Sublinear graph augmentation for fast query implementation

Artur Czumaj* Yishay Mansour† Shai Vardi‡

April 23, 2017

Abstract

We introduce the problem of augmenting graphs with sublinear memory in order to speed up replies to queries. As a concrete example, we focus on the following problem: the input is an (unpartitioned) bipartite graph $G = (V, E)$. Given a query $v \in V$, the algorithm's goal is to output v's color in some legal 2-coloring of G, using few probes to the graph. All replies have to be *consistent with the same 2-coloring.*

We show that for graphs with good mixing time, there exists a randomized algorithm that replies to queries using $\tilde{O}(\sqrt{n})$ probes and no additional memory. In contrast, we show that any deterministic algorithm for such graphs that uses no memory augmentation requires a linear number of probes. We give an algorithm for grids and tori that uses a sublinear number of probes and no memory. On the negative side, we show that even with unbounded preprocessing, a natural family of algorithms, probe-first local computation algorithms, requires $\Omega(n/\alpha)$ probes if the graph is augmented with α words of memory. Last, we give an algorithm for trees that errs on a sublinear number of edges (i.e., a sublinear number of edges are monochromatic under this coloring) that uses sublinear preprocessing, memory and probes.

1 Introduction

Consider a scenario in which we would like to implement query access to parts of a solution to a combinatorial problem on some huge graph G. For example, for the maximal/maximum matching problem, we would like to be able to query every vertex for an incident edge (if any) that belongs to an optimal (or approximately optimal) solution. In the classical model of algorithmic analysis, this task can be solved using global computations, by running an algorithm that takes as its input the entire graph, performs some computations, and returns the output; for massive graphs such an approach is not feasible - just reading the entire input may be too costly. In this paper, we take another approach, which is natural in the setting of large graphs. We consider a scenario when we have *probe access to* G and some *additional, strictly limited amount of memory* at our disposal. A probe[1] specifies a vertex v and a port number i, and the reply to the probe is v's i^{th} neighbor (for simplicity, we assume that the reply also includes the neighbor's degree). To measure the quality of performance of our algorithms, we consider four parameters in this work: the number of probes made to the graph (which also serves as a proxy for

*University of Warwick, UK. E-mail: a.czumaj@warwick.ac.uk.

†Tel Aviv University, Tel Aviv, Israel. E-mail: mansour@tau.ac.il.

‡California Institute of Technology, Pasadena, CA, USA. E-mail: svardi@caltech.edu.

[1]Feige et al., [7], differentiate between strong and weak probes. Our definition of probe is consistent with their definition of weak probe, which is also the definition of Goldreich and Ron [12].

runtime), the size of additional memory, the preprocessing time and the quality of the solution. Specifically, we consider the trade-offs between them.

To make this more precise, we focus on a concrete problem: 2-*coloring of a bipartite graph*. Given a graph $G = (V, E)$, and a coloring of the vertices $c : V \to \{\texttt{red}, \texttt{blue}\}$, we say an edge $(u, v) \in E$ is *monochromatic* if $c(u) = c(v)$. A coloring c is ϵ-*almost legal* if at most $\epsilon|E|$ edges are monochromatic.

Given an uncolored graph $G = (V, E)$ that is known to be bipartite, we would like to design an algorithm $\mathcal{A}$ that has probe access to the graph, takes queries of the form $v \in V$ and outputs a color $c(v) \in \{\texttt{red}, \texttt{blue}\}$ such that the following hold:

1. $\{c(v) : v \in V\}$ is an ϵ-almost legal coloring of G;
2. $\mathcal{A}$ uses at most k probes per query;
3. $\mathcal{A}$ uses at most α bits of (auxiliary) memory;
4. $\mathcal{A}$ uses no preprocessing (that is, A is not allowed to probe the graph until the first query).

Naturally, we would like ϵ, k and α to be as small as possible. If $\epsilon = 0$, we say that the algorithm is *exact*.

To see the difficulty of this scenario, first consider the case that we want $\epsilon = 0$, and G is an adversarial n-vertex cycle, with vertex IDs taking the values $\{1, 2, \ldots, n\}$. Assume the reply to the first queried vertex is `red`, and the next queried vertex is "on the other side" of the cycle. We can clearly not hope to give the correct color with $o(n)$ probes per query, regardless of how much memory we have. This example shows the main challenge in designing good algorithms in our setting: how to coordinate local computations to establish global properties of the inout graph, to maintain consistent local information. We prove a more general impossibility result (Theorem 1.5).

The problem considered naturally falls in the framework of *local computation algorithms* (LCAs), as studied by Rubinfeld et al. [28] (see Section 2 for a formal definition). This work differs from previous work on LCAs, e.g., [6, 15, 19, 21], in that we allow the LCA to modify the memory over its execution. In previous works the auxiliary memory was used exclusively to store a random seed that is needed for consistency in randomized LCAs, e.g., [7, 22, 27].

1.1 Our Contributions

For simplicity, in all of our results, the amount of memory is a hard constraint; the bound on the number of probes is satisfied with arbitrarily high probability. It is easy to adapt the (positive) results so that the converse is true (and hence also that both are satisfied w.h.p.) Intuitively, *the difficulty of the problem arises from the fact that no preprocessing is allowed*. If a linear amount of preprocessing is allowed, we can simply use BFS to color the graph, and then store some "central" vertices along with their color. We show that this can be implemented by selecting a random subset of the vertices to be central nodes. Then a simple BFS from each queried vertex is guaranteed to quickly find a central node w.h.p.

Theorem 1.1. *For any $\alpha > 0$, and any connected bipartite graph $G = (V, E)$, there exists an exact 2-coloring algorithm for G that uses $O(|E| + |V|)$ preprocessing time, $\alpha \log |V|$ words of memory and performs $O\left(\frac{|V|}{\alpha}\right)$ probes w.h.p.*

In Appendix A, we extend Theorem 1.1 to arbitrary (not necessarily connected) graphs.

