Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 34 (2021)

Robust Self-Ordering versus Local Self-Ordering

Oded Goldreich*

December 17, 2022

Abstract

We study two notions that refers to asymmetric graphs, which we view as graphs having a unique ordering that can be reconstructed by looking at an unlabeled version of the graph.

A *local self-ordering* procedure for a graph G is given oracle access to an arbitrary isomorphic copy of G, denoted G', and a vertex v in G', and is required to identify the name (or location) of v in G, while making few (i.e., polylogarithmically many) queries to G'. A graph $G = (V, E)$ is *robustly self-ordered* if the size of the symmetric difference between E and the edge-set of the graph obtained by permuting V using any permutation $\pi : V \to V$ is proportional to the number of non-fixed-points of π and to the maximal degree of G; that is, any permutation of the vertices that displaces t vertices must "displace" $\Omega(t \cdot d)$ edges, where d is the maximal degree of the graph.

We consider the relation between these two notions in two regimes: The bounded-degree graph regime, where oracle access to a graph means oracle access to its incidence function, and the dense graph regime, where oracle access to the graph means access to its adjacency predicate.

We show that, *in the bounded-degree regime*, robustly self-ordering and local self-ordering are almost orthogonal; that is, even extremely strong versions of one notion do not imply very weak versions of the other notion. Specifically, we present very efficient local self-ordering procedures for graphs that possess permutations that displace all but one vertex while preserving all but a pair of edges. One the other hand, we show robustly self-ordered graphs having no local self-ordering procedures even when allowing a number of queries that is a square root of the graph's size.

In the dense graph regime, local self-ordering procedures are shown to yield a quantitatively weaker version of the robust self-ordering condition, in which the said proportion is off by a factor that is related to the query complexity of the local self-ordering procedure. Furthermore, we show that this quantitatively loss is inherent. On the other hand, we show how to transform any robustly self-ordered graph into one having a local self-ordering procedure, while preserving the robustness condition. Combined with prior work, this yields explicit constructions of graphs that are both robustly and locally self-ordered, and an application to property testing.

A preliminary version of this work was posted as TR21-034 of *ECCC*. The current version eliminates some inaccuracies and hand-wavings that appeared in the preliminary version.

*Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Rehovot, Israel. E-mail: oded.goldreich@weizmann.ac.il. Partially supported by the Israel Science Foundation (grant No. 1041/18) and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819702).

ISSN 1433-8092

Contents

1 Introduction

In this work we study two notions that refers to asymmetric graphs, which we view as graphs having a unique ordering that can be reconstructed by looking at an unlabeled version of the graph (equiv., by exploring any graph that is isomorphic to it). These notions, called *robust self-ordering* and *local self-ordering*, were introduced by Goldreich and Wigderson [6, 7], where the focus was on constructions and applications. The current work is devoted to studying the relation between these two notions.

Robustly self-ordered graphs. Loosely speaking, a *robustly self-ordered graph* is as far as possible from having non-trivial automorphisms; specifically, for any t, any permutation of the vertices that displaces t vertices must "displace" $\Omega(t \cdot d)$ edges, where d is an upper-bound on the degree of the graph. In other words, a graph $G = ([n], E)$ is called robustly self-ordered if for every permutation $\pi : [n] \to [n]$ it holds that G and $\pi(G)$ differ on $\Omega(d) \cdot |\{v \in [n] : \pi(v) \neq v\}|$ vertex-pairs, where $\pi(G) = ([n], \{\{\pi(u), \pi(v)\} : \{u, v\} \in E\})$; that is, if π has t non-fixed-points, then the symmetric difference between the graphs is $\Omega(t \cdot d)$.

We shall consider two regimes (or types of graphs). In the regime of *bounded-degree graphs* being robustly self-ordered means that the symmetric difference is at least a constant fraction of the number of non-fixed-points. In contrast, in the regime of *dense graphs*, where the degree bound is the trivial one (i.e., the number of vertices), being robustly self-ordered means that the symmetric difference is at least a $\Omega(n)$ times the number of non-fixed-points, where n denotes the number of vertices in the graph.

The two regimes also differ in what it means to have oracle access to the graph. In the regime of *bounded-degree graphs* this means having oracle access to the incidence function of the graph; that is, the query (v, i) is answered with the i^{th} neighbor of v (and by a special symbol in case v has less than than i neighbors). In the regime of *dense graphs* oracle access to the graph means access to its adjacency predicate; that is, the query (u, v) is answered 1 if u is adjacent to v in the graph, and is answered 0 otherwise. (Indeed, these oracles are the ones considered in the bounded-degree and dense graph models of the property testing literature (cf., e.g., [3, Chap. 8-9]).)[1]

Local self-ordering procedures. Loosely speaking, a *local self-ordering procedure for G*, is given oracle access to an arbitrary isomorphic copy of G, denoted G', and a vertex v in G', and is required to identify the name (or location) of v in G, while making few queries to G'. That is, a local self-ordering procedure for G is a randomized algorithm that, for every permutation $\pi : [n] \to [n]$, on input $v \in [n]$ and oracle access to $G' = \pi(G)$, makes $\text{poly}(\log n)$ queries to G' and outputs $\pi^{-1}(v)$ (with probability at least $1 - n^{-c}$, for any desired constant c). We stress that π is *a priori* unknown to this procedure, but indeed $\pi^{-1}(v)$ is partial information (about π) obtained by the procedure.

1.1 The question and a brief summary of the answers

It seems that the two aforementioned notions may be related. They both carry an air of resiliency of the self-ordering phenomenon; that it, they both require the graph to be self-ordered (a.k.a asymmetric) in a very strong sense. In the case of robust self-ordering this strong sense is expressed

[1] The definitions of robustness in the two regimes also fits the definition of distance in these two property testing models.

in global terms (i.e., the relative size of the symmetric difference between a graph and a graph in which some vertices are displaced), and in the case of local self-ordering the criteria is local (i.e., a local exploration suffices for identifying each vertex). This begs the following question:

> *Does robust self-ordering imply local self-ordering and is it implied by it?*
>
> That is, does any robustly self-ordered graph have a locally self-ordering procedure and does the existence of such a procedure imply that the graph is robustly self-ordered?

Loosely speaking, we show that in the *bounded-degree graph regime* the answers are extremely negative: Robustly self-ordered graphs may have no local self-ordering procedure, even when allowing such procedure to use $o(\sqrt{n})$ rather than $\mathrm{poly}(\log n)$ many queries, whereas local self-ordering is possible in graphs that are extremely non-robust in the sesnse that they have permutations that displace all but one vertex while preserving all but a pair of edges.

In the *dense graph regime* local self-ordering procedures exist only for graphs that satisfy a quantitatively weaker version of the robust self-ordering condition, but these graphs need not be robustly self-ordered in the full-fledged sense. We also show how to transform any robustly self-ordered graph into one having a local self-ordering procedure, while preserving the robustness condition. The question of whether all robustly self-ordered graphs have local self-ordering procedures is left open.

We interpret these results as saying that the global and local versions of the vague concept of resiliency of self-ordering are not as tightly related as one could have thought. Especially in the dense graph regime, this interpretation is quantitative in nature, just as the results on which it is based. Hence, a more clear discussion of the question does require more quantitative definitions than the ones outline above. We provide such definitions next, and will re-visit the foregoing (vaguely stated) results later.

1.2 Quantitative definitions

Although the following definitions are presented in terms of individual graphs, we view these graphs as belonging to some infinite sets (or families) of graphs and presents the various quantities as function of the size of the graph (i.e., the number of vertices in it). For example, when we talk of dense graphs we envision a family of graphs such the maximum degrees of vertices in an n-vertex graph is $\Omega(n)$. In contrast, when we talk of bounded-degree graphs we envision a family of graphs such the maximum degrees of vertices in an n-vertex graph is bounded by a constant d, which is independent of n. With these preliminaries in place, we turn to the actual definitions.

Definition 1.1 (robustly self-ordered graphs): *For a function $\gamma : \mathbb{N} \to \mathbb{R}$, we say that graph $G = (V, E)$ is γ-*robustly self-ordered *if for every permutation $\pi : V \to V$ it holds that*

$$|E \triangle \{\{\pi(u), \pi(v)\} : \{u, v\} \in E\}| \;\geq\; \gamma(|V|) \cdot |\{v \in V : \pi(v) \neq v\}|. \tag{1}$$

where $S \triangle T$ denotes the symmetric difference between the sets S and T.

Recall that in the bounded-degree regime, we say that a family is robustly self-ordered if γ is lower-bounded by a positive constant (i.e., $\gamma(n) = \Omega(1)$). In contrast, in the dense graph regime, we say that a family is robustly self-ordered if γ is lower-bounded by a positive linear function (i.e., $\gamma(n) = \Omega(n)$). Note that if $G = (V, E)$ has maximum degree d, then for every permutation

$\pi : V \to V$ the size of symmetric difference between G and $\pi(G) = (V, \{\{\pi(u), \pi(v)\} : \{u, v\} \in E\})$ is upper-bounded by $d \cdot |\{v \in V : \pi(i) \neq i\}|$.

Definition 1.2 (local self-ordering procedures):[2] *For a function $q : \mathbb{N} \to \mathbb{N}$, a q-*query local self-ordering procedure *for a self-ordered graph $G = ([n], E)$ is a randomized oracle machine that, given a vertex v in any graph $G' = ([n], E')$ that is isomorphic to G and oracle access to G', makes at most $q(n)$ queries, and outputs, with probability at least $2/3$, the vertex that corresponds to v in G; that is, it outputs $\phi(v) \in [n]$ for the unique bijection $\phi : [n] \to [n]$ such that $\phi(G') = G$* (i.e., the unique isomorphism of G' to G).

The definition outlined at the beginning of the introduction mandates that q is a polylogarithmic function, but we may consider arbitrary sub-linear functions. Recall that in the bounded-degree graph regime, with degree bound d, oracle access to G' means oracle access to the incidence function of G' (i.e., the function $g' : V \times [d] \to V \cup \{\perp\}$ such that $g'(v, i)$ is the i^{th} neighbor of v in G' and $g'(v, i) = \perp$ if v has less than than i neighbors). In contrast, in the regime of dense graphs, oracle access to G' means access to its adjacency predicate (i.e., the function $g' : V \times V \to \{0, 1\}$ such that $g'(u, v) = 1$ if and only if u is adjacent to v in G').

We stress that, in contrast to [6, Def. 4.6], Definition 1.2 does not place restrictions on the time complexity of the procedure. We shall, however, refer to the time complexity of local self-ordering procedures when proving that even efficient ones do not have certain implications, making such negative results stronger.

1.3 The bounded-degree graph regime

Recall that in the bounded-degree graph regime we seek graphs that are $\Omega(1)$-robustly self-ordered, and that oracle access to a graph (as provided to a local self-ordering procedure) means oracle access to its incidence function. We show that in this regime robustly self-ordering and local self-ordering are almost orthogonal.

Theorem 1.3 (robustness versus locality in the bounded-degree regime):

1. *There exist explicit n-vertex bounded-degree graphs that are not $(3/n)$-robustly self-ordered, but have $O(\log n)$-time deterministic local self-ordering procedures.*
2. *There exist n-vertex bounded-degree graphs that are $\Omega(1)$-robustly self-ordered, but have no $o(\sqrt{n})$-query local self-ordering procedure.*

Part 1 (proved in Theorem 3.1) asserts that even an extremely strong notion of local self-ordering (i.e., deterministic procedures of logarithmic time complexity) fails to imply an extremely weak notion of robust self-ordering (i.e., anything above merely being self-ordered). Note that, query complexity $O(\frac{\log n}{\log \log n})$ is the absolute minimum for any local self-ordering procedure, because the actual information contents of the answers to q queries is an unlabelled graph with q edges, whereas the number of such unlabelled graphs is $\exp(O(q \log q))$.[3] On the other hand, local self-ordering a n-vertex graph requires that the graph is asymmetric, which implies that it is $(2/n)$-robustly

[2] The requirement may be extended to n-vertex graphs G' with an arbitrary vertex-set, V', provided that V' is also given to the algorithm (as explicit input).

[3] Note that the q answers should determine an index in $[n]$. Hence, we get $\exp(O(q \log q)) \geq n$.

self-ordered (because any non-trivial permutation must displace at least one edge (contributing two units to the symmetric difference)), and so ruling out $3/n$-robustness is the extreme.[4]

Part 2 (proved in Theorem 3.2) is also very strong: It asserts that full-fledged robustly self-ordered graphs do not imply the existence of very weak local self-ordering procedure that are allowed $o(\sqrt{n})$ queries (rather than $\mathrm{poly}(\log n)$ queries). This result is proved using random $O(1)$-regular graphs.

The proofs of Parts 1 and 2 (see Theorems 3.1 and 3.2, resp.) clarify that, in the bounded-degree graph regime, the robust self-ordering feature has little to do with local self-ordering procedures. What matters for the latter is the ability to locate oneself based on exploring a small portion of the graph, where in bounded-degree graphs such an exploration is truly local. In particular, a directed and positional (binary) tree (used in the proof of Theorem 3.1) is the archetypical structure that supports such a localization. We note that these directed and colored edges can be implemented by constant-size gadgets (which are either trees (having vertices of different degrees) or have constant girth). Needless to say, these graphs may not be robustly self-ordered. In contrast, a regular graph of large girth (which may be robustly self-ordered) hinders any attempt at such localization, since the neighborhoods look identical anywhere. In fact, different vertex degrees and short cycles are the only ways one can gather information about one's location in the graph by exploring the neighborhood. (We stress that the foregoing discussion refers to the bounded-degree regime, and makes little sense in the dense graph regime.)

A glance at the techniques. Part 2 of Theorem 1.3 is proved by reducing the problem of establishing lower bounds on the query complexity of local self-ordering procedures for a *regular* graph G to analyzing the probability of closing a (simple) cycle in a random (non-backtracking) walk on G. Specifically, the (query complexity) lower bound is inversely proportional to the square root of the said probability. We note that non-backtracking random walks, introduced in [1], are harder to analyze than standard random walks. Nevertheless, we show that the said probability in a random regular graph is inversely proportional to the size of the graph. We also observe that the probability of closing a (simple) cycle in a random (non-backtracking) walk on a regular graph vanishes exponentially with its girth (see Proposition 3.3). It follows that, for any regular graph, the query complexity of local self-ordering is exponential in its girth.

Remark 1.4 (on the difference between the two robustly self-ordered graphs that were presented in [6, Part I]): *The proof of Theorem 3.2 actually shows that a random bounded-degree n-vertex graph, which is $\Omega(1)$-robustly self-ordered [6, Thm. 6.1], does not have $o(\sqrt{n})$-query locally self-ordering procedures. In contrast, the construction of $\Omega(1)$-robustly self-ordered n-vertex graphs presented in [6, Sec. 4] has a $\mathrm{poly}(\log n)$-time locally self-ordering procedure [6, Thm. 4.7]. This articulates a fundamental difference between these two results.*

1.4 The dense graph regime

Recall that in the dense graph regime we seek n-vertex graphs that are $\Omega(n)$-robustly self-ordered, and that oracle access to a graph (as provided to a local self-ordering procedure) means oracle access to its adjacency predicate. We show that, in this regime, local self-ordering procedures do not imply the full-fledged notion of robust self-ordering, but do imply a quantitatively weaker version of it.

[4] Indeed, $3/n$ stands for any quantity larger than $2/n$.

Theorem 1.5 (from locality to robustness in the dense graph regime):

1. *For every $\ell(n) \in [\Omega(\log n), o(n)]$, there exist n-vertex graphs that have a $O(\ell(n)^2)$-query local self-ordering procedure, but are not $\omega(n/\ell(n))$-robustly self-ordered. Furthermore, the procedure is non-adaptive and relies on inspecting a subgraph induced by $O(\ell(n))$ vertices.*

2. *Every n-vertex graph that has a $q(n)$-query local self-ordering procedure is $\Omega(n/q(n)^2)$-robustly self-ordered. Furthermore, if the procedure is non-adaptive, then the graph is $\Omega(n/q(n))$-robustly self-ordered.*

Recall that a procedure is called non-adaptive if it determines all its queries beforehand (i.e., based on its explicit input and internal coin tosses, but independently of the answers to previous queries).

Theorem 1.5 asserts that there is a quantitative relation between the query complexity of the local self-ordering procedure and the robustness parameter of the graph: As one may expect, the robustness parameter is inversely related to the query complexity of the local self-ordering procedure. Specifically, Part 2 asserts that the robustness is at least inversely proportional to the non-adaptive query complexity, and Part 1 assert that this relation is rather tight (i.e., in general, the robustness may be at most inversely proportional to a square root of the non-adaptive query complexity).

Part 1 of Theorem 1.5 suggests that graphs having local self-ordering procedures may be easier to construct than $\Omega(n)$-robustly self-ordered n-vertex graphs. The proof of Theorem 2.9 supports this feeling because it presents a relatively simple construction of n-vertex graphs coupled with efficient local self-ordering procedures. These graphs are not $\Omega(n)$-robustly self-ordered, and the construction is significantly simpler than the construction of the latter (presented in [6, Sec. 8]).

The foregoing seems to suggest that robustly self-ordered is a stronger condition than having a local self-ordering procedure. However, we do not know whether this "suggestion" is true (i.e., whether every $\Omega(n)$-robustly self-ordering graph has a $\mathrm{poly}(\log n)$-query local self-ordering procedure).

Open Problem 1.6 (does $\Omega(n)$-robustness imply $\mathrm{poly}(\log n)$-locality): *Is it the case that any $\Omega(n)$-robustly self-ordered n-vertex graph has a $\mathrm{poly}(\log n)$-query local self-ordering procedure?*

A positive answer to Problem 1.6 would have allowed us to establish [7, Thm. 1.2–1.4] while using efficiently recognizable graph properties (by using the explicit construction of robustly self-ordered graphs provided in [6, Thm. 8.10]).[5] Fortunately, these positive corollaries also follow from an efficient transformation of $\Omega(n)$-robustly self-ordered n-vertex graph to ones that have $\mathrm{poly}(\log n)$-query local self-ordering procedure. We do provide such a transformation.

Theorem 1.7 (from robustness to locality in the dense graph regime): *There exists a polynomial-time transformation of $\Omega(n)$-robustly self-ordered n-vertex graphs to $\Omega(n)$-robustly self-ordered $O(n)$-vertex graphs that have $\mathrm{poly}(\log n)$-time local self-ordering procedures. Furthermore, the transformation is local in the sense that the adjacency predicate of the resulting graph can be computed by a $\mathrm{poly}(\log n)$-time reduction to the adjacency predicate of the input graph.*

[5] Doing the same for [7, Thm. 1.5] would have required more significant modifications. In particular, the current proof of [7, Thm. 1.5] uses a non-explicit two-source extractor (see [7, Clm. 5.3]).

