Transitive-Closure Spanners: A Survey[*]

Sofya Raskhodnikova[†]

Abstract

We survey results on transitive-closure spanners and their applications. Given a directed graph $G = (V, E)$ and an integer $k \geq 1$, a *k-transitive-closure-spanner (k-TC-spanner)* of G is a directed graph $H = (V, E_H)$ that has (1) the same transitive-closure as G and (2) diameter at most k. These spanners were studied implicitly in different areas of computer science, and properties of these spanners have been rediscovered over the span of 20 years. The common task implicitly tackled in these diverse applications can be abstracted as the problem of constructing sparse TC-spanners.

In this article, we survey combinatorial bounds on the size of sparsest TC-spanners, and algorithms and inapproximability results for the problem of computing the sparsest TC-spanner of a given directed graph. We also describe multiple applications of TC-spanners, including property testing, property reconstruction, key management in access control hierarchies and data structures.

1 Introduction

A *spanner* is a sparse backbone of a graph that approximately preserves distances between every pair of vertices. More precisely, a subgraph $H = (V, E_H)$ is a *k-spanner* of $G = (V, E)$ if for every pair of vertices $u, v \in V$, the shortest path distance $d_H(u, v)$ from u to v in H is at most $k \cdot d_G(u, v)$. Since they were introduced by Awerbuch [Awe85] and Peleg and Schäffer [PS89] in the context of distributed computing, spanners for undirected graphs have found numerous applications, including efficient routing [Cow01, CW04, PU89b, RTZ02, TZ01], simulating synchronized protocols in unsynchronized networks [PU89a], parallel and distributed algorithms for approximating shortest paths [Coh98, Coh00, Elk01], and algorithms for distance oracles [BS06, TZ05].

In the directed setting, three notions of spanners have been proposed: the direct generalization of the above definition [PS89], *roundtrip spanners* [CW04, RTZ02] and *transitive-closure spanners* [BGJ$^+$09a]. In this survey, we focus on the latter definition. It captures the notion that a spanner should have a small diameter but preserve the connectivity of the original graph.

Recall that the *transitive closure* of a graph $G = (V, E)$ is a graph (V, E_{TC}) where $(u, v) \in E_{TC}$ if and only if there is a directed path from u to v in G.

Definition 1.1 (TC-spanner [BGJ$^+$09a])**.** *Given a directed graph $G = (V, E)$ and an integer $k \geq 1$, a **k-transitive-closure-spanner (k-TC-spanner)** is a directed graph $H = (V, E_H)$ with the following properties:*

1. *E_H is a subset of the edges in the transitive closure of G.*
2. *For all vertices $u, v \in V$, if $d_G(u, v) < \infty$, then $d_H(u, v) \leq k$.*

*The edges from the transitive closure of G that are added to G to obtain a TC-spanner are called **shortcuts**, and the parameter k is called the **stretch**.*

Notice that a k-TC-spanner of G is a directed k-spanner of the transitive-closure of G. Nevertheless, TC-spanners are interesting in their own right due to the multiple TC-spanner-specific applications.

[*]Parts of this survey are adapted from [BGJ$^+$09a, BGJ$^+$09b, BGRW10, BGJ$^+$10].

[†]Pennsylvania State University, USA. Email: `sofya@cse.psu.edu`. Supported by National Science Foundation (NSF/CCF award 0729171 and NSF/CCF CAREER award 0845701).

Before TC-spanners were introduced in [BGJ$^+$09a], they were studied implicitly in access control, property testing, and data structures, and properties of these combinatorial objects have been discovered and rediscovered over the span of 20 years. In this work, we survey results on TC-spanners and their applications. We start by discussing a simple example of a TC-spanner of a directed line, which has been studied in different areas under different guises.

1.1 A Simple Example: TC-spanners of the Directed Line

To illustrate the definition of TC-spanners, we construct sparse TC-spanners for the directed line L_n, consisting of nodes[1] $[n]$ and edges $\{(i, i+1) : 1 \leq i \leq n-1\}$. See Section 3 for the discussion of work on this question. It is easy to see that the transitive closure of L_n has $\binom{n}{2} = \Theta(n^2)$ edges. Even a TC-spanner of stretch 2 can be much sparser:

Lemma 1.1 (2-TC-spanner of the line)**.** *For all $n \geq 3$, the directed line L_n has a 2-TC-spanner with at most $n \log n - n$ edges.*

Proof. Our 2-TC-spanner, H, is a graph with vertex set $[n]$. We construct the edge set of H recursively. First, define the middle node $v_{mid} = \lceil \frac{n}{2} \rceil$. We use this node as a *hub*: namely, we add edges (v, v_{mid}) for all nodes $v < v_{mid}$ and edges (v_{mid}, v) for all nodes $v > v_{mid}$. Then recurse on the two line segments resulting from removing v_{mid} from the current line. Proceed until each line segment contains exactly one node.

H is a 2-TC-spanner of the line L_n, since every pair of nodes $u, v \in [n]$ is connected by a path of length at most 2 via a hub. This happens in the stage of the recursion where u and v are separated into different line segments, or one of these two nodes is removed.

There are $t = \lfloor \log n \rfloor$ stages of the recursion, and in each stage $i \in [t]$ every non-hub node connects to the hub in its current line segment. Since 2^{i-1} nodes are hubs in the ith stage, this stage contributes $n - (2^i - 1)$ edges. Thus, the total number of edges in H is $n \cdot t - (2^{t+1} - t - 2) \leq n \log n - n$. The last inequality holds for $n \geq 3$. ∎

The same idea can be extended to construct a 3-TC-spanner of the line graph:

Lemma 1.2 (3-TC-spanner of the line)**.** *The directed line L_n has a 3-TC-spanner with $O(n \log \log n)$ edges.*

Proof. Again we construct the edge set of our 3-TC-spanner H recursively. For simplicity, assume that $\sqrt{n}$ is an integer. Designate $\sqrt{n}$ equidistant nodes as hubs: $\sqrt{n}, 2\sqrt{n}, \dots (\sqrt{n}-1)\sqrt{n}$. Connect each non-hub node to the nearest hub before it and the nearest hub after it. More precisely, for each non-hub node v, let v_ℓ be v rounded down to the nearest multiple of $\sqrt{n}$ and let $v_r = v_\ell + \sqrt{n}$. Add edge (v_ℓ, v) if $v_\ell \in [n]$ and edge (v, v_r) if $v_r \in [n]$. Also, add edges between all hubs, orienting them from the smaller to the larger. Finally, remove the hubs from the current line and recurse on the $\sqrt{n}$ resulting line segments. Proceed until each line segment contains exactly one node.

H is a 3-TC-spanner of the line L_n, since every pair of nodes $u, v \in [n]$ is connected by a path of length at most 3 via a pair of hubs. This happens in the stage of the recursion where u and v are separated into different line segments, or one of these two nodes is removed. For example, we add a path (u, u_r, v_ℓ, v) if u and v are not hubs.

Denote the number of edges in the spanner by $T(n)$. At the first stage of recursion, we add $\binom{\sqrt{n}}{2} \leq n$ edges to connect the hubs and at most 2 edges per non-hub node to connect non-hubs to hubs. Therefore, $T(n)$ satisfies the following recurrence:

$$T(n) \leq \begin{cases} 0 & \text{if } n \leq 1; \\ 3n + \sqrt{n} \cdot T(\sqrt{n}) & \text{if } n > 1. \end{cases}$$

The solution to this recurrence is $T(n) = O(n \log \log n)$. ∎

This construction generalizes to TC-spanners of arbitrary stretch k, giving k-TC-spanners of size $O(n \cdot \lambda_k(n))$, where $\lambda_k(n)$ are very slowly growing functions of n, called the k^{th}-row inverse Ackermann functions.

[1]We use $[n]$ to denote $\{1, 2, \dots, n\}$.

Definition 1.2 (Inverse Ackermann functions[2])**.** *For a given function $f : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ such that $f(n) < n$ for all $n > 1$, define the function $f^*(n) : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ as:*

$$f^*(n) = \min\{k \in \mathbb{Z}^{\geq 0} : f^{(k)}(n) < 2\}, \text{ where } f^{(k)} \text{ denotes } f \text{ composed with itself } k \text{ times.}$$

Then $\lambda_0(n) = n/2$ and $\lambda_1(n) = \sqrt{n}$. For $k \geq 2$, the k^{th}-row inverse Ackermann function *is $\lambda_k(n) = \lambda^*_{k-2}(n)$.*

Intuitively, $f^*(n)$ represents the number of times f can be applied before the answer drops below 2. If $f(n) = n/2$ this number is $\Theta(\log n)$, if $f(n) = \sqrt{n}$ it is $\Theta(\log\log n)$, and if $f(n) = \log n$ it is $\log^* n$. Therefore, $\lambda_2(n) = \Theta(\log n)$, $\lambda_3(n) = \Theta(\log\log n)$ and $\lambda_4(n) = \Theta(\log^* n)$. Also note that $\lambda_k(n)$ is a non-decreasing function of n for all $k \geq 0$.

Lemma 1.3 (k-TC-spanner of the line)**.** *The directed line L_n has a k-TC-spanner with $O(n \cdot \lambda_k(n))$ edges.*

Proof. Lemmas 1.1 and 1.2 imply Lemma 1.3 for $k = 2$ and 3. For larger k, we proceed as for $k = 3$, but select even more nodes as hubs, connect them using a $(k-2)$-TC-spanner, add edges from each node to the nearest hub before and the nearest hub after it, and recurse. Then each pair of nodes (u, v) will be connected by a path that jumps from u to the smallest hub greater than u, then follows a path of length at most $k-2$ in the $(k-2)$-TC-spanner on the hubs to reach the largest hub smaller than v, and uses one more edge to jump to v.