If the input is large, however, even linear preprocessing may be infeasible. Our main focus in this paper is therefore the case when little to no preprocessing is allowed. We first consider what can be done using no memory. We give algorithms for graphs that have good mixing time and for grids (note that grids are poorly mixing) that require no preprocessing and no memory. The high level of both is the same: given the queried vertex, we find a path from it so some predetermined vertex (in our case, we choose the vertex with ID 0), whose color is set to be `red`. If the path between q and 0 is of even length, we color q `red`, otherwise we color it `blue`. On graphs with fast mixing time graphs, we perform two random walks, one from q and one from 1, until they intersect. On grids, we show how to efficiently construct a hyperplane containing q and 1-dimensional paths from vertex 0; we use their intersection to find the path. The results on grids immediately apply to tori as well. For graphs that have good expansion, we give a lower bound of $\Omega(n)$ probes for deterministic algorithms to complement the upper bound of $\tilde{O}(\sqrt{n})$ (where n is the number of vertices), showing that randomization is necessary to obtain algorithms with small probe complexity for this problem on graphs with good expansion (we use $\tilde{O}$ to hide logarithmic dependencies, for clarity of presentation). The graphs with good expansion that we use to prove the lower bound are complete bipartite graphs. Note that complete bipartite graphs are excellent expanders.

Theorem 1.2. *There exists a randomized exact 2-coloring algorithm for bipartite graphs with mixing time $\tau = O(\log n)$ that use no preprocessing, no memory and $O(\sqrt{n}\log n)$ probes.*

Theorem 1.3. *There does not exist a deterministic algorithm for exact bipartite 2-coloring complete bipartite graphs that uses no memory and fewer than $\frac{n}{4}$ probes.*

Theorem 1.4. *There exists a deterministic exact 2-coloring algorithm for r-dimensional grids with no preprocessing and no memory that performs $O\left(rn^{\left(\frac{r-1}{r}\right)}\right)$ probes.*

Theorem 1.3 holds for algorithms for complete bipartite graphs that use no auxiliary memory. We give a more general lower bound for a natural family of deterministic algorithms: deterministic probe-first LCAs. Probe-first LCAs are ones whose probes are independent of the memory: the LCA first performs its probes and only then consults the memory. While the memory may provide useful information, it doesn't provide any encoding of the graph itself. All of the LCAs in this paper can be thought of as probe-first LCAs, and indeed, we are not aware of any deterministic LCA in the literature that is not (and can not be formulated as one). The randomized LCAs in the literature only read the memory to obtain the random seed, and so are not technically probe-first LCAs. We remark upon this further in Section 2. In Section 6, we show that if a probe-first LCA is allowed access to α bits of memory (where α is possibly a function of n), then it requires $\Omega(n/\alpha)$ probes to find an exact 2-coloring.

Theorem 1.5. *For any $\alpha > 0$, $k \geq 0$, and $n \geq (k+2)\alpha + 2$, there does not exist a deterministic probe-first LCA for exact bipartite 2-coloring even-length cycles of length n that uses at most k probes and less than α bits of memory, even with unbounded preprocessing.*

An interesting open problem is whether this lower bound extends to arbitrary LCAs.

Our results for grids and graphs with good expansion implicitly use the fact that these graphs have (relatively) small diameter. What if the diameter is unbounded? For the special family of trees, we show the following.

Theorem 1.6. *For any $\alpha > 0$, there exists a bipartite 2-coloring LCA for trees with $O(\alpha)$ preprocessing and $O(\alpha)$ memory that performs $O(\frac{n \log n}{\alpha})$ probes with probability $1 - \frac{1}{\text{poly}(n)}$ with at most $\alpha - 1$ violated edges.*

We summarize these results in Table 1.

Graph type	Preproc. Time	Memory (words)	Probes	Failure Probability[2]	Edges Violated
General	$O(m+n)$	$O(\sqrt{n}\log n)$	$O(\sqrt{n})$	$1/\operatorname{poly}(n)$	0
Cycle (det. pf.)	any	α	$\Omega(n/\alpha)$	0	0
Expander	0	0	$\tilde{O}(\sqrt{n})$	$1/\operatorname{poly}(n)$	0
Expander (det.)	0	0	$\Omega(n)$	0	0
r-dim Grid	0	0	$O\left(rn^{\left(\frac{r-1}{r}\right)}\right)$	0	0
Tree	$O(\alpha)$	$O(\alpha)$	$\tilde{O}(n/\alpha)$	$1/\operatorname{poly}(n)$	$\alpha - 1$

Table 1: Summary of results - n is the number of vertices; m is the number of edges in the input graph. "det." means the bound holds for deterministic algorithms. "det. pf." means the bound holds for deterministic probe-first algorithms.

1.2 Related Work

If preprocessing is allowed, our model is similar in spirit to graph sparsifiers e.g., [4,10,30], in particular graph spanners, e.g., [3,8,25]: given a connected, edge-weighted graph $G = (V, E)$, a spanner is a sparse subgraph H of G that approximately preserves all pairwise distances. A spanner can be thought of as adding some auxiliary information to the graph: which edges of G are in H. This additional information allows for saving in query reply time. Spanners are useful in routing [26,32], by allowing for small routing tables, and in distance oracles [5,31,33]. For example, Chechik [5] shows that it is possible to augment a graph with $O(kn^{1+1/k})$ bits of memory, such that it is possible to reply to queries of the form "what is the distance between u and v?" in time $O(1)$, where the reply is a $2k-1$ approximation to the real distance. The two main conceptual differences between this work and the previous work on spanners are (1) we are interested in performing only *sublinear* or even *no preprocessing* (all of the above works use preprocessing that is polynomial in the input size), and (2) we wish to augment the graph with *sublinear memory*.