(Indeed, the time bounds on these local procedures imply corresponding query complexity bounds.) Combining Theorem 1.7 with [6, Thm. 8.10] (or even the weaker [6, Thm. 1.4]), we obtain graphs that are both robustly and locally self-ordered.

Corollary 1.8 (explicit constructions obtaining linear robustness and polylogarithmic locality): *There exists an explicit construction of n-vertex graphs that are $\Omega(n)$-robustly self-ordered and have local self-ordering procedures that run in $\mathrm{poly}(\log n)$-time. Furthermore, the adjacency predicate of these graphs can be computed in $\mathrm{poly}(\log n)$-time.*

As stated above, substituting the non-explicit constructions used in the proofs of [7, Thm. 1.2–1.4] by the (strongly) explicit construction of Corollary 1.8, we obtain an explicit version of [7, Thm. 1.4] (which generalizes [7, Thm. 1.2&1.3]).

Corollary 1.9 (explicit separation between adaptive and non-adaptive testers in the dense graph model): *There exists a polynomial-time recognizable graph property that is testable by an adaptive oracle machine that make $O(\epsilon^{-1} \cdot \sqrt{n} \cdot \mathrm{poly}(\log n))$ queries but testing it non-adaptively requires $\Omega(n)$ queries, where n denotes the number of vertices in the graph. Furthermore, for every functions $f, g : \mathbb{N} \to \mathbb{N}$ such that $f(n) \leq \sqrt{n}$ and $g(m) \leq m$, there exists a polynomial-time recognizable graph property Π that satisfies the following two conditions:*

1. *There exists a general* (i.e., adaptive) *tester of Π that runs in time $O(\epsilon^{-1} \cdot f(n) \cdot \mathrm{poly}(\log n))$, and any tester for Π must make $\Omega(f(n))$ queries.*
2. *Any non-adaptive tester of Π must make $\Omega(g(f(n)) \cdot f(n))$ queries, and there exists such a tester that runs in time $O(\epsilon^{-2} \cdot g(f(n)) \cdot f(n) \cdot \mathrm{poly}(\log n))$.*

Actually, the same holds with respect to the more general result of [7, Thm. 1.7], which we avoid stating here. We stress that Corollary 1.9 provides an upper bound on the computational complexity of the testers, whereas the results in [7] only upper-bound the query complexity. However, the query complexity of the testers asserted by Corollary 1.9 is a $\mathrm{poly}(\log n)$ factor larger than the upper bounds stated in [7, Thm. 1.2–1.4].

A glance at the techniques. With the exception of Part 2 of Theorem 2.6, our results in this regime are proved by constructions of adequate graphs. The common theme in these constructions is combining several graphs that are "locally distinguishable" and locating the vertices within each part of the combined graph based on their adjacencies in some (but not all) of the parts. The simplest case occurs in the proof of Part 1 of Theorem 2.6, where we combine three (non-explicit) robustly self-ordered graphs that are each locally self-ordered. More sophisticated constructions appear in the proofs of Theorems 2.9 and 1.7, where we combine logarithmically many parts and locate vertices in each part based on their adjacencies in some of the other parts.

1.5 The story: From the initial motivation to a general study

Our initial motivation was proving Corollary 1.9, which asserts explicit graph properties that match the results proved in [7] using non-explicit graph properties. Specifically, the results proved in [7, Thm. 1.2–1.4] assert various separation between adaptive and non-adaptive testers in the dense graph model. The properties used there were non-explicit, because they relied on $\Omega(n)$-robustly

self-ordered n-vertex graphs that have $\mathrm{poly}(\log n)$-query local self-ordering procedures. At the time, such graphs were only known to exist (via a probabilistic argument [7, Cor. 2.7]), but no explicit construction was known. In contrast, explicit constructions of $\Omega(n)$-robustly self-ordered n-vertex graphs were given in [6, Thm. 1.4], but they were not known to have a $\mathrm{poly}(\log n)$-query local self-ordering procedures.

This gave rise to Problem 1.6 (i.e., does every $\Omega(n)$-robustly self-ordered n-vertex graph have a $\mathrm{poly}(\log n)$-query local self-ordering procedure). While we did not resolve Problem 1.6, the transformation provided by Theorem 1.7 is good enough for the original motivation, since it efficiently transforms the graphs of [6, Thm. 1.4] into ones that are both linearly robust and have local self-ordering procedures of polylogarithmic query complexity.

A natural question that arose was whether the converse implication holds; that is, does the existence of a $\mathrm{poly}(\log n)$-query local self-ordering procedure for a dense n-vertex graph imply that the graph is $\Omega(n)$-robustly self-ordered? This question was answered negatively in Part 1 of Theorem 1.5, whereas Part 2 asserts that such a graph must be $\Omega(n/\mathrm{poly}(\log n)))$-robustly self-ordered.

While the foregoing refers to the dense graph regime, a comparative study of the situation in the bounded-degree graph regime seemed begging. Note that in the latter regime explicit constructions of n-vertex graphs that are $\Omega(1)$-robustly self-ordered and have $\mathrm{poly}(\log n)$-query local self-ordering procedures were known [6, Thm. 4.7].[6] Nevertheless, it is interesting to see whether phenomena that occur in one regime also occur in the other regime, and Theorem 1.3 indicates that this is not the case with respect to the current question (since in the bounded-degree graph regime robustly self-ordering and local self-ordering are almost orthogonal).

1.6 Organization

The next two sections can be read independently of one another: Section 2 studies the dense graph regime, whereas Section 3 is devoted to the study of the bounded-degree graph regime.

The dense regime. Section 2 starts with a study of local self-ordering procedures per se, which related general (adaptive) procedures to non-adaptive ones, and presents a simplified form of the latter (which will be used in the proof of Theorem 1.5). This section (Section 2.1) also contains an exposition of a strong notion of local self-ordering, captured by the notion of a reliable locator (Definition 2.4) and poses the question of whether this is actually a real strengthening (Problem 2.5). In Section 2.2 we prove a rather tight quantitative implication from local self-ordering to robust self-ordering, establishing Theorem 1.5. In Section 2.3 we present a relatively simple construction of dense graphs with an efficient local self-ordering procedures (Theorem 2.9). In Section 2.4, we present a transformation of robustly self-ordered graph into ones having local self-ordering procedures, establishing Theorem 1.7.

The bounded-degree graph regime. In Section 3 we prove Theorem 1.3, which asserts that – in the bounded-degree graph regime – robust self-ordering and local self-ordering are almost orthogonal. Part 1 of the theorem is proved in Section 3.1 by observing that directed and positional (binary) trees have extremely simple local self-ordering procedures. On the other hand, Part 2 of

[6] Interestingly, our study uncovers a fundamental different between this construction and an alternative construction presented in [6, Sec. 3] (see Remark 1.4).

Theorem 1.3 is proved in Section 3.2 by proving that regular graphs have no local self-ordering procedure of complexity q if the probability that a random (non-backtracking) walk on them closes a (simple) cycle with probability $o(1/q^2)$.

2 The Dense Graph Regime

Recall that in the dense graph regime we seek n-vertex graphs that are $\Omega(n)$-robustly self-ordered, and that oracle access to a graph (as provided to a local self-ordering procedure) means oracle access to its adjacency predicate.

It seems that, in the current regime, robustly self-ordering and locally self-ordering are related. On the one hand, the fact that an n-vertex graph is $\Omega(n)$-robustly self-ordered implies that the neighborhoods of its n vertices are pairwise far apart, and so random samples of a $O(\log n)$-vertices would yield n different adjacency-patterns. Furthermore, *if* such a random sample induces a subgraph that is not isomorphic to any subgraph induces by a different set, *which is a big IF*, then we get a locally self-ordering procedure. The foregoing unsettled conjecture leaves us with Problem 1.6.

On the other hand, the fact that an n-vertex graph has a $\mathrm{poly}(\log n)$-query local self-ordering procedure seems to imply that a random set of $\mathrm{poly}(\log n)$ vertices induces a self-ordered subgraph (w.h.p.). This seems to suggest that the graph is $\Omega(n/\mathrm{poly}(\log n))$-robustly self-ordered. It turns out that the conclusion is correct (i.e., an n-vertex graph that has a $\mathrm{poly}(\log n)$-query local self-ordering procedure is $\Omega(n/\mathrm{poly}(\log n))$-robustly self-ordered), but our proof of this fact (see Theorem 2.8) does not quite follow the foregoing reasoning. In any case, the fact that such a local self-ordering procedure does *not* imply that the graph is $\Omega(n)$-robustly self-ordered is no coincidence; see Theorem 2.6. This establishes the two parts of Theorem 1.5.

Organization of this section. We start this section with a study of local self-ordering procedures per se. The results of this study, presented in Section 2.1, will be used in the proof of Theorem 2.8 (asserting that an n-vertex graph that has a $\mathrm{poly}(\log n)$-query local self-ordering procedure is $\Omega(n/\mathrm{poly}(\log n))$-robustly self-ordered). In contrast, the notion of reliable locators (Definition 2.4), reviewed in Section 2.1, will not be used in the rest of this section, but raises a couple of open problems (e.g., Problem 2.5). The rest of this section is organized as detailed in Section 1.6. In particular, Theorem 1.5 is proved in Section 2.2, Theorem 1.7 is proved in Section 2.4, and Theorem 2.9 (which is briefly mentioned in the introduction) is proved in Section 2.3.

2.1 On local self-ordering procedures

In this section, we present "canonical" forms of local self-ordering procedures (akin the ones presented for property testers in [5, Sec. 4]). Things are more tricky in the current setting, because the algorithm is given an explicit designated vertex (which it has to locate in G) rather than only oracle access to a graph (i.e., an isomorphic copy of the graph G).

Non-adaptivity. A local self-ordering procedure is called non-adaptive if it determines all its queries upfront, based solely on its explicit input (i.e., a vertex) and its randomness. Following [5, Sec. 4.1], we first observe that adaptivity can be disposed of at the cost of squaring the query complexity.

Proposition 2.1 (obtaining non-adaptive local self-ordering procedures): *If $G = ([n], E)$ has a q-query local self-ordering procedure, then it has a* non-adaptive *$3q^2$-query local self-ordering procedure. Furthermore, the non-adaptive procedure makes queries in $\binom{\{v\}\cup S}{2}$, where v is its input vertex and S is a random set of $2q$ vertices* (which excludes v).

Proof: We adapt the proof strategy of [5, Sec. 4.1] to the current context. First, we convert the general algorithm into a "vertex-uncovering" one; that is, an algorithm that works in iterations such that at each iteration it selects a new vertex (based on all information available to it) and queries all pairs that consist of this new vertex and one of the old vertices. Indeed, any q-query algorithm can be emulated by a vertex-uncovering algorithm that uses $2q$ iterations and makes $\binom{2q+1}{2} \leq 3q^2$ queries, where the extra unit accounts for the input vertex.

Next, we observe that, without loss of generality, the new vertices can be selected uniformly at random (among all vertices that were not used so far). Formally, given a vertex-uncovering algorithm A', consider an algorithm A'' that on input v, and oracle access to a graph $G' = ([n], E)$, selects a random permutation $\pi : [n] \to [n]$ such that $\pi(v) = v$, and replaces each new vertex w selected by A' with $\pi(w)$. Equivalently, we may think of A'' as selecting q' random vertices, denoted $u_1, ..., u_{q'}$, for its q' iterations, and defining $\pi(v_i) = u_i$ if v_i is the i^{th} vertex selected by A'. Then, A'' satisfies the syntactic condition of the proposition (e.g., it is non-adaptive). On the other hand, on input v and oracle access to G', algorithm A'' emulates an execution of A' on input v and oracle access to $\pi(G')$, which implies that A'' is a local self-ordering procedure for G. ■

Label-obliviousness. Intuitively, the labels of the various vertices are irrelevant to the task (of local self-ordering) at hand. All that matters is the adjacencies among vertices. Indeed, we show that one may assume, without loss of generality, that the local self-ordering procedure is "label-oblivious" (i.e., it is oblivious of the label of the input vertex as well as of the labels of all other vertices that appear in its queries).

Definition 2.2 (label-oblivious procedures): *We say that a $q(n)$-query non-adaptive* (local self-ordering) *procedure is* label-oblivious *if on input a vertex $s_0 = v$ and randomness $\overline{r} = (s_1, ..., s_\ell, \omega)$, where $\ell = 2q(n)$ and $\{s_1, ..., s_\ell\} \in \binom{[n]\setminus\{s_0\}}{\ell}$, it makes the query $\{s_i, s_j\}$ if and only if $\{i, j\} \in P(\omega)$, where $P(\omega) \in \binom{\{0,1,...,\ell\}}{2}$ is a set of pairs determined solely by ω, and decides based solely on the answers to the queries and on ω.*

We stress that $P(\omega)$ as well as the final decision are oblivious of $s_0, s_1, ..., s_\ell$. Note that the non-adaptive $O(q^2)$-query algorithm derived in the proof of Proposition 2.1 is label-oblivious. More importantly, any non-adaptive q-query algorithm can be transformed into label-oblivious algorithm, while preserving the number of queries (cf. [5, Sec. 4.2]).

Proposition 2.3 (obtaining label-oblivious local self-ordering procedures): *If G has a* non-adaptive *q-query local self-ordering procedure, then it has a* label-oblivious *q-query local self-ordering procedure.*

Proof: Here we adapt the strategy of [5, Sec. 4.2]. Let A be an algorithm as in the hypothesis. Recall that A determines its queries based on its own input, denoted v, and its own randomness, denoted r, and decides based on v, r and the answers obtained from the oracle. Suppose that the foregoing queries refer to the vertices $v_0 \stackrel{\text{def}}{=} v$ and $v_1, ..., v_\ell$, where the v_i's are determined based

on v and r. Then, we define $P(v,r) \subseteq \binom{\{0,1,...,\ell\}}{2}$ such that it includes $\{i,j\}$ if and only if the query $\{v_i, v_j\}$ is made by $A(v,r)$. Using these notation, we present the following label-invariant algorithm.

On input $s_0 \stackrel{\text{def}}{=} u$, our label-oblivious algorithm, denoted $\overline{A}$, uses randomness $(s_1, ..., s_\ell, \omega)$ such that $\omega = (v,r)$, and emulates the execution of A making the query $\{s_i, s_j\}$ if and only if $\{i,j\}$ is in $P(v,r)$. That is, $\overline{A}$ make queries to pairs in $\binom{\{s_0,s_1,...,s_\ell\}}{2}$, such that the choice of specific queries is determined by $P(v,r)$, where $(v,r) = \omega$ is the residual randomness, and the final decision depends on the answers as well as on (v,r), but not on u and $s_1, ..., s_\ell$.

The performance of $\overline{A}$ is analyzed by observing that, on input a vertex u, randomness $(s_1, ..., s_\ell, (v,r))$ and oracle access to a graph G', algorithm $\overline{A}$ emulates the execution of $A_r(v) \stackrel{\text{def}}{=} A(v,r)$ with oracle access to $\phi(G')$ such that $\phi(v) = u$ and $\phi(v_i) = s_i$ for every $i \in [\ell]$, where the v_i's are as above. Hence, the probability that $\overline{A}$ is correct (on input u and access to $G' = \pi(G)$) equals the probability that A is correct on a random input v and access to a random isomoprohic copy of G (i.e., to the graph $\phi(G') = \phi(\pi(G))$, where ϕ is a random permutation mapping u to v). Formally, for fixed $s_0 = u$ and $G' = \pi(G)$, we have

$$\begin{aligned} \Pr_{s_1,...,s_\ell,(v,r)}[\overline{A}^{G'}_{s_1,...,s_\ell,(v,r)}(u) = \pi^{-1}(u)] &= \Pr_{r,\phi}[A_r^{\phi(G')}(\phi(u)) = \pi^{-1}(u)] \\ &= \Pr_{r,\phi}[A_r^{\phi(\pi(G))}(\phi(u)) = (\phi \circ \pi)^{-1}(\phi(u))] \qquad (2) \end{aligned}$$

where the first equality follows by the fact that $\overline{A}$ emulates A on input $v_0 \stackrel{\text{def}}{=} v = \phi(u)$ and oracle access to $\phi(G')$, because $\overline{A}$ replaces queries of the form (v_i, v_j) made by A by the queries $(s_i, s_j) = (\phi(v_i), \phi(v_j))$. Note that, for every r and uniformly distributed $v \in [n]$, it holds that $(v, s_1, ..., s_\ell)$ is distributed identically to $(\phi(u), \phi(v_1), ..., \phi(v_\ell))$, where ϕ is a random permutation.[7] Lastly, note that Eq. (2) equals the probability that A is correct on a random input v and access to a random isomoprohic copy of G. ■

Reliable locators. We seize the opportunity provided by this section to review the notion of reliable locators, which was presented by Goldreich and Wigderson [7]. Loosely speaking, a set S of the vertices of G is called a reliable locator if the subgraph of G induced by S is self-ordered and unique in G, and every other vertex has a unique "profile" with respect to S (i.e., its sequence of adjacencies with S is unique). As sketched below, an abundence of reliable locators yields a local self-ordering procedure, but the converse is not known.

Definition 2.4 (reliable locator of a self-ordered graph [7, Def. 2.5]): *A set of vertices $S \subset [n]$ is called a* reliable locator *of a graph $G = ([n], E)$ if the following two conditions hold*

1. *The subgraph of G induced by S is self-ordered and is not isomorphic to any other induced subgraph of G.*
2. *For every $v \in [n] \setminus S$, the adjacencies of v with S uniquely determine v; that is, for every $w \neq v$ in $[n] \setminus S$ there exists $s \in S$ such that $\{w,s\} \in E$ if and only if $\{v,s\} \notin E$.*

[7] A random $v \in [n]$ corresponds to a random setting of $\phi(u)$, and also determines $v_1, ..., v_\ell$ (via $A(v,r)$). Then, a random choice of $\{s_1, ..., s_\ell\} \in \binom{[n]\setminus\{u\}}{\ell}$ corresponds to a random setting of ϕ at $v_1, ..., v_\ell$, which are distinct elements of $[n] \setminus \{v\}$.

Note that if almost all ℓ-sized subsets in a graph are reliable locators, then this graph has a $O(\ell^2)$-query local self-ordering procedure; specifically, on input a vertex v and oracle access to a graph G', this procedure selects a random ℓ-set of vertices R and inspects the subgraph of G' induced by $R \cup \{v\}$, which uniquely identifies v. The point is that if $G' = \pi(G)$ and $\pi^{-1}(R)$ is a reliable locator of G, then we obtain the location of v in G (i.e., $\pi^{-1}(v)$).

