Local Property Restoring

EXTENDED ABSTRACT

Zvika Brakerski
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot, ISRAEL
zvika.brakerski@weizmann.ac.il

April 7, 2008

Abstract

In this paper we present the new notion of *local restoring* of properties. A *restorer* gets implicit access, in the form of oracle queries, to a function (which may correspond to an object such as a graph, a code or a matrix) that is close to having a property, such as function monotonicity or graph bipartiteness. The restorer then, using a small number of queries to the function, can locally compute the value of a "corrected" function in any desired point. The corrected function is required to be both close to the original function and *strictly* have the property. In the case of error correcting codes, this corresponds to local self-correcting. Here we extend the definition and study for general functions and properties.

We define the new notion and go on to give restorers for properties in the dense graph model: Bipartiteness and ρ-Clique, and for function monotonicity. We also show some general relations between property restoring and other notions such as property testing and tolerant property testing.

1 Introduction

Error correcting codes encode messages into codewords in a way that allows decoding, i.e. recovering the original message even from codewords corrupted by some noise. Error correcting codes were introduced by Shannon [27] and Hamming [20], for applications such as communication over a noisy channel or storage over a noisy device, and have since found many applications in other areas such as complexity, algorithms, and cryptography.

While classical use of codes required recovering the entire message, it has been proved useful to use codes that enable *local* operations on codewords, see a survey by Goldreich in [12]. In such cases, the codeword is given in an implicit way via an oracle and global properties of the codeword are to be inferred by making a small number (preferably constant) of queries to the oracle. In *local self testing*, the goal is to decide whether the oracle indeed represents a valid codeword or whether it is far from one. In *local self correcting*, the goal is to recover bits of the original uncorrupted codeword.[1]

The notion of inferring global properties of objects by making local queries to an implicit representation of the object has been extended beyond the scope of error correcting codes in the field of *property testing*. Property testing has been defined by Rubinfeld and Sudan in [26] (and implied in the earlier work of Blum, Luby and Rubinfeld [4]), and has been extended by Goldreich, Goldwasser and Ron in [14] beyond algebraic properties to combinatorial graph properties. A property tester for property $\mathcal{P}$ is provided with query access to some function f and is required to decide, using a small number of queries, whether f has property $\mathcal{P}$ or whether f is *far* from having $\mathcal{P}$. The

[1] Two related notions that are less relevant to this work are *local decoding* where extracting a single bit of the original message is required and *local list decoding* where finding *all* possible messages whose codeword is close to the corrupted codeword is required.

local self testing of codes is thus a special case of property testing with the property being membership in the code. Property testers have been shown for a large class of graph properties, such as bipartiteness and ρ-clique, in the dense graph model. Function monotonicity has also been studied. See Section 1.3.

More recent research addressed *tolerant property testing* and *distance estimation*, both defined by Parnas, Ron and Rubinfeld in [25]. Whereas in property testing the tester must accept only when the function f has property $\mathcal{P}$, in tolerant property testing, the tester must also accept when the tested function f is *close* to a function having the property. In distance estimation the goal is to approximate the distance between f and the closest function having property $\mathcal{P}$. These notions have also been studied by Fischer and Fortnow [9] who show that some testable properties are not tolerantly testable, and by Fischer and Newman [11] who show that for all testable properties in the dense graph model, the distance can be approximated up to any additive constant.

1.1 Our Contribution

In this paper we define the notion of *restorers*, that given implicit access to a function/object that is close to having some property, locally reconstruct a function that has the property and is close to the original corrupt one. Obviously, there may be many functions that are close to the corrupt one and have the property. Our restorer will output one such function in a consistent local manner.

An (ϵ_1, ϵ_2)-restorer for property $\mathcal{P}$ is provided with query access to a function f that is ϵ_1-close to having property $\mathcal{P}$, meaning that there exists a function $h^* \in \mathcal{P}$ for which $\mathrm{dist}(f, h^*) \leq \epsilon_1$. On input x, the restorer makes a sub-linear (possibly constant) number of queries to f and outputs $h(x)$ where $h \in \mathcal{P}$ and $\mathrm{dist}(f, h) \leq \epsilon_2$. We note that even when $f \in \mathcal{P}$ the restorer may not return $f(x)$ but rather another function that is close to it.

Recall that there may be many possible "correct" functions/objects that are close to the input function, but we would like all local answers to be consistent with one such function.[2] Therefore we define the restorer as consisting of two (sub-linear) algorithms: the *initializer* creates a string that succinctly defines a single correct function and the *evaluator* takes this string and outputs the value of this function for each input.

Following the definition, we show some relations between restorers and other existing notions (see Section 1.2 for a detailed description of these notions).

- **Restoring and Property Testing yield Tolerant Testing.** We show in Theorem 3.1 that if a property has both a restorer and a property tester, then it is straightforward to construct a tolerant tester for that property.
- **Restoring does not imply Property Testing.** We show in Theorem 3.2 that there exist properties that are restorable but are not property testable. This is a straightforward consequence of a recent work by Grigorescu, Kaufman and Sudan [18]. They show that there exists an error correcting code that is 2-transitive, its dual contains low-weight codewords and in spite of that, is not locally testable. Their construction involves taking a non-testable subset of the RM code. Since the RM code is locally self correctable, this yields that the new code is both locally self correctable and is not testable.
- **Restoring (and tolerant testing) of Bipartiteness is easy for some parameters but hard for others.** In Corollary 3.3 we show that bipartiteness in the dense graph model has a $(\mathrm{poly}(\epsilon), \epsilon)$-restorer that runs in time $\mathrm{poly}(1/\epsilon)$ but for any c it does not have an $(\epsilon, c\epsilon)$-restorer with that time complexity (assuming the Unique Game Conjecture by Khot [22]). In fact we show in Theorem 3.4 that the above statement also applies to tolerant testers, thus showing a complexity gap between raw and fine tolerant testing and providing some answer to a question in [11].

  We use a result by Khot et al. [23] stating that under the Unique Games Conjecture, it is hard to distinguish instances of the MAX-CUT problem that are almost satisfiable and instances that are a little further from being satisfiable.

[2] Otherwise a trivial restorer for bipartiteness would be to always returns 0, since for every edge there is a bipartite graph where it is not present.

Most importantly, we demonstrate the viability of restoring by presenting restorers for several testable properties.

- **Graph Bipartiteness.** We present an $(\epsilon^4, O(\epsilon))$-restorer for bipartiteness in the dense graph model. That is, for given access to an adjacency matrix of a graph that is ϵ^4-close to a bipartite graph, we can locally construct a graph that is both bipartite and $O(\epsilon)$-close to the original graph. This restorer makes $\mathrm{poly}(1/\epsilon)$ queries to the graph and runs in $\mathrm{poly}(1/\epsilon)$ time.

- **Graph ρ-Clique.** We present an $(\epsilon^3, O(\epsilon))$-restorer for the ρ-Clique property in the dense graph model. This restorer makes $\mathrm{poly}(1/\epsilon)$ queries to the graph and runs in $\exp(\mathrm{poly}(1/\epsilon))$ time. Our construction uses an observation made in [14] incorporated with hashing.

- **1-Dimensional Function Monotonicity.** Given access to a one-dimensional function $f : [n] \to \mathcal{R}$, for any (totally) ordered set $\mathcal{R}$, we present an $(\epsilon, 2\epsilon + \delta)$-restorer for monotonicity of the function, for all constant $\delta > 0$. This restorer makes $\mathrm{polylog}(n)$ queries to the function and runs in time $\mathrm{polylog}(n)$. This construction uses a data structure defined in [25].

No attempt has been made in this extended abstract to optimize the constants or the powers in the $\mathrm{poly}(\cdot), \mathrm{polylog}(\cdot)$ notation in the various constructions.

1.2 Related Notions

The notion of restoring shares common characteristics with many previously defined notions. We present these notions and their relation to restoring.

- **Locally Self Correctable Codes.** A restorable property can be viewed as a relaxation of a locally self correctable code. Namely, a locally self correctable code is an error correcting code that has an (ϵ, ϵ)-restorer for all small enough ϵ. In Theorem 3.2 we use this relation to show that some restorable properties are not testable.

- **Property Testing, Tolerant Testing and Distance Estimation.** In these notions we are required to evaluate (to some extent) the difference between the object at hand and the nearest object that has the required property. While the setting of the problem is almost identical to that of restoring, the goal is very different. In restoring we are required to locally compute an object that has the property exactly and that is close to the given object. As we show in Theorem 3.1, restorers can assist in construction of tolerant testers from property testers. In Theorem 3.2, however, we show that a restorer does not necessarily imply even the weakest notion of the three (that of property testing).

- **Approximate Find Algorithms.** In [14] it is shown how to use property testing algorithms for certain graph properties to create linear-time *approximate find algorithms*. Such algorithm can be interpreted as finding a graph that is both ϵ-close (for some ϵ) to the original graph, and has the property $\mathcal{P}$.[3] However, there are two differences: *(i)* Their method was only guaranteed to work on graphs that have been accepted by the original property testers (which include all graphs that have the property and possibly some other graphs that are ϵ-close to having it); *(ii)* Approximate find algorithms are required to go over the entire graph and therefore are not local in the aforementioned sense. Thus, such algorithms can be considered as non-local self correction algorithms.

[3] In the original formulation, approximate find algorithms were used to approximately find the structure that defines the property, such as a k-coloring that respects all but an ϵ fraction of the edges, or a dense subset of the vertices that is almost a clique. Clearly, given the output of such algorithm it is immediate to "repair" the graph so it has the property.

- **Program Testing and Correcting.** In program testing and correcting, introduced in [4], there is some function f to be computed. A query access is given to a procedure P that allegedly computes f and we consider the hamming distance between f and P. Program testing resembles tolerant testing as it is required to decide whether P is close (to some extent) to f or far from it (to some larger extent). This is similar to the case of tolerant property testing with the difference being that f itself is known and can be computed efficiently. Therefore instead of requiring sub-linear run time in the size of the object, we require that the running time is sub-linear in the running time of any (alternatively, in other formulations, the best known) program that computes f.

  Program correcting is a notion that is very closely related to the one presented in this paper. The setting is similar to that of program testers and the corrector is required to output the correct value of f (with high probability) while maintaining sub-linear running time (again, relative to the best possible computation of f). We can see that when f ceases to be a known function, the notion of correcting naturally leads towards the definition of restorers.

1.3 Related Work

Bipartiteness in the dense graph model has been introduced in the context of property testing in [14] where a tester with query complexity $\tilde{O}\left(1/\epsilon^3\right)$ and $\mathrm{poly}(1/\epsilon)$ time complexity is presented. This has been improved to $\tilde{O}\left(1/\epsilon^2\right)$ query complexity by Alon and Krivelevich [2]. A lower bound of $\Omega(1/\epsilon^{1.5})$ on the query complexity of the problem has been introduced by Bogdanov and Trevisan [5]. The problem has also been studied in other models suited for sparse or general graphs [15, 16, 21]. $(\mathrm{poly}(\epsilon), \epsilon)$-Tolerant testing for this property can be derived from its property tester using constructions in Goldreich and Trevisan [17] and in [25]. Distance approximation up to an additive factor of all testable properties in the dense graph model, including this one, is presented in [11] and has $(1/\epsilon)$-tower query complexity.