It remains to specify the number of hubs and to analyze the size of the spanner. Let $f(n) = \lambda_{k-2}(n)$. We recurse on the segments of size $f(n)$. Since we choose at most $\frac{n}{f(n)}$ hubs, we need at most $\frac{n}{f(n)} \cdot f\left(\frac{n}{f(n)}\right) \leq \frac{n}{f(n)} \cdot f(n) \leq n$ edges to connect them. As before, each non-hub connects to at most 2 hubs. Therefore, the number of edges in constructed spanner, $T(n)$, satisfies the following recursion:

$$T(n) \leq \begin{cases} 0 & \text{if } n \leq 1; \\ 3n + \frac{n}{f(n)} \cdot T(f(n)) & \text{if } n > 1. \end{cases}$$

The solution is $T(n) = 3n \cdot f^*(n)$. This follows from the fact that $f^*(f(n)) = f^*(n) - 1$ for $n > 1$. Thus, $T(n) = O(n \cdot \lambda^*_{k-2}(n)) = O(n \cdot \lambda_k(n))$. ∎

As discussed in Section 3.1, the bound in Lemma 1.3 is tight.

1.2 A Brief Overview

TC-spanners were defined in [BGJ+09a] as a common abstraction for several applications. Prior to that, Thorup [Tho92] considered a special case of TC-spanners of graphs G that have at most twice as many edges as G, and conjectured that for all directed graphs G on n nodes there are such k-TC-spanners with k polylogarithmic in n. He proved his conjecture for planar graphs [Tho95], but later Hesse [Hes03] gave a counterexample to Thorup's conjecture for general graphs. TC-spanners were also studied for directed trees: implicitly in [AS87, AFB05, Cha87, DGL+99, Yao82] and explicitly in [BTS94, Tho97]. The implicit results were interpreted as TC-spanner constructions in [BGJ+09a].

[BGJ+09a] presented several applications of TC-spanners: testing monotonicity of functions, key management in an access hierarchy and data structures for computing partial products in a semigroup. They also studied the computational problem of finding a sparsest k-TC-spanner for a given directed graph. They presented algorithms and inapproximability results for this problem. Finally, they gave sparse TC-spanner constructions for new graph families.

Later, [BGJ+09b, BGJ+10] studied TC-spanners for the directed hypercube and hypergrid and presented an application of TC-spanners to property reconstruction. Steiner TC-spanners (see Definition 3.1 were formally introduced in [BGRW10], but studied before that in the context of access control hierarchies by [ABFF09] and [SFM07]. Berman, Raskhodnikova and Ruan [BRR10] improved algorithms presented in [BGJ+09a]. Finally, Jha and Raskhodnikova [JR10] pointed out the application of TC-spanners to testing if a function is Lipschitz.

[2]We define these functions, following the spirit of the presentation by Seidel [Sei]. The prevalent definition (see, e.g., [AS87]) is complicated and yields asymptotically equivalent functions.

1.3 Organization of This Survey

We start by introducing notation and basic graph-theoretic background in Section 2. In Section 3, we describe structural results on TC-spanners, that is, combinatorial bounds on their size. Structural results for specific graph families are surveyed in Section 3.1, while general structural results, applicable to all graphs, appear in Section 3.2. In Section 4, we survey results on the computational problem of finding a sparsest TC-spanner of a given directed graph and the more general problem of finding directed spanners. We describe approximation algorithms and hardness results for these problems. Finally, Section 5 presents multiple applications of TC-spanners, including property testing, property reconstruction, key management in access control hierarchies and data structures for computing partial product in a semigroup.

2 Preliminaries and Notation

We write $x \preceq y$ to denote that vertex v is reachable from vertex u in graph G. When the graph is clear from the context, we omit G. The *transitive closure* of a directed graph $G = (V, E)$, denoted $TC(G)$, is the directed graph (V, E'), where $E' = \{(u, v) : u \preceq_G v\}$. Vertices u and v are *comparable* if either $(u, v) \in TC(G)$ (that is, u is *below* v or, equivalently, *smaller than* v) or $(v, u) \in TC(G)$ (that is, u is *above* v or, equivalently, *larger than* v). This terminology and notation is usually used for partially-ordered sets (posets), which are equivalent to directed acyclic graphs, but can be also applied to general directed graphs.

The *transitive reduction* of G, denoted $TR(G)$, is a digraph G' with the fewest edges for which $TC(G') = TC(G)$. As shown by Aho *et al.* [AGU72], $TR(G)$ can be computed efficiently via a greedy algorithm. The algorithm contracts each strongly connected component C to a vertex $v(C)$ to get a supergraph H, obtains a supergraph H' by greedily removing arcs in H that do not change its transitive closure, and finally uncontracts the $v(C)$, choosing a representative vertex of C to be incident to the edges incident to $v(C)$ in H'. For directed acyclic graphs $TR(G)$ is unique. We say G is *transitively reduced* if $TR(G) = G$.

A digraph G is *weakly connected* if replacing each directed edge in G with an undirected edge results in a connected undirected graph. A digraph is *strongly connected* if each vertex in the graph is reachable from every other vertex via a directed path. The graph of strongly connected components of a digraph G is the digraph obtained by contracting each strongly connected component into one vertex, while maintaining all the edges between these components.

3 Overview of Structural Results on TC-spanners

For a directed graph G, we denote the number of edges in G by $|G|$ and the size of the sparsest k-TC-spanner of G by $S_k(G)$. (The size refers to the number of edges.) To put the following results in proper context, observe that if G has n vertices, $S_k(G)$ is at most $O(n^2)$. Unlike in the undirected setting, where for every $k \geq 1$, all graphs on n vertices have a $(2k-1)$-spanners with $O(n^{1+1/k})$ edges [ADD$^+$93, Pel00, TZ05], TC-spanners (and hence, directed spanners) can have $\Omega(n^2)$ edges. An example of a graph with a 2-TC-spanner of size $\Omega(n^2)$ is a complete bipartite graph $K_{\frac{n}{2},\frac{n}{2}}$ with $n/2$ vertices in each part and all edges directed from the first part to the second. Therefore, most constructions surveyed below are for TC-spanners of specific graph families. Nevertheless, there are several general results, described in Sections 3.2.

3.1 TC-spanners of Specific Graph Families

TC-spanners of lines and trees. TC-spanners of the directed lines and directed trees were discovered under many different guises. They were studied implicitly in [AS87, AFB05, Cha87, DGL$^+$99, Yao82] and explicitly in [BTS94, Tho97]. Alon and Schieber [AS87] implicitly gave tight bounds on $S_k(L_n)$. They showed that the size of the sparsest k-TC-spanner of the directed line is $\Theta(n \cdot \lambda_k(n))$, where $\lambda_k(n)$ is the k^{th}-row inverse Ackermann function (see Definition 1.2 and Lemma 1.3). [AS87] also showed that the smallest k for which $S_k(L_n) = O(n)$ is $O(\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann function. (The inverse Ackermann function is defined by $\alpha(n) = \min\{k \in \mathbb{Z}^{\geq 0} : \lambda_{2k}(n) \leq 3\}$.) Note that the size of any TC-spanner of L_n is at least $n-1$, since all edges of the form $(i, i+1)$ must be present in a TC-spanner to ensure the same connectivity as in L_n. [AS87, Cha87, Tho97] proved that sparsest k-TC-spanners of rooted directed trees asymptotically have the same number of edges as k-TC-spanners of the line.

TC-spanners of planar graphs. Thorup [Tho92] considered a special case of TC-spanners of graphs G that have at most twice as many edges as G. In [Tho95], he proved that all directed planar graphs G on n nodes have such TC-spanners with stretch polylogarithmic in n.

TC-spanners of H-minor-free graphs. A graph H is a *minor* of G if H is a (not necessarily induced) subgraph of a graph obtained from G by a sequence of edge contractions. A graph family $\mathcal{F}$ is *minor-closed* if it contains every minor of every graph in $\mathcal{F}$. For a fixed graph H (e.g., K_5), a minor-closed family $\mathcal{F}$ is *H-minor-free* if $H \notin \mathcal{F}$. Examples of such families include planar graphs, bounded treewidth graphs, and bounded genus graphs, explicitly studied in applications in Section 5. Bhattacharyya *et al.* [BGJ+09a] gave an efficient construction of k-TC-spanners of H-minor-free graphs. The size of the spanners is $O(n \cdot \log n \cdot \lambda_k(n))$, where $\lambda_k(\cdot)$ is the k^{th}-row inverse Ackermann function. This result allowed [BGJ+09a] to drastically improve monotonicity testers of Fischer *et al.* [FLN+02]. The application to monotonicity testing is described in Section 5.

The construction in [BGJ+09a] uses divide-and-conquer approach. A natural first attempt would be to use separators of Lipton and Tarjan [LT79]. Recall that an s-separator for a graph G on n nodes is a set of s nodes whose removal disconnects G into connected components of size at most $2n/3$. Observe that the proofs of Lemmas 1.1–1.3 use this approach for the special case of the line graph. There, at every stage graph separators play a role of hubs. To come up with efficient constructions for a wider family of graphs, Bhattacharyya *et al.* use the *path separators* for undirected H-minor free graphs due to Abraham and Gavoille [AG06]. An s-path-separator[3] for a graph G on n nodes is a set of s paths whose removal disconnects G into connected components of size at most $2n/3$. For some graph families, path separators can be much smaller than ordinary separators. For example, planar graphs require ordinary separators of size $\Theta(\sqrt{n})$, but are 3-path separable [AG06]. For a simple case of a 2-dimensional $m \times m$ grid, a Lipton-Trajan separator has size of $\Omega(m)$, but it is enough to remove m nodes on one path, say, a horizontal line that cuts through the middle, to separate it.

Path separators of Abraham and Gavoile were constructed for undirected graphs. In general, a path separator for a directed graph may be the union of many directed paths. Here we only explain the construction for the simple case when a separator consists of a small number of directed paths, as is the case, for instance, for a 2-dimensional grid $[m] \times [m]$ where all edges are directed towards vertices with larger coordinates. (See [BGJ+09a] for the general treatment.)