We stress that, although one may be tempted to think that a $q(n)$-query local self-ordering procedure for an n-vertex graph yields an abundance of (poly$(q(n) \cdot \log n)$-sized) reliable locators (via error reduction)[8], it is not clear if this is the case. Indeed, we ask

Open Problem 2.5 (do local self-ordering procedures yield reliable locators?) *Does the existence of local self-ordering procedure of polylogarithmic query complexity imply that almost all polylogarithmically sized sets are reliable locators?*

Oddly enough, the only graphs known to have reliable locators of polylogarithmic size are random graphs (see [7, Thm. 2.6]).[9] In particular, at the current time, we do not know of any efficient construction of graphs that have reliable locators of polylogarithmically size (let alone an abundance of such sets).[10]

2.2 Proof of Theorem 1.5

We first prove a special case of Part 1 of Theorem 1.5 (i.e., for $\ell = \Theta(\log n)$ rather than for every $\ell = \Omega(\log n)$).

Theorem 2.6 (local self-ordering does not imply linearly-robust self-ordering): *There exist n-vertex graphs that have $O((\log n)^2)$-query local self-ordering procedures, but are not $(n/\log_2 n)$-robustly self-ordered. Furthermore, the procedure is non-adaptive and is based on examining the subgarph induced by the input vertex and $O(\log n)$ random vertices. Moreover, each vertex in the graph has degree $\Theta(n)$.*

Theorem 2.6 can be generalized by replacing $O(\log n)$ with any $\ell = \Omega(\log n)$. That is, *for every $\ell \in [\Omega(\log n), o(n)]$, there exist n-vertex graphs that are not $\omega(n/\ell)$-robustly self-ordered, but have local self-ordering procedures that are based on examining the subgraph induced by the input vertex and ℓ random vertices.*[11]

Proof: We consider n-vertex graphs that consists of three dense $n/3$-vertex graphs that are connected by random bipartite graphs of low edge density. Specifically, for any constant $p \in (0,1)$, let $\mathcal{G}(k,p)$ be the Erdos–Renyi random k-vertex graph in which each pair of vertices is connected with probability p independently of all other vertex-pairs. Then, for $k = n/3$, we consider a distribution on n-vertex graphs that is generated as follows (see Figure 1).

[8] Error reduction does imply that (for every G' that is isomorphic to G) almost all $O(q(n) \cdot \log n)$-long sequences of vertices can be used to localize all vertices (of G' in G), but this falls short from yielding a reliable locator. In particular, we need the same sequence of vertices to correctly localize all vertices when accessing any graph G' that is isomorphic to G.

[9] Their proof extends to Erdos–Renyi random graphs of any constant edge probability (see [7, Clm. 5.5]).

[10] Indeed, we retract statements to the opposite that were made in a prior version of this work.

[11] This can be seen by replacing all occurrences of $\log_2 n$ in the following proof with $\ell/O(1)$.

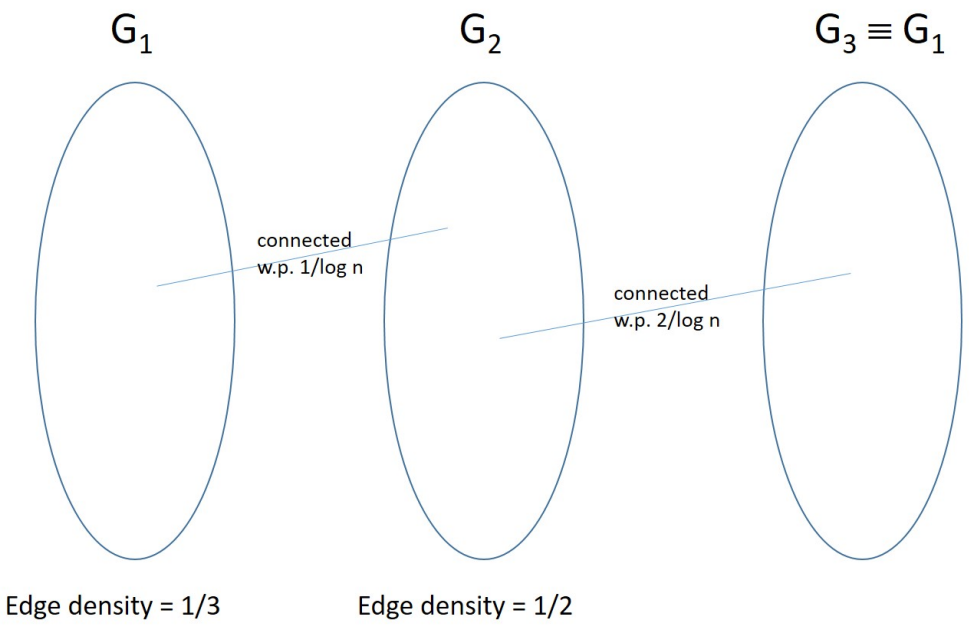

Figure 1: The construction demonstrating Theorem 2.6.

- Pick $G_1 = ([k], E_1)$ at random from $\mathcal{G}(k, 1/3)$, and $G_2 = (\{k+1, ..., 2k\}, E_2)$ at random from $\mathcal{G}(k, 1/2)$. Let $G_3 = (\{2k+1, ..., 3k\}, E_3)$ be an isomorphic copy of G_1, obtained by letting $E_3 = \{\{2k+u, 2k+v\} : \{u, v\} \in E_1\}$.
- Connect G_1 to G_2 by a random bipartite graph in which each edge is included with probability $1/\log_2 n$, independently of all other choices.
- Connect G_3 to G_2 by a bipartite graph in which each edge is included with probability $2/\log_2 n$, independently of all other choices.

Clearly, with overwhelmingly high probability, each vertex in the resulting graph G has degree $\Theta(n)$. We now turn to establishing the main claims.

On the one hand, with overwhelmingly high probability, the resulting graph G is not $(n/\log_2 n)$-robustly self-ordered. To see this, consider a bijection π that switches the vertices of G_1 with the corresponding vertices of G_3. The symmetric difference between G and $\pi(G)$ is due solely to the edges between G_2 and the other G_i's, but the expected number of these edges is $3k^2/\log_2 n$, whereas the number of non-fixed-points of π is $2k$. Noting that $\frac{3k^2/\log_2 n}{2k} = \frac{n}{2\log_2 n}$, the non-robustness claim follows.

On the other hand, we show that G has a local self-ordering procedure that rules according to the subgraph induced by a set containing the input vertex and $O(\log n)$ random vertices. Let S be a random set of $\ell = O(\log n)$ vertices in G, and let S_i denote its intersection with the vertices of G_i. Then, with probability at least $1 - \exp(-\Omega(\ell))$ over the choice of S and the choice of the graph G, the following conditions hold:

1. Each S_i has size approximately $\ell/3$.

This claim relies only on the randomness of S; that is, it holds for every graph G and all but a $1/\mathrm{poly}(n)$ fraction of the possible choices of the set S. In contrast, all subsequent claims hold (with probability $1 - \frac{1}{\mathrm{poly}(n)}$ over the choice of G) for any set S that satisfies the current condition.

2. Each vertex of S_1 (resp., S_3) has approximately $\ell/9$ neighbors in S, whereas each vertex of S_2 has approximately $\ell/6$ neighbors in S.

   (In all cases, almost all of the neighbors of vertex $v \in S$ in S reside in the same part S_i as v (i.e., in the same G_i); see Condition 4.)

3. The number of edges between S_1 and S_2 is approximately $\ell^2/9 \log_2 n$, whereas the number of edges between S_3 and S_2 is approximately $2\ell^2/9 \log_2 n$.

4. Each vertex of G_i has at least $\ell/4$ neighbors in S_i, but at most $\ell/100$ neighbors in $S \setminus S_i$.

   (Here we use a union bound over all n vertices. Note that, with probability $1 - \exp(-\Omega(\ell))$, a vertex in G_i has at least $\ell/4$ neighbors in S_i and at most $\ell/100$ neighbors in $S \setminus S_i$. A sharper dichotomy can be obtained for vertices in S, since, with probability at least $1 - o(1/\ell)$, a vertex in S_i has $o(\ell)$ neighbors in $S \setminus S_i$.)

5. Each S_i constitutes a reliable-locator for G_i.

   This follows by a generalization of [7, Thm. 2.6]. Specifically, while the original assertion refers to $\mathcal{G}(k, 1/2)$, the generalization to $\mathcal{G}(k, p)$ for any constant $p \in (0, 1)$ is quite straightforward (cf. [7, Clm. 5.5]).

Condition 2 allows to distinguish $S' = S_1 \cup S_3$ from S_2, whereas Conditions 3–4 allows to distinguish S_1 from S_3. Specifically, once the set S' is identified, we determine its bi-partition into (S_1, S_3) by considering the subgraph of G induced by S and using Condition 4 (since (S_1, S_3) is the only bi-partition that satisfies Condition 4, alas this does not determine which part of the bi-partition is S_1). Then, by Condition 3, we tell which of the two parts of S' is S_1 and which is S_3. Combining Conditions 4 and 5, we conclude that S can be used to locate each vertex in G, where Condition 4 allows to determine to which G_i each vertex belongs (and Condition 5 allow to locate it within this G_i). The local self-ordering claim follows. ■

Comment. The foregoing proof falls short of proving that (w.v.h.p.) the $O(\log n)$-vertex set S is a reliable locator, but it seems that this is the case. Specifically, we conjecture that every set S that satisfies the foregoing Condition 1 (i.e., each S_i is of size approximately $\ell/3$) is a reliable locator for G with probability at least $1 - (1/\mathrm{poly}(n))$, where the probability is taken over the random construction of the graph G. Proving this conjecture reduces to proving that (w.p. at least $1 - (1/\mathrm{poly}(n))$) the subgraph of G induced by S is self-ordered and not isomorphic to any other induced subgraph of G. The actual challenge is establishing the latter part[12], whereas the self-ordering of the subgraph induced by S follows from the foregoing argument.

[12] We envision proving that (w.v.h.p. over the choice of G) the subgraph of G induced by S is not isomorphic to any other induced subgraph of G by extending the proof of [7, Clm. 2.6.1], which in turn extends [4, Clm. 3.2.2].

An alternative to Part 1 of Theorem 1.5. In light of the foregoing comment, we provide an alternative result in which the local self-ordering condition refers to the abundance of reliable locators, but the non-robustness parameter is a larger. This means that the local self-ordering condition is seemingly stronger, but the non-robustness result is weaker.

Theorem 2.7 (local self-ordering does not imply linearly-robust self-ordering, take two): *For every $\ell \in [\Omega(\log n), o(n)]$ and $\gamma \geq O(\log n)/\ell$, there exist n-vertex graphs in which all but a $\exp(-\Omega(\gamma\ell) + O(\log n))$ fraction of the ℓ-vertex subsets are reliable locators, but the graph is not γ-robustly self-ordered.*

In particular, for $\ell = \omega(\log n)$, we can use $\gamma = o(1)$.

Proof Sketch: The proof simplifies the strategy used in the proof of Theorem 2.6. Specifically, the n-vertex graph G consists of two identical copies of a random $n/2$-vertex graph that are connected by a bipartite graph in which each edge appears with probability $\gamma/2$. The non-robustness of G is witnessed by a permutation that switches its two parts (while maintaining the order within each part). The density of ℓ-vertex sets that are reliable locators can be established by extending the proof of [7, Clm. 2.6.1], which in turn extends [4, Clm. 3.2.2]. Specifically, the proof presented in [7, Apdx. B] refers to the case that each edge appears in the random graph with probability exactly $1/2$, and yields a probability bound of $n \cdot \ell^3 \cdot (1/2)^{\Omega(\ell)}$. Extending this proof to the case that each edge occurs with an arbitrary probability in $[p, q]$ only means that the base $1/2$ of the exponent in all expressions (e.g., [7, Eq. (10)]) is replaced by

$$\max_{p_1,p_2\in[p,q]}\{p_1 \cdot p_2 + (1-p_1)\cdot(1-p_2)\} = \max_{x\in\{p,q\}}\{x^2 + (1-x)^2\}, \tag{3}$$

which represents the probability that two different pairs of vertex are either both connected or both unconnected (by edges in the random graph). In our case, where $p = \gamma < 1/2$ and $q = 1/2$, Eq. (3) equals $\gamma^2 + (1-\gamma)^2 \leq \exp(-\Omega(\gamma))$, which implies that the density of reliable locators is $n \cdot \ell^3 \cdot \exp(-\Omega(\gamma \cdot \ell))$. ■

Proving Part 2 of Theorem 1.5. While local self-ordering procedures do not imply linear-robustness, they do imply a sublinear level of robust self-ordering.

Theorem 2.8 (local self-ordering implies weak robust self-ordering): *Suppose that $G = ([n], E)$ has a local self-ordering procedure that makes $q(n)$ non-adaptive queries and errs with probability at most $1/3$. Then, G is $\Omega(n/q(n))$-robustly self-ordered.*

It follows that if G has a local self-ordering procedure that uses $q(n)$ general (adaptive) queries, then it is $\Omega(n/q(n)^2)$-robustly self-ordered, since q adaptive queries can be emulated by $3q^2$ non-adaptive ones (see Proposition 2.1). Recall that Theorem 2.6 implies that G is not necessarily $\omega(n/q(n)^{1/2})$-robustly self-ordered.

Proof: To gain some intuition, consider a permutation π that switches two vertices of G and keeps the rest intact. We claim that in this case the symmetric difference between G and $\pi(G)$ is $\Omega(n/q(n))$. This is shown by first showing that, without loss of generality, a $q(n)$-query local self-ordering procedure queries the graph on some adjacencies among the input vertex and $2q(n)$ random vertices. (The argument, which appears below, is similar to the one in [5, Sec. 4.2].) Next,

we note that if the symmetric difference between G and $\pi(G)$ is $o(n/q(n))$, then such a procedure cannot distinguish the two vertices that are switched by π, since distinguishing these two cases mandates hitting a pair of vertices that is in the symmetric difference, whereas each query made by the algorithm has at least one endpoint that is almost uniformly distributed among the fixed-points. This argument can be extended to the case that π has $O(1)$ non-fixed-points, but extending it to the general case requires a more careful counting, which we detail below.

Using Proposition 2.3 we reduce the analysis to the case of label-oblivious algorithms (as in Definition 2.2). Next, we consider an arbitrary q-query label-oblivious algorithm, denoted A. Recall that on input v and randomness $\overline{r} = (s_1, ..., s_\ell, \omega)$, when given oracle access to the graph G', the output of A, denoted $A^{G'}_{\overline{r}}(v)$, is determined by ω and the sequence of answers provided to the ordered sequence of queries $\{(s_i, s_j) : (i,j) \in P(\omega)\}$ where $s_0 = v$ (and the ordering is determined by a fixed ordering of $P(\omega) \in \binom{\{0,1,...,\ell\}}{2}$). We let $A^{G'}(v)$ denote the random variable $A^{G'}_{\overline{r}}(v)$ such that $\overline{r}$ is uniformly distributed (in $\binom{[n]\setminus\{v\}}{\ell} \times \Omega$, for some adequate Ω).

Towards analysing the robust self-ordering of G, we consider an arbitrary bijection $\pi : [n] \to [n]$. Let $T = \{v \in [n] : \pi(v) \neq v\}$ denote the set of non-fixed-points of π. Then, recalling that A is correct on each input, with probability at least $2/3$, it follows that, for every $v \in T$, it holds that

$$\Pr[A^G(v) \neq A^G(\pi(v))] \;\geq\; \Pr[A^G(v) = v] - \Pr[A^G(\pi(v)) = v] \;\geq\; 1/3, \tag{4}$$

where the foregoing holds regardless of the dependency between the two random invocations of A in the first probabilistic expression (i.e., the l.h.s of Eq. (4)). Recalling that $A^G_{\overline{r}}(x)$ denote the output of A^G on input x and randomness $\overline{r} = (\overline{s}, \omega) = (s_1, ..., s_\ell, \omega)$, and letting $\mathcal{S}_v \stackrel{\text{def}}{=} \binom{[n]\setminus\{v\}}{\ell}$, we can write Eq. (4) as

$$\Pr_{(\overline{s},\omega) \in \mathcal{S}_v \times \Omega}[A^G_{\overline{s},\omega}(v) \neq A^G_{\pi(\overline{s}),\omega}(\pi(v))] \geq 1/3, \tag{5}$$

since $\pi(\overline{s}) = (\pi(s_1), ..., \pi(s_\ell))$ is a uniformly distributed (ordered) ℓ-subset of $[n] \setminus \{\pi(v)\}$.

The key observation is that, since A is label-oblivious, the execution $A^G_{\overline{s},\omega}(s_0)$ makes the queries $\{(s_i, s_j) : (i,j) \in P(\omega)\}$, whereas $A^G_{\pi(\overline{s}),\omega}(\pi(s_0))$ makes the queries $\{(\pi(s_i), \pi(s_j)) : (i,j) \in P(\omega)\}$. Furthermore, both executions rule based solely on the answers and ω. Hence, Eq. (5) implies that *there exists $\omega \in \Omega$ such that, with probability at least $1/3$ over the choice of $\overline{s}$, there exists $(i,j) \in P(\omega)$ such that the answer to $\{s_i, s_j\}$ is different from the answer to $\{\pi(s_i), \pi(s_j)\}$* (i.e., $\{s_i, s_j\} \in E$ if and only if $\{\pi(s_i), \pi(s_j)\} \notin E$). Letting $\chi : [n]^2 \to \{0,1\}$ denote the adjacency predicate of the graph G (i.e., $\chi(u,v) = 1$ if and only if $\{u,v\}$ is an edge of G), and letting $s_0 = v$, we get

$$\Pr_{\overline{s}=\{s_1,...,s_\ell\} \in \mathcal{S}_v}[\exists (i,j) \in P(\omega) \text{ s.t. } \chi(s_i, s_j) \neq \chi(\pi(s_i), \pi(s_j))] \;\geq\; 1/3. \tag{6}$$

(Again, this holds because $A^G_{\overline{s},\omega}(s_0) = A^G_{\pi(\overline{s}),\omega}(\pi(s_0))$ holds if the same answers were provided to all corresponding queries.) Fixing ω as above, and letting $Q = P(\omega) \cap \binom{[\ell]}{2}$, we get

$$\Pr_{\overline{s}=\{s_1,...,s_\ell\} \in \mathcal{S}_v}\left[\begin{array}{l} \exists i \in [\ell] \text{ s.t. } \chi(v, s_i) \neq \chi(\pi(v), \pi(s_i)) \\ \vee \quad \exists \{i,j\} \in Q \text{ s.t. } \chi(s_i, s_j) \neq \chi(\pi(s_i), \pi(s_j)) \end{array}\right] \;\geq\; 1/3. \tag{7}$$

We now consider two cases, while letting $\chi(w,w) = 0$ for every w.

