Local Reconstructors and Tolerant Testers for Connectivity and Diameter

Local reconstructors and tolerant testers for connectivity and diameter

Andrea Campagna* Alan Guo† Ronitt Rubinfeld‡

August 12, 2012

Abstract

A local property reconstructor for a graph property is an algorithm which, given oracle access to the adjacency list of a graph that is "close" to having the property, provides oracle access to the adjacency matrix of a "correction" of the graph, i.e. a graph which has the property and is close to the given graph. For this model, we achieve local property reconstructors for the properties of connectivity and k-connectivity in undirected graphs, and the property of strong connectivity in directed graphs. Along the way, we present a method of transforming a local reconstructor (which acts as a "adjacency matrix oracle" for the corrected graph) into an "adjacency list oracle". This allows us to recursively use our local reconstructor for $(k-1)$-connectivity to obtain a local reconstructor for k-connectivity.

We also extend this notion of local property reconstruction to parametrized graph properties (for instance, having diameter at most D for some parameter D) and require that the corrected graph has the property with parameter close to the original. We obtain a local reconstructor for the low diameter property, where if the original graph is close to having diameter D, then the corrected graph has diameter roughly $2D$.

We also exploit a connection between local property reconstruction and property testing, observed by Brakerski, to obtain new tolerant property testers for all of the aforementioned properties. Except for the one for connectivity, these are the first tolerant property testers for these properties.

*This research was done while at IT University of Copenhagen and while visiting the Blavatnik School of Computer Science of Tel Aviv University. acam@itu.dk.

†CSAIL, Massachusetts Institute of Technology, Cambridge MA 02139. aguo@mit.edu. Research supported in part by NSF grants CCF-0829672, CCF-1065125, CCF-6922462, and an NSF Graduate Research Fellowship.

‡CSAIL, Massachusetts Institute of Technology, Cambridge MA 02139 and the Blavatnik School of Computer Science, Tel Aviv University. ronitt@csail.mit.edu. Research supported by NSF grant 1065125 and the Israel Science Foundation grant no. 1147/09.

1 Introduction

Suppose we are given a very large graph G that is promised to be close to having a property $\mathcal{P}_s$. For example, $\mathcal{P}_D$ might denote the property of having diameter at most D. Local reconstruction algorithms provide very fast query access to a "corrected" version of G. That is, the local reconstruction algorithm should have in mind some $\widetilde{G}$ which has the property $\mathcal{P}_s$ and is also close to the original graph G. The goal of the local reconstruction algorithm is to provide very fast query access to the edges of $\widetilde{G}$ — that is, given a pair of vertices u, v in G, the algorithm should in sublinear time determine whether the edge (u, v) is in $\widetilde{G}$. We call such an algorithm a *local reconstructor* for $\mathcal{P}_s$. It can be useful to relax the condition that $\widetilde{G}$ has property $\mathcal{P}_s$ and only require that $\widetilde{G}$ has property $\mathcal{P}_{\phi(s)}$ which contains $\mathcal{P}_s$ but is possibly larger. For instance, if $\mathcal{P}_D$ is the property of having diameter at most D, we might only require $\widetilde{G}$ to have property $\mathcal{P}_{4D}$, i.e. having diameter at most $4D$.

In this paper we study local reconstruction algorithms for some of the most basic problems in graph theory, namely connectivity in undirected graphs, strong connectivity in directed graphs, k-connectivity in undirected graphs, and small diameter in undirected graphs. Such algorithms might be used to efficiently repair connectivity or small diameter in graphs. These are common issues that have been considered in various models in wireless networks and robotics (see for example [7, 23]).

Techniques for designing local reconstructors are often borrowed from property testing, and as noted by Brakerski [5] (as well as in this paper), reconstructors can be used to design property testers. Property testers have been studied extensively in the literature (see, for instance, [20, 8, 9] for some early works on the subject). A property tester for a property $\mathcal{P}$ takes as input a graph G and parameter ϵ, and accepts with high probability if G has $\mathcal{P}$ and rejects with high probability if G is ϵ-far from having $\mathcal{P}$. Similarly, a tolerant tester gets a graph G and parameters $\epsilon_1 \leq \epsilon_2$, and accepts with high probability if G is ϵ_1-close to having $\mathcal{P}$ and rejects with high probability if G is ϵ_2-far from having $\mathcal{P}$.

Our results. Our specific results are the following.

- We give local reconstructors for the following properties: connectivity in undirected graphs; strong connectivity in directed graphs; k-connectivity in undirected graphs; having diameter at most D, for some diameter parameter D, in undirected graphs.
- We present a method of transforming our local reconstructor for k-connectivity (which provides query access to the adjacency *matrix* of the corrected graph) into an algorithm which provides query access to the adjacency *list* of the corrected graph (see Section 6 for the case $k = 1$). We note that this is *not* a black-box transformation, and is specific to our the local reconstructors that we design.
- We exploit a connection between local property reconstruction and property testing, observed by Brakerski [5], which we generalize to the setting of parametrized graph properties, in order to obtain tolerant property testers for all of the above graph properties (property testing notions will be defined shortly).

Our approach. Our techniques are simple, yet seem to be quite powerful given their simplicity. For each of the above properties, the strategy for constructing a local reconstructor is the same.

First, we designate a "super-node"; then elect "leader nodes" from which we add edges to the super-node. The main technique we use to elect leaders is to initially independently assign a random rank to every node, and to declare a node a leader if it has the lowest rank among all nodes within a small neighborhood. This is more useful than simply choosing leaders at random, since sometimes we would like to guarantee we have a leader; for instance, for connectivity, we need at least one leader in each connected component if we want to *guarantee* that the corrected graph is connected.

Brakerski [5] gave a way to construct a tolerant tester from a local reconstructor and property tester. Brakerski's idea (see [5]) behind the construction of the tolerant tester is to attempt to use the local reconstructor on G to get $\widetilde{G}$ which has the property and is close to G. If G is indeed close to having the property, then the attempt will be successful. If G is far from having the property, then either $\widetilde{G}$ will be far from having the property (which can be detected by running the property tester on $\widetilde{G}$) or G will be far from $\widetilde{G}$ (which can be detected via sampling). Since previous works ([9], [3], [13], [18]) give property testers for these properties we study, we obtain, as corollaries, that these properties have tolerant testers.

Related work. The notion of locally reconstructing a data set was introduced under the name *local filter* in [1], where the property considered was monotonicity of sequences. A model of local filters in which the requirements are strengthened is presented in [22], where the property of monotonicity is again considered. A closely related work is [5], which introduces the concept of local reconstruction under the name *local restoring* and also shows a special case of our relationship between local reconstructors, property testers, and tolerant testers. Several other properties of graphs, functions, and geometric point sets have been studied in the context of local reconstruction. A local reconstruction algorithm for expander graphs is given in [11]. [5] gives local reconstructors for the properties of bipartiteness and ρ-clique in the dense graph model, as well as monotonicity. A more recent work ([10]) concerns locally reconstructing Lipschitz functions. On the geometric side, [6] studies local reconstruction of convexity in two and three dimensions. The problem of testing the properties for which we give local reconstructors has been studied in multiple works—[9] for connectivity, [3] for strong connectivity, [9] and [13] for k-connectivity, and [18] for diameter.

Organization. In Section 2, we formally define our model and the notions of local reconstructors, property testers, and tolerant testers, and in Section 3 we formally state our main results. In Section 4 and 5, we present our results for connectivity and strong connectivity. Section 6 serves as a brief interlude where we show how our local reconstructor for connectivity can be modified to give a neighbor oracle for the corrected graph G'. This procedure will then be used in Sections 7 and 8, in which we present our results for k-connectivity and small diameter respectively.

2 Preliminaries

We adopt the general sparse model of graphs as presented in [18], i.e. the graph is given as an adjacency list, and a query for a vertex v is either its degree, or an index i on which the i-th neighbor of v is returned (with respect to the representation of the neighbors as an ordered list). We assume there is some upper bound m on the number of edges of the graphs we work with, and distances are measured according to this, i.e. if k is the minimum number of edge deletions and insertions necessary to change one graph to the other, then their distance is k/m. We assume $m = \Omega(n)$ where n is the number of vertices in the graph.