ρ-Clique in the dense graph model has also been introduced in [14] where a tester with query complexity $\tilde{O}\left(1/\epsilon^6\right)$ and time complexity $\exp\left(\tilde{O}\left(1/\epsilon^2\right)\right)$. The results in [17, 25] imply an $(\mathrm{poly}(\epsilon), \epsilon)$-tolerant tester for this problem as well, and [11] implies distance approximation with $(1/\epsilon)$-tower query complexity.

Function Monotonicity has been extensively studied in the field of property testing [13, 6, 7, 19, 10, 3]. An $(\frac{1}{2}\epsilon, \epsilon + \delta)$-tolerant property tester with run time and query complexity $\mathrm{polylog}(n)$ (where the domain of the function is $[n]$) is shown in [25] and also extended to the d-dimensional case. An improved result for the d-dimensional case is presented by Fattal and Ron [8]. Ailon et al. [1] presented a tolerant tester with time complexity $\log(n)$.

1.4 Paper Organization

In Section 2 we formally define the notion of restorers. Positive and negative results as to relations with other notions are presented in Section 3. Sections 4, 5 are dedicated to providing restorers for properties in the dense graph model: Bipartiteness and ρ-Clique, and for function 1-dimensional monotonicity.

2 Restorer — a Definition

Throughout this paper, we use hamming distance.

Definition 1 (Hamming Distance)**.** *Let* $F_1, F_2 : \mathcal{D} \to \mathcal{R}$*, then* $\mathrm{dist}_H(F_1, F_2) = \Pr_{x \overset{R}{\leftarrow} \mathcal{D}}[F_1(x) \neq F_2(x)]$.

Since this is the only distance measure we use, we use $\mathrm{dist}(\cdot, \cdot)$ instead of $\mathrm{dist}_H(\cdot, \cdot)$. Motivated by the discussion in the introduction, we present a formal definition of a restorer. We use *w.h.p* (with high probability) to denote a probability of more than $2/3$.

Definition 2 (Restorer)**.** *Let $\mathcal{D} \to \mathcal{R}$ be a class of functions, $\mathcal{P} \subseteq \mathcal{D} \to \mathcal{R}$ be a property and* $\mathrm{dist} : (\mathcal{D} \to \mathcal{R}) \times (\mathcal{D} \to \mathcal{R}) \to \mathbb{R}$ *be a metric. Let $(\mathcal{I}, \mathcal{E})$ be a pair of randomized algorithms with query access to $F : \mathcal{D} \to \mathcal{R}$, referred to as the* initializer *and the* evaluator *respectively. Then $(\mathcal{I}, \mathcal{E})$ is an (ϵ_1, ϵ_2)-restorer for property $\mathcal{P}$ if the following hold:*

1. *$\mathcal{I}, \mathcal{E}$ are local randomized algorithms. That is, they make $o(|\mathcal{D}|)$ queries to F.*
2. *$\mathcal{I}$ takes no input and its output is called the* initial state *of the restorer.*
3. *$\mathcal{E}$ takes an initial state σ and a value $x \in \mathcal{D}$ and outputs a value $\mathcal{E}^F(\sigma, x) \in \mathcal{R}$.*
4. *Let $\sigma = \mathcal{I}^F()$. If $\min_{H \in \mathcal{P}} \mathrm{dist}(F, H) \leq \epsilon_1$, then w.h.p over the coin tosses of $\mathcal{I}$, there exists $H_\sigma \in \mathcal{P}$ such that $\mathrm{dist}(F, H_\sigma) \leq \epsilon_2$ and for all x it holds w.h.p that $\mathcal{E}^F(\sigma, x) = H_\sigma(x)$.*

- **A note on proximity.** We stress that the restored function H_σ may not be identical to F even if $F \in \mathcal{P}$.
- **A note on randomness.** It is always possible to derandomize the evaluator at a cost of a $\log |\mathcal{D}|$ factor in the query and time complexities. In fact, all restorers presented in this paper have deterministic evaluators. One can amplify the success probability of the evaluator to $\left(1 - \frac{\delta}{|\mathcal{D}|}\right)$ using $O\left(\log |\mathcal{D}|\right)$ repetitions, thus by the union bound use the same random string for all inputs with success probability at least $1 - \delta$. Creating this common random string in the initializer derandomizes the evaluator.

  Also note that the initializer can be considered as only flipping random coins and exporting them as the initial state. This might, however, significantly increase the evaluator's complexity as it will have to simulate the initializer at every invocation.

3 Restorers vs. Testers

The notion of restoring is related to the notions of property testing and tolerant testing defined below.

Definition 3 (Property Testing and Tolerant Testing [25])**.** *Let $\mathcal{P}$ be a property and let $0 \leq \epsilon_1 < \epsilon_2 \leq 1$. An* (ϵ_1, ϵ_2)-Tolerant Testing *algorithm for property $\mathcal{P}$ is given query access to an unknown function F. The algorithm should accept w.h.p if F is ϵ_1-close to having property $\mathcal{P}$, and should reject w.h.p if F is ϵ_2-far from having the property. A* Fully Tolerant Testing *algorithm is given $0 \leq \epsilon_1 < \epsilon_2 \leq 1$ as input and has the above behavior. An* ϵ-Property Testing *algorithm for $\mathcal{P}$ is an $(0, \epsilon)$-Tolerant Testing algorithm.*

The following theorem states that if a property has a property tester and a restorer then it also has a tolerant property tester.

Theorem 3.1. *Let $\mathcal{P}$ be a property. Let $(\mathcal{I}, \mathcal{E})$ be an (ϵ_1, ϵ_2)-restorer for $\mathcal{P}$ and let $\mathcal{T}$ be an ϵ'-property tester for $\mathcal{P}$. Then for any $\beta > 0$ there exists an $(\epsilon_1, \epsilon_2 + \epsilon' + \beta)$ tolerant tester for $\mathcal{P}$.*

Proof Sketch: For oracle function F, the tolerant tester is as follows:

1. Run $\sigma = \mathcal{I}^F()$.
2. Take a random sample of the domain, run (an amplified version of) $\mathcal{E}^F(\sigma, \cdot)$ on that sample to estimate $\mathrm{dist}(F, H_\sigma)$ to within $\beta/2$. If the estimate is more than $\epsilon_2 + \beta/2$ then reject.
3. Run $\mathcal{T}$ on H_σ using (amplified) $\mathcal{E}^F$ to obtain the values of H_σ. Accept if and only if $\mathcal{T}$ accepts.

If F is ϵ_1-close to $\mathcal{P}$ then $\mathrm{dist}(F, H_\sigma) = \epsilon_2$ and $\mathcal{T}$ must accept. If F is $(\epsilon_2 + \epsilon' + \beta)$-far from $\mathcal{P}$, then either $\mathrm{dist}(H_\sigma, F) > \epsilon_2 + \beta$ and the distance test fails, or $\mathrm{dist}(H_\sigma, F) \leq \epsilon_2 + \beta$ and thus H_σ is ϵ'-far from having $\mathcal{P}$ and $\mathcal{T}$ rejects. ∎

But is the condition that $\mathcal{P}$ has a property tester necessary? It may seem like restoring is a stronger notion then testing and thus every restorable property is also testable. We use an example from the world of error correcting codes to show that at least when considering constant query complexity, this is not the case.

Theorem 3.2. *There exists a property $\mathcal{P}$ that has an (ϵ, ϵ)-restorer with constant query complexity, but $\mathcal{P}$ is not ϵ-testable with constant query complexity.*

Proof Sketch: The proof is implicit in [18] where an error correcting code that is locally self correctable (in constant number of queries) but not locally self testable in k queries is presented. Taking $k = \omega(1)$ and defining $\mathcal{P}$ as the codewords of the code completes the proof. ∎

We direct our attention to the relation between ϵ_1, ϵ_2. Can they be made as close as we want (with running time depending on, say, their difference)? In the case of tolerant testing and distance estimation in the dense graph model, this is answered in the affirmative in [11]. The query complexity required to achieve this, however, has super-exponential dependence on the difference. [11, Section 7] raise the question of whether this dependence can be improved.

The following theorem shows that for bipartiteness in the dense graph model (see Section 4 for model definition), even though tolerant property testing to within $(\mathrm{poly}(\epsilon), \epsilon)$ is achievable in time $\mathrm{poly}(1/\epsilon)$, this is not the case for $(\epsilon, c\epsilon)$-tolerant testers assuming the Unique Games Conjecture [22]. We stress that we do not answer the question of whether there exists such a tolerant tester with query complexity $\mathrm{poly}(1/\epsilon)$ and super-polynomial time complexity.

We use a corollary made by Khot et al. in [24].

Corollary 3.3 ([24, Corollary 1])**.** *Assuming the Unique Games Conjecture, for all sufficiently small $\eta > 0$, it is hard to distinguish between instances of MAX-CUT that are at least $(1 - \eta)$ satisfiable and instances that are at most $(1 - \frac{2}{\pi}\sqrt{\eta})$ satisfiable.*

Theorem 3.4. *Assume the Unique Games Conjecture. Let $\mathcal{P}$ be the bipartiteness property in the dense graph model (defined in Section 4). Then:*

1. *For all $\epsilon > 0$, $\mathcal{P}$ has a $(\mathrm{poly}(\epsilon), \epsilon)$-tolerant tester that runs in time $\mathrm{poly}(1/\epsilon)$.*
2. *For all $c > 1$, $\mathcal{P}$ has no $(\epsilon, c\epsilon)$-tolerant tester that for all $\epsilon > 0$ runs in time $\mathrm{poly}(1/\epsilon, n)$.*

Proof: A tolerant tester for bipartiteness in the dense graph model is implicit in [14] and can also be obtained by Theorem 3.1 by combining the restorer presented in Section 4.1 and the property tester from [14].

For the negative part, we use Corollary 3.3. Let $G = (V, E)$ be an instance for MAX-CUT that is $1 - \delta$ satisfiable. Denote $|V| = n$, $|E| = m$. Then the maximal cut in G is of size $(1-\delta)m$. Therefore δm is the minimal and sufficient number of edges to be removed from G to make it bipartite. Thus the result in [24] asserts that $(\eta\frac{m}{n^2}, \frac{2}{\pi}\sqrt{\eta}\frac{m}{n^2})$-tolerant testing for bipartiteness is hard. Take $\epsilon = \frac{2}{c\pi}\sqrt{\eta}\frac{m}{n^2}$, for some $\eta < \left(\frac{2}{c\pi}\right)^2$, then an $(\epsilon, c\epsilon)$-tolerant testing is hard. Therefore, assuming the conjecture, it cannot be done in time $\mathrm{poly}(1/\epsilon, n) = \mathrm{poly}(n)$. ∎

The corollary that follows extends this conclusion to the field of restorers.