If a separator consists of a constant numbers of directed paths, we can construct a k-TC-spanner of each path P in the separator as in the proof of Lemma 1.3. We also need to make sure that our TC-spanner contains short paths between all pairs of nodes that were using P to connect. To accomplish this, for each node u with a path to some node in separator path P, let u' be the smallest node in P comparable to u. As we are constructing a k-TC-spanner of P, at each stage of the recursion, we add an edge from u to a hub h whenever we add an edge (u', h). We deal symmetrically with each node v with a path from some node in P. Now, if there is a path from u to v via some vertex it P, there is a path of length at most k in the spanner we are constructing. This is because u and v are connected to the same hubs as u' and v' and, as demonstrated in the proof of Lemma 1.3, u' and v' are connected by a path of length at most k via the hubs. Now, we can safely remove the paths in the separator and recurse on the resulting components. To distinguish this recursion from the recursion in the construction of the TC-spanners of the paths, we call it an *outer* recursion. Observe that at each stage of outer recursion we are adding the same number of edges per node as in the construction for the line— namely, $O(\lambda_k(n))$ edges. This results in $O(n \cdot \lambda_k(n))$ edges per stage. Since there are $O(\log n)$ stages of outer recursion, the constructed k-TC-spanner has size $O(n \cdot \log n \cdot \lambda_k(n))$.

TC-spanner of hypergrids. The *directed hypergrid*, denoted $\mathcal{H}_{m,d}$, has vertex set $[m]^d$ and edge set $\{(x, y) : \exists \text{ unique } i \in [d] \text{ such that } y_i - x_i = 1 \text{ and for } j \neq i, y_j = x_j\}$. For the special case $m = 2$, $\mathcal{H}_{2,d}$ is called a *hypercube* and is also denoted by $\mathcal{H}_2$. 2-TC-spanners of hypergrids are especially relevant for applications in property testing and property reconstruction. TC-spanners of hypergrids of general stretch k are used in the application to key management in an access hierarchy. The following results on TC-spanners of hypergrids are from [BGJ+09b, BGJ+10].

[3]For a graph G to be s-path-separable one needs to be able to disconnect the graph by removing nodes on at most s paths from any minimum spanning tree of G. To keep our high-level overview simple, we do not get into details.

As a comparison point for bounds below, note that the obvious bounds on $S_2(\mathcal{H}_d)$ are the number of edges in the d-dimensional hypercube, $2^{d-1}d$, and the number of edges in the transitive closure of $\mathcal{H}_d$, which is $3^d - 2^d$. (An edge in the transitive closure of $\mathcal{H}_d$ has 3 possibilities for each coordinate: both endpoints are 0, both endpoints are 1, or the first endpoint is 0 and the second is 1. This includes self-loops, so we subtract the number of vertices in $\mathcal{H}_d$ to get the desired quantity.) Thus, $2^{d-1}d \le S_2(\mathcal{H}_d) \le 3^d - 2^d$. Similarly, the straightforward bounds on the number of edges in a 2-TC-spanner of $\mathcal{H}_{m,d}$ in terms of the number of edges in the directed grid and in its transitive closure are $dm^{d-1}(m-1)$ and $\left(\frac{m^2+m}{2}\right)^d - m^d$, respectively.

The following theorem gives upper and lower bounds on $S_2(\mathcal{H}_{m,d})$:

Theorem 3.1 (Hypergrid [BGJ$^+$09b, BGJ$^+$10])**.** *Let $S_2(\mathcal{H}_{m,d})$ denote the number of edges in the sparsest 2-TC-spanner of $\mathcal{H}_{m,d}$. Then*[4] *for $m \ge 3$,*

$$S_2(\mathcal{H}_{m,d}) = \Omega\left(\frac{m^d \log^d m}{(2d \log\log m)^{d-1}}\right) \quad \textit{and} \quad \le m^d \log^d m.$$

The upper bound in Theorem 3.1 follows from a general construction of k-TC-spanners for graph products for arbitrary $k \ge 2$. The lower bound is proved by a reduction of the 2-TC-spanner construction for $[m]^d$ to that for the $2 \times [m]^{d-1}$ grid and then directly analyzing the number of edges required for a 2-TC-spanner of $2 \times [m]^{d-1}$. The authors show a tradeoff between the number of edges in the 2-TC-spanner of the $2 \times [m]^{d-1}$ grid that stay within the hyperplanes $\{1\} \times [m]^{d-1}$ and $\{2\} \times [m]^{d-1}$ versus the number of edges that cross from one hyperplane to the other. The proof proceeds in multiple stages. Assuming an upper bound on the number of edges staying within the hyperplanes, each stage is shown to separately contribute a substantial number of edges crossing between the hyperplanes.

Theorem 3.1 is most useful when m is large. When m is small, it is superseded by another set of bounds on $S_2(\mathcal{H}_{m,d})$, given in [BGJ$^+$09b, BGJ$^+$10], which are optimal up to a factor of d^{2m}. These bounds are formulated in terms of a complicated combinatorial expression, but value of this expression can be estimated numerically. Specifically, $S_2(\mathcal{H}_{m,d}) = 2^{c_m d}\,\mathrm{poly}(d)$, where $c_2 \approx 1.1620$, $c_3 \approx 2.03$, $c_4 \approx 2.82$ and $c_5 \approx 3.24$, each significantly smaller than the exponents corresponding to the transitive closure sizes for the different m. More precisely, for the hypercube, $S_2(\mathcal{H}_d) = O(d^3 2^{c_2 d})$ and $\Omega(2^{c_2 d})$. The upper bound $S_2(\mathcal{H}_d)$ is proved via a randomized construction of a 2-TC-spanner of the directed hypercube. Curiously, even though the upper and lower bounds above differ by a factor of $O(d^3)$, it is known that the randomized construction yields a 2-TC-spanner of $\mathcal{H}_d$ of size within $O(d^2)$ of the optimal.

Steiner TC-spanners of d-dimensional posets. In some applications (in particular, to access control hierarchies [ABF06, AFB05, SFM07, ABFF09]), the shortcuts can use *Steiner* vertices, that is, vertices not in the original graph G. The resulting spanner is called a *Steiner TC-spanner.*

Definition 3.1 (Steiner TC-spanner [BGRW10])**.** *Given a directed graph $G = (V, E)$ and an integer $k \ge 1$, a **Steiner k-transitive-closure-spanner (Steiner k-TC-spanner)** of G is a directed graph $H = (V_H, E_H)$ satisfying:*

1. *$V \subseteq V_H$;*
2. *for all vertices $u, v \in V$, if $d_G(u, v) < \infty$ then $d_H(u, v) \le k$ and if $d_G(u, v) = \infty$ then $d_H(u, v) = \infty$.*

*Vertices in $V_H \backslash V$ are called **Steiner vertices**.*

For some graphs, Steiner TC-spanners can be significantly sparser than ordinary TC-spanners. Before, our example of a graph with a 2-TC-spanner of size $\Omega(n^2)$ was a complete bipartite graph $K_{\frac{n}{2},\frac{n}{2}}$ with $n/2$ vertices in each part and all edges directed from the first part to the second. This graph has a Steiner 2-TC-spanner of size n: it is enough to add one Steiner vertex v, edges to v from all nodes in the left part, and edges from v to all nodes in the right part. Thus, for $K_{\frac{n}{2},\frac{n}{2}}$ there is a linear gap between the size of the sparsest Steiner 2-TC-spanner and the size of an ordinary 2-TC-spanner.

However, Bhattacharyya *et al.* [BGRW10] show that for directed hypergrids, Steiner vertices do not help: sparsest Steiner TC-spanners have the same size as TC-spanners with no Steiner vertices.

[4]Logarithms are always to base 2 unless otherwise indicated.

stretch k	Upper Bounds on $S_k(G)$	Lower Bounds on $S_k(G)$	Reference
$k = 2$	$O(n \log^d n)$	$\Omega\left(n\left(\frac{\log n}{\log\log n}\right)^d\right)$ for constant d	[BGRW10]
constant $k \geq 3$		$n \log^{\Omega(d)} n$ for constant d	[BGRW10]
$k = 2t + 1$ for $t \in [d]$	$O(3^{d-t}t \cdot n \log^{d-1} n \log\log n)$		[SFM07]

Table 1: The size of the sparsest Steiner k-TC-spanners of d-dimensional posets on n vertices for $d \geq 2$

Lemma 3.2 ([BGRW10])**.** *If $\mathcal{H}_{m,d}$ has a Steiner k-TC spanner H, it also has a k-TC spanner of size $|H|$.*

Proof. We show how to replace one Steiner vertex in H with a grid vertex while keeping the same number of edges in the Steiner k-TC-spanner. This step can be repeated to remove all Steiner vertices.

Since $\mathcal{H}_{m,d}$ is acyclic, a cycle in H can contain at most one non-Steiner vertex, and therefore H will still remain a Steiner k-TC-spanner of $\mathcal{H}_{m,d}$ if this cycle is contracted to one vertex. Thus, we can assume without loss of generality that H is acyclic.

Let s be a Steiner vertex in H which does not have any other Steiner vertices below it. Let s' be the smallest vertex in $\mathcal{H}_{m,d}$ which is above all vertices v in $\mathcal{H}_{m,d}$ satisfying $v \preceq s$. (If there are no such v then s' is the grid vertex with all coordinates equal to 1.) Observe that s' always exists and is unique. Moreover, every vertex in $\mathcal{H}_{m,d}$ that is above all such v is also above s'. By definition, s' is above all vertices in $\mathcal{H}_{m,d}$ which have a path to s. It is also below all vertices in $\mathcal{H}_{m,d}$ which are reachable from s. We replace all edges in H that have s as an endpoint with the corresponding edges with s' as an endpoint, and remove s from H. Every pair of vertices that was connected via a path of length at most k is still connected via the same path, with s replaced by s' if necessary. No new pair (u, v) of vertices in $\mathcal{H}_{m,d}$ got connected via s' since if u was below s and v was above s then $u \preceq s \preceq v$. The number of edges in H has not increased. ∎

Atallah *et al.* [ABFF09], De Santis *et al.* [SFM07] and Bhattacharyya *et al.* [BGRW10] study Steiner TC-spanners of directed *acyclic* graphs or, equivalently, partially ordered sets. Motivated by the application to access control hierarchies (described in Section 5), they focus on the relationship between the dimension of a poset and the size of its sparsest Steiner TC-spanner.