Case 1: The first event in Eq. (7) ***is likely.*** That is,

$$\Pr_{\{s_1,...,s_\ell\}\in\binom{[n]\setminus\{v\}}{\ell}}[\exists i\in[\ell] \text{ s.t. } \chi(v,s_i)\neq\chi(\pi(v),\pi(s_i))>0.1.$$

Using the union bound, it follows that $\Pr_{s\in[n]\setminus\{v\}}[\chi(v,s)\neq\chi(\pi(v),\pi(s))]>0.1/\ell$. Since this holds for every $v\in T$, we get

$$\begin{aligned}|\{(v,s)\in[n]^2:\chi(v,s)\neq\chi(\pi(v),\pi(s))\}| &\geq \sum_{v\in T}|\{s\in[n]\setminus\{v\}:\chi(v,s)\neq\chi(\pi(v),\pi(s))\}|\\ &= \sum_{v\in T}(n-1)\cdot\Pr_{s\in[n]\setminus\{v\}}[\chi(v,s)\neq\chi(\pi(v),\pi(s))]\\ &> \frac{0.1\cdot(n-1)}{\ell}\cdot|T|,\end{aligned}$$

which is $\Omega(n/q(n))\cdot|T|$, since $\ell=2q(n)$.

Case 2: The second event in Eq. (7) ***is likely.*** That is,

$$\Pr_{\{s_1,...,s_\ell\}\in\binom{[n]\setminus\{v\}}{\ell}}[\exists(i,j)\in Q \text{ s.t. } \chi(s_i,s_j)\neq\chi(\pi(s_i),\pi(s_j))]>0.1.$$

It follows that $\Pr_{\{r,s\}\in\binom{[n]\setminus\{v\}}{2}}[\chi(r,s)\neq\chi(\pi(r),\pi(s))]>0.1/|Q|$, which implies

$$|\{(r,s)\in[n]^2:\chi(r,s)\neq\chi(\pi(r),\pi(s))\}| > 0.1\cdot(n-1)^2/|Q|.$$

This is definitely $\Omega(n/q(n))\cdot|T|$.

Hence, in each of the cases, the size of the symmetric difference between G and $\pi(G)$ (i.e., $|E\triangle\{\{\pi(u),\pi(v)\}:\{u,v\}\in E\}|$) is $\Omega(n/q(n))\cdot|\{v\in[n]:\pi(v)\neq v\}|$. Since this holds for every bijection $\pi:[n]\to[n]$, it follows that G is $\Omega(n/q(n))$-robustly self-ordered. ■

2.3 Explicit construction of graphs with local self-ordering procedures

In light of Theorem 2.6, which asserts that n-vertex graphs that have $(O(\log n)^2)$ local self-ordering procedures are not necessarily $\Omega(n)$-robustly self-ordered, it is natural to ask whether it is easier to construct graphs having self-ordering procedures than to construct $\Omega(n)$-robustly self-ordered graphs. This seems to be the case, as demonstrated next: The point is that the construction presented in the following proof is much simpler than the known construction of $\Omega(n)$-robustly self-ordered graphs, which relies on non-malleable two-source extractors (cf. [6, Sec. 8]).

Theorem 2.9 (explicit construction with an efficient procedure): *There exists an efficient construction of a family of graphs that have a local self-ordering procedure. Furthermore, the n-vertex graphs are locally constructable* (i.e., the adjacency of two vertices in the graph can be determined in polynomial-time) *and the local self-ordering procedure runs in* $\mathrm{poly}(\log n)$*-time. On the other hand, the graph is not $\omega(n/\log^2 n)$-robustly self-ordered and each vertex in the graph has degree $\Theta(n/\log n)$.*

Using Theorem 2.8, it follows that these n-vertex graphs are $(n/\mathrm{poly}(\log n))$-robustly self-ordered. We note that Theorem 2.9 does not supersede Theorem 2.6, because the vertex degrees are linear in Theorem 2.6 and the quantitative relation is tighter there.[13]

Proof: The graph consists of logarithmically many cliques of noticeably different sizes that are connected in a way that reflects the location of the vertices in each clique: The (approximate) degree of each vertex identifies to which clique it belongs, whereas its exact location in this clique is indicated by the identity of the cliques in which it has neighbors. Indeed, each vertex has relatively few neighbors in other cliques, but sufficiently many such neighbors so to enable the localization procedure.

Specifically, For $\ell = O(\log n)$ and $m = \Theta(n/\ell^2)$, we consider a graph that consists of ℓ cliques of different sizes that are connected by bipartite graphs of edge-density $1/\mathrm{poly}(\ell)$. The i^{th} clique has $(\ell + i) \cdot m = \Theta(n/\ell)$ vertices, and each of its vertices is connected to at most $m/2$ vertices in other cliques. The identity of the cliques to which each vertex is connected encode its location; that is, we encode each vertex by an $\ell/3$-bit long string and connect it to at most $\ell/3$ of the (subsequent) cliques. Furthermore, we partition the vertices in each clique to cells, and connect vertices only if they reside in corresponding cells. This guarantees that the degrees of vertices are dominated by the size of the clique in which they reside; in particular, vertices in different cliques have sufficiently different degrees in the (final) graph. Details follow (see Figure 2).

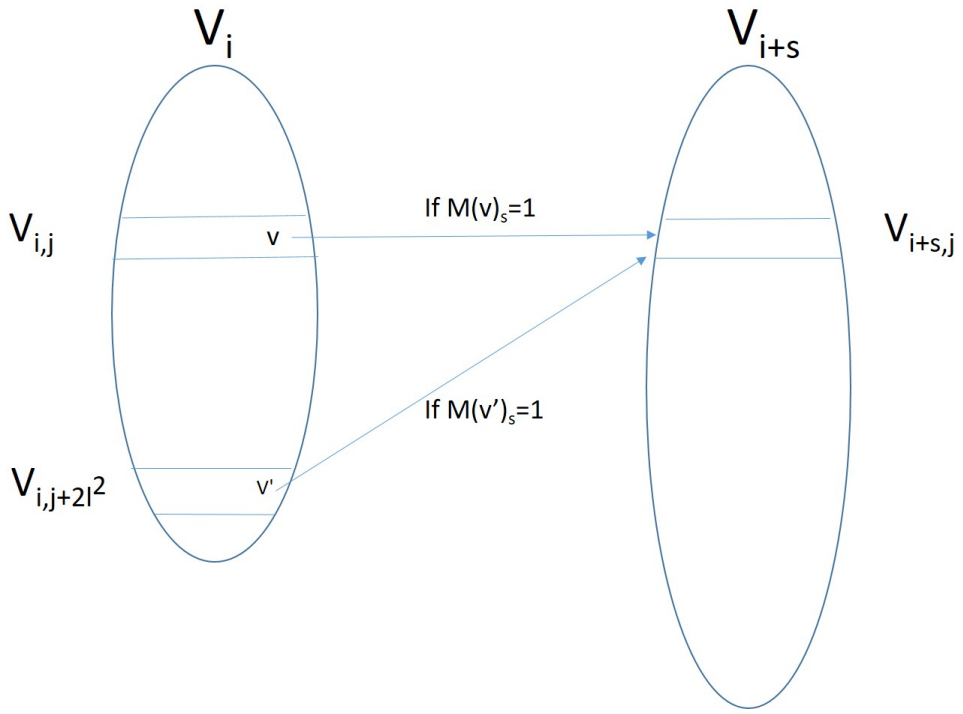

Figure 2: Detail in the construction demonstrating Theorem 2.9.

[13] Recall that Theorem 2.6 refers to n-vertex graphs in which each vertex has degree $\Theta(n)$. Also recall that Theorem 2.6 asserts a $O((\log n)^2)$-query local self-ordering procedure for an n-vertex graph that is not $(n/\log_2 n)$-robustly self-ordered. In contrast, the proof of Theorem 2.9 yields a local self-ordering procedure of query complexity $\widetilde{O}((\log n)^7)$.

***The cliques*:** For each $i \in \mathbb{Z}_\ell = \{0, 1, ..., \ell-1\}$, let C_i be an $(\ell+i)\cdot m$-vertex clique, with vertex-set $V_i = \{\langle i, j\rangle : j \in [(\ell + i) \cdot m]\}$.

Let $V = \bigcup_{i\in\mathbb{Z}_\ell} V_i$, and note that $n = |V| = \Theta(\ell^2 \cdot m) < 2^{\ell/3}$.

***Cells in cliques*:** For each $i \in \mathbb{Z}_\ell$, we partition V_i into $2\ell^2 + 2i\ell$ cells, each of size $m' = m/2\ell$; that is, for every $j \in \mathbb{Z}_{2\ell^2+2i\cdot\ell}$, let $V_{i,j} = \{\langle i, r\rangle : r \in [j \cdot m' + 1, ..., j \cdot m' + m']\}$.

(Indeed, $|V_i| = (2\ell^2 + 2i\ell) \cdot m' = (\ell + i) \cdot m$.)

***Connecting the cells*:** Using any (efficiently computable and invertible) injective map $M : V \to \{0, 1\}^{\ell/3}$ as an encoding, for every i, j (as above) and $s \in [\ell/3]$, we connect each vertex $v = \langle i, r\rangle \in V_{i,j}$ to all vertices in $V_{i+s \bmod \ell\,,\, j \bmod 2\ell^2}$ if and only if the s^{th} bit of $M(v)$ is 1; that is, $v \in V_{i,j}$ is connected to $u \in V_{i',j'}$ if and only if the following three conditions hold:

1. distance between cliques: $i' = i + s \bmod \ell$ for some $s \in [\ell/3]$;
2. corresponding cells: $j' = j \bmod 2\ell^2$;
3. the encoding of v matches the distance: $M(v)_s = 1$.

Hence, for $j \in \mathbb{Z}_{2\ell^2}$, vertices in $V_{i,j}$ are only connected to vertices in either V_i or in $V_{i+s \bmod \ell\,,\, j}$ or in $V_{i-s \bmod \ell\,,\, j} \cup V_{i-s \bmod \ell\,,\, j+2\ell^2}$ such that $s \in [\ell/3]$. As for $j \in \mathbb{Z}_{2\ell^2+2i\ell} \setminus \mathbb{Z}_{2\ell^2}$, vertices in $V_{i,j}$ are only connected to vertices in either V_i or in $V_{i+s \bmod \ell\,,\, j-2\ell^2}$ such that $s \in [\ell/3]$. See Figure 2. Note that, for every i, j and $s \in [\ell/3]$, if a vertex $v \in V_i$ is connected to some vertex in $V' \stackrel{\text{def}}{=} V_{i+s \bmod \ell\,,\, j}$, then v is connected to all $m' = \Omega(n/\ell^3)$ vertices in V'.

Envisioning the C_i's as residing on a cycle, the vertices in each C_i are only connected to the $C_{i'}$'s that are at distance at most $s \in [\ell/3]$ from C_i on the cycle, where the edges between V_i and V_{i+s} are confined to $\bigcup_{j\in\mathbb{Z}_{2\ell^2+2i\ell}}(V_{i,j} \times V_{i+s\,,\, j \bmod 2\ell^2})$.

Denoting the resulting graph by $G = (V, E)$, we observe that vertices of the different cliques can be easily told apart based on their degree, because vertices in C_i have degree at least $|V_i| - 1 = (\ell + i) \cdot m - 1$ and at most

$$(\ell + i) \cdot m + \frac{\ell}{3} \cdot \frac{m}{2\ell} + \frac{\ell}{3} \cdot 2 \cdot \frac{m}{2\ell} = (\ell + i + 0.5) \cdot m,$$

where the first term is due to the edges of C_i, and the second (resp., third) term is due to the edges that connect C_i and $C_{i+s \bmod \ell}$ (resp., C_i and $C_{i-s \bmod \ell}$) for $s \in [\ell/3]$.

A vertex v in a graph $G' = (V', E')$ that is isomorphic to G is located (in G) by taking a sample S of $\text{poly}(\log n)$ random vertices in G', and querying the subgraph of G' induces by $S \cup \{v\}$. Using the degrees of the vertices in this induced subgraph, we identifying the clique to which each of these vertices belongs, and using the adjacencies of v with S we obtain the encoding of the corresponding vertex of G under M, which yields its identity. That is, we decide that v corresponds to vertex $\langle i, r\rangle$ of G if v appears to reside in a clique of size $(\ell+i+0.25\pm0.25)\cdot m$ and for every $s \in [\ell/3]$ the vertex v is connected to a vertex that seems to resides a clique of size $(\ell + (i + s \bmod \ell) + 0.25 \pm 0.25) \cdot m$ if and only if $M(\langle i, r\rangle)_s = 1$.

More specifically, approximating the degree of $\text{poly}(\ell)$ many vertices upto $\pm 0.1m/\ell$ (w.h.p.) can be done by using a sample of $\widetilde{O}(\ell^6)$ vertices, since $m = \Theta(n/\ell^2)$, and so the foregoing procedure can be implemented using $\widetilde{O}(\ell^{12})$ queries. A closer look reveals that it suffices to use a sample of

$\widetilde{O}(\ell^3)$ vertices in order to hit all the $V_{i,j}$'s (w.h.p.), since $m' = \Theta(n/\ell^3)$, whereas we only need to approximate the degrees of the vertices in this sample upto $\pm 0.1m$ (w.h.p.), which can be done using a sample of size $\widetilde{O}(\ell^4)$. Hence, local self-ordering can be done using $\widetilde{O}(\ell^7)$ non-adaptive queries.

Lastly, we upper-bound the robustness parameter of G by considering a permutation π that switches C_1 and $\ell \cdot m$ of the vertices of C_2. The contribution of each non-fixed-point to the symmetric difference between G and $\pi(G)$ is smaller than $2m$, since the number of edges that go outside each clique is at most $m/2$ (and the difference in the sizes of C_1 and C_2 is m). Hence, G is not $2m$-robustly self-ordered, whereas $m = O(n/\log^2 n)$. ■

2.4 From robustness to locality: Proof of Theorem 1.7

While we do not know the answer to Problem 1.6 (i.e., whether any $\Omega(n)$-robustly self-ordered n-vertex graph has a $\mathrm{poly}(\log n)$-query local self-ordering procedure), we show an efficient transformation of $\Omega(n)$-robustly self-ordered n-vertex graphs into ones that are $\Omega(n)$-robustly as well as have a $\mathrm{poly}(\log n)$-query local self-ordering procedure.

Theorem 2.10 (from robustness to robustness plus locality): *There exists an efficient transformation of n-vertex graphs that are $\Omega(n)$-robustly self-ordered to $3n$-vertex graphs that are $\Omega(n)$-robustly self-ordered and have locally self-ordering procedures that runs in $\mathrm{poly}(\log n)$-time. Furthermore, the transformation is local in the sense that the adjacency predicate of the resulting graph can be computed by a $\mathrm{poly}(\log n)$-time reduction to the adjacency predicate of the input graph.*

Proof: The construction borrows some elements from the proof of Theorem 2.9. In particular, we use the idea of locating a vertex according to its adjacency with large subsets in a fixed partition of the vertex set (e.g., large cliques). Specifically, we shall combine any n-vertex $\Omega(n)$-robustly self-ordered graph G with $\Theta(\log n)$ locally distinguishable $\Theta(n/\log n)$-vertex cliques that will be used to locate vertices in G, where the vertices in the cliques will be located either via the other cliques or via the vertices of G. In addition, we connect the various graphs such that the robust self-ordering of G induces robust self-ordering of the larger resulting graph, denoted G'. The latter issue implies that we cannot use relatively sparse connections as in the proof of Theorem 2.9, which will complicate our construction and analysis.

More specifically, the graph G' consists of G and $\Theta(\log n)$ many $\Theta(n/\log n)$-vertex cliques, where the cliques are locally distinguishable by virtue of having a noticeably different number of vertices. The cliques will be used to locate vertices of G via assigning each vertex v of G a different codeword $C(v)$ that determines to which cliques the vertex v is connected such that v is connected to all vertices of the i^{th} clique if $C(v)_i = 1$ and is connected to none of these vertices otherwise. The robust self-ordering of G' will follow from the robust self-ordering of G and the edges connecting G and the rest of G'. Since the edges between G and the rest of G' are used in two different ways (i.e., to locate vertices of G and induce robustness on G'), we partition the rest of G' to two parts and connect them to G in different ways. Details follow.

Our construction of $G' = (V', E')$ uses a few ingredients. First, we use the n-vertex graph $G = ([n], E)$ that is guaranteed by the hypothesis. Let $\gamma > 0$ be a constant such that G is $\gamma \cdot n$-robustly self-ordered (and assume for simplicity that $\gamma < 1/50$). Furthermore, for simplicity, we

assume that each vertex in G has degree at least $\gamma \cdot n$.[14] The second ingredient we use is an efficiently computable error correcting code, $C : [n] \to \{0,1\}^\ell$, of constant relative distance, where $\ell = O(\log n)$. That is, the code C has distance $\Omega(\ell)$; for simplicity, we assume that the distance is at least $\ell/10$. In addition, for $\gamma' = \gamma/3$, we require C to satisfy the following two conditions, where the constant 0.25 is an arbitrary constant in $(0, 0.5)$:

1. Each codeword of C has Hamming weight $(0.25 \pm \gamma') \cdot \ell$; that is, for every $x \in [n]$, it holds that $|\{i \in [\ell] : C(x)_i = 1\}| = (0.25 \pm \gamma') \cdot \ell$.
2. For each coordinate $i \in [\ell]$, it holds that $\Pr_{x \in [n]}[C(x)_i = 1] = 0.25 \pm \gamma'$.

(Such a code can be constructed using the concatenated code paradigm.)[15] As stated above, we shall also use 2ℓ cliques of different sizes, where these sizes are evenly distributed between $0.9n/\ell$ and $1.1n/\ell$. Specifically, the i^{th} clique will have n_i vertices such that

$$n_i = \left(10\ell + (-1)^{i \bmod 2} \cdot \lceil i/2 \rceil\right) \cdot \frac{n}{10\ell^2}.$$

Hence, $n_{2j-1} + n_{2j} = 2 \cdot 10\ell \cdot n/10\ell^2 = 2n/\ell$ and $\sum_{i=1}^{\ell} n_i = n = \sum_{i=\ell+1}^{2\ell} n_i$ follows (assuming ℓ is even). Lastly, for simplicity, we shall first assume that n is close to a power of two, and let $\mathtt{IP}_2(v, u)$ denote the inner-product mod 2 (of the binary representation) of the vertices v and w (which are viewed as elements of $[n]$).