Corollary 3.5. *Bipartiteness in the dense graph model has, for all $\epsilon > 0$, a $(\mathrm{poly}(\epsilon), \epsilon)$-restorer that runs in time $\mathrm{poly}(1/\epsilon)$ but has no $(\epsilon, c\epsilon)$-restorer for all sufficiently small $\epsilon > 0$ that runs in time $\mathrm{poly}(1/\epsilon, n)$.*

4 Restorers for Graph Properties

In this section we present restorers for graph properties in the dense graph model. In this model, a graph $G = (V, E)$ is represented by its adjacency function (or matrix) $F_G : V \times V \to \{0, 1\}$ s.t. $F_G(u, v) = 1$ if and only if $(u, v) \in E$. We sometimes use F_G as notation for G. Note that we only handle undirected simple graphs. In this representation, the Hamming distance between two graphs $F_1 = (V, E_1), F_2 = (V, E_2)$ is $\mathrm{dist}(F_1, F_2) = \Pr_{(u,v)\overset{R}{\leftarrow}V^2}[F_1(u, v) \neq F_2(u, v)] = 2\frac{|E_1 \Delta E_2|}{|V|^2}$. The factor of 2 is due to the graph's being undirected (and thus

every edge $(u, v) \in E$ appears twice in $V \times V$). Throughout this section we denote $N = |V|$ (where V is clear from context).

We present restorers for two properties:

- **Bipartiteness.** A graph $G = (V, E)$ is bipartite if there exist V_1, V_2 s.t.$V_1 \cap V_2 = \emptyset$, $V_1 \cup V_2 = V$ and $(V_1 \times V_2) \cap E = \emptyset$.
- **ρ-Clique.** A graph $G = (V, E)$ is said to have a ρ-Clique if it has a clique of size ρN. That is, there exists $C \subseteq V$ s.t. $|C| = \rho N$ and $C \times C \subseteq E$.

Since we often refer to sets of vertices or subgraphs that are reduced to a sample set, the following notation is useful (think of S a small sample set and A as some big structure, say a large clique in the graph). Let $G = (V, E)$ be a graph. Let $S \subseteq V$ be a subset of the vertices and let $A \subseteq V$ be another subset. Then $A_{|S} \stackrel{\text{def}}{=} A \cap S$ and $G_{|S} \stackrel{\text{def}}{=} (S, E_{|S \times S})$.

4.1 Bipartiteness

We present an $(\epsilon^4, O(\epsilon))$-restorer for bipartiteness in the dense graph model. First we only consider the case where the graph has minimum degree cN for some $c > 0$ and show an $(\epsilon^2, O\left(\frac{1}{c}\epsilon\right))$-restorer for this case. Then we show how to use this construction to obtain a restorer in the general setting.

An overview of the main idea is presented below. Statements of the properties of the restorers in the high degree case and in the general case appear in Theorems 4.1, 4.2 respectively. The restorers themselves, along with a detailed analysis and proof of the theorems appear in Appendix A.

Overview: Consider a bipartite graph of minimum degree cN. The graph is augmented with $\epsilon^2 N^2$ edges that violate bipartiteness. Consider the partition induced by bipartiteness. All but ϵN vertices of the graph are adjacent to less than ϵN violating edges. Consider two vertices x, y that are adjacent to less than ϵN violating edges and have more than ϵN common neighbors. Obviously x, y belong to the same side of the partition. If we create a large enough set of vertices, all belonging to the same side of the partition, then any good vertex z neighboring at least ϵN of them belongs to the other side of the partition. We continue this process iteratively to obtain two sets that must be separated by the partition. We can then pick a new x that is not in either set and repeat the process. We do this until all of the graph is covered by pairs of sets.

Several difficulties of the above description come to mind: *(i)* The process is not local. We deal with this later; *(ii)* How can we tell whether a vertex has many adjacent violating edges? Even though we cannot do that, we can verify at the end of the process that we didn't remove too many edges, in which case we don't care whether we picked the "right" vertices; *(iii)* Do all pairs of sets must agree? We show that "repairing" their disagreement is not too expensive in terms of edge removals.

Locality is achieved by sampling a small subset of the vertices in the initializer and applying the procedure above to its induced subgraph. The partition in the induced subgraph, in turn, defines a partition of the entire graph that can be computed locally by the evaluator.

In Appendix A.1, the restorer $(\mathsf{InitDenseBipartite}, \mathsf{EvalDenseBipartite})$ is presented and analyzed.

Theorem 4.1. *For all $\epsilon < a \cdot c^2$ (for some $a > 0$).* $(\mathsf{InitDenseBipartite}, \mathsf{EvalDenseBipartite})$ *is an $(\epsilon^2, O\left(\frac{1}{c}\epsilon\right))$-restorer for bipartiteness in the dense graph model, restricted to input graphs with minimum degree cN. Let F be such graph, then the following hold.*

1. $\mathsf{InitDenseBipartite}^F$ *makes $\mathrm{poly}(1/\epsilon)$ queries to its oracle. It runs in time $\mathrm{poly}(1/\epsilon)$ and succeeds with probability at least $1 - \mathsf{negl}(1/\epsilon)$.*
2. $\mathsf{EvalDenseBipartite}^F$ *makes $\mathrm{poly}(1/\epsilon)$ queries to its oracle. It runs in time $\mathrm{poly}(1/\epsilon)$ and is deterministic.*

In Appendix A.2, the special case above is extended to the general case with somewhat worse parameters, presenting and analyzing the restorer (InitBipartite, EvalBipartite).

Theorem 4.2. (InitBipartite, EvalBipartite) *is an* $(\epsilon^4, O(\epsilon))$*-restorer for bipartiteness in the dense graph model. The following hold.*

1. InitBipartite *makes* $\mathrm{poly}(1/\epsilon)$ *queries to its oracle. It runs in time* $\mathrm{poly}(1/\epsilon)$ *and succeeds with probability at least* $1 - \mathsf{negl}(1/\epsilon)$.
2. EvalDenseBipartite *makes* $\mathrm{poly}(1/\epsilon)$ *queries to its oracle. It runs in time* $\mathrm{poly}(1/\epsilon)$ *and is deterministic.*

Theorems 4.1, 4.2 are proven in Appendices A.1, A.2 respectively.

4.2 ρ-Clique

We present an $(\epsilon^3, O(\epsilon))$-restorer for the ρ-Clique property. The initializer and evaluator algorithms are presented in Figures 1, 2 respectively. An auxiliary procedure for the evaluator is presented in Figure 3. The initializer makes $\mathrm{poly}(1/\epsilon)$ queries and runs in time $\exp(\mathrm{poly}(1/\epsilon)) + \mathrm{polylog}(N)$. The evaluator makes $\mathrm{poly}(1/\epsilon)$ queries, runs in time $\mathrm{poly}(1/\epsilon) + \mathrm{polylog}(N)$ and is completely deterministic.

Overview: We extend an observation made in [14]: assume there exists a set in the graph that is "almost" a ρ-Clique (almost all vertices in it are connected to almost all other vertices), denote it by C. Let T be the set of all vertices with "many" (almost ρN) neighbors in C. Let W be the set of ρN vertices of T that have the largest number of neighbors *within* T. Obviously almost all vertices in C are also in T. Furthermore, almost all vertices of C neighbor almost all vertices of T (this follows by counting since we require that each vertex in T neighbors almost all vertices of C). Therefore there exist almost ρN vertices in T that neighbor almost all vertices of T. Taking this set and completing it to a clique should not require adding "too many" edges.

Clearly, this skeleton of an algorithm is non-local. We approximate it by a local algorithm that finds (small) sets that represent T, W and use a pairwise independent hash family to enforce consistency of the solution on all edges.

We note that our restorer contains a predicate which may be interesting in its own right: Procedure CheckNode (Figure 3) locally provides explicit information on whether an input node is a member of the reconstructed clique in H_σ (which may be a little larger than ρN).

The restorer (InitClique, EvalClique) is presented in Figures 1, 2 respectively. Theorem 4.3 below summarizes its properties. Its proof uses lemmas presented and proven in Appendix B.

For a vertex set A, the set of vertices that neighbor "almost all" of A is defined as follows.

Definition 4. $K_\gamma(A) \stackrel{\mathrm{def}}{=} \{v \in V : |\Gamma(v) \cap A| \geq (1-\gamma)|A|\}$ *for all* $A \subseteq V$.

Theorem 4.3. (InitClique, EvalClique) *is an* $(\epsilon^3, O(\epsilon))$*-restorer for the* ρ*-Clique property. The following hold.*

1. InitClique *makes* $\mathrm{poly}(1/\epsilon)$ *queries to its oracle. It runs in time* $\exp(\mathrm{poly}(1/\epsilon)) + \log N$ *and its success probability is at least* $1 - \mathsf{negl}(1/\epsilon) - \frac{\mathrm{poly}(1/\epsilon)}{N}$.
2. EvalClique *makes* $\mathrm{poly}(1/\epsilon)$ *queries to its oracle. It runs in time* $\mathrm{poly}(1/\epsilon) + \mathrm{polylog}(N)$ *and is deterministic.*

Proof Sketch: The query and time complexities follow immediately from the code of the restorer. We define a *proper initialization* of the restorer (Definition 10 in Appendix B) and show in Lemma B.1 (Appendix B) that such initialization happens with probability $1 - \mathsf{negl}(1/\epsilon)$. Lemmas B.2, B.3 (Appendix B) prove that in case of proper initialization, if F is indeed ϵ^3-close to having a ρ-clique then any subset of size $(\rho - 6\epsilon)N$ of $K_{2\epsilon^2/\rho}(U^*) \cap K_{2\epsilon}(T^*)$ (denoted K^* from here on), if appended with any additional $6\epsilon N$ vertices, only needs $O(\epsilon)N^2$ additional edges to become a clique. Furthermore, w^* is a good estimate on the relative size of K^*.

Algorithm InitClique

Input: Query access to graph F.
Output: Initial state $\sigma = (U^*, T^*, w^*, h)$.
Code:

(1) Sample uniformly at random 3 sets of vertices $S_1, S_2, S_3 \stackrel{R}{\leftarrow} V$ of sizes $m_1 = \Theta(1/\epsilon^3)$, $m_2 = \Theta(1/\epsilon^5)$, $m_3 = \Theta(1/\epsilon^5)$ respectively.

(2) Query F on all pairs of vertices in $S_1 \times S_2$, $S_1 \times S_3$, $S_2 \times S_3$.

(3) Initialize $U^* \leftarrow \emptyset$.

(4) Recall that $K_\gamma(A) \stackrel{\text{def}}{=} \{v \in V : |\Gamma(v) \cap A| \geq (1-\gamma)|A|\}$. For all $U \subseteq S_1$ s.t. $|U| = \frac{\rho}{2} m_1$ do:

4a: Define $T = K_{2\epsilon^2/\rho}(U)_{|S_2}$. If $\frac{|T|}{m_2} < \frac{\rho}{2}$, go to the next U.

4b: Define $W = (K_{2\epsilon^2/\rho}(U) \cap K_{2\epsilon}(T))_{|S_3}$. If $w = \frac{|W|}{m_3} < (\rho - 5\epsilon)$, go to the next U.

4c: Set $U^* \leftarrow U$, $T^* \leftarrow T$, $W^* \leftarrow W$, $w^* \leftarrow w$.

(5) Let $H_{\log N}^{\log(1/\epsilon^2)} \subseteq [N] \to [1/\epsilon^2]$ be a family of pairwise independent hash functions. Randomly select a function $h \stackrel{R}{\leftarrow} H_{\log N}^{\log(1/\epsilon^2)}$.