Rubinfeld et al. [28] formally introduced the LCA model, though several well studied models fit within the framework. For example, *local reconstruction* [1,29]: given access to a function g that is close to having a certain property (e.g., monotonicity) the goal is to reply to a query x with some value $f(x)$ such that f is close to g, using few probes to g. *Locally decodable codes* e.g., [16,34] allow the decoding of part of a code without decrypting it in its entirety. In the past few years, many papers have studied LCAs for maximal independent set (e.g., [2,11]), maximal and approximate maximum matching (e.g., [19,23]) and coloring (e.g., [6,9]). Most of the work has been on graphs of bounded degree graphs; recently Feige et al. [7] considered LCAs on graphs of unbounded degree, employing sparsification techniques to obtain LCAs for weak coloring and approximate maximum matching. London et al. [20] showed how to apply LCAs to convex optimization to obtain distributed algorithms for e.g., network utility maximization, that are robust

to link failures. Levi et al., [18] describe LCAs that reply to queries of the form "is this edge in a sparse spanner of G?"

Göös et al. [14] show that for certain problems, remote probes do not help. A remote probe is a probe to a vertex ID that was neither given as the query nor received as a reply to a probe. In this work, we give the first non-trivial[3] LCAs that make use of remote probes. The result of [14] applies to LCL (locally checkable labeling) problems [24] on bounded degree graphs when the number of probes is $o(\sqrt{\log n})$. 2-coloring is also an LCL problem, but the number of probes our algorithm uses is greater than their threshold, hence the results do not contradict. We conjecture that there is no LCA for exact 2-coloring bipartite graphs that uses $\tilde{O}(\sqrt{n})$ probes, and uses neither remote probes nor any additional memory.

A large body of work in the property testing literature has been devoted to testing bipartiteness e.g., [12, 13, 17]. Property testing differs conceptually from this line of work in that we are guaranteed that the input is bipartite. Testing for bipartiteness usually involves sampling a subgraph and coloring it [12, 17]; it is not clear whether it is possible to adapt these testers to LCAs, while ensuring the consistency of the coloring. It is interesting to extend the results of the paper to graphs that are only guaranteed to be "almost" bipartite.

2 Preliminaries

We denote the set of integers $\{0, 1, 2, \ldots, n-1\}$ by $[n]$. Logarithms are natural. Our input is a simple undirected bipartite graph $G = (V, E)$, in which every vertex has an ID and all IDs are distinct, $|V| = n$; each vertex is assigned a unique ID in $\{0, 1, 2, \ldots, n-1\}$. The neighborhood of a vertex v, denoted $N(v)$, is the set of vertices that share an edge with v: $N(v) = \{u : (u, v) \in E\}$. The *degree* of a vertex v is denoted $d_v = |N(v)|$.

We think of each vertex v as having d_v *ports*, $1, 2, \ldots, d_v$, where d_v is v's degree. Each of v's neighbors is connected to v via a single unique port. There are two probe models for LCAs: *strong* and *weak*. A *strong probe* is given a vertex ID and returns a list of the vertex's neighbors. A *weak probe* is given a vertex ID v and a port number i and returns v's i^{th} neighbor u, and u's degree. In this paper (with the exception of Appendix A), we focus on weak probes.

2.1 LCAs.

Our definition of LCAs is slightly different from earlier definitions considered in the literature. The main differences are the following:

1. We allow the LCA to write on the auxiliary memory. Although not explicitly disallowed in most previous work, the enduring memory has thus-far (e.g., [2, 19, 22, 23, 27]) only been used for storing a random seed.

2. The randomness we use is not identical between queries. In previous work, randomized LCAs fixed their randomness before the first query; thereafter, they behaved deterministically: if the same query was given several times, the LCA would perform exactly the same steps. In this work, we allow LCAs to perform different actions if the same query is given (under the condition that the reply to the query is the

[3] For example, a trivial problem is the following: if vertex number 1 has an even degree, color all vertices `blue`, otherwise color all vertices `red`.

same). For this reason, our failure probability is per probe and not over all possible queries.

Definition 2.1 (Local computation algorithm)**.** *A $p(n)$-probe $m(n)$-memory local computation algorithm $\mathcal{A}$ for a computational problem is an algorithm that receives an input of size n. Given a query x, $\mathcal{A}$ makes at most $p(n)$ probes to the input in order to reply. $\mathcal{A}$ is allocated a memory of $m(n)$ bits in addition to the memory required to reply to each query. $\mathcal{A}$ must be* consistent*; that is, the algorithm's replies to all possible queries must combine into a single feasible solution to the problem.*

If the LCA is randomized, it also has a failure probability *$\delta(n)$, which is the probability per query that $\mathcal{A}$ uses more than $p(n)$ probes.*

We note that we define randomized LCAs as *Las Vegas* LCAs.[4] One can easily transform a Las Vegas LCA to a Monte Carlo LCA by letting the LCA return "fail" or an arbitrary output whenever the allowed number of probes is exceeded.

Definition 2.2 (Probe-first LCAs)**.** Probe-first *LCAs are LCAs whose choice of probes is not a function of the memory. In other words, the LCA first performs the probes and only then accesses the memory.*

All of the LCAs of this paper are either probe-first LCAs or they are Las Vegas LCAs and their Monte Carlo variants can be viewed as probe-first LCAs.

2.2 Breadth First Search

Several of the LCAs herein use Breadth-First Search (BFS). During its execution, the BFS algorithm maintains a data structure that contains so called "gray" vertices - vertices that have been encountered, but whose neighbors have not yet been probed. At any time, the vertices in the data structure are all at distance i or $i+1$ from the root of the BFS tree, for some i. In the generic BFS description, when it "pops" a new vertex from the data structure, it arbitrarily chooses one of the vertices at distance i. For consistency, we always break ties by ID; that is, our BFS always chooses the vertex at distance i with the lowest ID. The tie breaking with respect to the ports is done similarly: lower ID ports are chosen first.

3 Connected General Graphs with Preprocessing

Given linear preprocessing, there is an LCA for 2-coloring arbitrary (connected) bipartite graphs that uses k probes α and bits of memory such that $k\alpha = \tilde{O}(n)$.