Construction 2.10.1 (the graph $G' = (V', E')$): *The graph G' consists of a copy of $G = ([n], E)$ and 2ℓ cliques, denoted $G_1, ..., G_{2\ell}$, where $G_i = (V_i, E_i)$ is a clique such that $|V_i| = n_i$, that are connected as follows* (see Figure 3).

- *Each vertex $v \in [n]$ of G is connected to all vertices in G_i such that $C(v)_i = 1$* (and $i \in [\ell]$).
- *Each vertex v in $\bigcup_{j \in [\ell]} V_j \equiv [n]$ is connected to all vertices in $G_{\ell+i}$ such that $C(v)_i = 1$* (and $i \in [\ell]$).
- *Each vertex v in $\bigcup_{j \in [\ell]} V_{\ell+j} \equiv [n]$ is connected to vertex w in G if and only if $\mathtt{IP}_2(v, w) = 1$.*

  (Both v and w are viewed as elements of $[n]$.)[16]

[14] By the robustness feature, G may contain at most one vertex of degree smaller than $0.5\gamma n$. We can make this vertex isolated at the cost of decreasing the robustness constant to $\gamma/2$, and leave this vertex isolated in the following construction of G'.

[15] We start with a Reed-Solomon code $C' : [n] \to \mathrm{GF}(p)^p$ such that $p = \ell/\ell'$ where $\ell' = \Theta(\log \ell)$, and note that $p^p = \ell^{(1-o(1)) \cdot \ell/\ell'} = n^{(1-o(1)) \cdot \frac{\ell}{\log_2 n} \cdot \frac{\log_2 \ell}{\ell'}} = n^c$, and that the constant $c > 0$ can be made arbitrary large (by picking a suitable ℓ). Hence, each codeword of C' has at most $p/c < \gamma' \cdot p/2$ zero coordinates, and each coordinate of a random codeword is uniformly distributed in $\mathrm{GF}(p)$. Next, we find an inner-code $C'' : \mathrm{GF}(p) \to \{0,1\}^{\ell'}$ that satisfies all conditions (with γ' replaced by $\gamma'/2$) by an exhaustive search on a suitable sample space of pseudorandom $p \times \ell'$ matrices. Specifically, we use an $\exp(-\ell')$-biased space on $2 \cdot p \cdot \ell'$-bit long strings in order to define $p \times \ell'$ Boolean matrices such that each entry is 1 with probability approximately $1/4$, and test that the distance between rows as well as the weight of rows and of columns are all as postulated. In the concatenated code C each codeword is a sequence of C''-codewords, which implies that Condition 1 holds. As for Condition 2, note that each coordinate in a *random* C-codeword corresponds to a coordinate in a *random* C''-codeword. Lastly, the relative distance of C is lower-bounded by the product of the relative distances of C' and C''.

[16] Actually, we use two injective mappings $\mu_1 : [n] \to S_n$ and $\mu_2 : \bigcup_{i \in [\ell]} V_{\ell+j} \to S_n$, where S_n is the set of the n lexicographically first strings in $\{0,1\}^{\lceil \log_2(n+1) \rceil} \setminus \{0^{\lceil \log_2(n+1) \rceil}\}$. Hence, $\mathtt{IP}_2(v, w)$ stands for $\mathtt{IP}_2(\mu_2(v), \mu_1(w))$.

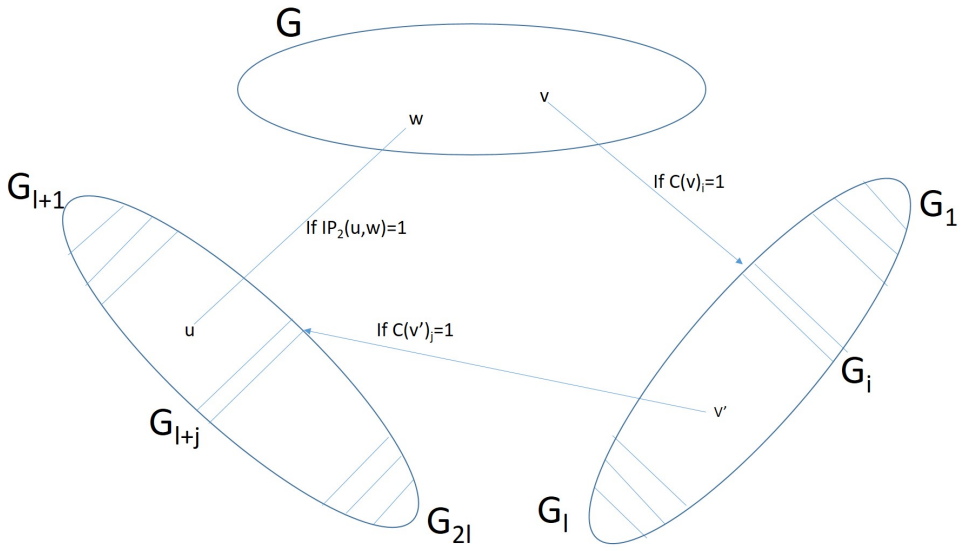

Figure 3: Illustration of Construction 2.10.1.

Recall that $n_i = (10\ell + (-1)^{i \bmod 2} \cdot \lceil i/2 \rceil) \cdot n/10\ell^2 \in [0.9n/\ell, 1.1n/\ell]$. *We shall refer to a three-way partition of* G', *calling* G *the* first part, *and* $\bigcup_{i\in[\ell]} G_i$ (resp., $\bigcup_{i\in[\ell]} G_{\ell+i}$) *the* second (resp., third) part. *Assuming that* ℓ *is even, each part has* n *vertices.*

Note that the bipartite graph connecting the first and third parts has edge-density approximately $1/2$, whereas the other two bipartite graphs (which connect other pairs of parts) have edge-density approximately $1/4$. These densities are reflected in the number of neighbors that each vertex has in the other parts. In particular, each vertex in the first part of G' has degree at least $\gamma n + (0.25n - \gamma' n) + (0.5n - o(n))$, where the first term is due to its neighbors in G and the second (resp., third) term is due to its neighbors in the second (resp., third) part. In contrast, each vertex in the second (resp., third) part has degree $2 \cdot (0.25n \pm \gamma' n) + o(n) = 0.5n \pm 2\gamma' n + o(n)$ (resp., $(0.25n \pm \gamma' \cdot n) + (0.5n \pm o(n)) + o(n) = 0.75n \pm \gamma' n \pm o(n)$), where the main contribution comes from neighbours in the other two parts.

The foregoing facts imply that the approximate degree of a vertex in G' determines in which of the three parts of G' it resides. Furthermore, for any vertex in the second (resp., third) part of G', we can determine in which G_i it resides by approximating the size of the maximum clique that is induced by its neighbors that reside in the second (resp., third) part. Both observations will be used in the localization process described next.

Claim 2.10.2 (G' is locally self-ordered): *The graph* G' *has a* $\mathrm{poly}(\log n)$*-time local self-ordering procedure. Furthermore, the procedure is non-adaptive and is based on examining the subgarph induced by the input vertex and* $O((\log n)^5)$ *random vertices.*

Proof: Intuitively, our localization process is anchored in the 2ℓ cliques (i.e., the G_i's). As hinted above, using a small sample of random vertices, it is easy to identify the vertices that reside in G (based on the vertex degree), and identify the clique in which each other vertex resides in (based on the subgraph induced by vertices that are not in G). Using this identification, we can locate the

vertices of the first (resp., second) part based on whether or not they neighbor each of the G_i's. The vertices of the third part are located using the locations of their neighbors in the first part. Details follow.

Letting S be a random $O(\log^5 n)$-subset of V', we show how to locate each vertex v of V' (including $v \in S$) based on the subgraph of G' induced by $S \cup \{v\}$. Our localization process proceeds as follow.

1. *Determining which vertices reside in the first part*: We first determine whether or not v is located in the first part of G', by relying on the degree dichotomy outlined above. Specifically, we rule that v resides in the first part if and only if it has more than $(0.75 + 0.5\gamma) \cdot |S|/3$ neighbors in S.

   Similarly, we determine for each vertex in S whether it resides in the first part of G', and let S_1 denote the corresponding set of vertices.

2. *Determining which vertices reside in the second part*: Assuming that v is in $V' \setminus V$, we determine whether or not v is located in the second part of G', by relying the number of neighbors v has in S_1. Specifically, we rule that v resides in the second part if and only if it has less than $0.375 \cdot |S_1|$ neighbors in S_1. (Indeed, we rely on the fact that vertices in the second (resp., third) part have approximately $0.25n$ (resp., $0.5n$) neighbors in the first part.)

   Similarly, we determine for each vertex in $S \setminus S_1$ whether or not it resides in the second part of G', and let S_2 and S_3 denote the corresponding sets of vertices.

3. *Determining the partition of S_2 and S_3 into cliques*: We determine the partition of S_2 (resp., S_3), among the ℓ cliques by considering the subgraph of G' induced by S_2 (resp., S_3); specifically, for $i \in [\ell]$, we denote by $S_{2,i} \subseteq S_2$ the vertices that reside in G_i (resp., that $S_{3,i} \subseteq S_3$ contains the vertices that reside in $G_{\ell+i}$). Note that, with high probability (over the choice of S), it holds that $|S_{2,i}| = (1 \pm o(1/\ell^2)) \cdot \frac{n_i}{3n} \cdot |S|$ and $|S_{3,i}| = (1 \pm o(1/\ell^2)) \cdot \frac{n_{\ell+i}}{3n} \cdot |S|$. In this case, the various sizes uniquely determine the indices of the various sets.

   Furthermore, the partition can be found in poly($|S|$)-time, since (w.h.p.) the subgraph of G' induced by S_2 (resp., S_3) consists of ℓ cliques.

4. *Locating vertices in the first part*: If v is located in the first part of G', then we determine its exact location in G according to its neighborhood in each of the G_i's of the *second* part. Specifically, denoting the (unknown) location of v in G by x, we determine $C(x)_i$ by checking whether v has a neighbor in $S_{2,i}$. Note that, unless $S_{2,i} \neq \emptyset$, it holds that $C(x)_i = 1$ if and only if $S_{2,i}$ contains a vertex that is connected to v (and in this case v is actually connected to all vertices in $S_{2,i}$).

   Similarly, we locate each vertex in S_1 (i.e., we determine the exact location of each vertex in S_1).

5. *Locating vertices in the second part*: If v is located in the second part of G', then we determine its exact location in G according to its neighborhood in each of the G_i's of the *third* part (i.e., according to the adjacencies in the various $S_{3,i}$'s). (Indeed, this is done analogously to Step 4.)[17]

[17] We can similarly locate each vertex in S_2, but this is not useful for us.

6. *Locating vertices in the third part*: If v is located in the third part of G', then we determine its exact location in G according to its neighborhood in the *first* part (or rather its adjacencies to vertices in S_1). We do so while relying on the fact that we have already located all vertices in S_1 (see Step 4).

   Specifically, a vertex $s \in S_1$ that is located at vertex w in G yields a linear equation on the location of vertex v in G'. Denoting the latter (unknown) location by x, we get the equation $\mathtt{IP}_2(w,x) = \chi(s,v)$, where $\chi(s,v) = 1$ if and only if s neighbors v (in the subgraph of G' induced by $S \cup \{v\}$). Hence, we get $\Omega(|S|) \gg \log_2 n$ linear equations in x (viewed as an Boolean vector), where the equations are almost uniformly and independently distributed (when S is random).[18] Solving this (full rank) linear system, we obtain x.

Indeed, once the partition of S according to the parts and cliques of G' is determined (i.e., once S_1 and the $S_{2,i}$'s and $S_{3,i}$'s are determined), the process of localization proceeds from the first part of G' to its second and third parts. That is, we first locate each vertex of the first part based on the *cliques it neighbors* in the second part. Likewise, we locate each vertex of the second part based on the *cliques it neighbors* in the third part. Lastly, we locate each vertex of the third part based on the *location of the vertices of* S_1 *in* G, which means that the localization of a vertex in the third part uses the prior localization of the vertices in S_1.

The foregoing process works provided that S hits each of the cliques (i.e., the G_i's) in proportion to its size and that S (or rather S_1) hits a set of vertices in G whose locations (viewed as $\lceil \log_2(n+1) \rceil$-dimensional vectors) contain a basis of the vector space that encodes $[n]$. These events occur with probability at least $1 - 1/\mathrm{poly}(n)$. Hence, with probability at least $1 - 1/\mathrm{poly}(n)$ over the choice of S, we can locate each vertex v in G' according to the subgraph induced by $S \cup \{v\}$. ■

Partial summary. Claim 2.10.2 asserts that the graph G' (as defined in Construction 2.10.1) has a $\mathrm{poly}(\log n)$-time locally self-ordering procedure. The next claim (i.e., Claim 2.10.3) asserts that G' is $\Omega(n)$-robustly self-ordered. Note that, so far (i.e., in the proof of Claim 2.10.2), we did not use the relative distance of the bijection C, although we did use the bounds on the weight of its codewords (in order to distinguish the vertices of the different parts according to their degrees). More importantly, the hypothesis that G is $\gamma \cdot n$-robustly self-ordered was not used so far either. Both hypotheses will be used next.

Claim 2.10.3 (G' is $\Omega(n)$-robustly self-ordered): *Recall that C has distance at least 0.1ℓ, and that $G = ([n], E)$ is γn-robustly self-ordered, where $\gamma < 1/50$. Then, $G' = (V', E')$ is $\Omega(\gamma n)$-robustly self-ordered.*

Proof: The current proof goes in an opposite direction to the proof of Claim 2.10.2: Whereas the localization process is anchored in the 2ℓ cliques (i.e., the G_i's), the analysis of the robust self-ordering of G' is anchored in the first part (i.e., the robustly self-ordered graph G). Recall that, for every permutation μ of the vertices of G', we need to compare the number of non-fixed-points in μ to the size of the symmetric difference between G' and $\mu(G')$. Intuitively, focusing at permutations that preserve the three-way partition of G', we first observe that the claim follows in the case that many of the non-fixed-points of the permutation reside in the first part, because in this case we can

[18] Here we used the hypothesis that $[n]$ is close to $\{0,1\}^{\lfloor \log_2 n \rfloor}$, which implies that the locations of the vertices in S are distributed almost uniformly and identically in $\{0,1\}^{\lfloor \log_2 n \rfloor}$.

rely on the subgraph of G' induced by the first part (i.e., G). Otherwise, we get a contribution (to the symmetric difference) of vertices in the third part that is due to their different adjacencies with the vertices of the first part. An analogue argument holds for vertices of the second part, based on their different adjacencies with the vertices of the third part. Details follow.

We consider an arbitrary permutation $\mu : V' \to V'$, and its set of non-fixed-points $T = \{x \in V' : \mu(x) \neq x\}$. Recalling that $V' = [n] \cup \bigcup_{i\in[2\ell]} V_i$, we let T' denote the set of vertices that are mapped by μ to a different part of G'; that is,

$$T' \stackrel{\text{def}}{=} \bigcup_{j\in[3]} \left\{v \in V'_j : \mu(v) \notin V'_j\right\},$$

where V'_j denotes the j^{th} part of G' (i.e., $V'_1 = [n]$, $V'_2 = \bigcup_{i\in[\ell]} V_i$ and $V'_3 = \bigcup_{i\in[\ell]} V_{\ell+i}$). We consider the following cases.

Case 1: $|T'| > \gamma \cdot |T|/2$**.** In this case, we consider the contribution of the vertices of T' to the symmetric difference between G' and $\mu(G')$. Each such vertex $v \in T'$ contributes $\Omega(n)$ units to the difference, where this contribution is due to the difference between the degree of vertices in the three parts. Specifically, recall that the vertices in the first part have degree at least $\gamma n + (0.75n - \gamma' n - o(n))$, whereas the vertices of the second and third parts have degree $0.5n \pm 2\gamma' n + o(n)$ and $0.75n \pm \gamma' n \pm o(n)$, respectively. (Recall that $\gamma' = \gamma/3 = \Omega(1)$.) Hence, in this case, the size of the symmetric difference between G' and $\mu(G')$ is $|T'| \cdot \Omega(n) = \Omega(\gamma \cdot |T| \cdot n)$.

Let $T_1 \stackrel{\text{def}}{=} \{v \in V'_1 : \mu(v) \in V'_1 \setminus \{v\}\}$, and note that $T_1 \subseteq T \setminus T'$.

Case 2: $|T'| \leq \gamma \cdot |T|/2$ ***and*** $|T_1| > |T|/100$**.** In this case, we consider the contribution of the vertices in T_1 (which move inside the first part of G' (i.e., G)) to the symmetric difference between G' and $\mu(G')$. Intuitively, by the robust self-ordering of G, each such vertex contributes at least $\gamma \cdot n$ units on the average, but we have to discount the neighbors that moved out of the first part. By the case hypothesis, the number of these potential neighbors (which reside in T') is at most $\gamma \cdot n/2$, and so each vertex $v \in T_1$ contributes at least $\gamma \cdot n/2$ units (on the average). Hence, in this case, the size of the symmetric difference between G' and $\mu(G')$ is at least $|T_1| \cdot \gamma n/2 = \Omega(\gamma \cdot |T| \cdot n)$.

Let $T_3 \stackrel{\text{def}}{=} \{v \in V'_3 : \mu(v) \in V'_3 \setminus \{v\}\}$, and note that $T_3 \subseteq T \setminus (T' \cup T_1)$.

Case 3: $|T'| \leq \gamma \cdot |T|/2$ ***and*** $|T_1| \leq |T|/100$ ***and*** $|T_3| > |T|/100$**.** In this case, we consider the contribution of the vertices in T_3 (which move within the third part of G') to the symmetric difference between G' and $\mu(G')$. The contribution of each such vertex $v \in T_3$ is due to the difference in its adjacencies in the first part; that is, to vertices $w \in V'_1$ that neighbor either v or $\mu(v)$ but not both. Recalling that the adjacency between the third and first parts is determined by $\mathtt{IP}_2$, the number of such vertices w equals $|\{w \in V'_1 : \mathtt{IP}_2(v, w) \neq \mathtt{IP}_2(\mu(v), w)\}| \approx n/2$.

As in Case 2, we need to discount vertices w that move out of V'_1; actually, here we need to discard any vertex $w \in V'_1$ that is a non-fixed-point of μ. This means that we should discard at most $|T'| + |T_1| \leq 2|T|/100 \leq 6n/100$ vertices, where the first inequality uses the case hypothesis (as well as $\gamma < 1/50$). Hence, each $v \in T_3$ contributes $\Omega(n)$ units; so, in this case, the size of the symmetric difference between G' and $\mu(G')$ is $|T_3| \cdot \Omega(n) = \Omega(|T| \cdot n)$.