Definition 3.2 (Poset dimension)**.** *The* dimension *of a poset G is the smallest d such that G can be embedded into a d-dimensional hypergrid $\mathcal{H}_{m,d}$ via an order-preserving embedding. A mapping from a poset G to a poset G' is called an* order-preserving embedding *if it respects the partial order, that is, all $x, y \in G$ are mapped to $x', y' \in G'$ such that (i) $x \preceq_G y$ iff $x' \preceq_{G'} y'$, and (ii) x is incomparable to y iff x' is incomparable to y'.*

Each poset has a dimension. In particular, each poset with n elements can be embedded into a hypergrid $\mathcal{H}_{n,d}$, so that for all $i \in [d]$, the ith coordinates of images of all points are distinct.

Poset dimension is a fundamental and well-studied parameter in poset theory. For instance, Dilworth's famous chain partitioning theorem was originally intended as a lemma for proving a theorem about the dimension of distributive lattices [Dil50]. A survey on poset dimension can be found in Trotter's monograph [Tro92]. One important result for the discussion below, proved by Dushnik and Miller [DM40], is that that for all m, the hypergrid $\mathcal{H}_{m,d}$ has dimension exactly d. Atallah *et al.* argue that many access control hierarchies are low-dimensional posets that come equipped with an embedding demonstrating low dimensionality.

Observe that the only poset of dimension 1 is the directed line. Tight bounds for the size of (Steiner) TC-spanners of directed lines were discussed in the beginning of Section 3. Table 1 summaries the best bounds for $d \geq 2$. The upper bounds hold for all posets of dimension d. The TC-spanners in the upper bounds can be constructed efficiently, given an explicit embedding of the poset into a d-dimensional grid. (Finding such an embedding is NP-hard [Yan82].) Paths of length at most k between all pairs of vertices in the resulting k-TC-spanners can be found efficiently. (This is important for the application to access control hierarchies).

The lower bounds mean that there exists a poset of dimension d for which every Steiner k-TC-spanner has the specified number of edges. The lower bound for Steiner 2-TC-spanners holds for the hypergrid $\mathcal{H}_{m,d}$

and follows from Theorem 3.1 and Lemma 3.2. The lower bound on the size of a Steiner k-TC-spanner for $k \geq 3$ is proved by the probabilistic method.

Note that the Steiner vertices used in constructions for d-dimensional posets are necessary to obtain sparse TC-spanners. Recall our example of a bipartite graph $K_{\frac{n}{2},\frac{n}{2}}$ for which every 2-TC-spanners required $\Omega(n^2)$ edges. $K_{\frac{n}{2},\frac{n}{2}}$ is a poset of dimension 2, and thus, by the upper bound in [BGRW10], has a Steiner 2-TC spanner of size $O(n \log^2 n)$. To see that $K_{\frac{n}{2},\frac{n}{2}}$ is embeddable into a $[n] \times [n]$ grid, map each of the $n/2$ left vertices of $K_{\frac{n}{2},\frac{n}{2}}$ to a distinct grid vertex in the set of incomparable vertices $\{(i, n/2+1-i) : i \in [n/2]\}$, and similarly map each right vertex to a distinct vertex in the set $\{(n+1-i, i+n/2) : i \in [n/2]\}$. It is easy to see that this is a proper embedding. As we mentioned before, for this graph there is an even better Steiner 2-TC spanner with $O(n)$ edges.

3.2 General TC-spanner Constructions

Graphs that require a large number of shortcuts. We have seen in the beginning of Section 3 that, in general, TC-spanners can be large. However, in the example we looked at, the graph itself was large and, in fact, we did not have to add any shortcuts to construct a 2-TC-spanner of that graph. Can one always construct a TC-spanner by adding a small number of edges to the original graph? Thorup [Tho92] conjectured that all directed graphs G on n nodes have TC-spanners with stretch polylogarithmic in n and size at most $2|G|$. As mentioned before, he proved his conjecture for planar graphs [Tho95], but later Hesse [Hes03] gave a counterexample to Thorup's conjecture for general graphs. He constructed a family of graphs with $n^{1+\epsilon}$ edges for which all n^{ϵ}-TC-spanners require $\Omega(n^{2-\epsilon})$ edges, for small $\epsilon > 0$.

2-TC-spanners from $2k$-TC-spanners. Berman, Raskhodnikova and Ruan [BRR10] show how to (efficiently) obtain a 2-TC-spanner of a graph G with diameter at most $2k$ by adding $O(n^{1-1/k} \cdot |G|)$ shortcuts. They prove that this relationship is nearly tight in the following sense: there are graphs with $2k$-TC-spanners of size $n^{1+(1-\epsilon)/k}$ and no 2-TC-spanners of size less than $n^{2-\epsilon}$, for small $\epsilon > 0$. These graphs are obtained by adjusting the parameters in the Hesse's construction described above.

For the transformation from 3-TC-spanners to 2-TC-spanners, the number of additional edges is asymptotically optimal, as evidenced by the 4-layered graph with m^2 nodes in layers 1 and 4 and m nodes in layers 2 and 3, where the edges are directed from smaller to larger layers and are formed as follows. There is a complete bipartite graph between layers 2 and 3. Each node in layer 2 is connected to m nodes in layer 1, and each node in layer 1 has outdegree 1. The edges between layers 3 and 4 are constructed in the same manner. The resulting graph has $3m^2$ edges and is a 3-TC-spanner. To construct a 2-TC-spanner of this graph, we need to connect m^4 pairs of vertices in layers 1 and 4 via paths of length at most 2. Each shortcut edge can be used by at most m such pairs. Therefore, at least m^3 shortcuts are required. Setting $n = 2m^2 + 2m$, we obtain a graph with a 3-TC-spanner of size $O(n)$, for which every 2-TC-spanner requires $\Omega(n^{3/2})$ edges.

TC-spanners with large stretch. Improving on the first result in this vain in [BGJ+09a], Berman, Raskhodnikova and Ruan show that one can obtain a k-TC-spanner of any graph by adding $O(n^2/k^2)$ shortcut edges. This construction is efficient.

TC-spanners of Graphs with Cycles. Here we give a reduction from constructing TC-spanners of general directed graphs to constructing TC-spanners of directed acyclic graphs (DAGs).

Lemma 3.3. *Let G be a directed graph on n vertices, and G' be the graph of strongly connected components of G. Then $S_{k+2}(G) \leq S_k(G')$ +2n. Moreover, given a k-TC-spanner H' of G', one can efficiently construct a $(k+2)$-TC-spanner H of G with at most $|H'| + 2n$ edges.*

Proof. For each strongly connected component C of G, pick an arbitrary vertex v_C and call it a representative of C. To construct a $(k+2)$-TC-spanner H of G from a k-TC-spanner H' of G', first connect representatives of connected components to mimic the structure of H': namely, add an edge (v_{C_1}, v_{C_2}) to H for every edge (C_1, C_2) in H'. Second, for every vertex u in the component C, where $u \neq v_C$, add edges (u, v_C) and (v_C, u) to H.

Problem	Setting of k	Approximability	Previous Work
DIRECTED k-SPANNER (and k-TC-SPANNER)	$k = 2$	$O(\log n)$ [EP01]	
	$k = 3$	$O(\sqrt{n} \cdot \log n)$ [BRR10]	[EP00, BGJ$^+$09a]
	$k \geq 3$	$O(kn^{1-1/\lceil k/2 \rceil} \cdot \log n)$ [BRR10]	[BGJ$^+$09a]
k-TC-SPANNER only	$k \geq 3$	$O(n^{1-1/\lceil k/2 \rceil} \cdot \log n)$ [BRR10]	[BGJ$^+$09a]
	$k = \Omega\left(\frac{\log n}{\log \log n}\right)$	$O(n/k^2)$ [BRR10]	[BGJ$^+$09a]

Table 2: Summary of Algorithmic Results on DIRECTED k-SPANNER and k-TC-SPANNER

The resulting H has the same number of edges as H' plus at most 2 edges per vertex, added to connect each vertex to the representative of its strongly connected component. That is, $|H| \leq |H'| + 2n$. To see that H is a $(k+2)$-TC-spanner of G, consider vertices u_1, u_2 in G, where u_2 is reachable from u_1. Let v_1 and v_2 be the representatives of the components of u_1 and u_2, respectively. Since H' is a k-TC-spanner of G', there is a path of length at most k from the component of u_1 to the component of u_2 in H'. Therefore, H contains a path of length at most k from v_1 to v_2. Since H also contains edges (u_1, v_1) and (v_2, u_2), it contains a path of length at most $k + 2$ from u_1 to u_2. ∎

4 Overview of Computational Results on Directed Spanners

The computational problem of finding the size of the sparsest k-TC-spanner of a given graph, called k-TC-SPANNER, was first considered in [BGJ$^+$09a]. k-TC-SPANNER is a special case of a well-studied problem, called DIRECTED k-SPANNER, of finding the size of the sparsest k-spanner of a given (not necessarily transitively closed) directed graph [EP01, EP00, BGJ$^+$09a, BRR10]. In this section, we survey approximation algorithms and inapproximability results for these two problems. All known algorithms on DIRECTED k-SPANNER also apply to two other variants, CLIENT/SERVER DIRECTED k-SPANNER and k-DIAMETER SPANNING SUBGRAPH, defined by Elkin and Peleg [EP01].

Algorithms for DIRECTED k-SPANNER **and** k-TC-SPANNER**.** All algorithms for DIRECTED k-SPANNER immediately yield algorithms for k-TC-SPANNER with the same approximation ratio because k-TC-SPANNER on input graph G is equivalent to DIRECTED k-SPANNER on input TC(G). Table 2 summarizes the best known approximation algorithms for these problems for different stretch k. Elkin and Peleg [EP01] gave an $O(\log n)$-approximation algorithm for DIRECTED 2-SPANNER. For $k = 3$, approximation algorithms were proposed in [EP00, BGJ$^+$09a, BRR10] with the best ratio, $O(\sqrt{n} \cdot \log n)$, due to [BRR10]. In general, for $k > 3$, [BRR10] prove an approximation ratio $O(kn^{1-1/\lceil k/2 \rceil} \cdot \log n)$, improving the first polynomial time algorithm for this problem, given in [BGJ$^+$09a]. For the special case of k-TC-SPANNER, [BRR10] give a slightly better ration of $O(n^{1-1/\lceil k/2 \rceil} \cdot \log n)$. For large k, the best approximation ratio is $O(n/k^2)$, due to [BRR10], again an improvement over the first approximation algorithm for this range of parameters, proposed in [BGJ$^+$09a].