Definition 2.1 ([18])**.** The *distance between two graphs* G_1, G_2, denoted $\text{dist}(G_1, G_2)$, is equal to the number of unordered pairs (u, v) such that (u, v) is an edge in one graph but not in the other, divided by m. A *property* is a subset of graphs. Throughout this paper we say that a graph has property $\mathcal{P}$ if it is contained in the subset $\mathcal{P}$. The *distance between a graph* G *and a property* $\mathcal{P}$, denoted $\text{dist}(G, \mathcal{P})$, is equal to $\text{dist}(G, \mathcal{P}) = \min_{G' \in \mathcal{P}} \text{dist}(G, G')$. If $\text{dist}(G, \mathcal{P}) \leq \epsilon$, then G is ϵ*-close to* $\mathcal{P}$, otherwise G is ϵ*-far from* $\mathcal{P}$.

Parametrized properties. Our result relating local reconstructors to tolerant testers (Theorem 3.1) generalizes a result of Brakerski ([5]) to parametrized properties. A *parametrized property* $\mathcal{P}_s$ is a property belonging to a family $\{\mathcal{P}_s\}_s$ of properties parametrized by some parameter s. For example, the property $\mathcal{P}_D$ of having diameter at most D is a parametrized property, with the diameter D as the parameter.

2.1 Local reconstructors

Definition 2.2. For an undirected graph $G = (V, E)$, the *neighbor set of* $v \in V$ is the set $N_G(v) = \{u \in V \mid (u, v) \in E\}$. For a directed graph $G = (V, E)$, the *in-neighbor set and out-neighbor set of* $v \in V$ are respectively $N_G^{\text{in}}(v) = \{u \in V \mid (u, v) \in E\}$ and $N_G^{\text{out}}(v) = \{u \in V \mid (v, u) \in E\}$

Definition 2.3 (Neighbor and edge oracles)**.** A *neighbor oracle* for a graph G is an algorithm which, given query $v \in V$ and either query deg or i, returns $\deg(v)$ or the i-th neighbor of v (with respect to some fixed ordering of the neighbor set) in $O(1)$ time. An *edge oracle for* G is an algorithm E_G which returns in $O(1)$ time $E_G(u, v) = 1$ if $(u, v) \in E$ and $E_G(u, v) = 0$ otherwise.

One can use a neighbor oracle to implement an edge oracle with query complexity $\deg(v)$, since given a query pair (u, v), one can check if $u \in N_G(v)$ with $\deg(v)$ queries to the neighbor oracle. We now formally define local reconstructors. Roughly speaking, a local reconstructor uses a neighbor oracle for G, which is close to $\mathcal{P}$, to implement an edge oracle for $\widetilde{G} \in \mathcal{P}$ which is close to G.

Definition 2.4. Let $\epsilon_1, \epsilon_2, \delta > 0$ and let $\phi : \mathbb{N} \to \mathbb{N}$. An $(\epsilon_1, \epsilon_2, \delta, \phi(\cdot))$*-local reconstructor (LR)* $\mathcal{R}$ for a parametrized graph property $\mathcal{P}_s$ is a randomized algorithm with access to a neighbor oracle of a graph G that is ϵ_1-close to $\mathcal{P}_s$, which satisfies the following:

- $\mathcal{R}$ makes $o(m)$ queries to the neighbor oracle for G per query to $\mathcal{R}$
- There exists $\widetilde{G} \in \mathcal{P}_{\phi(s)}$ with $\text{dist}(G, \widetilde{G}) \leq \epsilon_2$ such that $\mathcal{R}$ is an edge oracle for $\widetilde{G}$, with probability at least $1 - \delta$ (over the coin tosses of $\mathcal{R}$)

An $(\epsilon_1, \epsilon_2, \delta)$*-local reconstructor* for a non-parametrized graph property $\mathcal{P}$ is simply a $(\epsilon_1, \epsilon_2, \delta, \phi(\cdot))$-local reconstructor where $\mathcal{P}$ is viewed as the only property in its parametrized family and ϕ is the identity function. The *query complexity* of the local reconstructor is the number of queries $\mathcal{R}$ makes to the neighbor oracle for G on any query (u, v). We note that this definition differs from that of [10] because even if $G \in \mathcal{P}$, the reconstructed graph $\widetilde{G}$ may not equal G in general.

2.2 Tolerant testers

Tolerant testers (see [19]) are a generalization of property testers where the tester may accept if the input is close enough to having the property, where for property testers "close enough" means "distance zero".

Definition 2.5. Let $\epsilon_1, \epsilon_2 > 0$ and let $\phi : \mathbb{N} \to \mathbb{N}$. An $(\epsilon_1, \epsilon_2, \phi(\cdot))$-*tolerant tester* $\mathcal{T}$ for a parametrized graph property $\mathcal{P}_s$ is a randomized algorithm with query access to a neighbor oracle of an input graph G that satisfies the following:

- $\mathcal{T}$ makes $o(m)$ queries to the neighbor oracle for G
- If G is ϵ_1-close to $\mathcal{P}_s$, then $\Pr[\mathcal{T} \text{ accepts }] \geq \frac{2}{3}$
- If G is ϵ_2-far from $\mathcal{P}_{\phi(s)}$, then $\Pr[\mathcal{T} \text{ accepts }] \leq \frac{1}{3}$

For a non-parametrized graph property $\mathcal{P}$, an (ϵ_1, ϵ_2)-*tolerant* tester is defined similarly by viewing $\mathcal{P}$ as the single member of its parametrized family and taking ϕ to be the identity function.

For a parametrized graph property, an $(\epsilon, \phi(\cdot))$-*property tester* is simply a $(0, \epsilon, \phi(\cdot))$-tolerant tester, and for a non-parametrized graph property, an ϵ-*property tester* is defined analogously.

3 Local reconstructors and tolerant testers

We now show that our notion of local reconstructors can be used alongside property testers to construct tolerant testers for properties of sparse graphs. This idea is not new and can be found as [5, Theorem 3.1], but we extend the result for parametrized properties.

Theorem 3.1. *Let $\mathcal{P}_s$ be a parametrized graph property with an $(\epsilon_1, \epsilon_2, \delta, \phi(\cdot))$-local reconstructor $\mathcal{R}$ with query complexity $q_{\mathcal{R}}$ and suppose $\mathcal{P}_{\phi(s)}$ has a $(\epsilon', \psi(\cdot))$-property tester $\mathcal{T}$ with query complexity $q_{\mathcal{T}}$. Then for all $\beta > 0$, $\mathcal{P}_s$ has an $(\epsilon_1, \epsilon_2 + \epsilon' + \beta, (\psi \circ \phi)(\cdot))$-tolerant tester with query complexity $O\left((1/\beta^2 + q_{\mathcal{T}})q_{\mathcal{R}}\right)$.*

Proof. The algorithm and proof follows that of [5, Theorem 3.1].

The tolerant tester for $\mathcal{P}_s$ is as follows:

1. Run $\mathcal{R}$ on G and estimate $\text{dist}(G, \widetilde{G})$ to within additive error of $\beta/2$ by sampling $(u, v) \in V \times V$.
2. If estimate of $\text{dist}(G, \widetilde{G})$ exceeds $\epsilon_2 + \beta/2$, **reject**.
3. Run $\mathcal{T}$ on $\widetilde{G}$ using $\mathcal{R}$, and **accept** if and only if $\mathcal{T}$ accepts.

If G is ϵ_1-close to $\mathcal{P}_s$, then with high probability $\text{dist}(G, \widetilde{G}) \leq \epsilon_2$. Therefore, the algorithm passes step 2 and with high probability $\mathcal{T}$ accepts $\widetilde{G}$, since $\widetilde{G} \in \mathcal{P}_{\phi(s)}$. If G is $(\epsilon_2 + \epsilon' + \beta)$-far from $\mathcal{P}_{\psi(\phi(s))}$, then either $\text{dist}(G, \widetilde{G}) > \epsilon_2 + \beta/2$, in which case step 2 fails with (constant) high probability, or $\text{dist}(G, \widetilde{G}) \leq \epsilon_2 + \beta$ in which case $\widetilde{G}$ is ϵ'-far from $\mathcal{P}_{\psi(\phi(s))}$ and so $\mathcal{T}$ rejects with high probability. □

Taking ϕ and ψ to be identity, one gets as a special case the result of [5, Theorem 3.1] that if $\mathcal{P}$ is a graph property with an $(\epsilon_1, \epsilon_2, \delta)$-local reconstructor and an ϵ'-property tester, then for all $\beta > 0$ it has an $(\epsilon_1, \epsilon_2 + \epsilon' + \beta)$-tolerant tester.