Theorem 1.1. *For any $\alpha > 0$, and any connected bipartite graph $G = (V, E)$, there exists an exact 2-coloring LCA for G that uses $O(|E| + |V|)$ preprocessing time, $\alpha \log |V|$ words of memory and performs $O\left(\frac{|V|}{\alpha}\right)$ probes w.h.p.*

Proof. Let $G = (V, E)$ be a connected bipartite graph. In the preprocessing stage, (arbitrarily) 2-color G. Uniformly at random, choose a set $S \subset V$ of $\alpha \log n$ vertices, and save their ID and color to memory.

For any queried vertex v, perform BFS until some vertex $s \in S$ is encountered; v is colored according to the parity of a path between v and s. After k probes, the probability that no vertex $s \in S$ is encountered is

[4] Las Vegas algorithms always return a correct answer; Monte Carlo algorithms always obey their allotted runtime.

$$\prod_{i=0}^{k-1}\left(1-\frac{\alpha\log n}{n-i}\right)\leq\left(1-\frac{\alpha\log n}{n}\right)^{k}.$$

Setting $k=O(\frac{n}{\alpha})$ gives that this probability is at most $e^{-\Omega(\log n)}$. □

In Appendix A, we extend Theorem 1.1 to arbitrary (not necessarily connected) graphs.

4 Graphs with fast mixing time

In this section, we consider regular bipartite graphs whose mixing time is logarithmic in the number of vertices. More formally, let $G=(V,E)$ be a connected bipartite graph with maximal degree Δ, we define a Markov chain for a random walk on G as follows: for every vertex v, move to each of v's neighbors w.p. $\frac{1}{2\Delta}$, and stay at v w.p. $1-\sum_{i=1}^{d_v}\frac{1}{2\Delta}$. This chain is clearly aperiodic (every vertex has a self-loop) and irreducible, and as its transition matrix is symmetric, its stationary distribution is the uniform distribution, i.e., $\forall v\in V:\pi(v)=\frac{1}{n}$.

With regard to the mixing time, we require the following: Starting from any vertex, following a random walk of length $\tau=\Theta(\log n)$, the probability that we are at any vertex u is approximately $1/n$. For concreteness, we require that the probability that we are at vertex u is at least $\frac{1}{n}-\frac{1}{n^2}$. We call τ the *mixing time*. We note that such graphs are ubiquitous, e.g., a complete bipartite graph with self loops of constant probability has constant mixing time. Our main positive result of this section is the following:

Theorem 1.2. *There exists a randomized exact 2-coloring LCA for bipartite graphs with mixing time $\tau=O(\log n)$ that uses no preprocessing, no memory and $O\left(\sqrt{n}\log n\right)$ probes.*

Proof. Our LCA is the following: given a vertex q as a query, perform a random walk of length $\tau\sqrt{n}$ from vertex with ID 0. Store the vertex IDs of the vertices that were encountered on this random walk; call this set S. Now perform a random walk from q until one of the vertices of S is encountered. This creates (at least one) path p between vertices 1 and q. If p is of odd length, return `blue`, otherwise return `red`.

This clearly returns a valid coloring. It remains to show that $O(\sqrt{n}\log n)$ probes are used w.h.p. This is shown in Lemma 4.1, completing the proof. □

Lemma 4.1. *Let S be the set of vertices encountered in the first random walk. The probability that none of these vertices is encountered in the second random walk after $4\tau\log n$ steps is at most $1/n^2$.*

Proof. Consider the vertices at intervals of τ steps from one another in the first random walk. Denote this set by $T\subset S$. We choose the constant c_1 so that $|T|\geq\sqrt{n}$ w.h.p. We bound the probability that none of the vertices from the second walk are in T. For each vertex $u\in S$, $\Pr[u\notin T]\leq\left(\frac{n-1}{n}+\frac{1}{n^2}\right)^{\sqrt{n}}$. Let X denote the event that none of the vertices from the second walk are in T. By the union bound

$$Pr[X]\leq\left(1-\frac{1}{n}+\frac{1}{n^2}\right)^{4n\log n}\leq\left(1-\frac{1}{2n}\right)^{4n\log n}\leq e^{2\log n}.$$

□

We complement the upper bound with a lower bound on deterministic algorithms, showing that randomness is indeed necessary for obtaining a sublinear probe complexity if no auxiliary memory is used.

Theorem 1.3. *There does not exist a deterministic LCA for exact 2-coloring complete bipartite graphs that uses no memory and fewer than $\frac{n}{4}$ probes.*

Proof. Consider any LCA $\mathcal{A}$ on the vertex set $[n]$ that uses at most $k = \frac{n}{4} - 1$ probes. Assume w.l.o.g. that $\mathcal{A}$ uses exactly k probes. It is easy to construct a subgraph G' such that given a query vertex 1, $\mathcal{A}$ will probe the vertices $2, 3, \ldots, n/4$. Similarly, it is easy to construct a subgraph G'' that guarantees that when $\mathcal{A}$ is given as a query vertex $n/4+1$, it probes vertices $n/4+2, \ldots, n/2$. As each of these subgraphs contains exactly $n/4$ vertices, and each side of an n vertex bipartite graph consists of $n/2$ vertices, it is possible to generate two complete bipartite graphs G_1, G_2, each of which have both G' and G'' as subgraphs, but so that in G_1, vertices 1 and $n/4+1$ are on the same side, and in G_2 they are on different sides. Clearly there is no coloring of the vertices 1 and $n/4+1$ that satisfies both G_1 and G_2. □

5 Grids and Tori

We describe the results for grids, noting that they hold for tori as well. Designing algorithms for a tori is intuitively at least as hard as for grids, as there is no notion of an end, and hence no absolute position, which could hypothetically be leveraged by algorithms for grids.

5.1 Warm up - 2-dimensional grids

We assume that a $\sqrt{n} \times \sqrt{n}$ grid is arranged so that each interior vertex has four neighbors - to the north, south, east and west. However, the vertices do not know which neighbor is in which direction. When we say that it is possible to *land on* a vertex we mean to reach it and "know that we are there" (to distinguish from the usual notion of reachability).