We briefly describe the two TC-spanner-specific approximation algorithms in [BRR10]. They are based on the structural results mentioned in Section 3.2. The first algorithm runs the $O(\log n)$-approximation algorithm from [EP01] for DIRECTED 2-SPANNER on the transitive closure of the input graph G. Since $S_2(G) \leq S_k(G) + O(n^{1-1/\lceil k/2 \rceil} \cdot |TR(G)|)$, and the algorithm is guaranteed to output a 2-TC-spanner of size $O(\log n \cdot S_2(G)) = O(\log n \cdot S_k(G) + n^{1-1/\lceil k/2 \rceil} \log n \cdot |TR(G)|)$, the result is an $O(n^{1-1/\lceil k/2 \rceil} \log n)$-approximation. (Recall that $|TR(G)|$ is a lower bound on the size of a TC-spanner.) The algorithm for large k is based on an efficient procedure that obtains a k-TC-spanner by adding $O(n^2/k^2)$ shortcut edges. It can be run on each weakly connected component separately. For a weakly connected component with n nodes, the size of a k-TC-spanner is at least $n - 1$, so the resulting graph is a $O(n/k^2)$-approximation.

It is important to note that the algorithm for large k has a better approximation ratio than the corresponding hardness result for DIRECTED k-SPANNER, demonstrating that k-TC-SPANNER is a strictly easier problem for this range of parameters.

Setting of k	Inapproximability	Assumption	Notes
$k = 2$	$\Omega(\log n)$	P$\neq$ NP	Matches the upper bound
constant $k \geq 3$	$\Omega(2^{\log^{1-\epsilon} n})\ \ \forall \epsilon \in (0,1)$	NP$\subseteq$DTIME($n^{poly \log n}$)	Improvement implies breakthrough
$k \leq n^{1-\delta}\ \ \forall \delta \in (0,1)$	$\Omega(1+\delta)$	P$\neq$ NP.	

Table 3: Summary of Hardness Results on k-TC-Spanner in [BGJ+09a]

Inapproximability of Directed **k-**Spanner**.** For completeness, we state the inapproximability results for Directed k-Spanner, even though they do not imply anything for k-TC-Spanner. Kortsarz [Kor01] showed that the $O(\log n)$ approximation ratio for Directed 2-Spanner cannot be improved unless P=NP. For all $\delta, \epsilon \in (0,1)$ and $3 \leq k \leq n^{1-\delta}$, it is impossible to approximate Directed k-Spanner within a factor of $2^{\log^{1-\epsilon} n}$ in polynomial time, assuming NP$\not\subseteq$DTIME($n^{poly \log n}$) [EP00, EP07]. Thus, according to Arora and Lund's classification [Hoc97] of NP-hard problems, Directed k-Spanner is in class III, for $3 \leq k = O(n^{1-\delta})$. Moreover, [EP07] showed that proving that Directed k-Spanner is in class IV, that is, inapproximable within n^{δ} for some $\delta \in (0,1)$, would resolve a long standing open question in complexity theory, and cause classes III and IV to collapse into a single class.

Inapproximability of k-TC-Spanner**.** Table 3 summarizes inapproximability results for k-TC-Spanner, all due to [BGJ+09a], for different values of k. For constant k, the hardness results are the same as for Directed k-Spanner, even though the reductions are much more technically involved. Observe that a stronger inapproximability result for $k > 2$ would imply the same inaproximability for Directed-k-Spanner and, as shown in [EP07], collapse classes III and IV in Arora and Lund's classification. For nonconstant $k \leq n^{1-\gamma}$ for all $\gamma > 0$, we know that the problem is NP-hard, but not much beyond that. This contrasts sharply with the known hardness of Directed k-Spanner, but, as mentioned previously, k-TC-Spanner is known to be strictly easier for some (but not all) k in that range.

The $2^{\log^{1-\epsilon} n}$-inapproximability of k-TC-Spanner for constant $k \geq 3$ in [BGJ+09a] matches the inapproximability of Directed k-Spanner for the same stretch in [EP07]. As is the case for Directed k-Spanner, the reduction is from a problem called MIN-REP, which has the same inapproximability as Symmetric Label Cover. However, as illustrated in [BGJ+09a], all known hard instances for Directed k-Spanner cannot imply anything better than $\Omega(1)$-hardness for k-TC-Spanner. Intuitively, inapproximability of k-TC-Spanner is harder to prove than inapproximability of Directed k-Spanner because an instance of k-TC-Spanner must be transitively-closed, and thus, have more "shortcut" routes between pairs of vertices. The construction of hard instances of k-TC-Spanner in [BGJ+09a] uses so-called *generalized butterfly* and *broom* graphs. The paths in the generalized butterfly are well-structured, making it possible to analyze many different routes in the transitive closure of a hard instance.

5 Applications of TC-spanners

We describe four types of applications that use sparse TC-spanners: property testing, property reconstruction, key management in an access hierarchy and data structures for computing partial products in a semigroup. For property testing, we give two applications: to testing monotonicity of functions and to testing if a function is Lipschitz. All these applications, with the exception of testing Lipschitz functions and property reconstruction, were pointed out and described in [BGJ+09a]. The application to Lipschitz functions is from [JR10]. The application to property reconstruction is from [BGJ+10].

5.1 Applications to Property Testing

We start by describing the application to testing monotonicity of functions. We also point out the limitations of TC-spanner techniques and related open questions in the area.

Monotonicity testing. Monotonicity of functions [EKK$^+$00, GGL$^+$00, DGL$^+$99, BRW05, Fis04, FLN$^+$02, HK04, AC06, SMCB10] is one of the most studied properties in property testing [GGR98, RS96]. Fischer *et al.* [FLN$^+$02] prove that testing monotonicity is equivalent to several other testing problems. Let V_n be a poset of n elements and $G_n = (V_n, E)$ be the relation graph, i.e., the Hasse diagram, for V_n. A function $f : V_n \to \mathbb{R}$ is called *monotone* if $f(x) \le f(y)$ for all $(x, y) \in E$. We say f is ϵ-far from monotone if f has to be changed on $\ge \epsilon$ fraction of the domain to become monotone, that is, $\min_{\text{monotone } g} |\{x : f(x) \ne g(x)\}| \ge \epsilon n$. A monotonicity tester on G_n is an algorithm that, given an oracle for a function $f : V_n \to \mathbb{R}$, passes if f is monotone but fails with probability $\ge \frac{2}{3}$ if f is *ϵ-far* from monotone. For instance, if G_n is a directed line L_n, the tester needs to determine whether the input sequence, specified by f, is sorted or ϵ-far from sorted. If G_n is a 2-dimensional grid $\mathcal{H}_{m,2}$, the goal is to determine whether the input matrix has non-decreasing rows and columns. The optimal monotonicity tester for the directed line L_n, proposed by Dodis *et al.* [DGL$^+$99], is based on the sparsest 2-TC-spanner for that graph. Implicit in the proof of Proposition 9 in [DGL$^+$99] is a lemma relating the complexity of a monotonicity tester for L_n to the size of a 2-TC-spanner of L_n. Bhattacharyya *et al.* [BGJ$^+$09a] generalized this lemma by observing that a sparse 2-TC-spanner for any partial order graph G_n implies an efficient monotonicity tester on G_n.

Lemma 5.1 ([BGJ$^+$09a])**.** *If a directed acyclic graph G_n has a 2-TC-spanner with $s(n)$ edges, then there exists a monotonicity tester on G_n that runs in time $O\left(\frac{s(n)}{\epsilon n}\right)$.*

Proof. The tester selects $\frac{8s(n)}{\epsilon n}$ edges of the 2-TC-spanner H uniformly at random. It queries function f on the endpoints of all the selected edges and rejects if some selected edge (x, y) is *violated* by f, that is, $f(x) > f(y)$.

If the function f is monotone on G_n, the algorithm always accepts. The crux of the proof is to show that functions that are ϵ-far from monotone are rejected with probability at least $\frac{2}{3}$. Let $f : V_n \to \mathbb{R}$ be a function that is ϵ-far from monotone. It is enough to demonstrate that f violates at least $\frac{\epsilon n}{4}$ edges in H. Then each selected edge is violated with probability $\frac{\epsilon n}{4s(n)}$, and the lemma follows by elementary probability theory.

Denote the transitive closure of G by $TC(G)$. We say a vertex $x \in V_n$ is assigned a *bad* label by f if x has an incident violated edge in $TC(G_n)$; otherwise, x has a *good* label. Let V' be a set of vertices with good labels. Observe that f is monotone on the induced subgraph $G' = (V', E')$ of $TC(G)$. This implies ([FLN$^+$02], Lemma 1) that f can be changed into a monotone function by modifying it on at most $|V_n - V'|$ vertices. Since f is ϵ-far from monotone, it shows that there are at least ϵn vertices with bad labels.