In this work, we give local reconstructors for several graph properties: connectivity in undirected graphs, strong connectivity in directed graphs, and small diameter in undirected graphs. To be precise, we prove the following in Sections 4, 5, 7 and 8 respectively.

Theorem 3.2. *There is an $(\epsilon, (1+\alpha)\epsilon, \delta)$-LR for connectivity with query complexity $O\left(\frac{1}{\delta\alpha\epsilon}\right)$.*

Theorem 3.3. *There is an* $(\epsilon, (4+\alpha)\epsilon, \delta)$*-LR for strong connectivity with query complexity* $O\left(\frac{1}{\delta\alpha\epsilon}\right)$.

Theorem 3.4. *There is an* $(\epsilon, (2+\alpha)\epsilon + ck/2, k(\delta+\gamma))$*-LR for* k*-connectivity with query complexity* $O\left(\left(\left(\frac{1}{c}+1\right)k\right)^k t^{3k}(t+k)^k \log^k(t+k)\log^k(Cn)\right)$ *for* $n \geq \frac{k}{c}$*, where* $C = \frac{1}{\ln(1/(1-\gamma))}$ *and* $t = \frac{\ln(Cn)}{\delta\alpha\epsilon}$.

Theorem 3.5. *There is an* $(\epsilon, (3+\alpha)\epsilon + \frac{1}{m} + c, \delta + \frac{1}{n}, \phi(s) = 2s+2)$*-LR for diameter at most* D *with query complexity* $O(\frac{1}{c\delta\alpha\epsilon}\Delta^{O(\Delta \log \Delta)} \log n)$ *where* $\Delta = (\overline{d}/\epsilon)^{O(1/\epsilon)}$ *and* $\overline{d} = 2m/n$ *is the bound on average degree.*

Combining Theorem 3.1 with each of Theorems 3.2, 3.3, 3.4, and 3.5, along with property testers for each of the four properties (see [9], [3], [13], and [18]), we immediately obtain the following tolerant testers.

Corollary 3.6. *For all* $\alpha, \beta, \epsilon > 0$*, there is an* $(\epsilon, (1+\alpha)\epsilon + \beta)$*-tolerant tester for connectivity.*

Corollary 3.7. *For all* $\alpha, \beta, \epsilon > 0$*, there is an* $(\epsilon, (4+\alpha)\epsilon + \beta)$*-tolerant tester for strong connectivity.*

Corollary 3.8. *For all* $\alpha, \beta, c, \epsilon > 0$*, there is an* $(\epsilon, (2k+\alpha)\epsilon + ck/2 + \beta)$*-tolerant tester for* k*-connectivity.*

Corollary 3.9. *For all* $D, \alpha, \beta, \epsilon > 0$ *and constant* $c < 1$*, there is an* $(\epsilon, (3+\alpha)\epsilon + \frac{1}{m} + c + \beta, 4D+6)$*-tolerant tester for diameter at most* D*, for* $n \geq \frac{k}{c}$.

4 Local reconstruction of connectivity

In this section we prove Theorem 3.2. We begin by giving the high level description of the algorithm and then we present the implementation and analysis of the algorithm.

4.1 High level description

The basic idea behind the algorithm is as follows. We designate a "super-node" v_0 and add edges from a few special vertices to v_0 so that the resulting graph is connected. Ideally, we have exactly one special vertex in each connected component, since this number of edges is both necessary and sufficient to make the graph connected. Therefore, we reduce the problem to defining a notion of "special" that can be determined quickly, that ensures that at least one node per component is special and that likely not too many extra nodes per component are special. How does a given vertex know whether it is special? Our algorithm tosses coins to randomly assign a rank $r(v) \in (0, 1]$ to each $v \in V(G)$. Then v can explore its connected component by performing a breadth-first search (BFS) and if v happens to have the lowest rank among all vertices encountered, then v is special. The only problem with this approach is that if v lies in a large connected component, then the algorithm makes too many queries to G to determine whether v is special. We fix this by limiting the BFS to K vertices, where K is a constant depending only on a few parameters, such as success probability and closeness. We then show that components larger than size K do not contribute many more special vertices.

4.2 Algorithm

We now give the algorithm. We first do some preprocessing. In particular, we arbitrarily fix some vertex $v_0 \in V(G)$. Additionally, we have a random oracle which, for each vertex $v \in V(G)$, it assigns a random number $r(v) \in (0, 1]$.

```
procedure CONNECTED(v_1, v_2)
    if E_G(v_1, v_2) = 1 then                    ▷ if edge is already in graph, it stays in the graph
        return 1
    else
        if v_0 ∉ {v_1, v_2} then                 ▷ do not add edge if neither endpoint is v_0
            return 0
        else
            Let v ∈ {v_1, v_2} \ {v_0}
            BFS from v up to K vertices and let U be the set of vertices visited
            if r(v) < r(u) for all u ∈ U then    ▷ check if v is special
                return 1
            else
                return 0
            end if
        end if
    end if
end procedure
```

The following lemma shows that the procedure does not add too many extra edges. To prove it, we use the fact that if G has at least $\epsilon m + 2$ connected components, then G is ϵ-far from being connected.

Lemma 4.1. *With probability at least $1 - \delta$,* CONNECTED *adds at most $n/\delta K + \epsilon m + 1$ edges.*

Proof. Let X be the number of edges added by the local reconstructor. Call a connected component C *small* if $|C| < K$ and *large* otherwise. Let Y be the number of edges contributed by large components. Each small component contributes exactly one edge, hence $X = i + Y$ where i is the number of small components. Since G is ϵ-close to being connected, $i \leq \epsilon m + 1$. Moreover, each vertex in a large component contributes an edge with probability at most $1/K$. By Markov's inequality,

$$\Pr\left[Y > \frac{n}{\delta K}\right] \leq \Pr\left[Y > \frac{E[Y]}{\delta}\right] \leq \delta.$$

hence

$$\Pr\left[X > \frac{n}{\delta K} + \epsilon m + 1\right] \leq \Pr\left[Y > \frac{n}{\delta K}\right] \leq \delta.$$

□

Proof of Theorem 3.2. Let $\mathcal{P}$ be the family of connected graphs on n vertices. The local reconstructor $\mathcal{R}$ will run CONNECTED with $K = \frac{m}{\delta\alpha\epsilon m - 1} = O\left(\frac{1}{\delta\alpha\epsilon}\right)$. Clearly $\mathcal{R}$ has query complexity $O(K) = O\left(\frac{1}{\delta\alpha\epsilon}\right)$. To see that $\widetilde{G}$ is connected, observe that, on each connected component, the

rank function attains a minimum on some vertex, and that vertex therefore must get an edge to v_0. Furthermore, if G is ϵ-close to $\mathcal{P}$, then by Lemma 4.1 with probability at least $1-\delta$ the procedure will add no more than $(1+\alpha)\epsilon m$ edges and hence $\text{dist}(G, \widetilde{G}) \leq (1+\alpha)\epsilon$. □

5 Local reconstruction of strong connectivity

In this section we prove Theorem 3.3. We first go over some preliminary definitions and properties of directed graphs. Then we give the high level description of the algorithm. Finally we end by presenting the implementation and analysis of the algorithm.

5.1 Preliminaries

Throughout this section, we will use *arc* to mean *directed edge*.

Definition 5.1. A directed graph G is *connected* if it is connected when viewing arcs as undirected edges, and it is *strongly connected* if there is a path between every ordered pair of vertices. A *connected component* of G is a maximal connected subgraph of G, and a *strongly connected component* is a maximal strongly connected subgraph of G.