Lemma 5.1. *Given an edge (v, u), assume w.l.o.g. that u is to the north of v. It is possible to land on the vertex north of u using at most 39 weak probes.*

Proof. Vertex u has three neighbors. Not counting paths through u, two of u's neighbors are at distance 2 from v, and the third, w is to the north of u. There is no path of length 2 from w to v that does not pass through u. It takes it most 12 probes to find all vertices at distance 2 from any vertex x - three probes to find each of x's neighbors and 3 more per neighbor. Therefore 12 probes per vertex are sufficient to rule out two of u's neighbors as the northern neighbor. Adding the 3 probes made by u completes the claim. □

It is also possible to land on the vertex south of v using the same reasoning. It is easy to know when we reach the edge of the grid, as the edge vertices have one fewer neighbor; the following is immediate

Corollary 5.2. *It is possible to traverse the graph in a straight line from side to side through any vertex using $39\sqrt{n}$ probes.*

It is not possible to know in which direction the graph is being traversed (north-south or east-west), however it is not necessary.

Theorem 5.3. *There exists a deterministic bipartite 2-coloring LCA for 2-dimensional grids with no preprocessing and no memory that performs $O(\sqrt{n})$ probes.*

Proof. When queried on a vertex v, traverse the graph in both directions (north-south and east-west). Traverse the graph in one direction from the vertex with ID 0 (it will be either north-south or east-west). The line described by the traversal from vertex 1 must intersect exactly one of the lines emanating from v (unless v's ID is 0, in which case it intersects both). Coloring vertex 0 `red` determines v's color. □

5.2 k-dimensional grids

Each interior vertex of a k-dimensional grid is connected to $2k$ other vertices. The LCA is the following: When a vertex v is queried, the LCA discovers all vertices of some hyperplane passing through it; the LCA then traverses the grid along the k straight lines that pass through the vertex with ID 0. The hyperplane intersects with at least one of these lines; coloring vertex 0 `red` determines v's color.

Lemma 5.4. *Let $G = (V, E)$ be a k-dimensional grid, $k \geq 1$ and let u and v be two interior vertices such that $(u, v) \in E$. Then there exists a single $w \in N(u) \setminus \{v\}$ such that there does not exist a path of length 2 from w to v that does not include u.*

Proof. Denote $W = N(u) \setminus \{v\}$. There exists one vertex $w \in W$ for which u, v, w lie on a line. The shortest path between v and w that does not include u must have length 4. For all other $w' \in W$, u, v, w define a two dimensional plane, and there is a vertex x such that v, u, w, x comprise a square. The path w, x, v has length 2. □

Lemma 5.5. *Given an edge (v, u), assume w.l.o.g. that u is to the north of v. It is possible to land on the vertex north of u using at most $(2k-1)^3 + (2k-1)^2 + 2k - 1$ weak probes.*

Proof. Vertex u has $2k - 1$ neighbors that are not v. By Observation 5.4, all but one of them has a path of length 2 to v that doesn't pass through u. For each of the neighbors of u, it is possible to enumerate all length 2 paths using $(2k-1)^2 + 2k - 1$ probes. □

Note that it is also possible to land on the vertex to the south of v using the same reasoning, therefore:

Corollary 5.6. *It is possible to traverse the k-dimensional grid in a straight line from side to side through any vertex using $O\left(k^3 \sqrt[k]{n}\right)$ probes.*

Note that even though we can traverse the grid, we cannot know the absolute direction of the line. We show how to build on the line construction to construct a hyperplane.

Lemma 5.7. *Given a k-dimensional grid $G = (V, E)$ and a vertex $v \in V$, it is possible to construct a hyperplane that passes through v using at most $O\left(k^3 n^{\frac{1}{k}}\right) + O\left(kn^{\left(\frac{k-1}{k}\right)}\right)$ weak probes.*

Proof. Construct $k - 1$ lines passing through v that traverse the grid, $p_1, \ldots p_{k-1}$. This takes $O\left(k^3 \sqrt[k]{n}\right)$ probes. We can complete these lines to a hyperplane as follows: Set $S = \bigcup_{i=1}^{k-1} p_i$. As long as there exist two vertices $u, v \in S$, such that $N(v) \cap N(u) \neq \emptyset$, add the vertex $w \in N(v) \cap N(u)$ (there is exactly one such neighbor) to S. This takes at most k probes per vertex in the hyperplane times.

We need to show that (1) all vertices of the hyperplane are added to S, and (2) no vertices that are not in the hyperplane are added to S. To show (1), assume that some vertices in the hyperplane are not added to S. Consider the distances from v in the metric space defined by the grid. Let v be the "origin", let x be the closest vertex to v (there may be more than one) that is not in S. But then there are two neighbors of x in the plane that contains v and x that are closer to v and are in S (as x is the closest vertex not in S) hence x is too. To show (2), note that any vertex not in the hyperplane has at most one neighbor in the hyperplane at distance 1. □

Theorem 1.4. *There exists a deterministic 2-coloring LCA for k-dimensional grids with no preprocessing and no memory that performs $O\left(kn^{\left(\frac{k-1}{k}\right)}\right)$ probes.*

Proof. If v's ID is 0, color v `red`. Otherwise, construct a hyperplane through v, and k lines that traverse the grid that pass through vertex 1. The hyperplane must intersect exactly one of these lines. Color 0 `red`, and color v accordingly. □

6 Lower Bound for Probe-First Graphs

Our main result in this section is the following:

Theorem 1.5. *For any $\alpha > 0$, $k \geq 0$, and $n \geq (k+2)\alpha + 2$, there does not exist a deterministic probe-first LCA for exact 2-coloring even cycles of length n that uses at most k probes and less than α bits of memory, even with unbounded preprocessing.*