Every function that is ϵ-far from monotone has a matching M of at least $\frac{\epsilon n}{2}$ violated edges in $TC(G)$ [DGL$^+$99]. We will establish a map from the set of edges in M to the set of violated edges in H, so that each violated edge in H is the image of at most 2 edges in M. For each edge (x, y) in the matching, consider the corresponding path from x to y of length at most 2 in the 2-TC-spanner H. If the path is of length 1, (x, y) is the violated edge in H corresponding to the matching edge (x, y). Otherwise, let (x, z, y) be a path of length 2 in H. At least one of the edges (x, z) and (z, y) is violated, and we map (x, y) to that edge. Since M is a matching, at most 2 edges in M can be mapped to one violated edge in $TC(G)$. Thus, the 2-TC-spanner H has $\ge \frac{\epsilon n}{4}$ violated edges, as required. ■

The fact that H is a 2-TC-spanner is crucial for the proof. If it was a k-TC-spanner for $k > 2$, the path of length k from x to y might not have any violated edges incident to x or y, even if $f(x) > f(y)$. Consider $G_{2n} = (V_{2n}, E)$ where $V_{2n} = \{x_1, \dots, x_{2n}\}, E = \{(x_i, x_n) \mid i < n\} \cup (x_n, x_{n+1}) \cup \{(x_{n+1}, x_j) \mid j > n + 1\}$. G_n is a 3-TC-spanner for itself. Now set $f(x_i) = 1$ for $i \le n$ and $f(x_i) = 0$ otherwise. Clearly, this function is $\frac{1}{2}$-far from monotone, but only one edge, (x_n, x_{n+1}) is violated in the 3-TC-spanner.

As demonstrated by Lemma 5.1, all the 2-TC-spanner constructions yield monotonicity testers for functions defined on the corresponding posets. This lemma led to significant improvements in monotonicity testers for several graph families, including planar graphs and, in general, H-minor-free graphs [BGJ$^+$09a]. Indeed, [BGJ$^+$09a] achieve testers with $O(\log^2 n/\epsilon)$ queries for H-minor-free graphs using their construction of sparse 2-TC-spanners for this graph family, whereas the previous tester, due to Fischer *et al.* [FLN$^+$02], worked only for planar graphs and required $\Theta(\sqrt{n}/\epsilon)$ queries.

We briefly discuss the limitations of the TC-spanner method for constructing monotonicity testers. The lower bounds in [BGJ$^+$09b, BGJ$^+$10] on the size of the sparsest 2-TC-spanners for the hypercube and the hypergrid (described in Theorem 3.1) rule out the TC-spanner approach for improving monotonicity testers on the hypercube and hypergrid. Currently, the running time of the best tester for monotonicity of functions of the form $f : \{0,1\}^d \to R$ and, more generally, $f : [m]^d \to R$, where R is an arbitrary range, is $O\left(\frac{d}{\epsilon}\log m \cdot \log|R|\right)$ [DGL+99]. The best known lower bound (for the hypercube with range $R = \{0,1\}$) is $\Omega(\log\log d)$ [FLN+02]. (There are better bounds for restricted classes of tests in [FLN+02] and [SMCB10].) Even though for a fixed d, it is known that the optimal monotonicity tester for the grid runs in time $\Theta(\frac{\log m}{\epsilon})$ [HK04, Fis04], bridging the gap between the lower and upper bounds for arbitrary d has remained elusive. Lemma 5.1 showed that if a 2-TC-spanner of size $o(2^d d^2)$ for the hypercube or, more generally, a 2-TC-spanner of size $o(m^d d^2 \log^2 m)$ for the hypergrid were found, the monotonicity tester of [DGL+99] would be improved. In the light of the lower bounds for the hypercube and the hypergrid, a fundamentally new approach is required.

Testing if a function is Lipschitz. In the important special case when G_n is the directed line, Lemma 5.1 yields an optimal tester for whether a function of the form $f : [n] \to R$ is monotone or, equivalently, of whether a list of n elements is sorted, that runs in time $O(n/\epsilon)$. (There is another optimal tester for this problem that was discovered first [EKK+00].) Jha and Raskhodnikova [JR10] observe that the test and analysis in Lemma 5.1 apply to any property of a list of numbers if (a) it can be expressed in terms of pairs of list elements and (b) it is transitive: namely, whenever (x,y) and (y,z) are not *violated*, (x,z) is also not violated. In particular, it applies to testing whether a function of the form $f : [n] \to \mathbb{R}$ is Lipschitz. A function $f : \mathcal{D} \to \mathcal{R}$ is called *Lipschitz* if $dist_{\mathcal{R}}(f(x), f(y)) = dist_{\mathcal{D}}(x,y)$ for all x, y in $\mathcal{D}$, where $dist_{\mathcal{R}}$ and $dist_{\mathcal{D}}$ denote the distance functions on the range and domain of f, respectively. Testing the Lipschitz property has applications to programs with noisy inputs and to data privacy.

Note that the Lipschitz property was defined in terms of pairs of domain elements. Consider a function $f : [n] \to \mathbb{R}$. We say a pair (x,y) is *violated* if $|f(y) - f(x)| > |y - x|$. Then if (x,y) and (y,z) are not violated, it implies that neither is (x,z). Thus, the requirements (a) and (b) above hold and, using their observation, Jha and Raskhodnikova get a $O(n/\epsilon)$ *Lipschitz* test for functions of the form $f : [n] \to \mathbb{R}$ via the optimal 2-TC-spanner construction of the line.

5.2 Application to Property Reconstruction

Property-preserving data reconstruction was introduced Ailon, Chazelle, Comandur and Liu [ACCL08]. In this model, a reconstruction algorithm, called a *filter*, sits between a *client* and a *dataset*. A dataset is viewed as a function $f : \mathcal{D} \to \mathcal{R}$. Client accesses the dataset using *queries* of the form $x \in \mathcal{D}$ to the filter. The filter *looks up* a small number of values in the dataset and outputs $g(x)$, where g must satisfy some fixed *structural* property $\mathcal{P}$. Extending this notion, Saks and Seshadhri [SS08] defined *local* reconstruction. A filter is *local* if it allows for a local (or distributed) implementation: namely, if the output function g does not depend on the order of the queries.

Definition 5.1 (Local filter)**.** *A* local filter *for reconstructing property $\mathcal{P}$ is an algorithm A that has oracle access to a function $f : \mathcal{D} \to \mathcal{R}$, and to an auxiliary random string ρ (the "random seed"), and takes as input $x \in \mathcal{D}$. For fixed f and ρ, A runs deterministically on input x to produce an output $A_{f,\rho}(x) \in \mathcal{R}$. As x varies over the domain $\mathcal{D}$, this defines a function $g : \mathcal{D} \to \mathcal{R}$, where $g(x) = A_{f,\rho}(x)$. (Note that a local filter has no internal state to store previously made queries.) The filter must satisfy the following conditions:*

- *For each f and ρ, the function g output by the filter satisfies $\mathcal{P}$.*
- *If f satisfies $\mathcal{P}$, then g is identical to f with probability at least $1-\delta$, for some $\delta \leq 1/3$. The parameter δ is called* error probability.

In answering query $x \in \mathcal{D}$, the filter A may ask for values of f at domain points of its choice using its oracle access to f. Each such access made to the oracle is called a *lookup* to distinguish it from the client query x. A local filter is *non-adaptive* if the set of domain points that the filter looks up to answer an input query x does not depend on answers given by the oracle.

Saks and Seshadhri also required that g must be sufficiently close to f: *With high probability (over the choice of ρ), $Dist(g,f) \leq B(n) \cdot Dist(f,\mathcal{P})$, where $B(n)$ is called the* error blow-up. ($Dist(g,f)$ is the number of points in the domain on which f and g differ. $Dist(f,\mathcal{P})$ is $\min_{g \in \mathcal{P}} Dist(g,f)$.) If a local filter along with Definition 5.1 satisfies this condition, we call it *distance-respecting.*

Local Monotonicity Reconstructors. The first property considered in the reconstruction [ACCL08] and local reconstruction [SS08] modes was monotonicity of functions. (See Section 5.1 for a definition.) A (distance-respecting) filter for monotonicity can be used, for example, when a program will run correctly only if its input is sorted. Then, instead of accessing the input directly, the program can access it via a filter, which will ensure that the program always sees a sorted input, making small corrections when necessary. A local filter can be implemented in a distributed manner with an additional guarantee that every program run on the same not-quite-sorted input will see the same corrected version. This can be done by supplying the same random string to each copy of the filter.

Saks and Seshadhri [SS08] give a *distance-respecting* local monotonicity filter for the directed hypergrid, $\mathcal{H}_{m,d}$, that makes $(\log m)^{O(d)}$ lookups per query. No non-trivial monotonicity filter for the hypercube $\mathcal{H}_d$ (performing $o(2^d)$ lookups per query) is known. One of the monotonicity filters in [ACCL08] is a local filter for the directed line $\mathcal{H}_{m,1}$ with $O(\log m)$ lookups per query (but a worse error blow up than in [SS08]). As observed in [SS08], this upper bound is tight. Notably, all known local filters for monotonicity property are *non-adaptive*. A lower bound of $2^{\alpha d}$, on the number of lookups per query for a *distance-respecting* local monotonicity filter on $\mathcal{H}_d$ with *error blow-up* $2^{\beta d}$, where α, β are sufficiently small constants, appeared in [SS08].

[BGJ+10] show how to construct sparse 2-TC-spanners from local monotonicity reconstructors with low lookup complexity. These constructions, in conjunction with lower bounds on the size of 2-TC-spanners of the hypergrid and hypercube, described in Section 3.1, imply lower bounds on lookup complexity of local monotonicity reconstructors with arbitrary blow-up. The transformations from non-adaptive and adaptive reconstructors are stated in Theorems 5.2 and 5.3, respectively.