Definition 5.2. A vertex is a *source* (*sink*) if it has no incoming (outgoing) arcs. A strongly connected component with no incoming (outgoing) arcs is a *source (sink) component*.

Query model. In our model, we assume our neighbor oracle has access to both the in-neighbor set and the out-neighbor set. This allows us to perform both backward and forward depth-first search (DFS), as well as undirected BFS, which is a BFS ignoring directions of edges.

5.2 High level description

The basic idea behind the algorithm is as follows. As in the undirected connectivity case, we designate a "super-node" v_0, but now we add arcs from a few special "transmitting" vertices to v_0 and also add arcs from v_0 to a few special "receiving" vertices. In order to make G strongly connected, we need to add at least one arc from the super-node to each source component and from each sink component to the super-node without adding too many extra arcs. A naïve approach is to emulate the strategy for connectivity: to decide if v is a transmitter, do a forward DFS from v and check if v has minimal rank (and analogously for receivers and backward DFS). Again, we can limit the search so that large components may have some extra special vertices. The problem with this approach is that a sink component could be extremely small (e.g. one vertex) with many vertices whose only outgoing arcs lead to the sink. In this case, all of these vertices would be special and receive an edge to v_0. Therefore we tweak our algorithm so that if v does a forward DFS and sees few vertices, then it checks if it is actually in a sink component. If so, then it is a transmitter; if not, then we do a limited *undirected BFS* from v and check minimality of rank. We then show that we do not add too many extra edges this way.

5.3 Algorithm

We now present the algorithm. We do the same preprocessing as in CONNECTED, i.e. we arbitrarily fix a vertex $v_0 \in V(G)$ and have access to a random oracle that randomly assigns ranks $r(v) \in (0, 1]$ to each $v \in V(G)$.

```
procedure StronglyConnected(v_1, v_2)
    if E_G(v_1, v_2) = 1 then                ▷ if edge is already in graph, it stays in the graph
        return 1
    else
        if v_0 ∉ {v_1, v_2} then             ▷ do not add arc if neither endpoint is v_0
            return 0
        else if v_2 = v_0 then               ▷ check if v_1 is a transmitter
            Forward DFS from v_1 up to K vertices
            if Forward DFS sees at least K vertices or InSmallSink(v_1) then
                return 1 if v_1 has lowest rank among the DFS vertices else return 0
            else
                Undirected BFS up to K vertices
                return 1 if v_1 has lowest rank among the BFS vertices else return 0
            end if
        else if v_1 = v_0 then               ▷ check if v_2 is a receiver
            Backward DFS from v_2 up to K vertices
            if Backward DFS sees at least K vertices or InSmallSource(v_2) then
                return 1 if v_2 has lowest rank among the DFS vertices, else return 0
            else
                Undirected BFS up to K vertices
                return 1 if v_2 has lowest rank among the BFS vertices, else return 0
            end if
        end if
    end if
end procedure
```

The procedure uses two subroutines: InSmallSource(v) and InSmallSink(v), which return True if v is in a source (respectively sink) component of size less than K. We will implement InSmallSink (InSmallSource is similar except reverse all the directions of edges) by running Tarjan's algorithm for finding strongly connected components, except stopping after only exploring all nodes reachable from v. If only one strongly connected component is returned, then return True, otherwise return False. This clearly runs in $O(K)$ time since Tarjan's algorithm runs in linear time and we are simply restricting the algorithm to the subgraph induced by all strongly connected components reachable from v.

The following three lemmas capture the fact that if a directed graph is almost strongly connected, then it cannot have too many source, sink, or connected components, and therefore our algorithm likely does not add too many arcs.

Lemma 5.3. *If G is ϵ-close to being strongly connected, then G has at most ϵm source components and at most ϵm sink components.*

Proof. Consider the directed graph $\widehat{G}$ of strongly connected components of G. To make G strongly connected, $\widehat{G}$ must have no sources or sinks, yet adding an arc eliminates at most one source and at most one sink from $\widehat{G}$. □

Lemma 5.4. *If G is ϵ-close to being strongly connected, then G has at most $\epsilon m + 1$ connected components.*

Proof. If G is ϵ-close to being strongly connected, then it is also ϵ-close to being connected. □

Lemma 5.5. *With probability at least $1-\delta$, the procedure will add no more than $\frac{2n}{\delta K} + 4\epsilon n + 2$ arcs.*

Proof. The analysis is similar to that of Lemma 5.5, except slightly more complicated. Let X be the random variable equal to the number of arcs added. Let S be the random variable equal to the number of arcs added that end at v_0, and let T be the random variable equal to the number of arcs added that start at v_0. We will focus on S, since the analysis for T is symmetrical. Any vertex v for which InSmallSink(v) is True lies in a sink component of size less than K; call these components small sink components. Also, any vertex that does an undirected BFS and sees less than K vertices must belong to a connected component of size less than K; call these components small connected components. Let S' be the number of arcs counted by S contributed by vertices not in small sink components or small connected components. Then $S \leq S' + 2\epsilon m + 1$ since each small sink component (of which there are at most ϵm by Corollary 5.3) contributes at most 1 outgoing arc and each small connected component (of which there are at most $\epsilon m + 1$ by Lemma 5.4) contributes at most 1 outgoing arc. Define T' analogously to S', except for source components instead of sink components, so that $T \leq T' + 2\epsilon m + 1$. Then we have $E[S'] \leq \frac{n}{K}$ and $E[T'] \leq \frac{n}{K}$ and therefore

$$\begin{aligned}
\Pr\left[X > \frac{2n}{\delta K} + 4\epsilon m + 2\right] &\leq \Pr\left[S' + T' > \frac{2n}{\delta K}\right] \\
&\leq \Pr\left[S' + T' > \frac{E[S' + T']}{\delta}\right] \\
&\leq \delta
\end{aligned}$$

where the final inequality follows from Markov's inequality. □

Proof of Theorem 3.3. The local reconstructor $\mathcal{R}$ runs StronglyConnected with $K = \frac{m}{\delta\alpha\epsilon m/2 - 1}$. To see that $\widetilde{G}$ is strongly connected, note that every source component has some vertex of minimal rank, and this vertex will have the lowest rank among its backward K-neighborhood, hence every source component gets an arc from v_0. By a similar argument, every sink component gets an arc to v_0. Now we must show that with high probability we did not add too many edges. By Lemma 5.5, with probability at least $1-\delta$ we added at most $(4+\alpha)\epsilon m$ edges and hence $\text{dist}(G, \widetilde{G}) \leq (4+\alpha)\epsilon$. □

6 Implementing a neighbor oracle with connectivity reconstructor

Our local reconstructors for k-connectivity and small diameter rely on the given graph G being connected. Even if G is not connected, it is close to being connected, and so one may hope to first make G into an intermediate connected graph G' using a local reconstructor for connectivity, and then run the local reconstructor for the desired property on G' to obtain $\widetilde{G}$. We would therefore like a neighbor oracle for G', but the local reconstructor only gives us an edge oracle for G'. We show how to modify the local reconstructor for connectivity to obtain a neighbor oracle for G', with only a slight loss in the parameters achieved.

As is, our connectivity reconstructor Connected from Section 4 is *almost* a neighbor oracle. Recall that the reconstructor works by selecting an arbitrary $v_0 \in G$ and adding edges from a

few special vertices to v_0—no other edges are added. For a vertex $v \neq v_0$, $N_G(v) \subseteq N_{G'}(v) \subseteq N_G(v) \cup \{v_0\}$, thus $N_{G'}(v)$ can be computed in constant time. However, the problem arises when one queries $N_{G'}(v_0)$. Potentially $\Theta(n)$ edges are added to v_0 by the reconstructor, so computing $N_{G'}(v)$ queries CONNECTED $O(n)$ times. We thus modify CONNECTED to obtain the following.