6.1 Summary of Proof

The high level of the proof is the following: assume that there exists an LCA that uses k probes and $\alpha - 1$ bits of memory and can correctly color all vertices of an n vertex graph. We build a family of 2^α graphs for which the LCA makes the same k probes when queried on the vertices $1, \ldots, \alpha$. We use these graphs to encode strings in $\{0,1\}^\alpha$. Specifically, let $\{s_1, \ldots, s_{2^\alpha}\}$ denote the set of all strings $\{0,1\}^\alpha$. We construct 2^α graphs $G^{s_1}, \ldots, G^{s_{2^\alpha}}$ such that for all $i \in [\alpha]$, the color of vertex i in G^{s_j} is `blue` iff $s_j(i) = 1$. Alice is then given a graph, corresponding to a string s, and sends $\alpha - 1$ bits to Bob. By the assumption that the LCA can color the vertices $1, \ldots, \alpha$ correctly, we get that Bob can recover s. This is true as Bob already knows the replies to all of the probes the LCA would make, and only needs the $\alpha - 1$ bits to decide the colors of $1, \ldots, \alpha$. But if this is true, we have just given a protocol for encoding α bits of information using $\alpha - 1$ bits, which is information-theoretically impossible. Hence no such LCA can exist.

Before going into more detail, we need a more precise description of LCAs.

6.2 A Concise Description of LCAs.

Assume that we have a deterministic probe-first LCA $\mathcal{A}$ that uses k probes and α bits of memory (which we sometimes call a *key*). We view the LCA is a set of functions $f_v^0, \ldots, f_v^{k-1}, v \in [V]$ that map sets of IDs (a history of replies to previous probes) to an ID and a port number:

$$f_v^i : V^i \to V \times \Delta,$$

where Δ is the maximum degree of the graph. The LCA includes one more set of functions that map the key and the history of replies to all k probes to a color:

$$f_v^k : 2^\alpha \times V^k \to \{\texttt{red}, \texttt{blue}\}.$$

As an example, the following is an excerpt of a possible set of probe functions for the query 1 (i.e., the vertex with ID 1), in a 3-probe LCA $\mathcal{A}$ that uses two bits of memory for cycles on the vertex set [9]. For clarity, instead of denoting the port numbers by the set $\{0,1\}$, we denote them by $\{\texttt{clockwise}, \texttt{anticlockwise}\}$.

$$f_1^0 = [1, \texttt{anticlockwise}],$$
$$f_1^1(2) = [1, \texttt{clockwise}], f_1^1(3) = [3, \texttt{anticlockwise}],$$
$$f_1^2(2,3) = [3, \texttt{clockwise}],$$
$$f_1^3(\{1,1\},(2,3,9)) = \texttt{red}, f_1^3(\{1,0\},(2,3,1)) = \texttt{red}$$

This can be interpreted as follows: When the LCA receives 1 as a query, it probes vertex 1's anticlockwise neighbor. If it receives as a reply that 1's anticlockwise neighbor is 2, it probes vertex 1's clockwise neighbor next. If it receives as a reply that 1's anticlockwise neighbor is 3, it probes 3's anticlockwise neighbor next. $f_1^2(2,3) = [3, \texttt{clockwise}]$ means that given that the replies to the previous two queries were 2 and 3, we next probe vertex 3's clockwise neighbor. Note that $f_1^3(2,3,1)$, while syntactically correct, is impossible for this particular LCA, as 1 cannot be 3's clockwise neighbor as it is 2's clockwise neighbor. We allow "impossible" histories, to simplify the notation and proof.

This interpretation can be extended in the natural way to general (not probe-first) LCAs, by having the key as part of the input to all of the functions.

6.3 Proof of Theorem 1.5

Proof. Assume there is a probe-first LCA $\mathcal{A}$ for bipartite 2-coloring even cycles on $n = (k+2)\alpha+2$ vertices that uses (at most) k probes and $\alpha-1$ bits of memory. For simplicity, we assume that k is even; the proof can easily be tweaked to accommodate odd k.

We use $\mathcal{A}$ to construct a set of $\alpha+1$ line segments as follows. Initialize the roots of segments $I_0, \ldots, I_\alpha$ to be vertices $0, 1, \ldots, \alpha$ respectively. We let *next* be the smallest positive integer that we have not yet used as a vertex name. Before the first probe, $next = \alpha+1$. We keep track of all the integers we have used as vertex names, labeling this set Γ. We then simulate $\mathcal{A}$ for queries $1, \ldots, \alpha$. Whenever $\mathcal{A}$ probes a vertex v and a direction $\texttt{dir} \in \{\texttt{clockwise}, \texttt{anticlockwise}\}$ such that v does not have a $\texttt{dir}$ neighbor, we let *next* be that neighbor, add *next* to Γ and update *next* to be the smallest integer not in Γ. If $\mathcal{A}$ probes a vertex v that we have not seen yet (i.e., is not in Γ), with direction $\texttt{clockwise}$ ($\texttt{anticlockwise}$), we let v be the $\texttt{anticlockwise}$ (resp. $\texttt{clockwise}$) neighbor of the $\texttt{anticlockwise}$-est (resp. $\texttt{clockwise}$-est) vertex in I_0. After simulating k probes for queries $1, \ldots, \alpha$, we have a set of $\alpha+1$ line segments. Denote this set by $\mathcal{I}$.

Note that given $\mathcal{I}$, $\mathcal{A}$ can perform all of the probes when queried on the vertices $1, \ldots, \alpha$.

If $|\mathcal{I}|$ is odd, add to $\mathcal{I}$ the next $\alpha+1$ integers in Γ, otherwise add the next α integers in Γ. We call these vertices *auxiliary* vertices. Note that $|\mathcal{I}|$ is now even.