Case 4: $|T'| \leq \gamma \cdot |T|/2$ ***and*** $|T_1| \leq |T|/100$ ***and*** $|T_3| \leq |T|/100$**.** In this case, most vertices in T are vertices v that reside in the second part such that $\mu(v) \neq v$ also resides in the second part. The contribution of each such vertex $v \in T_2 \stackrel{\text{def}}{=} T \setminus (T' \cup T_1 \cup T_3)$ is due to the difference in its adjacencies in the third part; that is, vertices $w \in V'_3$ that neighbor either v or $\mu(v)$ but not both. Recalling that the adjacency between the second and third parts is determined by C applied to vertices of the second part, the number of such vertices w (i.e., which neighbor either v or $\mu(v)$ but not both) equals

$$\begin{aligned}\sum_{i\in[\ell]} |\{w \in V_{\ell+i} : C(v)_i \neq C(\mu(v))_i\}| &\geq |\{i \in [\ell] : C(v)_i \neq C(\mu(v))_i\}| \cdot \min_{i\in[\ell]}\{n_{\ell+i}\} \\ &\geq 0.1\ell \cdot 0.9n/\ell,\end{aligned}$$

where the second inequality is due to the distance of C (which we assume to be at least $\ell/10$). Analogously to Case 3, we have to discard vertices of V'_3 that are non-fixed-points, whereas their number is upper-bounded by $|T' \cup T_3| \leq 2 \cdot |T|/100 \leq 6n/100$. Hence, each vertex $v \in T_2$ contributes at least $0.09n - 0.06n = \Omega(n)$ units, and we get a total contribution of at least $|T_2| \cdot \Omega(n) = \Omega(|T| \cdot n)$.

Hence, in each of the possible cases, the size of the symmetric difference between G' and $\mu(G')$ is $\Omega(\gamma \cdot |T| \cdot n) = \Omega(\gamma \cdot |V'|) \cdot |T|$. ■

Conclusion. Combining Claims 2.10.2 and 2.10.3, we established the theorem for any n that is closed to a power of two. This hypothesis was used for a single reason: It guaranteed that a uniformly distributed element of $[n]$ has a binary representation that is almost uniformly distributed in $\{0,1\}^{\lfloor \log_2 n \rfloor}$, which in turn implies that $\mathtt{IP}_2(u,v)$ is almost unbiased, when u is uniformly distributed in $[n]$ and $v \in [n]$ is fixed. We can obtain the latter effect, for any $n \in \mathbb{N}$, by using ideas as in [6, Rem. 8.9].

Specifically, we replace $\mathtt{IP}_2(u,v)$ by $\mathtt{IP}_2(G(u),G(v))$, where $G : [n] \to \{0,1\}^{\lceil \log_2(n+1) \rceil} \setminus \{0\}^{\lceil \log_2(n+1) \rceil}$ is a small-biased generator that is injective. In this case, for each fixed v and a uniformly distributed $u \in [n]$, it holds that $\mathtt{IP}_2(G(u),G(v))$ is almost unbiased, since this bit is a non-zero linear combination of the bits of $G(u)$. As for constructing the generator G, for $t = \omega(1)$ such that $\exp(\widetilde{O}(2^t)) \leq \mathrm{poly}(\log n)$, we let $\ell = \lceil \log_2(n+1) \rceil$, associate $[n]$ with the n first elements in $\lceil [n/2^{\ell-t}] \rceil \times \{0,1\}^{\ell-t}$ and define $G(s',s'') = (G'(s'),s'')$ such that $G' : [\lceil n/2^{\ell-t} \rceil] \to \{0,1\}^t \setminus \{0^t\}$ is a small-biased generator (which is also injective), which is found by exhaustive search. (Indeed, the upper bound on t was selected in order to make the exhaustive search feasible, whereas the lower bound on t was selected to make 2^{-t} be $o(1)$.) ■

Digest. Construction 2.10.1 (i.e., the graph G') combines a $\Omega(n)$-robustly self-ordered n-vertex graph (i.e., G) with a logarithmic number of $\Theta(n/\log n)$-vertex cliques arranged in two parts. These three parts are connected by bipartite graphs of constant edge-density. The cliques serve as an "anchor" for the local self-ordering procedure of G': Using the adjacencies of vertices to these cliques, we locate the vertices of the first and second parts of G', whereas the vertices of the third part are located via the already located vertices of the first part. In contrast, the first part (i.e., G) is the anchor for establishing the $\Omega(n)$-robust self-ordering of G': The hypothesis that G is $\Omega(n)$-robustly self-ordered implies that each non-fixed-point of μ that resides in the first part contributes $\Omega(n)$ units to the symmetric difference between G' and $\mu(G')$. On the other hand, when

almost all vertices in the first part of G' are fixed-points, each non-fixed-point in the third part of G' contributes $\Omega(n)$ units to the symmetric difference, by virtue of their neighbors in the first part. Likewise, if there are few non-fixed-points in the third part, then we lower-bound the size of symmetric difference by considering the non-fixed-points of the second part (and their neighbors in the third part).

Construction 2.10.1 is based on a three-way partition of the vertices, where the original graph G is defined as the first part, and the smaller cliques (i.e., the G_i's) are partitioned among the second and third parts. The reason we use a three-way partition is that we need two different types of edges between G and the rest of the graph. The first type of edges serves for locating the vertices of G according to their adjacencies in the cliques, whereas the second type is used for the analysis of the robust self-ordering of G' (in case the permutation fixes almost all vertices in G).

3 The Bounded-Degree Graph Regime

Recall that in the bounded-degree graph regime we seek graphs that are $\Omega(1)$-robustly self-ordered, and that oracle access to a graph (as provided to a local self-ordering procedure) means oracle access to its incidence function. We show that in this regime robustly self-ordering and local self-ordering are almost orthogonal; that is, even extremely strong versions of one notion do not imply very weak versions of the other notion.

3.1 Proof of Part 1 of Theorem 1.3

We show that very simple graphs, which are evidently far from being $\Omega(1)$-robustly self-ordered, have simple local self-ordering procedures.

Theorem 3.1 (local self-ordering does not imply robust self-ordering): *There exist explicit n-vertex graphs of maximal degree 3 that have $O(\log n)$-time locally self-ordering deterministic procedures, but are not $(3/n)$-robustly self-ordered.*

The failure of a potential implication is extremely strong. We use the strongest possible notion of local self-ordering (i.e., deterministic procedures of logarithmic[19] time complexity), and rule out an extremely weak notion of robust self-ordering (i.e., anything above merely being self-ordered).

Proof: We first present the construction in terms of directed graphs with edge-coloring. In these terms, the graph is a balanced binary (positioned) tree of depth $\log_2 n$ with edges directed from the root to the leaves, colored left and right, respectively. Specifically, each internal vertex has two children, and the edges are directed from it to its children such that one edge is colored 'left' and the other is colored 'right'.

This (directed and edge-colored) graph can be locally self-ordered by going from the given vertex towards the root (using edges that are directed in the opposite direction), and collecting the edge-colors, which determines the vertex's location. To see that this tree is not $3/n$-robustly self-ordered, consider the permutation that flips the two sides of the tree: This permutation has

[19] Actually, one can envision a procedure of query complexity $O(\frac{\log n}{\log \log n})$, but this is the absolute minimum. The reason is that the labels obtained by such a procedure are irrelevant, and so the actual information contents of the answers to q queries is an unlabelled graph with q edges, whereas the number of such unlabelled graphs is $\exp(O(q \log q))$.

$n-1$ non-fixed-points, but the corresponding symmetric difference is only 2 (i.e., the two edges incident to the root), which yields a robustness ratio of at most $2/(n-1)$.

To transport this example to our model (of undirected graphs with no edge colors), we replaced the two directed and colored edges by two constant-sized gadgets that encode the edge's direction and color. Specifically, an edge from u to v colored 'left' is replaced by a 5-path $(u, \mathrm{t}_{u,v}, \mathrm{t}'_{u,v}, \mathrm{h}_{u,v}, \mathrm{h}'_{u,v}, v)$ augmented by a vertex $\mathrm{l}_{u,v}$ that is connected to $\mathrm{t}_{u,v}$, and ditto for color 'right' except that $\mathrm{l}_{u,v}$ is connected to $\mathrm{t}'_{u,v}$, where 't' stands for tail, 'h' for head, and 'l' for leaf. (The direction is encoded by connecting the leaf to a vertex at distance at most 2 from a single endpoint of the path, whereas the color is encoded by whether this distance is 1 or 2.)

Note that each type of vertex in the resulting tree is easily identified by its constant distance neighborhood (which identifies the gadget in which it resides or those to which it is connected). Hence, the local self-ordering procedure of the former case can be modified at a constant factor overhead. On the other hand, the permutation that flips the two sides of the resulting tree has only one fixed-point (i.e., the original root), and the corresponding symmetric difference is only 2 (i.e., the edges connecting the leaves that appear in the two gadgets incident at the root). ■

3.2 Proof of Part 2 of Theorem 1.3

We show that $\Omega(1)$-robust self-ordered graphs may have no local self-ordering procedures of poly-logarithmic query complexity. In fact, we prove a stronger statement.

Theorem 3.2 (robust self-ordering does not imply local self-ordering): *There exist n-vertex bounded-degree graphs that are $\Omega(1)$-robustly self-ordered, but do not have local self-ordering procedures of query complexity $o(\sqrt{n})$. Furthermore, this holds for a $1-o(1)$ fraction of the $O(1)$-regular graphs.*

The failure of a potential implication is extremely strong. We use the strongest possible notion of robust self-ordering, and rule out local self-ordering procedures of high query complexity, regardless of their computational complexity. We warm, however, that the following proof is the most complex in the current paper, although its underlying idea is quite simple.

Proof: The key observation is that, in the case of regular graphs, the only meaningful information that a local exploration of the graph can obtain arises from encountering a simple cycle in the graph. Hence, proving lower bound on the query complexity of local self-ordering procedures can be reduced to proving upper bounds on the probability that such procedures find a simple cycle. Furthermore, as long as the procedure does not encounter a simple cycle, it is "practically non-adaptive" (see below). Hence, it suffices to upper-bound the probability that a "blind" exploration of the graph yields some simple cycle, which in turn reduces to upper-bounding the probability that the next exploration-step closes a simple cycle. Lastly, we observe that the latter probability can be upper-bounded in terms of the probability that a *non-backtracking* random walk closes a simple cycle.

A non-backtracking random walk is a walk that at each step chooses uniformly at random one of the neighbors of the current vertex other than the neighbor visited in the previous step [1]. Such a walk captures the foregoing "blind" exploration since in each step the explorer can avoid going back on the last edge it has traversed, but all unused edges are practically indistinguishable. Note that if a non-backtracking walk closes a simple cycle, then some sequence of steps in it constitutes a simple cycle. Hence, we consider the probability that a non-backtracking random walk consists of a simple cycle.

Definition 3.2.1 (probability of forming a simple cycle): *For a graph $G = ([n], E)$, the* probability of forming a simple cycle in an ℓ-step random walk*, denoted $\mathtt{sc}_\ell(G)$, is the probability that a non-backtracking random walk that starts at a uniformly distributed vertex s reaches s in its ℓ^{th} step after visiting $\ell - 1$ distinct vertices. The* probability of forming a simple cycle in G*, denoted $\mathtt{sc}(G)$, is $\max_{\ell \in [n]}\{\mathtt{sc}_\ell(G)\}$.*

We mention that $\mathtt{sc}_\ell(G) = 0$ if ℓ is either smaller than the girth of G or larger than n. Furthermore, $\mathtt{sc}_{\Omega(\log n)}(G)) = (1 \pm o(1))/n$ if G is a regular expander. The choice to take the maximum up to n is rather arbitrary; we could, as well, have taken the maximum up to the query complexity that we care about. We shall show (see Lemma 3.2.3) that most $O(1)$-regular n-vertex graphs G satisfy $\mathtt{sc}(G) = O(1/n)$. But before doing so, we prove that this result implies the claimed lower bound (on the query complexity of local self-ordering procedures for such graphs). More generally, we prove the following.

Lemma 3.2.2 (the reduction): *For $d \geq 3$ and any d-regular graph $G = ([n], E)$, the probability that a q-query procedure succeeds in locating a random vertex in a random isomorphic copy of G is upper-bounded by $O(q^2 \cdot \max(\mathtt{sc}(G), 1/n))$.*

Proof: We start with an overview of the proof, which refers to an arbitrary q-query algorithm that explores a graph G', which is a random isomorphic copy of G, starting at an arbitrary vertex given to it. We may assume, without loss of generality, that this algorithm is deterministic (by fixing the best possible sequence of coin tosses). Furthermore, we observe that the labels of the vertices that appear as queries or as answers to queries are immaterial; all that matters is the equality and inequality relation between these labels.

Next, we consider the event in which an answer to a query equals a vertex that was visited before, where the probability space is uniform over all G' obtained by letting $G' = \pi(G)$. An *uninformative* case is that we queried u for one of its neighbors, received w as an answer, and later queried w and received the answer u. The informative cases, which are actually unlikely and are called *surprising*, are of two types: Either the answer closes a cycle in the subgraph seen so far, or it joins two connected components of that subgraph. We observe that before any surprising event occurs, the subgraph seen so far is a directed forest, where the edges are directed from the queries to the answers. Furthermore, the structure of this forest is determined beforehand, since the answers are practically determined by this hypothesis; that is, the forest represents a sequence of queries that are all answered by vertices that were unvisited till the time of the query.

At this point we make two key observations. The first observation is that the probability of a surprising event is upper-bounded by $p \stackrel{\text{def}}{=} \binom{q}{2} \cdot \max(O(\mathtt{sc}(G)), 1/(n-q))$, where the first term is due to answers that close a cycle and the second term is due to answers that merge two trees. The second observation is that if no surprising event occurs, then (w.v.h.p) the algorithm fails to locate the input (i.e., start) vertex. The latter observation relies on the hypothesis that graph is regular, which implies that no information is leaked by a vertex's degree. Needless to say, the precise argument (presented next) involves providing a more precise description of the operation of a generic algorithm and the "exploration forest" defined by it.

The actual proof: preliminaries. To simplify the exposition, we define the incidence queries as providing not only the neighbor w of the queried vertex u but also the index of u in the list of neighbors of w; that is, the query (u, j) is answered by (w, k) such that w is the j^{th} neighbor of u, which is the k^{th} neighbor of w. Needless to say, this only makes the algorithm stronger, but it

offers the benefit of assuming (w.l.o.g.) that after making the query (u, j) and receiving the answer (w, k), the algorithm never queries (w, k). This corresponds to avoiding traversing an edge in both directions (i.e., uninformative events are avoided). In addition, we assume (also w.l.o.g.) that the algorithm never makes the same query twice.

Next, we note that the labels of the vertices given as answers to the incidence queries are immaterial; all that matters is the equality and inequality relation between these labels. Likewise, the indices of vertices in their neighbor's incidence list do not matter; all that matters is that our queries refer to indices that were not given as answer (i.e., we may query (u, j) only if (u, j) was not provided as an answer to a prior query). Hence, we may replace the query (u, j) by a request for a neighbor of u that was not known already to be a neighbor of u. Lastly, incidence queries regarding a vertex that was not visited so far are replaced by a request for a new vertex followed by an incidence query to it. In light of the above, the algorithm may refer to vertices by the order in which they were first visited.

Hence, the algorithm's queries are of two types: (1) an incidence query of the form "provide a new neighbor of the i^{th} vertex visited so far" (assuming that this is possible)[20], and (2) a request to provide a new vertex (which, once obtained, is defined as visited). Note that queries of type (1) may be answered by a vertex that was not visited before (and is defined as visited at this time) or by a vertex that was visited before (either via a different vertex or via a request of type (2)). In the latter case we call the answer surprising and note that it either closes a cycle (within a connected component of the subgraph seen before) or causes two connected components (of this subgraph) to merge.

The connected components that we refer to are those in the directed graph defined by the queries and answers, where edges are directed from a query to its answer. This directed graph contains also the input (or start) vertex. Note that if none of the answers is surprising, then this graph is is directed forest.

Assuming that none of answers is surprising, the sequence of queries made by the algorithm is uniquely determined, since all vertices have the same degree and so the question of whether all neighbors of a vertex were visited is predetermined by the queries (under the assumption that no answer is surprising). The i^{th} element in this query-sequence is either a request for a new vertex or a request for a new neighbor of the j^{th} visited vertex for some $j \in \{0, 1, ..., i-1\}$. In both cases, the answer is defined to be the i^{th} visited vertex, where *the input vertex is defined as the* 0^{th} *visited vertex*. (Recall that the input vertex is the vertex to be located by the algorithm.)

Note that the definition of the sequence of queries has been presented as referring to an actual execution of the algorithm, when the answers are provided by the graph G', and while assuming that none of the answers is surprising. Nevertheless, the same definition applies to a mental experiment in which the algorithm is invokes and the answers are fictitiously emulated such that none of them is surprising. Indeed, the two definitions collide if no surprising answer occurs in the real execution (where the answers are provided by G'). When this condition does not hold in a real execution, it is the case that the i_1^{th} visited vertex (according to this definition) equals the i_2^{th} visited vertex.

Hence, we shall analyze what happens in a real execution while referring to the forest that is defined by the foregoing mental experiment. This makes sense because our analysis is focused at the case that no surprising event occurs, and as long as this holds both definitions are equivalent. Such a

[20] If the i^{th} vertex was visited from the j^{th} vertex, then the j^{th} vertex is excluded from the list of possible answers. Also excluded are all neighbors previously visited from the i^{th} vertex. Hence, an answer is possible if and only if less than d neighbors were excluded so far.

(real) random execution is described by a sequence of random variables $(X_0, ..., X_q)$ such that X_0 is the input vertex (given to the algorithm), and X_i is the answer to the i^{th} query. Specifically, if the i^{th} query is a request for a new vertex, then X_i is uniformly distributed in $[n] \setminus \{X_0, X_1, ..., X_{i-1}\}$, and otherwise X_i in a new neighbor of a previous X_p (i.e., if the i^{th} query is a request for a new neighbor of X_p, where $p \in \{0, 1, ..., i-1\}$, then X_i is uniformly distributed among the unknown neighbors of X_p).[21] A surprising event occurs if $X_i = X_j$ for some $i > j$, and we upper-bound the probability for the occurrence of any surprising event by taking a union bound over all pairs (i, j) and upper-bounding the probability that the corresponding event is the first one. That is, for each (i, j) such that $i > j$, we upper-bound the probability that $X_i = X_j$ and $X_{i'} \neq X_{j'}$ for every $i' < i$ and $j' < i'$. Denoting the combined event by $\chi_{i,j}$, we shall prove that $\Pr[\chi_{i,j}] = O(\max(\mathtt{sc}(G), 1/(n-q)))$.