Theorem 6.1. *Fix a positive constant $c < 1$. There is a randomized algorithm N, given access to the neighbor oracle N_G for G that is ϵ-close to being connected such that, with probability at least $1-\delta$, there exists a connected graph G' that is $((1+\alpha)\epsilon + c)$-close to G and N is a neighbor oracle for G', with query complexity $O\left(\frac{1}{c\delta\alpha\epsilon}\right)$.*

Proof. We modify CONNECTED as follows. Instead of designating one super-node v_0, we designate $c \cdot n$ super-nodes. Partition V into sets of size $1/c$ and assign each set in the partition to a distinct super-node. This can be implemented, for instance, by identifying $V = \{1, \ldots, n\}$, designating the super-nodes to be $\{1, \ldots, cn\}$, and for a given vertex $v \in V$, assign v to the super-node $h(v) = \lceil v/c \rceil$. For any v that would be connected to v_0, we instead connect it to $h(v)$. Additionally, we add the edges $(i, i+1)$ for all $i \in \{1, \ldots, cn-1\}$ to ensure connectivity. This adds a total of $c-1$ edges, which constitute at most c-fraction of the edges. Call this modified local reconstructor MOD-CONNECTED. It is straightforward to see that MOD-CONNECTED has the same query complexity as CONNECTED. We now implement an algorithm to compute $N_{G'}$ as follows. Given a non-super-node v, its neighbor set could have grown by at most adding $h(v)$, so $N_{G'}(v)$ can be computed with $O(1)$ calls to MOD-CONNECTED. For a super node w, its neighbor set could have grown by at most $1/c+2$, since at most $1/c$ non-super-nodes could have been connected to w, and w is further connected to at most two super-nodes. Therefore $N_{G'}(w)$ can be computed with $O(1/c)$ calls to MOD-CONNECTED. □

7 Local reconstruction of k-connectivity

In this section we prove Theorem 3.4. We first go over some preliminary definitions and concepts related to k-connectivity. Then we give the high level description of the algorithm, and finally we end by presenting the implementation of the algorithm. All graphs in this section are undirected unless otherwise specified. Throughout this section, we assume $k > 1$.

7.1 Preliminaries

For a subset $U \subsetneq V$ of the vertices in a graph $G = (V, E)$, the *degree of* U, denoted $\deg(U)$, is equal to $\deg(U) = |\{(u, v) \in E \mid u \in U, v \in V \setminus U\}|$.

Definition 7.1 (k-connectivity)**.** An undirected graph $G = (V, E)$ is *k-connected* if for every $U \subsetneq V$, $\deg(U) \geq k$.

An equivalent definition of k-connectivity, a result of Menger's theorem (see [14]), is that every pair of vertices has at least k edge-disjoint paths connecting them. An important notion in the context of k-connectivity is that of an extreme set. Extreme sets are a generalization of connected components to the k-connectivity setting. Connected components are 0-extreme sets.

Definition 7.2. A set $U \subseteq V$ is ℓ-extreme if $\deg(U) = \ell$ and $\deg(W) > \ell$ for every $W \subsetneq U$.

It is straightforward from the definition that if a graph is $(k-1)$-connected and has no $(k-1)$-extreme sets, then it is in fact k-connected. Extreme sets satisfy some nice properties, which

are used by [13] as well for property testing and distance approximation for k-connectivity. Two extreme sets are either disjoint or one is contained in the other (see [16]. If $W \subsetneq U$ and W is ℓ_W-extreme and U is ℓ_U-extreme, then $\ell_W > \ell_U$. Consequently, distinct ℓ-extreme sets are disjoint.

One may hope there is some relationship between distance from k-connectivity and the number of extreme sets, analogous to the relationship between distance from connectivity and the number connected components. Indeed, for a $(k-1)$-connected graph, there is. A graph G that is $(k-1)$-connected cannot have any ℓ-extreme sets for $\ell < k-1$. Moreover, the number of additional edges required to make G k-connected is at least half the number of $(k-1)$-extreme sets in G. This is simply because each $(k-1)$-extreme set requires at least one additional edge, and adding an edge to G meets the demand of at most two such sets.

7.2 High level description

The idea behind the algorithm is to simply iteratively make the graph j-connected, for $j = 1, 2, \ldots, k$. Let G_j be the corrected j-connected graph obtained from G. It suffices to use a neighbor oracle for G_{k-1} to implement a neighbor oracle for G_k. The base case $k = 1$ is addressed by Section 6.

Now suppose we have a neighbor oracle for $(k-1)$-connectivity and we wish to implement a neighbor oracle for k-connectivity. Again, we use a similar idea as in Sections 4 and 6. Specifically, we fix a positive constant $k/n \leq c < 1$ and designate a set $V_0 \subset V$ of $c \cdot n \geq k$ super-nodes, connecting them in a certain way to make the subgraph induced by V_0 k-connected (details in the next subsection). The idea is then to ensure at least one vertex from each extreme set contributes a new edge to a super-node. Again, we implement this by assigning all vertices a random rank independently and uniformly in $[0, 1)$ and searching a neighborhood of v up to t vertices, where t is appropriately chosen, and checking if it has minimal rank. Instead of doing this search via BFS, we use the extreme set search algorithm of [9]. The basic procedure satisfies the following: if v lies in a t-bounded extreme set, it finds this set with probability $\Theta(t^{-2})$, otherwise it never succeeds. We iterate the basic procedure a polylogarithmic number of times. If every iteration fails (which happens if v does not lie in a t-bounded extreme set) then we tell v to connect to a super-node with probability $\Theta\left(\frac{\log(n/t)}{t}\right)$. We then show that with high probability the resulting graph G_k is k-connected and that we do not add too many edges. This completes the edge oracle. We will also show how implement these ideas carefully so that the edge oracle can be transformed into a neighbor oracle in order to make the recursion work.

7.3 Algorithm

Given our previous discussion, all that remains to implement the algorithm is to implement the following tasks:

- **Search:** Given $v \in V$ which lies in an extreme set S, find a neighborhood $U \subseteq S$ containing v such that $|U| \leq t$.
- **Decision:** Given $v \in V$, determine whether v should contribute an edge to V_0, and if so, to which $v' \in V_0$.

7.3.1 Implementing the search task

The goal of the search task is to detect that v lies in an extreme set of size at most t. We are now assuming the input graph is $(k-1)$-connected, so all extreme sets are $(k-1)$-extreme. The search task can be implemented by a method of [9] and [13] which runs in time $O(t^3 d \log(td))$ where d is a degree bound on the graph. Roughly, the procedure works by growing a set U' starting with $\{v\}$ and iteratively choosing a cut edge and adding the vertex on the other end of the edge into U'. The cut edge is chosen by assigning random weights to the edges, and choosing the edge with minimal weight. The procedure stops when the cut size is less than k or when $|U'| = t$. The pseudocode for the extreme set search is as follows.

procedure ExtremeSetSearch$(v, k-1)$
  For each edge, independently assign a random weight uniformly from $[0, 1)$
  $U' \leftarrow \{v\}$
  repeat
    $(u, w) \leftarrow \arg\min\{\mathrm{wt}(u_1, u_2) \mid u_1 \in U', u_2 \notin U', (u_1, u_2) \in E(G_{k-1})\}$ ▷ Implicitly uses $(k-1)$-Conn
    $U' \leftarrow U' \cup \{w\}$
  until $|U'| = t$ or $\deg(U') < k$
end procedure

Its running time is $O(td \log(td))$ but its success probability is only $\Omega(t^{-2})$ (that is, the probability that $U' = S$ when the procedure terminates), so the basic procedure is repeated $\Theta(t^2)$ times or until success. We can check if each run is successful by checking that the final set U' is a $(k-1)$-extreme set. This procedure is adapted in [18] to the general sparse model by noticing that no vertex of degree at least $t + k$ would ever be added to S, and hence the procedure has a running time of $O(t^3(t+k)\log(t+k))$. We actually want the probability that all vertices have a successful search to be at least $1 - \gamma/2$, so we repeat the basic procedure $O(t^2 \log(Cn))$ times for $C = \frac{1}{\ln(1/(1-\gamma/2))}$, yielding a time complexity of $O(t^3(t+k)\log(t+k)\log(Cn))$.