Lemma 6.1. *$\mathcal{I}$ contains at most $(k+2)\alpha+2$ vertices.*

Proof. After the initialization of the segments, $|\Gamma| = \alpha+1$. We perform αk probes, each one adds at most 1 vertex to Γ, for a total of $\alpha+1+k\alpha$. Adding the $\alpha+1$ extra vertices, this gives that $\mathcal{I}$ has at most $(k+2)\alpha+2$ vertices. □

Now add to $\mathcal{I}$ the next $(k+2)\alpha+2-|\mathcal{I}|$ integers in Γ, to ensure that $\mathcal{I}$ has exactly $(k+2)\alpha+2$ vertices. We have constructed $\alpha+1$ segments $I_0, I_1, \ldots, I_\alpha$, such that I_i contains vertex i. These segments can be joined by either

1. connecting the `clockwise`-est vertex in I_{i-1} directly to the `anticlockwise`-est vertex of I_i, or

2. connecting the `clockwise`-est vertex in I_{i-1} and the `anticlockwise`-est vertex of I_i to an intermediary vertex

for every $i \in [\alpha]$. The cycle is then completed by connecting the `anticlockwise`-est vertex of I_α to the `clockwise`-est vertex of I_1, possibly with some intermediary vertices (to ensure all graphs have the same number of nodes). The α choices of inserting or not inserting an intermediary vertex between I_{i-1} and I_i, $i \in [\alpha]$ describe the 2^α possible graphs that we can construct in this fashion from the $\alpha + 1$ intervals.

We show how to construct the graph $G(x)$ that encodes the string $x = x_1 x_2 \dots x_\alpha$. Note that the graph is a function of $\mathcal{A}$ and x. Let $\texttt{clockwise}_i$ and $\texttt{anticlockwise}_i$ be the number of vertices `clockwise` and `anticlockwise` of vertex i in I_i respectively. Set $x_0 = 0$. For all $i \in [\alpha]$, if $\texttt{anticlockwise}_{i-1} + \texttt{clockwise}_i$ is even and $x_{i-1} \neq x_i$, or $\texttt{anticlockwise}_{i-1} + \texttt{clockwise}_i$ is odd and $x_{i-1} = x_i$, add an auxiliary vertex between I_{i-1} and I_i, otherwise connect them directly. Then connect the `anticlockwise`-est vertex of I_α to the `clockwise`-est vertex of I_0, inserting the remaining auxiliary vertices between them, completing the cycle.

This cycle encodes x, because vertex i is an odd number of vertices away from vertex 0 iff $x_i = 1$, hence a 2 coloring of the graph that assigns vertex 0 the color `red`, will assign all vertices i s.t. $x_i = 0$ the color `red` and all vertices i such that $x_i = 1$ the color `blue`.

Set $|\mathcal{I}| = n$. We have shown a construction of 2^α different graphs $\{G(x) : x \in \{0,1\}^\alpha\}$, each of length n, for which $\mathcal{A}$ performs the same probes, and gets the same replies thereto, for queried vertices $1, \dots, \alpha$, but there are no two graphs $G(x) \neq G(y)$ for which it colors all the vertices $1, \dots, \alpha$ identically.

We give the one-way $\alpha - 1$-bit protocol for encoding an α-bit string x. Alice is given $G(x)$ and sends an $\alpha - 1$-bit encoding of $G(x)$ to Bob. Bob knows the replies to the probes that the LCA $\mathcal{A}$ would make when queried on vertices $1, \dots, \alpha$. The LCA, having the replies to the probes can use the key to correctly color all the vertices $1, \dots, \alpha$ by the assumption that $\mathcal{A}$ is a 2-coloring LCA for even-length cycles of length n. But from this coloring, it is easy to decode x. Therefore such an LCA implies an encoding protocol for α bits using $\alpha - 1$ bits, a contradiction. □

7 Trees

In this section, we show that if G is a tree, we can obtain a coloring that has $\alpha - 1$ violated edges by choosing α vertices uniformly at random and coloring them `blue`. The preprocessing and memory required are therefore proportional to α.

Fix α. Choose a set $S \subset V$, $|S| = \alpha$ of *focii* uniformly at random. Color the vertices of S `blue`. Whenever a vertex v is queried, if $v \notin S$, perform a BFS. Color v consistently with the first encountered focus. We use a charging argument to show that at most $\alpha - 1$ edges are monochromatic. Define the *skeleton* of the tree to be the union of the paths between the focii. For any vertex v in the skeleton, its *skeletal subtree* is the set of vertices W not on the skeleton, for which any path connecting $w \in W$ to any skeleton on the vertex passes through v. Let T be the tree in which every vertex is given as a query to the LCA and colored according the the reply.

Lemma 7.1. *If v^* is on the skeleton, and v^*'s focus is u, then u is the focus of every vertex in v^*'s skeletal subtree.*

Proof. Let w be some vertex in v's skeletal subtree. Clearly a focus is closest to v iff it is closest to w. Furthermore, the BFS tie-breaking rule is global, guaranteeing that the focus that will be found first (out of all the closest focii) in the BFS from v is the same one that will be found by BFS from w. □

Lemma 7.2. *There are at most $\alpha - 1$ violated edges in T.*

Proof. From Claim 7.1, edge violations can only be on the skeleton. Call any vertex of degree at least 3 (that is not a focus) a *junction*. We now charge violated edges to leaves. For some leaf, follow the skeleton until we either reach a violated edge or a junction. If we reach a junction, we contract the path, so that the focus takes place of the junction. The focus is not a leaf anymore, and we have reduced the size of the skeleton (this ensures the process terminates). Otherwise (we reach a violated edge) - we charge the violated edge e to the focus, and remove the focus, e and the paths between them and from e until the nearest junction. It remains to show that no edge between e and the junction can be violated. But this is implied by Claim 7.1 (with the junction as v^*). □

Lemma 7.3. *The number of probes used to reply to any query is $O\left(\frac{n \log n}{\alpha}\right)$ with probability $1 - \frac{1}{\text{poly}(n)}$.*

The proof is similar to the proof of Theorem 1.1 and is omitted. The algorithm description together with Lemmas 7.2 and 7.3 imply Theorem 1.6.