(6) Return $\sigma = (U^*, T^*, w^*, h)$.

Figure 1: Algorithm InitClique.

Algorithm EvalClique

Input: Initial state $\sigma = (U^*, T^*, w^*, h)$, input pair of vertices (u, v). Query access to graph F.
Output: Value of adjacency function of graph $H_\sigma(u, v)$.
Code:

(1) Query $F(u, v)$. If $F(u, v) = 1$, return 1.

(2) If $\mathsf{CheckNode}^F(\sigma, v) = 1$, $\mathsf{CheckNode}^F(\sigma, u) = 1$ then return 1, otherwise return 0.

Figure 2: Algorithm EvalClique.

We then use the hash function to consistently define the subsets of K^* and of $V \setminus K^*$ that form the clique. The values of ℓ_1, ℓ_2 in Procedure CheckNode are defined so that we can use the Mixing Lemma for pairwise independent hash families. Using the Mixing Lemma, we bound the probability that the number of vertices from K^* that are selected is $w^* \cdot N + \Theta(\epsilon)$ and the number of vertices from $V \setminus K^*$ is at most $\Theta(\epsilon)$ so that their sum is at least ρN and no more than $(\rho + \Theta(\epsilon))N$. We get that with probability at least $\left(1 - \Theta(\frac{1}{\epsilon^4 N})\right)$ on the choice of h, at least ρN vertices are selected by CheckNode and completing them to a clique requires $O(\epsilon)N^2$ additional edges. ∎

5 Restorer for 1-Dimensional Monotonicity

In this section we extend the notion of restoring beyond the scope of graph properties and show a restorer for 1-Dimensional function monotonicity. An overview of the construction and statement of performance follows. Full details and analysis appear in Appendix C.

Our objects are functions $f : [n] \to \mathcal{R}$, where $\mathcal{R}$ is a (totally) ordered set (by relation $<$). The Hamming distance between functions is simply the number of locations on which they differ, divided by n. A function is *monotone* if $\forall_{i \in [n-1].}\ f(i) \leq f(i+1)$. As explained in [25], we can assume w.l.o.g that all values of f are

Procedure CheckNode

Input: Initial state $\sigma = (U^*, T^*, w^*, h)$, a vertex v. Query access to graph F.
Output: Returns 1 if v is a member of the ρ-clique in H_σ.
Code:

(1) Denote $\ell_1 = \left\lceil \frac{\rho}{\epsilon^2(w^*-\epsilon)} \right\rceil$ and $\ell_2 = \left\lceil \frac{\rho-(w^*-2\epsilon)}{\epsilon^2(1-w^*-\epsilon)} \right\rceil$.

(2) Query F on all pairs $\{v\} \times U^*$, $\{v\} \times T^*$.

(3) If $v \in K_{2\epsilon^2/\rho}(U^*) \cap K_{2\epsilon}(T^*)$ then return 1 if and only if $h(v) \leq \ell_1$.

(4) Otherwise, return 1 if and only if $h(v) \leq \ell_2$.

Figure 3: Procedure CheckNode.

distinct. Denote the class of all monotone functions by $\mathcal{M} \subseteq ([n] \to \mathcal{R})$ and for all f denote f's distance from monotonicity by $\epsilon_{\mathrm{mon}}(f) = \min_{h \in \mathcal{M}} \mathrm{dist}(f, h)$.

Overview: In [25], the *index-value tree* data structure is presented. An index-value tree is a binary tree where an interval $I(v) \subseteq [n]$ is associated with each node v in the tree. At each level, the interval $I(v)$ is split between the children of v. Thus, with each $j \in [n]$ we can associate a path in the tree containing all nodes v such that $j \in I(v)$. In each node v, some of the indices in $I(v)$ are decided to be "bad". Many index-value trees can be associated with each function f, but for any of them, the set of all bad indices, denoted B, is shown to be a superset of the indices that violate monotonicity. That is, f is monotone on $[n] \setminus B$.

[25] then goes on to present a construction of an index-value tree in which both the height of the tree is $O(\log n)$ and B is not much larger than the minimal set of violating indices. Furthermore, each path in the tree can almost be computed locally in $O(\mathrm{polylog}(n))$ time and query complexities. "Almost" is since the local construction cannot decide whether $i \in B$ but rather whether $i \in B_1$ (see Definition 12 in the appendix). B_1 is shown to be a sufficiently small superset of B. In Corollary C.1 we present a slight modification to the algorithm of [25] that is required in order to make it consistent.

Being able to decide whether $i \in B_1$, however, is not sufficient in order to correct the function. For each $i \in B_1$, we would like to find the largest $j < i$ such that $j \notin B_1$, and define $h_\sigma(i) = f(j)$ (which restores monotonicity). This, however, may be difficult. Therefore we define yet another sufficiently small superset $B_2 \supseteq B_1$ (see Definition 14 in the appendix). The definition is such that for every node v in the tree, we can locally decide whether the entire subtree rooted at v is in B_2. Thus enabling to locally search the tree for the required $j \notin B_2$.

The initializer InitMonotone simply selects the randomness such that the tree and the sets B_1, B_2 are well defined. The evaluator EvalMonotone, when applied on input index i, searches the tree to find whether $i \in B_2$. If $i \notin B_2$ then its value doesn't change. Otherwise we search for the maximal $j \notin B_2$ such that $j < i$ and set $h_\sigma(i) = f(j)$, thus restoring monotonicity. Since the set B_2 is a small enough superset of the violating indices, correctness follows.

The properties of the restorer are stated below.

Theorem 5.1. *For all $\delta > 0$,* (InitMonotone, EvalMonotone) *is an $(\epsilon, 2\epsilon + 4\delta)$-restorer for function monotonicity. The following hold.*

1. InitMonotone *makes no queries to its oracle. It runs in time $\Theta(\log^4(n)/\delta^2)$ and its success probability is at least $1 - 1/\mathrm{poly}(n)$.*
2. EvalMonotone *makes at most $O(\log^8(n)/\delta^4)$ queries to its oracle. It runs in time $O(\log^8(n)/\delta^4)$ and is deterministic.*

The theorem is proven in Appendix C.

References

[1] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. *Random Struct. Algorithms*, 31(3):371–383, 2007. Preliminary version in RANDOM '04.

[2] N. Alon and M. Krivelevich. Testing k-colorability. *SIAM J. Discrete Math.*, 15(2):211–227, 2002.

[3] T. Batu, R. Rubinfeld, and P. White. Fast approximate pcps for multidimensional bin-packing problems. *Inf. Comput.*, 196(1):42–56, 2005. Preliminary version in RANDOM '99.

[4] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

[5] A. Bogdanov and L. Trevisan. Lower bounds for testing bipartiteness in dense graphs. In *CCC '04: Proceedings of the 19th IEEE Annual Conference on Computational Complexity*, pages 75–81, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. In *RANDOM-APPROX '99: Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, pages 97–108, London, UK, 1999. Springer-Verlag.

[7] F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *J. Comput. Syst. Sci.*, 60(3):717–751, 2000. Preliminary version in STOC '98.

[8] S. Fattal and D. Ron. Approximating the distance to monotonicity in high dimensions, 2008. Unpublished.

[9] E. Fischer and L. Fortnow. Tolerant versus intolerant testing for boolean properties. In *CCC '05: Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, pages 135–140, Washington, DC, USA, 2005. IEEE Computer Society.

[10] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 474–483, New York, NY, USA, 2002. ACM.

[11] E. Fischer and I. Newman. Testing versus estimation of graph properties. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 138–146, New York, NY, USA, 2005. ACM.

[12] O. Goldreich. Short locally testable codes and proofs (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, (014), 2005.

[13] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000. Preliminary version in FOCS '98.

[14] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *J. ACM*, 45(4):653–750, 1998. Preliminary version in FOCS '96.

[15] O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 406–415, New York, NY, USA, 1997. ACM.

[16] O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.

[17] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Struct. Algorithms*, 23(1):23–57, 2003. Preliminary version in FOCS '01.

[18] E. Grigorescu, T. Kaufman, and M. Sudan. 2-transitivity is insufficient for local testability. *Electronic Colloquium on Computational Complexity (ECCC)*, (033), 2008.

[19] S. Halevy and E. Kushilevitz. Distribution-free property-testing. *SIAM J. Comput.*, 37(4):1107–1138, 2007. Preliminary version in RANDOM '03.

[20] R. Hamming. Error-detecting and error-correcting codes. In *Bell System Technical Journal*, volume 29(2), pages 147–160, 1950.

[21] T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM J. Comput.*, 33(6):1441–1483, 2004. Preliminary version in RANDOM-APPROX '03.

[22] S. Khot. On the power of unique 2-prover 1-round games. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, New York, NY, USA, 2002. ACM.

[23] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 146–154, Washington, DC, USA, 2004. IEEE Computer Society.

[24] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357, 2007. Preliminary version in FOCS '04.

[25] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *J. Comput. Syst. Sci.*, 72(6):1012–1042, 2006. Preliminary version in STOC '05.

[26] R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

[27] C. E. Shannon. A mathematical theory of communication (parts I and II). *Bell System Technical Journal*, XXVII:379–423, 1948.

A Restorer for Graph Bipartiteness

A.1 A Restorer for Bipartiteness in High Degree Graphs

We present $(\mathsf{InitDenseBipartite}, \mathsf{EvalDenseBipartite})$ — an $(\epsilon^2, O\left(\frac{1}{c}\epsilon\right))$-restorer for graph bipartiteness for graphs with minimum degree cN. The main idea is explained in Section 4.1. The restorer is described in Figures 4, 5.

The initializer selects a sample of vertices S_0 of size m_0. For all $x \in S_0$, we recursively define two series of sets: $A_i(x), B_i(x)$. Intuitively, the sets $A_i(x)$ correspond to x's side of the partition and the sets $B_i(x)$ correspond to the opposite side. Note that $A_i(x), B_i(x)$ are not limited to S_0 (that is, they have vertices in $V \setminus S_0$), however computing whether a vertex belongs to $A_i(x), B_i(x)$ requires only knowledge of S_0.

Definition 5. *For all $x \in S_0$, we define x's "interest sets" as follows.*

$$\begin{aligned}
A_0(x) &\stackrel{\text{def}}{=} \{x\}\,, \\
B_0(x) &\stackrel{\text{def}}{=} \Gamma(x)\,, \\
A_i(x) &\stackrel{\text{def}}{=} A_{i-1}(x) \cup \{v \in V : (\Gamma(v) \cap B_{i-1}(x))_{|S_0} \geq 4\epsilon m_0\}\,, \\
B_i(x) &\stackrel{\text{def}}{=} B_{i-1}(x) \cup \{v \in V : (\Gamma(v) \cap A_i(x))_{|S_0} \geq 4\epsilon m_0\}\,.
\end{aligned}$$

Algorithm InitDenseBipartite

Input: Query access to graph F.
Output: Initial state σ.
Code:

(1) Randomly select a sample S_0 of $m_0 = \Theta(\frac{1}{\epsilon^2})$ vertices.

