertices, we obtain a runtime of $\widetilde{O}(n^\omega)$, while we desire runtime polylogarithmic in n.

We recall the tensor product of graphs.

▶ **Definition B.4.** *Given graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ the **tensor product** of G_1 and G_2, denoted $G_1 \times G_2$, is the graph with vertex set $V_1 \times V_2$ where $(v_1, v_2), (w_1, w_2)$ are adjacent if and only if $(v_1, w_1) \in E_1$ and $(v_2, w_2) \in E_2$.*

The projections of a random walk on the tensor product onto its component graphs induce independent random walks over each graph. Then we can easily decompose sampling conditional products to sampling on the components.

▶ **Lemma B.5.** *Given memoryless local access oracles $\mathcal{A}_1, \mathcal{A}_2$ for graphs G_1, G_2 with runtime $T(\mathcal{A}_1, c, L)$ and $T(\mathcal{A}_2, c, L)$, there is a memoryless local access oracle $\mathcal{A}_T$ for $G_1 \times G_2$ with runtime $T(\mathcal{A}_T) = T(\mathcal{A}_1, c, L) + T(\mathcal{A}_2, c, L) + O(\log(|G_1| \cdot |G_2|))$.*

Proof. The algorithm initializes both sub-oracles $\mathcal{A}_1, \mathcal{A}_2$ with maximum walk length L. Suppose we receive a query at time t. Then $\mathcal{A}_T$ itself queries $\mathcal{A}_1, \mathcal{A}_2$ with time t, where $\mathcal{A}_T$ allocates disjoint sections of the random tape $\mathbf{R}_1, \mathbf{R}_2$ to both. Let the obtained vertices be v', w' respectively. Then $\mathcal{A}$ returns (v', w'). Since the vertex in a walk on a tensor product is a deterministic function of the two (independent) component distributions, by Proposition 2.6 we obtain a $2/n^c$ approximation for any constant c. The runtime is composed of the required calls to the sub-oracles, plus the time to write the inputs to each and output the returned vertex. ◀

We then obtain efficient local access to walks on arbitrarily dense graphs.

▶ **Corollary B.6.** *Let G be an arbitrary graph. For all $k \geq 1$ there is a local access algorithm for $G^{\times k}$ with runtime $O(k \operatorname{polylog}(kL))$, where we hide factors polynomial in $|G|$.*

For G a regular graph with degree d, since $|G^{\times k}| = |G|^k$ and $\deg(G^{\times k}) = d^k$, we obtain fast memoryless local access oracles for walks on infinite families of degree $(|G|^k)^\delta$ graphs for any $\delta = \log_{|G|}(d) \in (0, 1]$. This indicates that high degree does not prevent constructing efficient memoryless local access oracles. We remark that a similar approach gives fast memoryless local access oracles for the Cartesian product.