Bounding the probability of a surprising event. Recall that $\chi_{i,j}$ holds only if the i^{th} query is a request for a new neighbor of some vertex X_p such that $p < i$. Furthermore, $X_0, ..., X_{i-1}$ are distinct. We distinguish two cases: The case that X_p and X_j reside in the same tree (where $X_i = X_j$ closes a simple cycle that contains the (non-tree) edge $\{X_p, X_j\}$), and the case that X_p and X_j reside in different trees (where $X_i = X_j$ merges these trees with the edge $\{X_p, X_j\}$). We stress that we consider a directed forest (i.e., a collection of trees directed from the roots to the leaves), but the cycle being closed is not necessarily directed (see below).

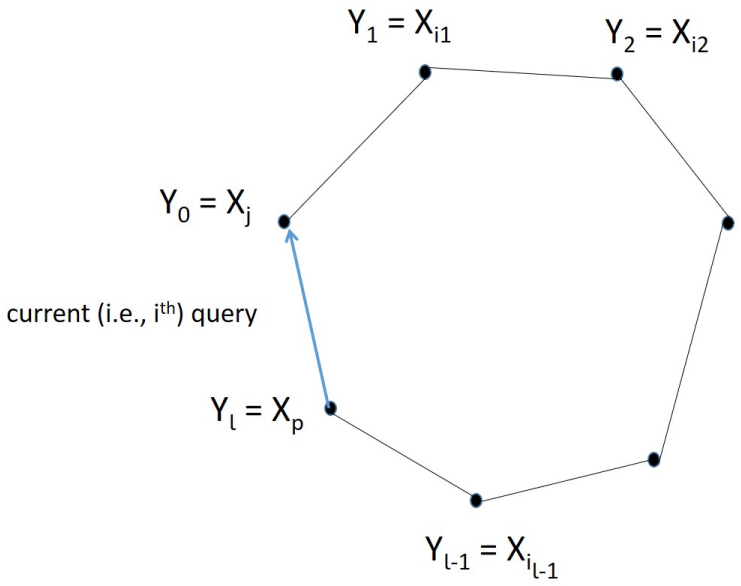

Figure 4: The query that closes a cycle (detail for the proof of Lemma 3.2.2).

We start with the first case (i.e., X_p and X_j reside in the same tree (and the current query closes a simple cycle that contains the edge $\{X_p, X_j\}$)), which is more interesting. Letting ℓ denote the (undirected) distance between X_j and X_p on the tree, note that the (undirected) path connecting X_j and X_p in the tree has the form $X_j, X_{i_1}, ..., X_{i_{\ell-1}}, X_p$. We will show that in this case $\Pr[\chi_{i,j}] = O(\mathtt{sc}_{\ell+1}(G))$, where the probability space is uniform over all permutations $\pi : [n] \to [n]$

[21] That is, X_k is excluded if either X_k was provided as a prior answer to a new neighbor request regarding X_p or X_p was provided as answer to a new neighbor request regarding X_k.

and the algorithm gets its answers from $G' = \pi(G)$. To clarify the exposition, we rename the vertices on the path between X_j and X_p by $Y_0, Y_1, ..., Y_\ell$ such that $Y_k = X_{i_k}$ is the vertex at (undirected) distance k from $Y_0 = X_j$ on the said path (e.g., $i_0 = j$ and $i_\ell = p$). Indeed, the i_k's are distinct. See Figure 4.

Note that the foregoing path (i.e., the sequence of adjacent vertices $Y_0, ..., Y_\ell)$) is not necessarily directed from Y_0 to Y_ℓ, and the following sub-cases will distinguish the two possibilities (i.e., a path directed from Y_0 to Y_ℓ versus two paths from some Y_i ending at Y_0 and Y_ℓ). Another distinction refers to the question of whether one of the Y_k's is the root of the tree on which all these Y_k's reside. Hence, we have four sub-cases, which are analyzed next, while recalling that in all sub-cases the undirected $(\ell+1)$-cycle consists of the foregoing undirected ℓ-path and the edge $\{Y_\ell, Y_0\}$. In each of the four sub-cases, we upper-bound the probability of forming a simple $(\ell+1)$-cycle (consisting of the vertices $Y_0, ..., Y_\ell$), while noting that the probability of the corresponding event (i.e., $\chi_{i,j}$) can only be lower (since it conditions on not having surprising events in other parts of the forest).

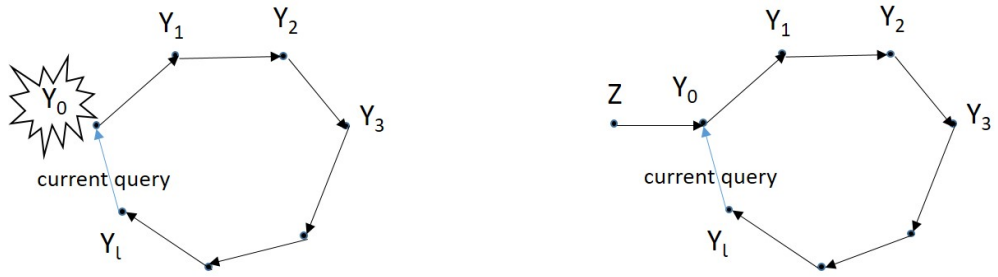

Figure 5: The first two sub-cases in the proof of Lemma 3.2.2.

1. The vertex Y_0 is a root of the tree in which it resides (i.e., Y_0 is a response to a "new vertex" request), and each other Y_i is the answer to a ("new neighbor") query regarding Y_{i-1}. In this sub-case, a simple directed path of length ℓ leads from Y_0 to Y_ℓ, and the edge from Y_ℓ to Y_0 closes a simple directed $(\ell+1)$-cycle. (See the l.h.s. image in Figure 5, where the explosion image represents the answer to a "new vertex" request.)

   The probability that the algorithm traverses this simple $(\ell+1)$-cycle equals the probability that a $(\ell+1)$-step non-backtracking random walk forms a simple cycle in G, which equals $\mathsf{sc}_{\ell+1}(G)$. The foregoing assertion uses the observation that the order in which the algorithm traverses edges of the forest is immaterial, and so we may assume that the path from Y_0 to Y_ℓ is traversed before any other edge is traversed. The latter assumption is justified by the hypothesis that Y_0 is the root of a tree; that is, the said cycle is formed by a single "new vertex" request followed by ℓ "new neighbor" queries each referring to the previous answer (i.e., last encountered vertex).

2. The vertex Y_0 is not the root of the tree in which it resides (i.e., Y_0 is a response to a "new neighbor" request regarding some Z), and (as in the previous sub-case) each other Y_t is the answer to a ("new neighbor") query regarding Y_{t-1}. Hence, as in the previous sub-case, we obtain a simple directed $(\ell+1)$-cycle that goes from from Y_0 to Y_ℓ, and then returns to Y_0. (See the r.h.s. image in Figure 5.)

The probability that the algorithm traverses this simple $(\ell+1)$-cycle equals the probability that a $(\ell+1)$-step non-backtracking random walk closes a *simple* cycle in G *conditioned on the first step not taking a specific neighbor of* Y_0 (i.e., not taking the aforementioned vertex Z). This means that the probability space of the walks taken by the algorithm is cut by a factor of $d/(d-1)$. It follows that in this sub-case the probability that a simple $(\ell+1)$-cycle is formed is upper-bounded by $\frac{d}{d-1} \cdot \mathtt{sc}_{\ell+1}(G)$.

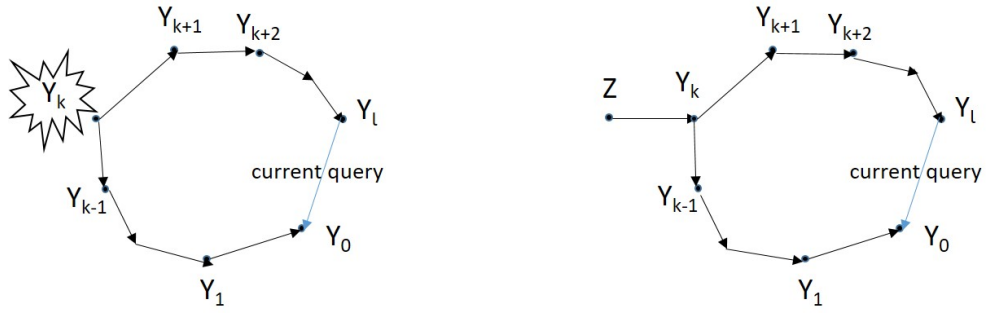

Figure 6: The other two sub-cases in the proof of Lemma 3.2.2.

3. For some $k \in [\ell]$, the vertex Y_k is a root of the current tree, and the edges are directed from Y_k to Y_ℓ (as in the previous sub-cases) and from Y_k to Y_0. That is, the path $(Y_0, ..., Y_\ell)$ consists of two directed subpaths, one going from Y_k to $Y_{k+1}, ..., Y_\ell$, and the other going from Y_k to $Y_{k-1}, ..., Y_0$. (Recall that all Y_t's are distinct, and see the l.h.s. image in Figure 6.)

   By reversing the order in which the second path is traversed, we note that this sub-case is "isomorphic" to the first sub-case. Specifically, in the current sub-case we have two "new neighbor" requests for Y_k and a single requests for each Y_t such that $t \in [\ell] \setminus \{k\}$, rather than a single "new neighbor" request for each Y_t such that $t \in \{0, 1, ..., \ell\}$. In other words, the space of possible choices corresponds to $[d] \times [d-1] \times [d-1]^{\ell-1}$ rather than to $[d] \times [d-1]^\ell$ (as in the first sub-case). Hence, the probability that a simple $(\ell+1)$-cycle is formed is upper-bounded again by $\mathtt{sc}_{\ell+1}(G)$.

4. The last sub-case is a hybrid of the second and third sub-cases. As in the third sub-case, for some $k \in [\ell]$, the edges are directed from Y_k to Y_ℓ and from Y_k to Y_0, but Y_k is not a root of the current tree (analogously to the second sub-case). (See the r.h.s. image in Figure 6.)

   In this sub-case, we make two choices for neighbors of Y_k, but have to avoid a specific neighbor (as in the second sub-case). Hence, the space of possible choices corresponds to $[d-1] \times [d-2] \times [d-1]^{\ell-1}$, which means that the probability space (of the non-backtracking random walks) is cut by a factor of $\frac{d(d-1)}{(d-1)(d-2)}$. Hence, in this sub-case, the probability that a simple $(\ell+1)$-cycle is formed is upper-bounded by $\frac{d}{d-2} \cdot \mathtt{sc}_{\ell+1}(G)$.

Thus, in all sub-cases, we got a probability bound of at most $3 \cdot \mathtt{sc}_{\ell+1}(G)$, since $d \geq 3$.

We now turn to the second case, which is the case that these two vertices (i.e., X_p and X_j) reside in different trees (which are merged by the edges $\{X_p, X_j\}$). The probability that this event occurs (i.e., that a new neighbor request for X_p is answered by X_j) is at most $1/(n-q)$, because there are at least $n-q$ possible locations for the root of the second tree and only one of them

yields the said event. Taking account of both cases, it follows that the probability of a surprising event occurring during the q-query execution on G' is at most $\epsilon \stackrel{\text{def}}{=} \binom{q}{2} \cdot \max(3 \cdot \mathtt{sc}(G), 1/(n-q))$, as claimed.

Showing that locating fails when there is no surprising event. Lastly, assuming that $\epsilon < 1/2$, we claim that the local self-ordering procedure errs with very high probability (when querying G' that is isomorphic to G), because, *when no surprising events occurs*, little information is revealed on the location of the start (i.e., input) vertex s in G. The key observation is that *the only information that is revealed to the algorithm is the non-occurring of a surprising event*. Hence, fixing an input $s \in [n]$, for a uniformly distributed permutation $\pi : [n] \to [n]$, we consider the distribution of the location of s in G (i.e., $\pi^{-1}(s)$) given that no surprising event occurs (when exploring $G' = \pi(G)$ on input s). Specifically, let $\chi^{G'}(s)$ denote the foregoing surprising event, where $G' = \pi(G)$ is a uniformly distributed isomorphic copy of G. Then, for every $i \in [n]$, it holds that

$$\begin{aligned}
\Pr_\pi[\pi^{-1}(s)=i \mid \neg\chi^{\pi(G)}(s)] &= \frac{\Pr_\pi[\neg\chi^{\pi(G)}(s) \mid \pi^{-1}(s)=i] \cdot \Pr_\pi[\pi^{-1}(s)=i]}{\Pr_\pi[\neg\chi^{\pi(G)}(s)]} \\
&\leq \frac{\Pr_\pi[\pi^{-1}(s)=i]}{\Pr_\pi[\neg\chi^{\pi(G)}(s)]} \\
&\leq \frac{1/n}{1-\epsilon}
\end{aligned}$$

which is less than $2/n$. This means that $\pi^{-1}(s)$ is hard to predict when all we know is that no surprising event occurred during an exploration of $\pi(G)$. Specifically, when exploring a random isomorphic copy $G' = \pi(G)$ of G, the probability that a q-query algorithm correctly locates the start vertex is smaller than $\epsilon + (2/n)$. ■

Proving the claim of the theorem. Using Lemma 3.2.2, it suffices to upper-bound the probability of closing a simple cycle in a random d-regular graph.

Lemma 3.2.3 (the cycle forming probability of random graphs): *For any $d \geq 3$ and $t > 1$, for at least $1 - O(1/t^2)$ of the d-regular n-vertex graphs G, it holds that $\mathtt{sc}(G) = O(t/n)$.*

Taking $t = \omega(1)$ and recalling that $1 - o(1)$ fraction of the $O(1)$-regular graphs are $\Omega(1)$-robustly self-ordered (see [6, Thm. 6.1]), the theorem follows. We mention that Proposition 3.3 implies a weak bound of $\mathtt{sc}(G) \leq n^{-\Omega(1)}$, since almost all d-regular n-vertex graphs G have logarithmic girth.

Proof: Using the fact that, with very high probability, the graph G is an expander, we may assume that $\mathtt{sc}_\ell(G) < 1.01/n$ for all sufficiently large $\ell = \Omega(\log n)$. Hence, we focus on upper-bounding $\mathtt{sc}_\ell(G)$ for $\ell = O(\log n)$.

To facilitate the proof, we prove the result in the *configuration model*, in which a random d-regular multi-graph is generated by selecting uniformly a perfect matching among the dn ports, where each vertex has d ports that represence its d possible incidences (cf., e.g., [2]). We note that a simple d-regular graph is generated with probability approximately $\exp(-d^2/4)$, and that each simple d-regular graph is generated with the same probability. Hence, an upper bound on the probability of a bad event in the configuration model yields an upper bound (which is at most an $\exp(d^2/4)$ factor larger) on the probability of this event in random d-regular graphs.

For every $\ell \in [O(\log n)]$, we consider random variables that represent all possible non-backtracking walks of length ℓ on a random d-regular multi-graph $G = ([n], E)$. Such a walk is represented by a start vertex $s \in [n]$ and a sequence of ℓ (non-backtracking) moves $\alpha \in D \stackrel{\text{def}}{=} [d] \times [d-1]^{\ell-1}$, where α_i is an index in the list of neighbors of the current vertex that exclude the previous vertex. (That is, the walk reprsented by the pair (s, α) has the form $(v_0{=}s, v_1, ..., v_\ell)$, where v_i is the α_i^{th} neighbor of v_i in a list that excludes v_{i-1}.) We stress that selecting a random d-regular multi-graph includes also ordering at random the d edges incident at each vertex.

We define 0-1 random variables $\zeta_{s,\alpha} = \zeta_{s,\alpha}(G)$ such that $\zeta_{s,\alpha} = 1$ if the walk on the random multi-graph G that corresponds to the pair (s, α) returns to s in its last step after visiting $\ell - 1$ different vertices. Note that the average of the $\zeta_{s,\alpha}(G)$'s equals the probability that a random non-backtracking ℓ-step walk on G closes a simple cycle; that is,

$$\mathtt{sc}_\ell(G) = \frac{1}{n \cdot |D|} \cdot \sum_{s \in [n], \alpha \in D} \zeta_{s,\alpha}(G). \tag{8}$$

Our aim is to prove that, for every $t > 1$, the sum (i.e., Eq. (8)) exceeds $O(t/n)$ with probability $O(1/t^2)$, where the probability space is uniform over the perfect matchings used to generate the d-regular multi-graphs G (in the configuration model).

Intuitively, it seems that $\mathrm{E}[\zeta_{s,\alpha}] \approx 1/n$ and that the $\zeta_{s,\alpha}$'s are almost pairwise independent. Indeed, we shall shortly show that this is essentially the case. But first, let us assume that $\mathrm{E}[\zeta_{s,\alpha}] \leq B/n$ and that the sum of the covariances of all (distinct) pairs is at most $C \cdot |D|^2$, where the covariance of $\zeta_{s,\alpha}$ and $\zeta_{s',\alpha'}$ equals $\mathrm{E}[\zeta_{s,\alpha}\zeta_{s',\alpha'}] - \mathrm{E}[\zeta_{s,\alpha}]^2$. Then, using Chebyshev's inequality we get

$$\Pr\left[\sum_{s \in [n], \alpha \in D} \zeta_{s,\alpha} > t \cdot |D|\right] \;<\; \frac{B \cdot |D| + C \cdot |D|^2}{(t - B)^2 \cdot |D|^2} \;=\; \frac{1}{(t-B)^2} \cdot \left(\frac{B}{d \cdot (d-1)^{\ell-1}} + C\right). \tag{9}$$

Combining Eq. (8) and Eq. (9), while assuming $t > 2B$, we get $\Pr_G[\mathtt{sc}_\ell(G) > t/n] \;=\; O(((d-1)^{-\ell} \cdot B + C)/t^2)$. Hence, $\Pr_G[\mathtt{sc}(G) > t/n] = O((B + C \cdot \log n)/t^2)$, because, for a suitable $L = O(\log n)$, we have

$$\begin{aligned}
\Pr_G[\mathtt{sc}(G) > t/n] &\leq \Pr_G\Big[\exists \ell \in [L+1, n] \text{ s.t. } \mathtt{sc}_\ell(G) > t/n\Big] + \Pr_G\Big[\exists \ell \in [L] \text{ s.t. } \mathtt{sc}_\ell(G) > t/n\Big] \\
&\leq O(1/n) + \sum_{\ell \in [L]} O(((d-1)^{-\ell} \cdot B + C)/t^2).
\end{aligned}$$

Thus, the claim follows once we establish $B = O(1)$ and $C = O(1/\log n)$, since then $\Pr[\mathtt{sc}(G) > t/n] = O(1/t^2)$.