(2) Randomly select a sample S_1 of $m_1 = \Theta(\frac{1}{\epsilon^2})$ *edges* (pairs of vertices).

(3) For all $x, y \in S_0$, query $F(x, y)$ to create the induced graph $F_{|S_0}$.

(4) For all $(u, v) \in S_1$ and for all $x \in S_0$, query $F(u, v), F(u, x), F(v, x)$.

(5) Set $Q \leftarrow \emptyset$, $T \leftarrow S_0$.

(6) While $|T| > 0$ do:

6a: Pick some $x \in T$ and compute $A_i(x)_{|S_0}$, $B_i(x)_{|S_0}$ starting from $i = 0$ and stopping at the first i where $\left|\left(A_i(x)_{|S_0} \setminus A_{i-1}(x)_{|S_0}\right)\right| \leq \epsilon m_0$. Denote this value of i by t_x and denote $A(x) = A_{t_x}(x)$, $B(x) = B_{t_x-1}(x)$.

6b: For every $e_j = (u_j, v_j) \in S_1$, if $F(u_j, v_j) = 1$ and either $\{u_j, v_j\} \subseteq A(x)$ or $\{u_j, v_j\} \subseteq B(x)$ then set $a_j = 1$. Otherwise $a_j = 0$.

6c: If $\sum_j a_j < \frac{7}{2}\epsilon m_1$ then set $Q \leftarrow Q \cup \{x\}$, $T \leftarrow T \setminus (A(x) \cup B(x))$. Otherwise set $T \leftarrow T \setminus \{x\}$.

(7) The output σ contains the pairs $(A_{t_x-1}(x)_{|S_0}, B_{t_x-1}(x)_{|S_0})$ for all $x \in Q$. Note that using these pairs it can easily be checked whether some vertex is in $A(x)$ or $B(x)$.

Figure 4: Algorithm InitDenseBipartite.

Proof of Theorem 4.1: The claims about query and time complexities are immediate from the description of (InitDenseBipartite, EvalDenseBipartite). Lemma A.2 guarantees that with probability $1 - \text{negl}(1/\epsilon)$ the restorer is properly initialized (see Definition 8). From Lemma A.1 it follows that H_σ is indeed bipartite and from Lemma A.8 is follows that proper initialization yields $\text{dist}(F, H_\sigma) = O\left(\frac{1}{c}\epsilon\right)$ so long as $\epsilon \leq c^2/23$. ∎

Let $G = (V, E_G)$ be a bipartite graph s.t. $\text{dist}(F, G) \leq \epsilon^2$. Further let V_1, V_2 be the partition of V in G. We assume w.l.o.g that $E_G \subseteq E$ and define $E_B = E \setminus E_G$, the set of violating edges. For all $v \in V$ we denote $\Gamma_B(v)$, the neighbors of v resulting from edges in E_B. We define the set of vertices that have many such neighbors.

Definition 6. $Z \stackrel{\text{def}}{=} \{v \in V : \Gamma_B(v) > \frac{4}{11}\epsilon N\}$.

The following lemma proves that regardless of how A, B are defined, the output graph is always bipartite. Denote $H_\sigma = (V, E_\sigma)$.

Lemma A.1. *The graph H_σ is bipartite.*

Proof Sketch: Assume towards contradiction that there is an odd cycle in H_σ. Namely $w_1, \ldots, w_{2k+1}$ s.t. $(w_i, w_{i+1}) \in E_\sigma$ for all $i = 1, \ldots, 2k$ and $(w_{2k+1}, w_1) \in E_\sigma$. From step 2 of the evaluator it follows that there exists an $x \in Q$ s.t. $w_1 \in A(x)$ or $w_1 \in B(x)$. Assume w.l.o.g that $w_1 \in A(x)$ for some $x \in Q$. Then by step 4 of the evaluator, it must be that $w_2 \in B(x)$. By induction it follows that $w_{2k+1} \in A(x)$ and hence by step 3 it must be that $(w_{2k+1}, w_1) \notin E_\sigma$ contradicting our assumption. ∎

Definition 7 (well represented sets)**.** *A set $U \subseteq V$ is* well represented *in S_0 if the following hold.*

1. *If $|U| > \frac{1}{2}\epsilon N$, then $\frac{9}{10} \cdot \frac{m_0}{N} \cdot |U| \leq \left|U_{|S_0}\right| \leq \frac{11}{10} \cdot \frac{m_0}{N} \cdot |U|$.*
2. *If $|U| \leq \frac{1}{2}\epsilon N$, then $\left|U_{|S_0}\right| \leq \frac{11}{20}\epsilon m_0$.*

Algorithm EvalDenseBipartite

Input: Initial state σ, input pair of vertices u, v. Query access to graph F.
Output: Value of adjacency function of graph $H_\sigma(u, v)$.
Code:

(1) Query $F(u, v)$. If $F(u, v) = 0$ then return 0.

(2) If $u \notin \bigcup_{x\in Q}(A(x) \cup B(x))$ or $v \notin \bigcup_{x\in Q}(A(x) \cup B(x))$ then return 0.

(3) For all $x \in Q$, if $\{u, v\} \subseteq A(x)$ or $\{u, v\} \subseteq B(x)$ then return 0.

(4) For all $x \in Q$, if $u \in A(x)$ and $v \notin B(x)$ or $v \in A(x)$ and $u \notin B(x)$ then return 0.

(5) Return 1.

Figure 5: Algorithm EvalDenseBipartite.

A set $Y \subseteq V \times V$ is well represented *in S_1 if the following hold.*

1. *If $|Y| > \frac{1}{2}\epsilon N^2$ then $\frac{9}{10} \cdot \frac{m_1}{N^2} \cdot |Y| \leq \left|Y_{|S_1}\right| \leq \frac{11}{10} \cdot \frac{m_1}{N^2} \cdot |Y|$.*
2. *If $|Y| \leq \frac{1}{2}\epsilon N^2$ then $\left|Y_{|S_1}\right| \leq \frac{11}{20}\epsilon m_1$.*

Definition 8 (proper initialization)**.** *The restorer is properly initialized if the following hold.*

1. *For all $x, y \in S_0$, $i \leq 1/\epsilon$ and also for all but $\mathsf{negl}(1/\epsilon)N$ vertices $y \in V \setminus S_0$, it holds that $\Gamma(y)$, $\Gamma_B(y)$, $A_i(x)$, $B_i(x)$, $\Gamma(y) \cap A_i(x)$, $\Gamma(y) \cap B_i(x)$ are well represented in S_0.*
2. *$V \setminus \bigcup_{x\in Q}(A(x) \cup B(x))$ is well represented in S_0.*
3. *For all $x \in S_0$, the set $\{(u, v) \in V \times V : F(u, v) = 1 \ \wedge (\{u, v\} \subseteq A(x) \ \vee \ \{u, v\} \subseteq B(x))\}$ is well represented in S_1.*
4. *Z is well represented in S_0.*

Lemma A.2. *The restorer is properly initialized with probability at least $1 - \mathsf{negl}(1/\epsilon)$.*

The proof is by applying Chernoff's bound and is omitted.

Lemma A.3. *Let Q be as defined in the initializer. Let $\epsilon \leq c^2/23$, then if the restorer is properly initialized then $|Q| \leq \frac{2}{c}$.*

Proof Sketch: We show that whenever x is appended to Q, at least $\Theta(cm_0)$ vertices are removed from T.

Let Q, T denote the set just before x is appended. Then since for all $y \in Q$, $A(y), B(y)$ are removed from T, it follows that $x \notin A(y) \cup B(y)$ for all $y \in Q$. Thus $\Gamma(x)_{|S_0}$ contains at most:

1. $4\epsilon m_0$ vertices of $A_{t_y-1}(y)_{|S_0}$ (since $x \notin B_{t_y-1}(y)$).
2. ϵm_0 vertices of $(A_{t_y}(y) \setminus A_{t_y-1}(y))_{|S_0}$ (since this set's size is at most ϵm_0).
3. $4\epsilon m_0$ vertices of $B_{t_y-1}(y)_{|S_0}$ (since $x \notin A_{t_y}(y)$).

On the other hand, $\Gamma(x)_{|S_0}$ is of size at least $\frac{9}{10}cm_0$ and therefore when x is appended to Q, the number of vertices removed from T is:

$$\left|((A(x) \cup B(x))_{|S_0}) \setminus \bigcup_{y\in Q}((A(y) \cup B(y))_{|S_0})\right| \geq \left(\frac{9}{10}c - 9\epsilon\,|Q|\right) \cdot m_0$$

For $\epsilon \leq c^2/23$ we get that $|Q| \leq \frac{2}{c}$. ∎

Now we show that for any $x \in S_0$, the cut between $A(x) \cup B(x)$ and the rest of the graph is bounded in size.

Lemma A.4. *If the restorer is properly initialized and $\epsilon \leq c^2/23$, then the cut $(A(x) \cup B(x), V \setminus (A(x) \cup B(x)))$ contains at most $11\epsilon N^2$ edges.*

Proof Sketch: Let $v \notin A(x) \cup B(x)$, then by the same argument as in Lemma A.3, $\Gamma(v)$ contains at most $9\epsilon m_0$ vertices in $(A(x) \cup B(x))_{|S_0}$. Therefore from proper initialization, all but a negligible number of such vs contain at most $10\epsilon N$ vertices in $A(x) \cup B(x)$. Therefore, the cut between $A(x) \cup B(x)$ and the rest of the graph contains at most $N \cdot 10\epsilon N + \mathsf{negl}(1/\epsilon)N \cdot N \leq 11\epsilon N^2$ edges. ∎

Now we show that for $x \in S_0 \setminus Z$, the separation of $A(x)$ from $B(x)$ doesn't remove many of the original edges. Recall that $G = (V_1 \cup V_2, E_G)$ is a bipartite graph whose distance from $F = (V_1 \cup V_2, E)$ is at most ϵ^2.

Lemma A.5. *Let $x \in S_0 \setminus Z$ and let $Y = \{(u,v) \in E_G : (u,v \notin Z) \wedge (\{u,v\} \subseteq A(x) \vee \{u,v\} \subseteq B(x))\}$. Then in a proper initialization $|Y| \leq \mathsf{negl}(1/\epsilon)N^2$.*

Proof Sketch: Let $(u,v) \in V_1 \times V_2$. Assume that $(u,v) \in Y$ and $\{u,v\} \subseteq A(x)$. Then all but a negligible fraction of vertices u, v have at least $4\epsilon m_0$ neighbors in $B_{t_x-1}(x)_{|S_0}$. For such u, v, good representation guarantees at most $\frac{2}{5}\epsilon m_0$ of these neighbors result from E_B (violating edges). Therefore $B_{t_x-1}(x)_{|S_0}$ contains at least $\frac{18}{5}\epsilon m_0$ vertices from V_1 and at least $\frac{18}{5}\epsilon m_0$ vertices from V_2. Note that $|Z| \leq \frac{11}{4}\epsilon N$ and therefore by well representation, $\left|Z_{|S_0}\right| \leq \frac{16}{5}\epsilon m_0$. Therefore there are at least $\frac{2}{5}\epsilon m_0$ vertices from V_1 and at least $\frac{2}{5}\epsilon m_0$ vertices from V_2 in $(B_{t_x-1}(x)_{|S_0}) \setminus Z$. Let w_1, w_2 be such vertices. Each of them has at least $4\epsilon m_0$ neighbors in $A_{t_x-1}(x)_{|S_0}$. Repeating the process $O(1/\epsilon)$ times yields that there are at least $\frac{18}{5}\epsilon m_0$ vertices from each V_1, V_2 in $B_0(x)_{|S_0} = \Gamma(x)_{|S_0}$. Note that at least on of these sets must be from violating edges, therefore $\left|\Gamma_B(x)_{|S_0}\right| \geq \frac{18}{5}\epsilon m_0$. Therefore $|\Gamma_B(x)| \geq \frac{10}{11} \cdot \frac{18}{5}\epsilon N > \frac{4}{11}\epsilon N$ in contradiction to $x \notin Z$. A similar argument can be used to show the same for $\{u,v\} \subseteq B(x)$.