7.3.2 Implementing the decision task

For the decision task, it is helpful to first think about how to do it globally. First, we must hash each $v \in V$ to a set $h(v)$ of k super-nodes in V_0. This can be implemented as follows. Label $V = \{1, \ldots, n\}$ and $V_0 = \{1, \ldots, cn\}$, and define $h(v) = \{\lceil v/c \rceil, \lceil v/c \rceil + 1, \ldots, \lceil v/c \rceil + (k-1)\}$ where the entries are taken modulo cn. The specific way we do this hashing is not important—it suffices to guarantee the following properties: (1) for every $v \in V$, $|h(v)| \geq k$ and (2) for every super-node $v' \in V_0$, there are at most a constant number, independent of n, of v such that $v' \in h(v)$, and that these v are easily computable given v'; our method guarantees the constant $\frac{k}{c}$, which is the best one can hope for given Property 1. Property 1 will be used later to ensure that the resulting graph is k-connected (Lemma 7.4). Property 2 ensures, by the same reasoning as in Section 6, that computing N_{G_k} makes at most $O\left(\frac{k}{c} + k\right)$ calls to E_{G_k}.

Now, to decide whether v should be connected to a super-node, do an extreme set search to find a neighborhood U, of size at most t, containing v, and check if v has minimal rank in U, where the rank is the randomly assigned rank given in the high level description. If so, then mark

v as *successful*. If the extreme set search fails, i.e. all $\Theta(t^2 \log n)$ iterations fail, then mark v as successful with probability $\frac{\ln(Cn/t)}{t}$ where $C = \frac{1}{\ln(1/(1-\gamma/2))}$ again, which can be implemented by checking if the rank of v is less than $\frac{\ln(Cn/t)}{t}$. If v is successful, then find the lexicographically smallest super-node $v' \in h(v)$ to which v is not already connected. If none exist, then do nothing; otherwise, connect v to v'.

We also want the subgraph induced by V_0 to be k-connected to help ensure that G_k is k-connected (Lemma 7.4). Globally, from each $i \in V_0$ we add an edge to $i+1, i+2, \ldots, i+\lceil k/2 \rceil$, taken modulo cn. This ensures that the subgraph induced by V_0 is k-connected (Lemma 7.3). Locally, this is implemented as follows. If $(i,j) \in [cn]^2$ is queried, if the edge is already in G_{k-1}, then the local reconstructor returns 1, otherwise it returns 1 if and only if $j \in \{i+1, i+2, \ldots, i+\lceil k/2 \rceil \pmod{cn}\}$ or $i \in \{j+1, j+2, \ldots, j+\lceil k/2 \rceil \pmod{cn}\}$.

7.3.3 Pseudocode for k-connectivity given $(k-1)$-connectivity

The pseudocode for the k-connectivity reconstructor is given below. It assumes the input graph G_{k-1} is $(k-1)$-connected.

```
procedure k-Conn(v_1, v_2)
    if (k-1)-Conn(v_1, v_2) = 1 then        ▷ if edge is already in graph, it stays in the graph
        return 1
    else
        if v_1 ∉ V_0 and v_2 ∉ V_0 then        ▷ do not add edge if neither endpoint is in V_0
            return 0
        else if v_1, v_2 ∈ V_0 then
            if v_1 ∈ {v_2 ± 1, ..., v_2 ± ⌈k/2⌉ (mod cn)} then
                return 1
            else
                return 0
            end if
        else
            Let v ∈ {v_1, v_2} \ V_0
            Let v' ∈ {v_1, v_2} \ {v}        ▷ Decide if v should be connected to v' ∈ V_0
            b ← FALSE                        ▷ connect v if b is true
            for i = 1, ..., t^2 log(Cn) do
                if ExtremeSetSearch(v, k-1) succeeds then
                    U ← ExtremeSetSearch(v, k-1)
                    if v has minimal rank in U then
                        b ← TRUE
                    end if
                    Break
                end if
            end for
            if b is false and r(v) < ln(Cn/t)/t then
                b ← TRUE
```

```
            end if
            if b is true and v' is lexicographically smallest in h(v) \ N_{G_{k-1}}(v) then
                return 1
            else
                return 0
            end if
        end if
    end if
end procedure
```

7.3.4 Analysis

The following lemma ensures the subgraph induced by V_0 after adding these edges is k-connected.

Lemma 7.3. *For $n \geq 2k+1$, let $V = \{0, 1, \ldots, n-1\}$, $n \geq 2k+1$, and $E = \{(i, i+j \pmod n) \mid i \in V, 1 \leq j \leq k\}$. Then $G = (V, E)$ is $2k$-connected.*

Proof. We claim it suffices to show that for every i, there are $2k$ edge-disjoint paths connecting i to $i+1 \pmod n$. Suppose this is true. Consider any cut $(C, V \setminus C)$ of the vertices. To show that this cut has $2k$ cut edges, it suffices to show that there exist $i \in C$, $j \in V \setminus C$ with $2k$ edge-disjoint paths connecting i to j. But there exists i such that $i \in C$ and $i+1 \in V \setminus C$. Hence our claim implies the assertion.

It remains to prove our claim. By symmetry, we may assume $i = 0$. We will exhibit $2k$ explicit edge-disjoint paths from 0 to 1. For $j = 2, \ldots, k$, we have paths which traverse edges $(0, j), (j, j+1), (j+1, 1)$ as well as paths which traverse edges $(0, n-j), (n-j, n-j+1), (n-j+1, 1)$. This accounts for $2k-2$ paths which are pairwise edge-disjoint. Note that, in these $2k-2$ paths, the only edges of the form $(a, a+1)$ used are for $a \in \{2, \ldots, k\} \cup \{n-k, \ldots, n-2\}$, and the only edges for the form $(b, b+k)$ used are for $b \in \{0, 1, n-k, n-k+1\}$. The final two paths are the path which is simply the edge $(0, 1)$, and the path which traverses edges $(0, n-1), (n-1, n-k-1), (n-k-1, n-k-2), (n-k-2, n-k-3), \ldots, (k+3, k+2), (k+2, 2), (2, 1)$. □

The following lemma ensures that the resulting graph is k-connected.

Lemma 7.4. *With probability at least $1-\gamma$, the resulting graph G_k is k-connected.*

Proof. We first show that with probability at least $1-\gamma$, every extreme set of G_{k-1} has a vertex which adds an edge to V_0. First, consider each small extreme set, i.e. extreme sets of size at most t. For each such set U, there is some $v_U \in U$ with minimal rank in U. By the choice of C, with probability at least $1-\gamma/2$ every v_U has a successful extreme set search and recognizes that it has minimal rank in U, hence every small extreme set contributes an edge to V_0. Next, consider each large extreme set, i.e. extreme sets of size more than t. The probability that a large extreme set contributes no edges, i.e. has no successful vertices, is at most $\left(1 - \frac{\ln(Cn/t)}{t}\right)^t < \frac{t}{Cn}$. Since there are at most n/t large extreme sets, the probability that every large extreme set contributes an edge is at least $1-\gamma/2$ by the choice of C. Hence with probability at least $1-\gamma$ every extreme set contributes an edge to V_0.

Now we assume that every extreme set contributes an edge to V_0 and show that G_k is k-connected. It suffices to show that, for every extreme set S in G_{k-1}, one of the edges added by the procedure crosses between S and $V \setminus S$. We have three cases.

- $V_0 \subseteq V \setminus S$: Consider $v \in S$ of minimal rank. With high probability, the neighborhood U of v found by the search algorithm is contained in S, so v is also minimally ranked in U. There must exist $v^* \in h(v)$ to which v is not already connected, for if not, then v is connected to $|h(v)| \geq k$ vertices outside of S, contradicting that S is $(k-1)$-extreme.
- $V_0 \subseteq S$: We claim there is another extreme set $S' \neq S$ in G_{k-1}. Suppose not. Then for every subset $S'' \subseteq V \setminus S$, $\deg(S'') \geq k$, for otherwise $V \setminus S$ contains an extreme set. But $\deg(V \setminus S) = \deg(S) = k-1$, so $V \setminus S$ is an extreme set, a contradiction. Now, recall that S' is disjoint from S. By the same argument as in the previous case, with high probability there is $v \in S'$ which gets connected to some super-node in $V_0 \subseteq S$.
- $V_0 \cap S$ and $V_0 \setminus S$ are both non-empty: We claim that in G_k, S has degree $\deg(S) \geq k$. Consider the cut $(V_0 \cap S, V_0 \setminus S)$ within the subgraph V_0. Any cut edge here must be a cut edge in the cut $(S, V \setminus S)$. By Lemma 7.3, the subgraph V_0 is k-connected, so the cut $(V_0 \cap S, V_0 \setminus S)$, and hence the cut $(S, V \setminus S)$, has at least k cut edges.