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008. 1.2

[2] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In *Proc. 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1132–1139, 2012. 1.2, 1

[3] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1):81–100, January 1993. 1.2

[4] András A. Benczúr and David R. Karger. Approximating s-t minimum cuts in Õ(n2) time. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 47–55, 1996. 1.2

[5] Shiri Chechik. Approximate distance oracles with constant query time. In *Symposium on Theory of Computing, STOC*, pages 654–663, 2014. 1.2

[6] Guy Even, Moti Medina, and Dana Ron. Best of two local models: Local centralized and local distributed algorithms. *CoRR*, abs/1402.3796, 2014. 1, 1.2

[7] Uriel Feige, Boaz Patt-Shamir, and Shai Vardi. On the probe complexity of local computation algorithms. *CoRR*, abs/1703.07734, 2017. 1, 1, 1.2

[8] Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC*, pages 9–17, 2016. 1.2

[9] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 625–634, 2016. 1.2

[10] Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 71–80, 2011. 1.2

[11] Mohsen Ghaffari. An improved distributed algorithm for maximal independent set. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 270–277, 2016. 1.2

[12] Oded Goldreich and Dana Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999. 1, 1.2

[13] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002. 1.2

[14] Mika Göös, Juho Hirvonen, Reut Levi, Moti Medina, and Jukka Suomela. Non-local probes do not help with many graph problems. *Distributed Computing - 30th International Symposium, DISC*, pages 201–214, 2016. 1.2

[15] Avinatan Hassidim, Yishay Mansour, and Shai Vardi. Local computation mechanism design. *ACM Trans. Economics and Comput.*, 4(4):21:1–21:24, 2016. 1

[16] Jonathan Katz and Luca Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 80–86, 2000. 1.2

[17] Tali Kaufman, Michael Krivelevich, and Dana Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM J. Comput.*, 33(6):1441–1483, 2004. 1.2

[18] Reut Levi, Dana Ron, and Ronitt Rubinfeld. A local algorithm for constructing spanners in minor-free graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM*, pages 38:1–38:15, 2016. 1.2

[19] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. Brief announcement: Local computation algorithms for graphs of non-constant degrees. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA*, pages 59–61, 2015. 1, 1.2, 1

[20] Palma London, Niangjun Chen, Shai Vardi, and Adam Wierman. Distributed optimization via local computation algorithms. `http://users.cms.caltech.edu/~plondon/loco.pdf`, 2017. 1.2

[21] Yishay Mansour, Boaz Patt-Shamir, and Shai Vardi. Constant-time local computation algorithms. In *Approximation and Online Algorithms - 13th International Workshop, WAOA*, pages 110–121, 2015. 1

[22] Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. Converting online algorithms to local computation algorithms. In *Proc. 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 653–664, 2012. 1, 1

[23] Yishay Mansour and Shai Vardi. A local computation approximation scheme to maximum matching. In *APPROX-RANDOM*, pages 260–273, 2013. 1.2, 1

[24] Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995. 1.2

[25] David Peleg and Alejandro A. Schffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989. 1.2

[26] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989. 1.2

[27] Omer Reingold and Shai Vardi. New techniques and tighter bounds for local computation algorithms. *J. Comput. Syst. Sci.*, 82(7):1180–1200, 2016. 1, 1

[28] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011. 1, 1.2

[29] Michael E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM J. Comput.*, 39(7):2897–2926, 2010. 1.2

[30] Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.*, 40(6):1913–1926, 2011. 1.2

[31] Mikkel Thorup and Uri Zwick. Approximate distance oracles. In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, (STOC)*, pages 183–192, 2001. 1.2

[32] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *SPAA*, pages 1–10, 2001. 1.2

[33] Christian Wulff-Nilsen. Approximate distance oracles with improved preprocessing time. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 202–208, 2012. 1.2

[34] Sergey Yekhanin. Locally decodable codes. *Foundations and Trends in Theoretical Computer Science*, 6(3):139–255, 2012. 1.2

A General Graphs with Preprocessing

It is easy to extend the result of Section 3 to graphs that are not necessarily connected.

A.1 Strong Probes

If the LCA uses strong probes, Theorem 1.1 holds without modification (except the omission of the word "connected"). It is restated here for completeness.

Theorem A.1. *For any $\alpha > 0$, and any bipartite graph $G = (V, E)$, there exists an exact 2-coloring LCA for G that uses $O(|E| + |V|)$ preprocessing time, $\alpha \log |V|$ words of memory and performs $O\left(\frac{|V|}{\alpha}\right)$ strong probes w.h.p.*

Proof outline. Similarly to the proof of Theorem 1.1, arbitrarily 2-color G in the preprocessing stage. Choose a set $S \subset V$ of $\alpha \log n$ vertices uniformly at random and save their ID and color to memory.

For any queried vertex v, perform BFS until either some vertex $s \in S$ is encountered or until v's entire connected component, C_v, is discovered. If the LCA visits a vertex $s \in S$, v is colored according to the parity of a path between v and s. Otherwise, let the color of the smallest ID in C_v be `blue`, and color v accordingly.

Set $k = O\left(\frac{n}{\alpha}\right)$. If the connected component of v is smaller than k, after k probes, it will be discovered in its entirety, hence v can be colored. If the connected component has not been discovered, then by the same argument as in the proof of Theorem 1.1, k probes will suffice to ensure a vertex of S will be encountered w.h.p. □

A.2 Weak Probes

In order to discover k vertices, we might need to perform $\Omega(k^2)$ weak probes, if the vertices are well connected (consider for example, a clique).

Theorem A.2. *For any $\alpha > 0$, and any bipartite graph $G = (V, E)$, there exists an exact 2-coloring LCA for G that uses $O(|E| + |V|)$ preprocessing time, $\alpha \log |V|$ words of memory and performs $O\left(\left(\frac{|V|}{\alpha}\right)^2\right)$ weak probes w.h.p.*

The proof is virtually identical to that for Theorem A.1 and is omitted.