Therefore Y contains only edges for which at least one end is in a set of size $\mathsf{negl}(1/\epsilon)N$ and hence $|Y| \leq \mathsf{negl}(1/\epsilon)N^2$. ∎

We now conclude that all vertices in $S_0 \setminus Z$ pass the test in steps 6b, 6c of the initializer.

Lemma A.6. *For all $x \in S_0 \setminus Z$, denote $Y' = \{(u,v) \in S_1 : F(u,v) = 1 \ \wedge \ (\{u,v\} \subseteq A(x) \vee \{u,v\} \subseteq B(x))\}$. Then in a proper initialization it holds that $|Y'| \leq \frac{7}{2}\epsilon m_1$.*

Proof Sketch: We use case analysis to count the total number of edges in

$$\{(u,v) \in V \times V : F(u,v) = 1 \ \wedge \ (\{u,v\} \subseteq A(x) \vee \{u,v\} \subseteq B(x))\} \ :$$

1. $(u,v) \notin E_G$. These are violating edges and therefore there are at most $\epsilon^2 N^2$ such edges.
2. $u \in Z$ or $v \in Z$. Since $|Z| \leq \frac{11}{4}\epsilon N$, there are at most $\frac{11}{4}\epsilon N^2$ such edges.
3. The remaining case is exactly the set Y from Lemma A.5 and $|Y| \leq \mathsf{negl}(1/\epsilon)N^2$.

We conclude that there are at most $3\epsilon N^2$ such edges. Since this set is well represented in S_1, the result follows.
∎

We still need to show that step 2 doesn't remove too many edges. We show that almost all vertices are in $\bigcup_{x \in Q}(A(x) \cup B(x))$.

Lemma A.7. *In a proper initialization, its holds that $\left|V \setminus \bigcup_{x \in Q}(A(x) \cup B(x))\right| \leq 4\epsilon N$.*

Proof Sketch: Let $y \in \left(S_0 \setminus \bigcup_{x \in Q}(A(x) \cup B(x))\right)$. It must be the case that in step 6c, y did not meet the condition. Therefore by Lemma A.6, $y \in Z \cap S_0$. Therefore $\left|S_0 \setminus \bigcup_{x \in Q}(A(x) \cup B(x))\right| \leq \left|Z_{|S_0}\right| \leq \frac{16}{5}\epsilon m_0$. Therefore by well representation $\left|V \setminus \bigcup_{x \in Q}(A(x) \cup B(x))\right| \leq 4\epsilon N$. ∎

We conclude that the restored graph H_σ is $O(\frac{1}{c}\epsilon)$-close to F.

Lemma A.8. *If $\epsilon \leq c^2/23$ then in a proper initialization* $\mathrm{dist}\,(F, H_\sigma) = O\left(\frac{1}{c}\epsilon\right)$.

Proof Sketch: We go over each step of the evaluator and see how many edges can be removed.

1. Step 2. From Lemma A.7, at most $4\epsilon N \cdot N = 4\epsilon N^2$ edges are removed.
2. Step 3. From well representation and by definition of steps 6b, 6c of the initializer, at most $\frac{7}{2} \cdot \frac{10}{9}\epsilon N^2$ edges are removed *for every* $x \in Q$. Hence the total is $\frac{2}{c} \cdot \frac{7}{2} \cdot \frac{10}{9}\epsilon N^2 < \frac{8}{c}\epsilon N^2$ edges.
3. Step 4. At this step, an edge is removed only if one of its endpoints is in $A(x) \cup B(x)$ for some $x \in Q$ and the other isn't. By Lemma A.4 There are at most $11\epsilon N^2$ such edges for every x, thus a total of at most $\frac{22}{c}\epsilon N^2$ edges are removed.

Therefore a total of at most $O\left(\frac{1}{c}\epsilon N^2\right)$ edges are removed. ∎

A.2 Deriving a Restorer for General Graphs

Procedure EstimateNeighborhood

Input: Test set $T \subseteq V$, input node v. Query access to graph F.
Output: Estimation of $\frac{|\Gamma(v)|}{N}$.
Code:

(1) Query F on all pairs $\{v\} \times T$.

(2) Return $\frac{1}{|T|}\sum_{u \in T} F(v, u)$.

Figure 6: Procedure EstimateNeighborhood.

Algorithm InitBipartite

Input: Query access to graph F.
Output: Initial state σ.
Code:

(1) Randomly select a set of vertices $T \subseteq V$ of size $\Theta(1/\epsilon^2)$.

(2) Simulate $\mathsf{InitDenseBipartite}^F()$ with parameters $c = 5\epsilon, \epsilon' = 25a\epsilon^2$.

(3) Whenever the simulation needs to sample a random vertex v:

- 3a: Randomly select $x \xleftarrow{R} V$ and run $\mathsf{EstimateNeighborhood}^F(T, x)$.
- 3b: If $\mathsf{EstimateNeighborhood}^F(T, x) > 6\epsilon$, continue simulation with $v = x$.
- 3c: Otherwise, go to 3a.

(4) Return the initial state σ returned by the simulated InitDenseBipartite.

Figure 7: Algorithm InitBipartite.

The restorer (InitBipartite, EvalBipartite) is presented in Figures 7, 8 respectively (the constant a is as defined in Theorem 4.1). An additional auxiliary Procedure EstimateNeighborhood is presented in Figure 6. A proof of correctness follows.

Proof of Theorem 4.2: Consider running (InitDenseBipartite, EvalDenseBipartite) with parameters $c = 5\epsilon, \epsilon' = 25a\epsilon^2$ (where a is as defined in Theorem 4.1 on graph F': a graph obtained by repeatedly removing all vertices with less than $5\epsilon N$ neighbors from F and maybe some vertices with less than $7\epsilon N$ neighbors. With probability $1 - \mathsf{negl}(1/\epsilon)$, the initializer $\mathsf{InitBipartite}^F$ perfectly simulates $\mathsf{InitDenseBipartite}^{F'}$.

Algorithm EvalBipartite

Input: Initial state σ, input pair of vertices (u, v). Query access to graph F.
Output: Value of adjacency function of graph $H_\sigma(u, v)$.
Code:
(1) Simulate $\mathsf{EvalDenseBipartite}^F(\sigma, (u, v))$ with parameters $c = 5\epsilon, \epsilon' = 25a\epsilon^2$ and return the result.

Figure 8: Algorithm EvalBipartite.

The restored graph H_σ must be bipartite by Lemma A.1. For vertices u, v in F', the evaluator $\mathsf{EvalBipartite}^F(\sigma, (u, v))$ outputs the same value as $\mathsf{EvalDenseBipartite}^{F'}(\sigma, (u, v))$ and thus removes at most $O\left(\frac{1}{c}\epsilon'\right) N^2 = O(\epsilon) N^2$ edges. All vertices not in F' neighbor $O(\epsilon) N^2$ edges altogether. Therefore $\mathrm{dist}(H_\sigma, F) = O(\epsilon)$. By Theorem 4.1 we obtain an $(\epsilon'^2, O\left(\frac{1}{c}\epsilon'\right)) = (\epsilon^4, O(\epsilon))$-restorer for the general case. ∎

B Analysis of the Restorer for ρ-Clique

Statement and analysis of Lemmas B.1, B.2, B.3 follow. The following definition provides with a way to bound sets that are random variables.

Definition 9 ((α, β)-Bounded Sets)**.** *Let X, L, U be (possibly random variable) sets of vertices. Sets (L, U) are said to (α, β)-bound X if with probability at least $1 - \alpha$ it holds that $|L \setminus X| \leq \beta N$ and $|X \setminus U| \leq \beta N$.*

Note that for small α, β, this means that approximately $L \subseteq X \subseteq U$. We omit the parameters and say that X is *bounded* by (L, U) if it is $(\mathsf{negl}(1/\epsilon), \mathsf{negl}(1/\epsilon))$-bounded by (L, U).

Let $G = (V, E_G)$ be a graph that is ϵ^3-close to F and has a ρ-clique. Since w.l.o.g we may assume that $E_G \supseteq E$, we may refer to F as resulting from a process where we start with G and remove at most $\epsilon^3 N^2$ edges. Let C be a ρ-clique in G.

Definition 10 (Proper initialization)**.** *The restorer is* properly initialized *if for* any U, T, W *defined in the initializer* InitClique *it holds that:*

1. *There exists $U' \subseteq C_{|S_1}$ of size $\frac{\rho}{2} m_1$ such that $K_{2\epsilon^2/\rho}(U')$ is bounded by $(K_{\epsilon^2/\rho}(C), K_{3\epsilon^2/\rho}(C))$.*
2. $\left| \frac{|T|}{m_2} - \frac{\left|K_{2\epsilon^2/\rho}(U)\right|}{N} \right| < \frac{\epsilon}{2}$.
3. *If $\frac{|T|}{m_2} \geq \frac{\rho}{2}$ then $K_{2\epsilon}(T)$ is bounded by $(K_\epsilon(K_{2\epsilon^2/\rho}(U)), K_{3\epsilon}(K_{2\epsilon^2/\rho}(U)))$.*
4. $\left| \frac{|W|}{m_3} - \frac{\left|K_{2\epsilon^2/\rho}(U) \cap K_{2\epsilon}(T)\right|}{N} \right| < \frac{\epsilon}{2}$.

Lemma B.1. *The restorer is properly initialized with probability* $1 - \mathsf{negl}(1/\epsilon)$.

The proof is by standard application of Chernoff's bound and is omitted from this extended abstract.

Lemma B.2. *Let $(U^*, T^*, w^*, h) = \mathsf{InitClique}^F()$. Then if the restorer is properly initialized and $U^* \neq \emptyset$, then there exists at least one set $C' \subseteq (K_{2\epsilon^2/\rho}(U^*) \cap K_{2\epsilon}(T^*))$ of size $(\rho - 6\epsilon)N$. For each such set, C' is missing $O(\epsilon) N^2$ edges to being a clique.*

Proof: Since $\frac{|W|}{m_3} \geq (\rho - 5\epsilon)$ then $\left|K_{2\epsilon^2/\rho}(U^*) \cap K_{2\epsilon}(T^*)\right| \geq (\rho - 6\epsilon)N$ and thus has subsets as required. Let C' be such subset and denote $X = K_{2\epsilon^2/\rho}(U^*)$. For all but a negligible fraction of the vertices $v \in C'$ it holds that $v \in K_{3\epsilon}(X)$ and therefore there exist $(\rho - 6\epsilon - \mathsf{negl}(1/\epsilon))N$ vertices in C' that have more than $(1 - 3\epsilon)|X|$

neighbors in X. Specifically this means that they have at least $|C'|-3\epsilon\,|X| \geq |C'|-3\epsilon N \geq (\rho-9\epsilon-\mathsf{negl}(1/\epsilon))N$ neighbors in C'.