Bounding B and C. We start by upper-bounding B; specifically, we prove that $p_\ell \stackrel{\text{def}}{=} \mathrm{E}[\zeta_{s,\alpha}] \leq 1/(n - \ell)$. This can be seen by fixing the first $\ell - 1$ steps in the walk that starts at s (and proceeds according to α), which means fixing $\ell - 1$ adjacencies in the random multi-graph G. Denoting the corresponding vertices by $v_1, ..., v_{\ell-1}$, where $v_0 = s$, we assume that they are distinct (or else $\zeta_{s,\alpha} = 0$), and observe that $n - \ell$ vertices were not visited at all and therefore each of them has d free incidences. In contrast, we used one incidence of each v_i, including $v_0 = s$, to get to v_{i+1}, which means that vertex s has only $d - 1$ free incidences. Hence, the probability (over the residual

choice of G) that s is chosen in the last step, as the α_ℓ^{th} neighbor of $v_{\ell-1}$, is at most $\frac{d-1}{d\cdot(n-\ell)}$, and it follows that $p_\ell \leq \frac{d-1}{d} \cdot \frac{1}{n-\ell}$.

Towards upper-bounding the covariances of all (distinct) variable pairs, we also establish a lower bound on p_ℓ (for $\ell \geq 3$).[22] This is done by noting that the first $\ell-1$ steps visit distinct vertices with probability $1-O(\ell/n)$, and by taking a closer look at the free incidences. The point is that we used one incidence of $v_0 = s$, two incidences of each v_i such that $i \in [\ell-2]$, and a single incidence of $v_{\ell-1}$. Hence, the number of free incidences is $(n-\ell)\cdot d + 2\cdot(d-1) + (\ell-2)\cdot(d-2) = n\cdot d - 2\cdot(\ell-1)$, and it follows that $p_\ell \geq (1-O(\ell/n)) \cdot \frac{d-1}{n\cdot d-2\cdot(\ell-1)} > \frac{d-1}{d} \cdot \frac{1-O(\ell/n)}{n}$.

We now show that, although the $\zeta_{s,\alpha}$'s are not pairwise independent, the sum of their covariances is at most $O(|D|^2 \cdot \ell/n)$; that is, we shall show that $C = O(\ell/n)$. (We comment that a weaker bound of $C = O(\ell)$ is easier to obtain, but it only implies a probability bound of $\text{poly}(\log n)/t^2$).)[23] Using these bounds (i.e., $B = O(1)$ and $C = O(\ell/n) = O(1/\log n)$), the claim follows (as detailed above).

Hence, it is left to prove the claim regarding the sum of all covariances; that is, the covariances of $\zeta_{s,\alpha}$ and $\zeta_{s',\alpha'}$ for all $s, s' \in [n]$ and $\alpha, \alpha' \in D$ such that $(s,\alpha) \neq (s',\alpha')$. (This is proved only for $\ell \geq 3$, since $\mathtt{sc}_1(G) = \mathtt{sc}_2(G) = 0$ holds for any graph G.) We consider two cases according to the relation between s and s'.

Case 1: $s = s'$**.** In this case $\alpha \neq \alpha'$ and a crude upper-bound on the covariance of $\zeta_{s,\alpha}$ and $\zeta_{s,\alpha'}$ will suffice, because there are relatively few such pairs. In particular, we upper-bound the covariance of $\zeta_{s,\alpha}$ and $\zeta_{s,\alpha'}$ by $\Pr[\zeta_{s,\alpha}=\zeta_{s,\alpha'}=1]$, and upper-bound the latter by the product of $\Pr[\zeta_{s,\alpha}=1]$ and the probability that both walks end at the same vertex (although they take different paths (i.e., $\alpha \neq \alpha'$)). Details follow.

Let γ denote the longest common suffix of α and α' (i.e., γ is longest sequence over $[d-1]$ such that $\alpha = \beta\gamma$ and $\alpha' = \beta'\gamma$), and note that $i \stackrel{\text{def}}{=} \ell - |\gamma| \geq 1$ and $\alpha_i \neq \alpha'_i$. Then, the probability that after i steps the two walks (corresponding to $\zeta_{s,\alpha}$ and $\zeta_{s,\alpha'}$) are at the same vertex is at most $1/(n-2\ell)$, since in the i^{th} step different incidences are used (regardless of whether or not the walks are at the same vertex after $i-1$ steps). This upper bound holds also when conditioning on $\zeta_{s,\alpha}=1$, since it is derived by fixing the first walk and considering the choice made in the i^{th} step of the second walk. On the other hand, if the two walks are at different vertices after i steps, then with probability at least $1-((\ell-i)/(n-2\ell)$ they will end-up in different vertices (since, as long as they are at different vertices, the next step depends on different incidences). Hence,

$$\Pr[\zeta_{s,\alpha}=\zeta_{s,\alpha'}=1] \;\leq\; \Pr[\zeta_{s,\alpha}=1] \cdot \left(\frac{1}{n-2\ell} + \frac{\ell-i}{n-2\ell}\right) \;=\; O(\ell/n^2).$$

Observing that there are less than $n \cdot |D|^2$ pairs in this case, it follows that the sum of their covariances is $n \cdot |D|^2 \cdot O(\ell/n^2) = O(|D|^2 \cdot \ell/n)$.

[22] Note that $p_1 = p_2 = 0$.

[23] Specifically, we (crudely) upper-bound the covariance of $\zeta_{s,\alpha}$ and $\zeta_{s,\alpha'}$ by $\Pr[\zeta_{s,\alpha}=\zeta_{s,\alpha'}=1]$, and upper-bound the latter by considering two cases. In case $s = s'$, we use a bound of $\Pr[\zeta_{s,\alpha}=1] = O(1/n)$, and the fact that there are less than $n \cdot |D|^2$ such pairs. In case $s \neq s'$, we upper-bound $\Pr[\zeta_{s,\alpha}=\zeta_{s,\alpha'}=1]$ by the product of $\Pr[\zeta_{s,\alpha}=1]$ and the probability that the second walk either hits the first one or ends at s', where each of the latter events occurs with probability $O(\ell/n)$. Hence, the sum of all covariances is upper-bounded by $(n\cdot|D|^2)\cdot O(1/n) + (n^2\cdot|D|^2)\cdot O(\ell/n^2) = O(\ell)\cdot|D|^2$.

Case 2: $s \neq s'$**.** In this case, we upper-bound the covariance of $\zeta_{s,\alpha}$ and $\zeta_{s',\alpha'}$ more carefully, while relating it to $\Pr[\zeta_{s,\alpha}=1]^2$. Fixing the first walk, we consider two sub-cases: In the first sub-case the second walk does not intersect the first walk, and the analysis of the second walk is very similar to the analysis of $\Pr[\zeta_{s,\alpha}=1]$, except that here at most 2ℓ vertices may have less than d free incidences. The complementary sub-case occurs with probability at most $O(\ell^2/n)$, and we can show that the residual walk (after the collision) will hit s' with probability $O(1/n)$. Combining both sub-cases, it follows that in this case $\Pr[\zeta_{s,\alpha}=\zeta_{s',\alpha'}=1]$ is at most $\Pr[\zeta_{s,\alpha}=1]^2 + O(\ell^2/n^3)$. Details follow.

Our focus is on upper-bounding $q \stackrel{\text{def}}{=} \Pr[\zeta_{s',\alpha'}=1 \,|\, \zeta_{s,\alpha}=1]$, and this is done by fixing an arbitrary walk that witnesses $\zeta_{s,\alpha}=1$. As stated above, the probability that the second walk intersects the first walk is $O(\ell^2/n)$, and in this sub-case we consider the intersection point v and argue that the probability that the residual walk that starts at v ends at s' is $O(1/n)$. Hence, the contribution of this sub-case to q is $O(\ell^2/n^2)$.

Our main concern is with the complementary sub-case in which the second walk does not intersect the first walk. In this sub-case the analysis of the second walk is very similar to the analysis of $\Pr[\zeta_{s',\alpha'}=1]$, except that here at most ℓ additional vertices (i.e., those on the first walk) may have less than d free incidences. Specifically, we may assume that the $\ell-1^{\text{st}}$ vertex in the walk is different from the first one (or else $\zeta_{s',\alpha'}=0$), and therefore the probability that the walk ends at s' is at most $\frac{d-1}{d}\cdot\frac{1}{n-2\ell}$, since at least $n-2\ell$ vertices have d free incidences. Hence, in this sub-case we get a probability bound of $p'_\ell \leq \frac{d-1}{d}\cdot\frac{1}{n-2\ell}$. Recalling that $p_\ell > \frac{d-1}{d}\cdot\frac{1-O(\ell/n)}{n}$, we get $p'_\ell \leq \frac{n}{n-O(\ell)}\cdot p_\ell$, which means that the contribution of this sub-case to q is at most $\frac{n}{n-O(\ell)}\cdot p_\ell = (1+O(\ell/n))\cdot p_\ell$.

Combining the contribution of both sub-cases, we upper-bound $\Pr[\zeta_{s,\alpha}=\zeta_{s,\alpha'}=1] = p_\ell \cdot q$ by

$$p_\ell \cdot \big(O(\ell^2/n^2) + (1+O(\ell/n))\cdot p_\ell\big) = (1+O(\ell/n))\cdot p_\ell^2,$$

since $p_\ell = \Omega(1/n) = \omega(\ell^2/n^2)$. Using $p_\ell = \Pr[\zeta_{s,\alpha}=1] = O(1/n)$, it follows that the corresponding covariance equals

$$\begin{aligned}\Pr[\zeta_{s,\alpha}=\zeta_{s,\alpha'}=1] - \Pr[\zeta_{s,\alpha}=1]^2 &\leq (1+O(\ell/n))\cdot p_\ell^2 - p_\ell^2\\ &= O(\ell/n)\cdot \Pr[\zeta_{s,\alpha}=1]^2\\ &= O(\ell/n^3).\end{aligned}$$

Hence, the sum of the covariances of this case is at most $n^2\cdot|D|^2\cdot O(\ell/n^3) = O(|D|^2\cdot\ell/n)$.

Combining both cases, we conclude that the sum of all covariances is $O(|D|^2\cdot\ell/n)$, and $C=O(\ell/n)$ follows. ■

Combining Lemmas 3.2.2 and 3.2.3, the theorem follows. ■

Digest. The proof of Theorem 3.2 reduces the problem of establishing lower bounds on the query complexity of local self-ordering procedures for a graph G to analyzing the probability of forming a simple cycle in a random walk on G (see Lemma 3.2.2). The definition of the latter probability referred to non-backtracking random walks (see Definition 3.2.1). Indeed, non-backtracking random walks, introduced in [1], are harder to analyze than standard random walks, but they fit our

application well. Nevertheless, an analogous definition of the probability of closing (i.e., containing) a simple cycle in a random walk is natural also in the context of other types of random walks (e.g., standard or lazy ones).[24] Turning back to Definition 3.2.1, we observe that the probability of forming a simple cycle decreases exponentially with the girth of the graph. That is –

Proposition 3.3 (sc decreases exponentially with the girth): *For every d-regular graph G of girth g it holds that $\mathtt{sc}(G) \leq (d-1)^{-\lfloor (g-1)/2 \rfloor}$.*

Using Lemma 3.2.2 it follows that any d-regular n-vertex graph that has logarithmic girth does not have local self-ordering procedures of query complexity $O(n^{o(1)})$.

Proof: Clearly, $\mathtt{sc}_\ell(G) = 0$ for every $\ell \in [g-1]$. For $\ell > \lfloor (g-1)/2 \rfloor$, we observe that the (distribution of the) vertex reached by an ℓ-step non-backtracking random walk (starting at a fixed vertex) is a convex combination of vertices reached by $\lfloor (g-1)/2 \rfloor$-step non-backtracking random walks (which start at various vertices), which represent the last $\lfloor (g-1)/2 \rfloor$ steps in the ℓ-step walk. (That is, letting $X_s(t)$ denote the vertex reached by an t-step non-backtracking random walk starting at vertex s, we observe that $X_s(\ell)$ is a convex combination of $X_v(\lfloor (g-1)/2 \rfloor)$'s.) On the other hand, the vertex reached by a $\lfloor (g-1)/2 \rfloor$-step non-backtracking random walk (starting at a fixed vertex) is uniformly distributed on a set of $M \stackrel{\text{def}}{=} d \cdot (d-1)^{\lfloor (g-1)/2 \rfloor - 1}$ vertices (i.e., $\Pr[X_s(\lfloor (g-1)/2 \rfloor) = v] \in \{1/M, 0\}$ for every s and v). Hence, an ℓ-step non-backtracking random walk returns to the start vertex with probability at most $1/M < (d-1)^{-\lfloor (g-1)/2 \rfloor}$, let alone doing so without revisiting any other vertex. ∎

Can Theorem 3.2 be strengthened? Recall that Theorem 3.2 establishes the existence of $O(1)$-regular n-vertex graphs G that are $\Omega(1)$-robustly self-ordered such that any local self-ordering procedure for G requires $\Omega(\sqrt{n})$ queries. We ask whether this lower bound can be increased.

Open Problem 3.4 (what is the query complexity of local self-ordering for the worst possible robustly self-ordered graph): *For every $d \in \mathbb{N}$ and $\gamma > 0$, determine the minimal function $Q : \mathbb{N} \to \mathbb{N}$ such that every n-vertex graph of maximal degree d that is γ-robustly self-ordered has a local self-ordering procedure of query complexity at most $Q(n)$. Specifically, two opposite extreme challenges are stated next.*

1. *Does $Q(n) = \Omega(n)$; that is, do there exist n-vertex bounded-degree graphs that are $\Omega(1)$-robustly self-ordered, but do not have local self-ordering procedures of query complexity $o(n)$.*
2. *Does $Q(n) = \widetilde{O}(\sqrt{n})$; that is, does every n-vertex bounded-degree graph that is $\Omega(1)$-robustly self-ordered have a local self-ordering procedure of query complexity $\widetilde{O}(\sqrt{n})$.*

We note that the robustness (self-ordering) condition is essential for Problem 3.4, since there exist n-vertex (asymmetric) graphs that require $\Omega(n)$ queries for local self-ordering: Consider an $(n-3)$-cycle that is augmented by an isolated vertex that is connected to one vertex and an 2-vertex path that is connected to its adjacent vertex (see Figure 7). This graph is asymmetric but any local self-ordering procedure for it requires $\Omega(n)$ queries; however, this graph is not $\omega(1/n)$-robustly self-ordered.

[24] Note that in the latter cases it makes little sense to consider the probability of forming a simple cycle, since these walks are likely to backtrack. On the other hand, one may consider the probability of containing a simple cycle (rather than forming one) also in the case of non-backtracking random walks. However, as noted above, Definition 3.2.1 fits our application best.

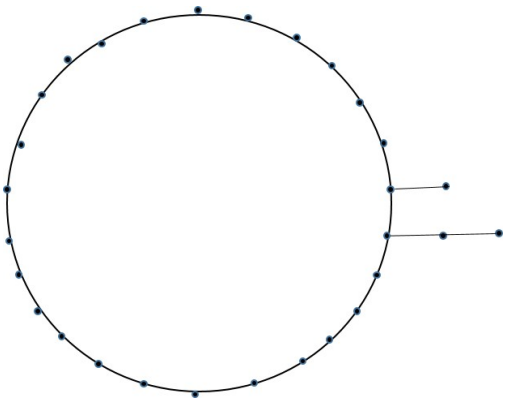

Figure 7: An augmented cycle requiring a linear number of queries for localization.

We also note that the lower bound asserted in Theorem 3.2 cannot be increased using the proof strategy used in our proof (i.e., using Lemma 3.2.2). Furthermore, it seems that such an improvement may not hold for random regular graphs, which (as shown in [6, Thm. 6.1]) are $\Omega(1)$-robustly self-ordered with probability $1 - o(1)$.

Conjecture 3.5 (random regular graphs have $\widetilde{O}(\sqrt{n})$-query self-ordering procedures): *For every $d \geq 3$, with probability $1-o(1)$, a random d-regular n-vertex graph has a local self-ordering procedure of query complexity $\widetilde{O}(\sqrt{n})$.*

We believe that Conjecture 3.5 can be proved using the following strategy. Let $\ell' = O(\log \log n)$ and $\ell = 0.5 \cdot \lceil \log_{d-1} n \rceil + \ell'$. For every vertex v in a random n-vertex d-regular graph, let χ_v be a random variable representing the number of collisions between pairs of vertices that are at distance exactly ℓ from v. Then, it seems that the expected value of χ_v is $O((d-1)^{2\ell'})$, and no value is assigned probability that exceeds $O((d-1)^{-\ell'})$. Furthermore, the values of different χ_v's seem sufficiently independent. Hence, it seems that a vertex v in a random d-regular graph G can be uniquely identified by the number of collisions in ℓ-step walks from the vertices that are at distance $\ell'/3$ from v; that is, letting S_v^G denote the set of vertices at distance $\ell'/3$ from v (in G) and m_w^G denote the number of collisions between pairs of vertices that are at distance ℓ from w (in G), it seems that the multi-set $\{m_w^G : w \in S_v^G\}$ uniquely identifies v (in G). In such a case, we get a local self-ordering procedure of query complexity $(d-1)^{\ell'/3} \cdot (d-1)^{\ell} = \widetilde{O}(n^{1/2})$.

Acknowledgements

I am grateful to Avi Wigderson for collaboration in early stages of this research.

References

[1] N. Alon, I. Benjamini, E. Lubetzky, and S. Sodin. Non-Backtracking Random Walks Mix Faster. *Communications in Contemporary Mathematics*, Vol. 9 (4), pages 585–603, 2007.

[2] D. Ellis. Lecture 13: The Expansion of Random Regular Graphs. Lecture notes, Algebraic Methods in Combinatorics, University of Cambridge, March 2011.
Available at `https://davidellis2.files.wordpress.com/2019/07/lecture13.pdf`

[3] O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

[4] O. Goldreich. On Testing Asymmetry in the Bounded Degree Graph Model *ECCC*, TR20-118, 2020.

[5] O. Goldreich and L. Trevisan. Three theorems regarding testing graph properties. *Random Structures and Algorithms*, Vol. 23 (1), pages 23–57, August 2003.

[6] O. Goldreich and A. Wigderson. Robustly Self-Ordered Graphs: Constructions and Applications to Property Testing. *ECCC*, TR20-149, 2020.

[7] O. Goldreich and A. Wigderson. Non-adaptive vs Adaptive Queries in the Dense Graph Testing Model. *ECCC*, TR20-160, 2020. (Third revision, Sept. 2021.)

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il