Theorem 5.2 (Transformation from non-adaptive Local Monotonicity Reconstructors to 2-TC-spanners, [BGJ+10])**.** *Let $G_n = (V_n, E)$ be a poset on n nodes. Suppose there is a* non-adaptive *local monotonicity reconstructor A for G_n that looks up at most $\ell(n)$ values to answer any query $x \in V_n$ and has* error *probability at most δ. Then there is a 2-TC-Spanner of G_n with at most $O(n\ell(n) \cdot \lceil \log n / \log(1/\delta) \rceil)$ edges.*

Proof. Let A be a local reconstructor given by the statement of the theorem. Let $\mathcal{F}$ be the set of pairs (x, y) with x, y in V_n such that $x \prec y$. Then, $\mathcal{F}$ is of size at most $\binom{n}{2}$. Given $(x, y) \in \mathcal{F}$, let $\mathsf{cube}(x, y)$ be the set $\{z \in V_n : x \preceq z \preceq y\}$. Define function $f^{(x,y)}(v)$ to be 1 on all $v \succeq x$ and all $v \succeq y$, and 0 everywhere else. Also, define function $f^{(\overline{x,y})}(v)$, which is identical to $f^{(x,y)}(v)$ for all $v \notin \mathsf{cube}(x, y)$ and 0 for $v \in \mathsf{cube}(x, y)$. Both, $f^{(x,y)}$ and $f^{(\overline{x,y})}$, are monotone functions for all $(x, y) \in \mathcal{F}$. Let A_ρ be the deterministic algorithm which runs A with the random seed fixed to ρ. We say a string ρ is *good* for $(x, y) \in \mathcal{F}$ if filter A_ρ on input $f^{(x,y)}$ returns $g = f^{(x,y)}$ *and* on input $f^{(\overline{x,y})}$ returns $g = f^{(\overline{x,y})}$.

Now we show that there exists a set S of size $s \le \lceil 2 \log n / \log(1/2\delta) \rceil$, consisting of strings used as random seeds by A, such that for every $(x, y) \in \mathcal{F}$ some string $\rho \in S$ is good for (x, y). We choose S by picking strings used as random seeds uniformly and independently at random. Since A has error probability at most δ, we know that for every monotone f, with probability at least $1 - \delta$ (with respect to the choice of ρ), the function $A_{f,\rho}$ is identical to f. Then, for fixed $(x, y) \in \mathcal{F}$ and uniformly random ρ,

$$\begin{aligned} \Pr[\rho \text{ is not } \textit{good} \text{ for } (x,y)] \le \quad & \Pr[A_\rho \text{ on input } f^{(x,y)} \text{ fails to output } f^{(x,y)}] \\ + \quad & \Pr[A_\rho \text{ on input } f^{(\overline{x,y})} \text{ fails to output } f^{(\overline{x,y})}] \le 2\delta. \end{aligned}$$

Since strings in S are chosen independently, $\Pr[\text{no } \rho \in S \text{ is good for } (x, y)] \le (2 \cdot \delta)^s$, which, for $s = \lceil 2 \log n / \log(1/2\delta) \rceil$, is at most $1/n^2 < 1/|\mathcal{F}|$. By a union bound over $\mathcal{F}$,

$$\Pr[\text{for some } (x, y) \in \mathcal{F}, \text{ no } \rho \in S \text{ is good for } (x, y)] < 1.$$

Thus, there exists a set S with required properties.

We construct our 2-TC-spanner $H = (V_n, E_H)$ of G_n using set S described above. Let $\mathcal{N}_\rho(x)$ be the set consisting of x and all vertices looked up by A_ρ on query x. For each string $\rho \in S$ and each vertex $x \in V_n$, connect x to all comparable vertices in $\mathcal{N}_\rho(x)$ (other than itself) and orient these edges according to their direction in G_n.

We prove H is a 2-TC-Spanner as follows. Suppose not, *i.e.,* there exists $(x, y) \in \mathcal{F}$ with no path of length at most 2 in H from x to y. Consider $\rho \in S$ which is *good* for (x, y). Define function h by setting

$h(v) = f^{(x,y)}(v)$ for all $v \notin \mathsf{cube}(x, y)$. Then $h(v) = f^{(\overline{x,y})}(v)$ for all $v \notin \mathsf{cube}(x, y)$, by definition of $f^{(\overline{x,y})}$. For a $v \in \mathsf{cube}(x, y)$, set $h(v)$ to 1 for $v \in \mathcal{N}_\rho(x)$ and to 0 for $v \in \mathcal{N}_\rho(y)$. All unassigned points are set to 0. By the assumption above, $\mathcal{N}_\rho(x) \cap \mathcal{N}_\rho(y)$ does not contain any points in $\mathsf{cube}(x, y)$. Therefore, h is well-defined. Since, ρ is *good* for (x, y) and h is identical to $f^{(x,y)}$ for all look ups made on query x, $A_\rho(x) = h(x) = 1$. Similarly, $A_\rho(y) = h(y) = 0$. But $x \prec y$, so $A_{h,\rho}(v)$ is not monotone. Contradiction.

The number of edges in H is at most $\sum_{x \in V_n, \rho \in S} |\mathcal{N}_\rho(x)| \leq n \cdot \ell \cdot s \leq n\ell \cdot \lceil 2 \log n / \log(1/2\delta) \rceil$. ∎

Next theorem applies even to *adaptive* local monotonicity reconstuctors. It takes into account how many lookups on query x are points incomparable to x. In particular, if there are no such lookups, then constructed 2-TC-spanner is of the same size as in Lemma 5.2.

Theorem 5.3 (Transformation from adaptive Local Monotonicity Reconstructors to 2-TC-spanners, [BGJ+10])**.** *Let $G_n = (V_n, E)$ be a poset on n nodes. Suppose there is a (possibly* adaptive*) local monotonicity reconstructor A for G_n that, for any query $x \in V_n$, looks up at most $\ell_1(n)$ vertices comparable to x and at most $\ell_2(n)$ vertices incomparable to x, and has* error *probability at most δ. Then there is a 2-TC-Spanner of G_n with at most $O(n\ell_1(n) \cdot 2^{\ell_2(n)} \lceil \log n / \log(1/\delta) \rceil)$ edges.*

Proof. Define $\mathcal{F}$, $f^{(x,y)}$, $f^{(\overline{x,y})}$, A_ρ and S as in the proof of Theorem 5.2. As before, for each $x \in V_n$, we define sets $\mathcal{N}_\rho(x)$, and construct the 2-TC-Spanner H by connecting each x to comparable points in $\mathcal{N}(x)$ for all $\rho \in S$ and orienting the edges according to G_n. However, now $\mathcal{N}_\rho(x)$ is a union of several sets $\mathcal{N}_\rho^{b,w}(x)$, indexed by $b \in \{0, 1\}$ and $w \in \{0, 1\}^{\ell_2(n)}$. (In addition, $\mathcal{N}_\rho(x)$ contains x.) For each $x \in V_n$, $b \in \{0, 1\}$ and $w \in \{0, 1\}^{\ell_2(n)}$, let $\mathcal{N}_\rho^{b,w}(x) \subseteq V_n$ be the set of lookups performed by A_ρ on query x, assuming that the oracle answers all lookups as follows. When a lookup y is comparable to x, answer 0 if $y \prec x$, b if $y = x$, 1 if $x \prec y$. Otherwise, if y is the i'th lookup made to an incomparable point for some $i \in [\ell_2]$, answer $w[i]$. Recall that we set $N_\rho(x)$ to be the union of $N_\rho^{b,w}$ for all $b \in \{0, 1\}$ and all $w \in \{0, 1\}^{\ell_2(n)}$. This completes the description of $\mathcal{N}_\rho(x)$ and construction of H.

The argument that H is a 2-TC-spanner proceeds similarly to that in the proof of Theorem 5.2. The caveat is that an adaptive local filter might choose lookups based on the answers to previous lookups. The constructed function h sets all points comparable to x to 0 if they are below x and 1 if they are above x. However, points incomparable to x might be comparable to y and might be set to 0 or 1, depending on whether they are above or below y. Since we included sets of points queried under all these possibilities in $\mathcal{N}_\rho(x)$, we can now conclude that $A_\rho(x) = h(x) = 1$. The same applies for y. So, $A_{h,\rho}$ outputs a non-monotone function, witnessed the pair (x, y). Contradiction.

We proceed to bound the number of edges E_H in H. For each $\rho \in S$, $x \in V_n$, $b \in \{0, 1\}$, and $w \in \{0, 1\}^{\ell_2(n)}$, the number of vertices in $N_{b,w}^\rho(x)$ comparable to x is at most $\ell_1(n)$. Therefore,

$$|E_H| \leq \ell_1(n) \cdot 2 \cdot 2^{\ell_2(n)} \cdot |S| \leq O\left(n \cdot \ell_1(n) \cdot 2^{\ell_2(n)} \lceil \log n / \log(1/\delta) \rceil\right).$$ ∎

In Theorems 5.2 and 5.3 when δ is sufficiently small the bounds on the 2-TC-Spanner size become $O(n\ell(n))$ and $O(n\ell_1(n) \cdot 2^{\ell_2(n)})$, respectively. As pointed out earlier, all known monotonicity reconstructors are non-adaptive. It is an open question whether it is possible to give a transformation from adaptive local monotonicity reconstructors to 2-TC-spanners without incurring an exponential dependence on the number of lookups made to points incomparable to the query point. This dependence is an artifact of the proof or an indication that lookups to incomparable points might be helpful for adaptive local monotonicity reconstructors.

Theorems 5.2 and Theorem 5.3 imply the following lower bounds on the lookup complexity of local monotonicity reconstructors. These lower bounds hold for any error-blow up.

Corollary 5.4 ([BGJ+10])**.** *Consider a nonadaptive local monotonicity filter with constant error probability δ. If the filter is for functions $f : \mathcal{H}_{m,d} \to \mathbb{R}$, it must perform $\Omega\left(\frac{\log^{d-1} m}{d^d (2 \log \log m)^{d-1}}\right)$ lookups per query. If the filter is for functions $f : \mathcal{H}_d \to \mathbb{R}$, it must perform $\Omega\left(2^{\alpha d}/d\right)$ lookups per query, where $\alpha \geq 0.1620$.*

Corollary 5.5 ([BGJ+10])**.** *Consider an (adaptive) local monotonicity filter with constant error probability δ, that for every query $x \in V_n$, looks up at most ℓ_2 vertices incomparable to x. If the filter is for functions*

$f : \mathcal{H}_{m,d} \to \mathbb{R}$, it must perform $\Omega\left(\frac{\log^{d-1} m}{2^{\ell_2} d^d (2\log\log m)^{d-1}}\right)$ lookups to vertices comparable to x per query x. If the filter is for functions $f : \mathcal{H}_d \to \mathbb{R}$, it must perform $\Omega\left(2^{\alpha d - \ell_2}/d\right)$ comparable lookups, where $\alpha \geq 0.1620$.