Therefore C' is missing at most

$$(\rho-6\epsilon)^2N^2-(\rho-6\epsilon-\mathsf{negl}(1/\epsilon))N\cdot(\rho-9\epsilon-\mathsf{negl}(1/\epsilon))N = O\left(\rho\epsilon\right)N^2 = O\left(\epsilon\right)N^2$$

edges to being a clique. ∎

But does InitClique ever succeed in finding U^*?

Lemma B.3. *Let $F=(V,E)$ be ϵ^3-close to having a ρ-clique. Then if the restorer is properly initialized, then Algorithm* $\mathsf{InitClique}^F()$ *returns $U^*\neq\emptyset$.*

Proof: Let $G=(V,E_G)$ be a graph that is ϵ^3-close to F as described above, and let U',C be the sets for which item 1 of Definition 10 holds. A simple counting argument shows that there exists a subset $\tilde{C}\subseteq C$ of size $(\rho-\epsilon)N$ such that each $v\in\tilde{C}$ is adjacent, in F, to at least $(\rho-\epsilon^2)N$ vertices of C. Otherwise, more than ϵN vertices lost more than ϵ^2N edges in contradiction to the distance guarantee. Using our notation, we have that $\tilde{C}\subseteq K_{\epsilon^2/\rho}(C)$.

By definition, $K_{2\epsilon^2/\rho}(U')$ contains all but a negligible fraction of $\tilde{C}$. On the other hand, all but a negligible fraction of $K_{2\epsilon^2/\rho}(U')$ is contained in $K_{3\epsilon^2/\rho}(C)$ and thus all but a negligible fraction of the vertices of $K_{2\epsilon^2/\rho}(U')$ neighbor at least $(1-3\epsilon^2/\rho)\,|C| = (\rho-3\epsilon^2)N$ of the vertices of C. We deduce that all but a negligible fraction of $K_{2\epsilon^2/\rho}(U')$ neighbor all but $3\epsilon^3N$ of the vertices of $\tilde{C}$.

Denote $X=\tilde{C}\setminus K_\epsilon(K_{2\epsilon^2/\rho}(U'))$. We use a simple counting argument on the number of edges between $\tilde{C}$ and $K_{2\epsilon^2/\rho}(U')$. A lower and upper bound on this size yield:

$$(\left|K_{2\epsilon^2/\rho}(U')\right|-\mathsf{negl}(1/\epsilon))\cdot(\left|\tilde{C}\right|-3\epsilon^2N) \leq |X|\,(1-\epsilon)\left|K_{2\epsilon^2/\rho}(U')\right| + (\left|\tilde{C}\right|-|X|)\cdot\left|K_{2\epsilon^2/\rho}(U')\right| \ .$$

Using basic arithmetic we have that $|X|\leq(3\epsilon+\mathsf{negl}(1/\epsilon))N$. We deduce that all but $(3\epsilon+\mathsf{negl}(1/\epsilon))N$ of $\tilde{C}$ are in $K_\epsilon(K_{2\epsilon^2/\rho}(U'))$ and therefore all but $(3\epsilon+\mathsf{negl}(1/\epsilon))N$ of $\tilde{C}$ are in $K_{2\epsilon}(T')$ (for T' that matches U').

We conclude that there are at least $(\rho-4\epsilon-\mathsf{negl}(1/\epsilon))N$ vertices in $K_{2\epsilon^2/\rho}(U')\cap K_{2\epsilon}(T')$ and therefore $\frac{|W|}{m_3}\geq\rho-5\epsilon$. Therefore for $U=U'$, InitClique reaches step 4c. ∎

C Restorer for 1-Dimensional Function Monotonicity

We present the restorer discussed in Section 5 and provide a detailed analysis thereof, concluded with the proof of Theorem 5.1. We start with a presentation of index-value trees [25].

Index-value trees. An index-value tree is a a binary tree in which each node v is characterized by the following properties: $\mathrm{ind}(v),\mathrm{lind}(v),\mathrm{rind}(v)\in[n]$, $\mathrm{val}(v),\mathrm{lval}(v),\mathrm{rval}(v)\in\mathcal{R}$ (we note that $\mathrm{lval},\mathrm{rval}$ can also be $\pm\infty$ which is handled in the standard way). Additional sets associated with v are defined by its properties:

$$\begin{aligned}
I(v) &\stackrel{\text{def}}{=} \{\mathrm{lind}(v),\ldots,\mathrm{rind}(v)\}\ ,\\
S(v) &\stackrel{\text{def}}{=} \{j\in I(v):\mathrm{lval}(v)<f(j)\leq\mathrm{rval}(v)\}\ ,\\
B_{\leq}(v) &\stackrel{\text{def}}{=} \{j\in S(v):j\leq\mathrm{ind}(v)\text{ and }f(j)>\mathrm{val}(v)\}\ ,\\
B_{>}(v) &\stackrel{\text{def}}{=} \{j\in S(v):j>\mathrm{ind}(v)\text{ and }f(j)\leq\mathrm{val}(v)\}\ .
\end{aligned}$$

We denote $s(v)=|S(v)|$, $b_{\leq}(v)=|B_{\leq}(v)|$, $b_{>}(v)=|B_{>}(v)|$.

The tree is defined recursively as follows.

1. Let r be the root of the tree. Then $\mathrm{lind}(r)=1$, $\mathrm{rind}(r)=n$, $\mathrm{ind}(r)=\lceil n/2\rceil$, $\mathrm{lval}(v)=-\infty$, $\mathrm{lval}(v)=\infty$. The value $\mathrm{val}(v)$ is defined by the specific construction.

2. For any node p, if $\mathrm{lind}(p) = \mathrm{rind}(p)$ or if $S(v) = \emptyset$ then p is a leaf. Otherwise let ℓ, r denote its left and right children respectively, then: $\mathrm{lval}(\ell) = \mathrm{lval}(p)$, $\mathrm{rval}(\ell) = \mathrm{val}(p)$, $\mathrm{lind}(\ell) = \mathrm{lind}(p)$, $\mathrm{rind}(\ell) = \mathrm{ind}(p)$ and $\mathrm{lval}(r) = \mathrm{val}(p)$, $\mathrm{rval}(r) = \mathrm{rval}(p)$, $\mathrm{lind}(r) = \mathrm{ind}(p) + 1$, $\mathrm{rind}(r) = \mathrm{rind}(p)$.
3. The values of $\mathrm{ind}(v), \mathrm{val}(v)$ for all nodes are determined by the specific tree construction algorithm so that $\mathrm{lind}(v) \leq \mathrm{ind}(v) < \mathrm{rind}(v)$ and $\mathrm{val}(v) = f(j)$ for some $j \in S(v)$.

For the entire tree, we define $B \stackrel{\text{def}}{=} \bigcup_{v \in T}(B_{\leq}(v) \cup B_{>}(v))$. For any index-value tree T constructed for f, it holds that f is monotone on $[n] \setminus B$ [25].

We use (a slight modification of) a construction described in [25] with the properties quoted below.

Corollary C.1 (Construction from [25])**.** *Let $f : [n] \to \mathcal{R}$ be some function. For all $\delta > 0$, define $\gamma = \delta / \log(n)$. Let $r_1 \in \{0,1\}^{m_1}$ be a random string of length $m_1 = \Theta(\log^2(n)/\gamma^2)$.*

There exists a procedure FindPath *that takes randomness r_1, an input $j \in [n]$ and has query access to f such that:*

1. FindPath *makes at most $\Theta(\log^2(n)/\gamma^2)$ queries to f.*
2. *For all r_1 there exists an index-value tree T_{r_1}, some of which nodes are marked as* empirically small*.*
3. *For all j, $\mathsf{FindPath}^f(r_1, j)$ returns a path in T_{r_1} starting from the root and ending in the first node v for which $j \in I(v)$ and either v is a leaf or v is an empirically small node. For each node x along the returned path, the algorithm returns $\mathrm{lind}(x), \mathrm{ind}(x), \mathrm{rind}(x), \mathrm{lval}(x), \mathrm{val}(x), \mathrm{rval}(x)$.*
4. *With probability at least $1 - 1/\mathrm{poly}(n)$, the tree T_{r_1} has the following properties:*
   - *4a: The tree height is at most $2 \log n$.*
   - *4b: Each empirically small node v either satisfies $s(v) \leq \gamma |I(v)|$ or one of its ancestors is empirically small.*
   - *4c: $\frac{1}{n} \sum_{v \in T_{r_1}} (b_{\leq}(v) + b_{>}(v)) \leq 2\epsilon_{\mathrm{mon}}(f) + 2\delta$.*

Proof: The proof is simply quoting the properties of [25]'s "Algorithm 2" combined with an additional observation. Algorithm 2 uses fresh randomness for computing each node in the path every time the algorithm is run. Since consecutive runs may not agree, there is no implicit tree T_{r_1}. In the analysis, however, it is shown that the error probability could be made smaller than any $1/\mathrm{poly}(n)$. Therefore, since there are at most $\Theta(n)$ possible queries, we can use a single random string to gain consistency and still have success probability at least $1 - 1/\mathrm{poly}(n)$. ∎

Definition 11 (semi-proper randomness)**.** *A random string $r_1 \in \{0,1\}^{m_1}$ is* semi-proper *if T_{r_1} satisfies item 4 of Corollary C.1.*

Definition 12 (small nodes)**.** *Let r_1 be semi-proper. Then a node $v \in T_{r_1}$ is* small *if it is empirically small or if one of its ancestors is empirically small. Denote $B_1 \stackrel{\text{def}}{=} \{j \in [n] : j \in B \text{ or } \exists_{\text{small } v \in T_{r_1}}.\ j \in I(v)\}$.*

B_1 extends B as it includes all of B but also indices that are not in B but are "hard to find" (since upon reaching a small node, FindPath may stop).

For any node v in T_{r_1}, we would like to estimate the fraction of the indices in $I(v)$ that are not in B_1. That is the fraction of indices in $I(v)$ that are both not in B and are reachable. Formally define $\beta(v) = \frac{|I(v) \setminus B_1|}{|I(v)|}$. Procedure FindRatio described in Figure 9 estimates the value of $\beta(v)$.