□

The following lemma shows that with high probability not too many additional edges are added.

Lemma 7.5. *With probability at least $1-\delta$, the number of edges added is at most*

$$2\epsilon m + ckn/2 + \frac{n \ln(Cn/t)}{\delta t}.$$

Proof. Each t-bounded extreme set will contribute at most one edge. There are at most $2\epsilon m$ extreme sets. On average $1/t$ of the remaining vertices contribute edges. Finally, the super-nodes themselves contribute at most $ckn/2$ edges. Therefore we add at most $2\epsilon m + ckn/2 + X$ edges, where X is a random variable with mean $\mathbb{E}[X] \leq \frac{n \ln(Cn/t)}{t}$. Markov's inequality implies

$$\Pr\left[X > \frac{\mathbb{E}[X]}{\delta}\right] \leq \delta.$$

□

Proof of Theorem 3.4. Set $C = \frac{1}{\ln(1/(1-\gamma))}$ and $t = \frac{\ln(Cn)}{\delta\alpha\epsilon}$. By Lemma 7.4 our resulting graph is k-connected with probability at least $1-\gamma$ and by Lemma 7.5, $\mathrm{dist}(G_{k-1}, G_k) \leq (2+\alpha)\epsilon + ck/2$ with probability $1-\delta$ and therefore by induction $\mathrm{dist}(G, G_k) \leq (2+\alpha)k\epsilon + ck^2/2$. This can be improved to $(2+\alpha)k\epsilon + ck/2$ by noting that the same super-nodes and the same edges between them can be used for all intermediate graphs $G_1, \ldots, G_{k-1}$. The success probability is at least $(1-\delta)^k(1-\gamma)^k \geq 1 - k(\delta+\gamma)$. The query complexity for correcting G_{k-1} to G_k is $O(t^3(t+k)\log(t+k)\log(Cn))$ queries to $N_{G_{k-1}}$. But each call to $N_{G_{k-1}}$ takes $O\left(\left(\frac{1}{c}+1\right)k\right)$ calls to $E_{G_{k-1}}$. By induction, the query complexity of E_{G_k} is $O\left(\left(\left(\frac{1}{c}+1\right)k\right)^k \cdot t^{3k}(t+k)^k \log^k(t+k)\log^k(Cn)\right)$. □

8 Local reconstruction of small diameter

In this section we prove Theorem 3.5. We first go over some preliminaries on graph diameter. We then give the high level description of the algorithm and prove some useful characteristics of graphs that are close to having small diameter before finally presenting the implementation and analysis of the algorithm.

8.1 Preliminaries

Definition 8.1. Let G be a graph with adjacency matrix A. For an integer k, let G^k be the graph on the same vertex set as G, with adjacency matrix A^k (boolean arithmetic).

It is not hard to show that there is an edge between u and v in G^k if and only if there is a path of length at most k between u and v in G. This observation immediately gives us the following.

Proposition 8.2. *Let G be a graph and let $D > 0$ be an integer. Then* $\operatorname{diam} G \leq D$ *if and only if G^D is a complete graph.*

8.2 High level description

The basic idea behind the algorithm is as follows. Again, we designate a "super-node" v_0 and add edges between v_0 and a few special vertices. If the input graph is close to having diameter at most D, then we aim for our reconstructed graph to have diameter at most $2D+2$. We show that if G is close to having diameter at most D, then we have an upper bound on the size of any independent set in G^D (Lemma 8.3 and Corollary 8.4).

Ideally, then, we want our special vertices (those that get an edge to v_0) to be a dominating set in G^D that is not too large. If we add edges from the dominating set in G^D to v_0, then our resulting graph $\widetilde{G}$ has diameter at most $2D+2$, since any vertex can reach some vertex in the dominating set within D steps, and hence v_0 within $D+1$ steps. If this dominating set is a maximal independent set, then our upper bound on the size of independent sets in G^D also upper bounds the number of edges we add. However, all known algorithms for locally computing a maximal independent set have query complexity bounded in terms of the maximum degree of the graph. A variant of the algorithm found in [15] has been analyzed by [24] and [17] to run in expected time bounded in terms of the average degree of the graph, but this average is taken over not only coin tosses of the algorithm but also all possible queries. This is undesirable for us since we want a uniform bound on the query complexity for any potential query vertex, given a "good" set of coin tosses. We want an algorithm such that, for most sets of coin tosses, we get the correct answer *everywhere*, whereas the algorithm of [15] leaves open the possibility of failure for some queries, regardless of the coin tosses. If some queries give the wrong answer, then the fact that our reconstructed graph retains the property is compromised. Instead, we do something less optimal but good enough. Note that adding edges does not increase diameter. Instead of using a maximal independent set, we settle for a dominating set in G^D as long as we can still control its size.

8.3 Properties of graphs close to having small diameter

The following lemma and subsequent corollary state that if a graph G is close to having diameter at most D, then no independent set in G^D can be very large.

Lemma 8.3. *Let $v_1, \ldots, v_k$ be an independent set in G^D and let H be the subgraph of G^D induced by this set. Let (s,t) be an edge not in $E(G)$ and let $G' = (V(G), E(G) \cup \{(s,t)\})$ be the graph obtained by adding (s,t) to G. Let H' be the subgraph of $(G')^D$ induced by $v_1, \ldots, v_k$. Then, for some $i \in \{1, \ldots, k\}$, all edges in H' (if any) are incident to v_i. In particular, if G^D has an independent set of size k, then $(G')^D$ has an independent set of size $k-1$.*

Proof. We will prove the assertion as follows. First, we will show that any two edges in H' must share a vertex. Then, we show that any third edge must also be incident to that vertex. This implies that all edges share a common vertex.

Suppose that the edges (v_i, v_j) and (v_k, v_l) are in H'. Then without loss of generality the paths from v_i to v_j and from v_k to v_l in G' must use (s,t). Let $a = d(v_i, s)$, $b = d(t, v_j)$, $c = d(v_k, s)$ and $d = d(t, v_l)$, so we have

$$\begin{aligned} a + b + 1 &\leq D \\ c + d + 1 &\leq D. \end{aligned}$$

Adding yields $a + b + c + d \leq 2D - 2$, so at least one of $(a + c)$, $(b + d)$ must be less than or equal to D. Without loss of generality, suppose $a + c \leq D$. This implies that $d(v_i, v_k) \leq D$ even in G, so it must be that $v_i = v_k$.

Now suppose yet another edge is in H'. By what we just showed, it must share an incident vertex with (v_i, v_j), and it must do likewise with (v_i, v_l). So either it is incident to v_i, in which case we are done, or the edge is (v_j, v_l). We will show the latter cannot happen. Define a, b, d as above. Note that we have

$$a + d + 1 \leq D.$$

Suppose there is path of length $\leq D$ from v_j to v_l in G' which traverses (s,t) by entering via s and exiting via t. Let $e = d(v_j, s)$. Then we have

$$e + d + 1 \leq D.$$

However, since any path from v_i to v_j that does not traverse (s,t) must have length greater than D, we have $a + e > D$ from which we can deduce

$$e > b + 1.$$

Similarly, since any path from v_j to v_l that does not traverse (s,t) must have length greater than D, we have

$$b + d > D.$$

But then $D \geq e + d + 1 > b + d + 2 > D + 2$, a contradiction. □

Corollary 8.4. *Suppose G is ϵ-close to having diameter $\leq D$. Then any independent set in G^D has size at most $\epsilon m + 1$.*

Proof. Suppose G^D has an independent set of size $\epsilon m + 2$. There exist a set of ϵm edges such that, when added to G, we obtain G' with $\text{diam}(G') \leq D$, and hence $(G')^D$ is a complete graph. But, by Lemma 8.3, $(G')^D$ has an independent set of size 2, a contradiction. □

Finally, we will use a result from [2, 18] which we will restate in our own terms below (the theorem number we reference is from [2]):

Theorem 8.5 ([2, Theorem 3.1]). *Any connected graph G is $(\frac{2n}{Dm})$-close to having diameter $\leq D$.*

This result implies that any connected graph is ϵ-close to having diameter at most $\frac{2n}{\epsilon m}$, and will come in handy when we want to bound our query complexity. In particular, the query complexity of our algorithm will depend on D, but if $D \geq \frac{2n}{\epsilon m}$, which is $O(1)$ for constant ϵ, then we can instead simply aim for diameter $\frac{2n}{\epsilon m}$ and bound our query complexity in terms of this constant instead.