Definition 13 (proper randomness)**.** *Random strings $r_1 \in \{0,1\}^{m_1}, r_2 \in \{0,1\}^{m_2}$ are* proper *if r_1 is semi-proper and for* all *$v \in T_{r_1}$ it holds that $\left|\mathsf{FindRatio}^f(r_1, r_2, v) - \beta(v)\right| \leq \gamma/4$.*

Lemma C.2. *Let $r_1 \in \{0,1\}^{m_1}, r_2 \in \{0,1\}^{m_2}$ be random strings. Then with probability at least $1 - 1/\mathrm{poly}(n)$ it holds that r_1, r_2 are proper.*

Procedure FindRatio

Input: Strings $r_1 \in \{0,1\}^{m_1}$, $r_2 \in \{0,1\}^{m_2}$ where $m_1 = \Theta(\log^2(n)/\gamma^2)$, $m_2 = \Theta(\log^2/\gamma^2)$. Node v (given as the set of parameters defining it). Query access to function f.
Output: Estimation of $\beta(v)$ in tree T_{r_1}.
Code:

(1) Let $s = \Theta(\log(n)/\gamma^2)$. Compute the path to v by running $\mathsf{FindPath}^f(r_1, j)$ for some $j \in I(v)$.

(2) If v is small then return 0.

(3) If $|I(v)| \leq s$, go over all indices $j \in I(v)$, run $\mathsf{FindPath}^f(r_1, j)$, check if $j \in B_1$ and compute $\beta(v)$.

(4) Otherwise, randomly sample s indices of $I(v)$ using r_2 as random bits (note that r_2 is sufficiently long). For each such index j run $\mathsf{FindPath}^f(r_1, j)$ and check if $j \in B_1$. Output the ratio of sampled indices that are not in B_1.

Figure 9: Procedure FindRatio.

Proof Sketch: By Corollary C.1, r_1 is semi-proper with probability at least $1 - 1/\mathrm{poly}(n)$. We use Chernoff's bound to bound the probability that r_1 is semi-proper but r_1, r_2 are not proper. The expected ratio of $I(v) \setminus B_1$ in the sample is $\beta(v)$, the chance of deviating by more than $\gamma/4$ is bounded by $\frac{1}{\mathrm{poly}(n)}$. We can take the constant in the $\Theta(\cdot)$ notation to be large enough so that even after applying a union bound over all $\Theta(n)$ possible nodes we still have a chance of failure of at most $1/\mathrm{poly}(n)$ and the result follows. ∎

We now define another useful extension of the set of "problematic" nodes in the tree.

Definition 14 (deserted nodes)**.** *For any proper r_1, r_2, for any level $h = 0, \ldots, 2\log n$ of T_{r_1}, define $\eta_h \stackrel{\text{def}}{=} (1 + 2\log n - h) \cdot \frac{\gamma}{2}$. A node v of level h in T_{r_1} is* deserted *if either $\mathsf{FindRatio}^f(r_1, r_2, v) < \eta_h$ or if one of its ancestors is deserted. Define $B_2 \stackrel{\text{def}}{=} \{j \in [n] : \exists_{\text{deserted } v \in T_{r_1}}.\ j \in I(v)\}$.*

Again we extend our set of "non-treatable" indices. Our new set B_2 has a hierarchial structure which will enable us to efficiently search the tree for indices either in or out of B_2. Details follow.

Lemma C.3. *Let r_1, r_2 be proper. Let $v \in T_{r_1}$ and let v_0, v_1 be its children. Then if v_0, v_1 are both deserted then v is deserted.*

Proof: Obviously if either v_0 or v_1 "inherited" the property then v must also have it. Let h be the level of v in the tree. If $\mathsf{FindRatio}^f(r_1, r_2, v_{0/1}) < \eta_{h+1}$ then by Lemma C.2 it holds that $\beta(v_{0/1}) < \eta_{h+1} + \frac{\gamma}{4}$. Thus

$$\beta(v) = \frac{\beta(v_0)\,|I(v_0)| + \beta(v_1)\,|I(v_1)|}{|I(v)|} < (\eta_{h+1} + \frac{\gamma}{4})\frac{|I(v_0)| + |I(v_1)|}{|I(v)|} = \eta_{h+1} + \frac{\gamma}{4}\,.$$

Applying Lemma C.2 to v yields $\mathsf{FindRatio}^f(r_1, r_2, v) < \eta_{h+1} + \frac{\gamma}{2} = \eta_h$ and the result follows. ∎

Lemma C.4. *Let r_1, r_2 be proper. Then $B_1 \subseteq B_2$.*

Proof: Let $j \in B_1$ then consider the leaf v that contains j (and possibly other indices), if $j \in B$ then $s(v) = 0$ and hence $\mathsf{FindRatio}^f(r_1, r_2, v) < \frac{\gamma}{4} < \eta_h$. If $j \in B_1 \setminus B$ then v is small and thus deserted, yielding $j \in B_2$. ∎

Lemma C.5. *Let r_1, r_2 be proper. Then $|B_2| \leq (2\epsilon_{\mathrm{mon}}(f) + 4\delta)n$.*

Proof: By item 4c of Corollary C.1, it follows that $|B| \leq (2\epsilon_{\mathrm{mon}}(f) + 2\delta)n$.

We now bound $|B_1 \setminus B|$. Let V be the set of vertices in T_{r_1} that are small but have no small ancestor. If $j \in B_1 \setminus B$ then $j \in S(v)$ for some $v \in V$ and from item 4b it follows that $\sum_{v \in V} |S(v)| \leq \sum_{v \in V} \gamma \, |I(v)| \leq \gamma n$. Therefore $|B_1 \setminus B| \leq \gamma n$.

We similarly bound $|B_2 \setminus B_1|$. Let U be the set of vertices in T_{r_1} that are deserted and have no deserted ancestor. All indices $j \in B_2 \setminus B_1$ belong to $I(v) \setminus B_1$ for some $v \in U$. Therefore

$$|B_2 \setminus B_1| \leq \sum_{v \in U} |I(v) \setminus B_1| = \sum_{v \in U} \beta(v) \, |I(v)| < \sum_{v \in U} (\eta_{h(v)} + \gamma/4) \, |I(v)| \leq (\gamma + \delta) \sum_{v \in U} |I(v)| \leq (\gamma + \delta) n \, .$$

We conclude that $|B_2| \leq (2\epsilon_{\mathrm{mon}}(f) + 3\delta + 2\gamma)n \leq (2\epsilon_{\mathrm{mon}}(f) + 4\delta)n$. ∎

Next we present Procedure FindVal (Figure 10). We show that for all i it can efficiently find the closest value $j < i$ such that $j \notin B_2$. The restorer will replace the values of all $i \in B_2$ with the value of the nearest $j \notin B_2$ (from below). Thus replacing (among others) all values of $i \in B$ and making the function monotone.

Procedure FindVal

Input: Random strings r_1, r_2. Index $i \in [n]$. Query access to f.
Output: Maximal $j \in ([n] \setminus B_2)$ such that $j < i$, if such j doesn't exist, (-1) is returned.
Code:

(1) Use $\mathsf{FindPath}^f(r_1, i)$ and $\mathsf{FindRatio}^f(r_1, r_2, \cdot)$ to find the first ancestor of the leaf containing i that is not deserted and whose left child is not deserted, denote it by v. If no such node exists, return (-1).

(2) Let u be the left child of v. Find its rightmost non-deserted leaf by going down the tree, taking the right child if it is not deserted and the left otherwise. Return the index in the non-deserted leaf.

Figure 10: Procedure FindVal.

Lemma C.6. *Let r_1, r_2 be proper. Then $\mathsf{FindVal}^f(r_1, r_2, i)$ makes $O(\log^4(n)/\gamma^4)$ queries to f and for $i \in B_2$ it returns the value of the maximal $j \in ([n] \setminus B_2)$ such that $j < i$. If such index does not exist then it returns (-1).*

Proof: If r_1, r_2 are proper, then the procedure calls FindPath and FindRatio at most $O(\log(n))$ times and thus it makes at most $O(\log^4(n)/\gamma^4)$ oracle calls.

If such j exists, then by definition all of its ancestors are not deserted. Let v be the least common ancestor of j and i. Since $j < i$, it must be a descendant of the left child of v. Therefore if such j exists then i has to have an ancestor that is not deserted and whose left child is not deserted. Thus the procedure returns (-1) if and only if no such j exists.

Since by Lemma C.3 if v is not deserted then it has at least one non-deserted child, then in the second step of the procedure, we first check if the right child is non-deserted, if so this means that j is in that subtree. Otherwise the whole subtree is deserted and j is in the left child. Therefore we always find the rightmost (that is, maximal) non-deserted leaf that is to the left of i, such leaf must contain $j \notin B_2$. ∎

The restorer $(\mathsf{InitMonotone}, \mathsf{EvalMonotone})$ is described in Figures 11, 12. Note that EvalMonotone uses Procedures FindPath, FindRatio, FindVal that have been described and analyzed above. We can now finally prove Theorem 5.1 from Section 5.

Proof of Theorem 5.1: The claims on the number of queries and on the running time follow immediately from definition and from Lemma C.6.

It remains to show that if r_1, r_2 are proper (which happens with probability at least $1 - 1/\mathrm{poly}(n)$), then h_σ is monotone and $\mathrm{dist}(f, h_\sigma) \leq 2\epsilon_{\mathrm{mon}}(f) + 4\delta$. From here on let r_1, r_2 be proper.

We note that $\{i \in [n] : f(i) \neq h_\sigma(i)\} \subseteq B_2$. Thus by Lemma C.5 we get that:

$$\mathrm{dist}(f, h_\sigma) = \Pr_{i \in [n]}[f(i) \neq h_\sigma(i)] \leq \Pr[i \in B_2] = \frac{|B_2|}{n} \leq 2\epsilon_{\mathrm{mon}}(f) + 4\delta \, .$$

Algorithm InitMonotone

Input: Query access to function f.
Output: Initial state $\sigma = (r_1, r_2)$.
Code:

(1) Randomly select $r_1 \xleftarrow{R} \{0,1\}^{m_1}$, $r_2 \xleftarrow{R} \{0,1\}^{m_2}$ for m_1, m_2 defined in Procedure FindRatio.

(2) Return $\sigma = (r_1, r_2)$.

Figure 11: Algorithm InitMonotone.

Algorithm EvalMonotone

Input: Initial state $\sigma = (r_1, r_2)$. Index $i \in [n]$. Query access to function f.
Output: Value of restored function $h_\sigma(i)$.
Code:

(1) Use $\mathsf{FindPath}^f(r_1, r_2, i)$ and $\mathsf{FindRatio}^f(r_1, r_2, \cdot)$ to check if $i \in B_2$.

(2) If $i \notin B_2$ then return $h_\sigma(i) = f(i)$.

(3) If $i \in B_2$ then compute $j = \mathsf{FindVal}^f(r_1, r_2, i)$. If $j = (-1)$ then return $h_\sigma(i) = \min \mathcal{R}$ (minimal value of $\mathcal{R}$), otherwise return $h_\sigma(i) = f(j)$.

Figure 12: Algorithm EvalMonotone.

As to monotonicity, note that $B_2 \supseteq B$ and therefore f is monotone on $[n] \setminus B_2$. Clearly h_σ does not violate this monotonicity as by Lemma C.6, for each value in $i \in B_2$ it takes $h_\sigma(i) = f(j)$ where j is the maximal index that is smaller than i and not in B_2 (or the smallest possible value if there is no such index). Therefore $h_\sigma(i)$ is at least as big as $f(j)$ but at most as big as $f(k)$ for any $B_2 \not\ni k > i$. ∎