8.4 Algorithm

We start by fixing a super-node $v_0 \in G$. Given our discussion in the high level description, it remains to implement the selection of a small dominating set. To this end, we create a dominating set S by first adding the set H of high-degree vertices into S and then using the local maximal independent set algorithm found in [21], which is based on Luby's algorithm ([12]), on G^D with H and its vertices' neighbors (in G^D) removed to create an independent set M. Then let $S = H \cup M$.

By high-degree vertex we mean a vertex with degree greater than $\overline{d}/\epsilon$, of which there are at most ϵn, where $\overline{d} = \frac{2m}{n}$ is a bound on the average degree. We will also consider the super-node v_0 a high-degree vertex for our purposes. To implement choosing a maximal independent set, we define a subroutine $\text{MIS}_D(v)$ as follows. On input v, simulate the local maximal independent set described in [21], except explore all neighbors within D steps from v rather than just immediate neighbors, and automatically reject if the algorithm encounters any vertex with degree exceeding $\overline{d}/\epsilon$. Note that the running time of the local MIS algorithm given in [21] is bounded in terms of the maximum degree of the graph. This is not problematic since our variant of the algorithm ignores any vertex with degree greater than $\overline{d}/\epsilon$, hence the effective maximum degree of our graph G for the purposes of the algorithm is $\overline{d}/\epsilon$, so the effective maximum degree of G^D is $(\overline{d}/\epsilon)^D$.

One final challenge is that if D is large, say $\Theta(\log n)$, then our query complexity bound in terms of our effective degree is no longer sublinear. We work around this by using a result of [2], which states that every connected graph is ϵ-close to having diameter at most $\frac{2n}{\epsilon m} = O(1/\epsilon)$. Therefore, we can aim for achieving diameter $K = \min\{D, \frac{2n}{\epsilon m}\}$ so that our effective degree is $(\overline{d}/\epsilon)^K = (\overline{d}/\epsilon)^{O(1/\epsilon)}$. Of course, this only works if our graph is connected to begin with. Therefore, we use the neighbor oracle for the connected correction G' of G, given in Section 6. The idea is then to first make G into a connected graph G', and then reconstruct a small diameter graph $\widetilde{G}$ out of G'.

```
procedure SmallDiam(v_1, v_2, D)
    K ← min{2n/(εm), D}
    if E_{G'}(v_1, v_2) = 1 then                ▷ if edge is already in graph, it stays in the graph
        return 1
    else
        if v_0 ∉ {v_1, v_2} then                ▷ do not add edge if neither endpoint is v_0
            return 0
        else
            Let v ∈ {v_1, v_2} \ {v_0}          ▷ check if v is special
            if deg_{G'}(v) > d̄/ε then
                return 1
            else
```

```
            return MIS_K(v)                    ▷ neighbor queries to G' instead of G
          end if
        end if
      end if
    end procedure
```

Proof of Theorem 3.5. Correcting G to G' with Mod-Connect adds $(1+\alpha)\epsilon m + cm$ edges. The query complexity is clearly dominated by that of $\mathrm{MIS}_K(v)$. The probability of success follows from that of the local MIS algorithm. Now we show correctness. Let H be the set of vertices in G with degree greater than $\overline{d}/\epsilon$, and let M be the set of vertices for which $\mathrm{MIS}(v) = 1$. Observe that M is an independent set in G^D, so $|M| \leq \epsilon m$. Note that since we forced v_0 to be in this union, the number of edges we added is actually $|H \cup M \setminus \{v_0\}| = |H| + |M| - 1 \leq \epsilon n + \epsilon m \leq 2\epsilon m$. Furthermore, let $v \in V(G)$. There exists some $u \in H \cup M$ such that $d(v,u) \leq K$, otherwise v would have been added to M. Therefore, $d(v, v_0) \leq K + 1$. For any other $v' \in V(G)$, we thus have $d(v,v') \leq d(v,v_0) + d(v_0,v') \leq 2K + 2 \leq 2D + 2$. □

References

[1] N. Ailon, B. Chazelle, S. Comandur, D. Liu. Property-preserving data reconstruction. In *Proc. 15th International Symposium on Algorithms and Computation*, pages 16–27, 2004.

[2] N. Alon, A. Gyàrfàs, and M. Ruszinkò. Decreasing the diameter of bounded degree graphs. *J. Graph Theory* **35** (2000), 161–172.

[3] M. A. Bender and D. Ron. Testing properties of directed graphs: acyclicity and connectivity. *Random Structures and Algorithms* **20** (2002), 184–205.

[4] A. Bhattacharyya, E. Grigorescu, M. Jha, K. Jung, S. Raskhodnikova, D. Woodruff. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. In *Proc. 14th International Workshop on Randomization and Computation*, pages 448–461, 2010.

[5] Z. Brakerski. Local property restoring. Manuscript, 2008.

[6] B. Chazelle, C. Seshadhri. Online geometric reconstruction. In *Proc. 22nd ACM Symposium on Computational Geometry*, pages 386–394, 2006.

[7] A. Derbakova, N. Correll, D. Rus. Decentralized self-repair to maintain connectivity and coverage in networked multi-robot systems. In *Proc. IEEE International Conference on Robotics and Automation*, 2011.

[8] O. Goldreich, S. Goldwasser, D. Ron. Property Testing and its Connection to Learning and Approximation. *J. ACM*, 45(4): 653-750 (1998)

[9] O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica* **32** (2002), 302–343.

[10] M. Jha and S. Raskhodnikova. Testing and reconstruction of Lipschitz functions with applications to data privacy. In *Proc. 52nd Annual IEEE Symposium on Foundations of Computer Science*, pages 433–442.

[11] S. Kale, Y. Peres, C. Seshadhri. Noise tolerance of expanders and sublinear expander reconstruction. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 719–728, 2008.

[12] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[13] S. Marko, D. Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms* 5(2) (2009).

[14] K. Menger. Zur allgemeinen Kurventheorie. *Fund. Math.* 10: 96–115, 1927.

[15] H. N. Nguyen and K. Onak. Constant-time approximation algorithms via local improvements. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336, 2008.

[16] D. Noar, D. Gusfield, C. Martel. A fast algorithm for optimally increasing the edge connectivity. *SICOMP*, 26(4):1139–1165, 1997.

[17] K. Onak, D. Ron, M. Rosen, R. Rubinfeld. A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1123–1131, 2012.

[18] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms* **20** (2002), 165–183.

[19] M. Parnas, D. Ron, R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences* **72** (2006), 1012–1042.

[20] R. Rubinfeld, M. Sudan. Robust Characterizations of Polynomials with Applications to Program Testing. *SIAM J. Comput.* 25(2): 252-271 (1996).

[21] R. Rubinfeld, G. Tamir, S. Vardi, and N. Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science*, pages 223–238, 2011.

[22] M. E. Saks and C. Seshadhri. Local monotonicity reconstruction. *SIAM Journal on Computing* **39** (2010), 2897–2926.

[23] E. Stump, A. Jadbabaie, V. Kumar. Connectivity management in mobile robot teams. In *Proc. IEEE International Conference on Robotics and Automation*, 2008.

[24] Y. Yoshida, M. Yamamoto, H. Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proc. 41st ACM Symposium on Theory of Computing*, pages 225–234, 2009.