Prior to [BGJ+10], no lower bounds for monotonicity reconstructors on $\mathcal{H}_{m,d}$ with dependence on both m and d were known. Unlike the bound in [SS08], the TC-spanner-based lower bounds hold for any error blow-up. These bounds are tight for reconstructors that are either non-adaptive or perform the number of incomparable lookups that is polylogarithmic in the number of points in the domain. Specifically, for the hypergrid $\mathcal{H}_{m,d}$ of constant dimension d, the number of lookups is $(\log m)^{\Theta(d)}$, and for the hypercube $\mathcal{H}_d$, it is $2^{\Theta(d)}$ for any error blow-up.

5.3 Application to Key Management in an Access Control Hierarchy

Atallah *et al.* [AFB05] used sparse Steiner TC-spanners to construct efficient key management schemes for access control hierarchies. An *access hierarchy* is a partially ordered set G of access classes. Each user is entitled to access a certain class and all classes reachable from the corresponding node in G. One approach for devising a cryptographic protocol that enforces the access hierarchy is to have the users follow a *key management scheme* [ABF06, AFB05, SFM07, ABFF09, BGRW10]. Here, each edge (i, j) has an associated public key $P(i, j)$, and each node i, an associated secret key k_i. Only users with the secret key for a node have the required permissions for the associated access class. The public and secret keys are designed so that there is an efficient algorithm A which takes k_i and $P(i, j)$ and generates k_j, but for each (i, j) in G, it is computationally hard to generate k_j without knowledge of k_i. Thus, a user can efficiently generate the required keys to access a descendant class, but not other classes. The number of runs of algorithm A needed to generate a secret key k_v from a secret key k_u is equal to $d_G(u, v)$. To speed this up, Atallah *et al.* [ABFF09] suggest adding edges and nodes to G to increase connectivity. To preserve the access hierarchy represented by G, the new graph H must be a Steiner TC-spanner of G. The number of edges in H corresponds to the space complexity of the scheme, while the stretch k of the spanner corresponds to the time complexity.

We note that the time to find the path from u to v is also important in this application. In the upper bounds from [BGRW10] listed in Table 1, this time is $O(d)$, which for, say, constant d is likely to be much less than $2g(n)$ or $3g(n)$, where $g(n)$ is the time to run algorithm A. This is because algorithm A involves the evaluation of a cryptographic hash function, which is very expensive: any hash function secure against poly(n)-time adversaries requires $g(n) \geq \text{polylog}\, n$ evaluation time under existing number-theoretic assumptions.

5.4 Application to Computing Partial Products in a Semigroup

Yao [Yao82] and Alon and Schieber [AS87] study space-efficient data structures for the following problem: Preprocess elements $\{s_1, \ldots, s_n\}$ of a semigroup $(S, \circ)$ to be able to compute partial products $s_i \circ s_{i+1} \circ \cdots \circ s_j$ for all $i, j \in [n]$ with at most k queries to a small database of pre-computed partial products. Examples of a semigroup $(S, \circ)$ include $(\mathbb{R}, \min)$, the space of real d-dimensional vectors with operation $(x_1, \ldots, x_d) \circ (y_1, \ldots, y_d) = (\min(x_1, y_1), \ldots, \min(x_d, y_d))$, and the space of real $d \times d$ matrices with equipped with the multiplication operation.

Bhattacharyya *et al.* [BGJ+09a] point out that the problem of computing partial products in a semigroup reduces to finding a sparsest k-TC-spanner for a directed line L_{n+1}. If the database stores a product $s_u \circ \cdots \circ s_v$ for each k-TC-spanner edge $(u, v+1)$, every product $s_i \circ \cdots \circ s_j$ can be computed by multiplying the products corresponding to the edges on a path of length at most k from i to $j+1$ in the k-TC-spanner for L_{n+1}.

Chazelle [Cha87] and Alon and Schieber [AS87] also consider a generalization of the above problem, where the input is an (undirected) tree T with an element s_i of a semigroup associated with each vertex i. The goal is to create a space-efficient data structure that allows to compute the product of elements associated with all vertices on the path from i to j, for all vertices i, j in T. As before, only k queries to the data structure are allowed for each product computation. The generalized problem reduces to finding a sparsest k-TC-spanner for a directed tree T' obtained from T by appending a new vertex to each leaf, and then selecting an arbitrary root and directing all edges away from it. A k-TC-spanner for T' with $s(n)$ edges yields a preprocessing scheme with space complexity $s(n)$ for computing products on T with at most $2k$ queries as follows. The database stores a product $s_{v_1} \circ \cdots \circ s_{v_t}$ for each k-TC-spanner edge (v_1, v_{t+1}) if the endpoints of that edge are connected by the path $v_1, \cdots, v_t, v_{t+1}$ in T'. Let $LCA(u,v)$ denote the lowest common ancestor of u and v in T. To compute the product corresponding to a path from u to v in T, we consider 2 cases: (1) if u is an ancestor of v (or vice versa) in T, query the products corresponding to the k-TC-spanner edges on the shortest path from u to a child of v (from v to a child of u, respectively); (2) otherwise, make queries corresponding to the k-TC-spanner edges on the shortest path from $LCA(u,v)$ to a child of u and on the shortest path from a child of $LCA(u,v)$ nearest to u to a child of u. This give a total of at most $2k$ queries.

References

[ABF06] Mikhail J. Atallah, Marina Blanton, and Keith B. Frikken. Key management for non-tree access hierarchies. In *SACMAT*, pages 11–18, 2006.

[ABFF09] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):1–43, 2009.

[AC06] Nir Ailon and Bernard Chazelle. Information theory in property testing and monotonicity testing in higher dimension. *Inf. Comput.*, 204(11):1704–1717, 2006.

[ACCL08] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property-preserving data reconstruction. *Algorithmica*, 51(2):160–182, 2008.

[ADD$^+$93] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.

[AFB05] Mikhail J. Atallah, Keith B. Frikken, and Marina Blanton. Dynamic and efficient key management for access hierarchies. In *ACM Conference on Computer and Communications Security*, pages 190–202, 2005.

[AG06] Ittai Abraham and Cyril Gavoille. Object location using path separators. In *PODC*, pages 188–197, 2006.

[AGU72] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972.

[AS87] Noga Alon and Baruch Schieber. Optimal preprocessing for answering on-line product queries. Technical Report 71/87, Tel-Aviv University, 1987.

[Awe85] Baruch Awerbuch. Communication-time trade-offs in network synchronization. In *PODC*, pages 272–276, 1985.

[BGJ$^+$09a] Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Transitive-closure spanners. In *SODA*, pages 932–941, 2009.

[BGJ$^+$09b] Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Transitive-closure spanners of the hypercube and the hypergrid. ECCC Report TR09-046, 2009.

[BGJ$^+$10] Arnab Bhattacharyya, Elena Grigorescu, Madhav Jha, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. 2010.

[BGRW10] Arnab Bhattacharyya, Elena Grigorescu, Sofya Raskhodnikova, and David Woodruff. Steiner transitive-closure spanners of d-dimensional posets. Manuscript, 2010.

[BRR10] Piotr Berman, Sofya Raskhodnikova, and Ge Ruan. Finding sparser directed spanners. Manuscript, 2010.

[BRW05] Tugkan Batu, Ronitt Rubinfeld, and Patrick White. Fast approximate PCPs for multidimensional bin-packing problems. *Inf. Comput.*, 196(1):42–56, 2005.

[BS06] Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $\tilde{O}(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.

[BTS94] Hanls L. Bodlaender, Gerard Tel, and Nicola Santoro. Tradeoffs in non-reversing diameter. *Nordic Journal of Computing*, 1(1):111 – 134, 1994.

[Cha87] Bernard Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.

[Coh98] Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998.

[Coh00] Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *JACM*, 47(1):132–166, 2000.

[Cow01] Lenore Cowen. Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183, 2001.

[CW04] Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *J. Algorithms*, 50(1):79–95, 2004.

[DGL$^+$99] Yevgeniy Dodis, Oded Goldreich, Eric Lehman, Sofya Raskhodnikova, Dana Ron, and Alex Samorodnitsky. Improved testing algorithms for monotonicity. In *RANDOM*, pages 97–108, 1999.

[Dil50] R. P. Dilworth. A decomposition theorem for partially ordered sets. *The Annals of Mathematics, Second Series*, 51(1):161–166, 1950.

[DM40] B. Dushnik and E.W. Miller. Concerning similarity transformations of linearly ordered sets. *Bulletin Amer. Math. Soc.*, 46:322–326, 1940.

[EKK$^+$00] F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *JCSS*, 60(3):717–751, 2000.

[Elk01] M. Elkin. Computing almost shortest paths. In *PODC*, pages 53–62, 2001.

[EP00] Michael Elkin and David Peleg. Strong inapproximability of the basic k-spanner problem. In *ICALP*, pages 636–647, 2000.

[EP01] Michael Elkin and David Peleg. The client-server 2-spanner problem with applications to network design. In *SIROCCO*, pages 117–132, 2001.

[EP07] Michael Elkin and David Peleg. The hardness of approximating spanner problems. *Theory Comput. Syst.*, 41(4):691–729, 2007.

[Fis04] Eldar Fischer. On the strength of comparisons in property testing. *Inf. Comput.*, 189(1):107–116, 2004.

[FLN$^+$02] Eldar Fischer, Eric Lehman, Ilan Newman, Sofya Raskhodnikova, Ronitt Rubinfeld, and Alex Samorodnitsky. Monotonicity testing over general poset domains. In *STOC*, pages 474–483, 2002.

[GGL$^+$00] Oded Goldreich, Shafi Goldwasser, Eric Lehman, Dana Ron, and Alex Samorodnitsky. Testing monotonicity. *Combinatorica*, 20(3):301–337, 2000.

[GGR98] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.

[Hes03] William Hesse. Directed graphs requiring large numbers of shortcuts. In *SODA*, pages 665–669, 2003.

[HK04] Shirley Halevy and Eyal Kushilevitz. Testing monotonicity over graph products. In *ICALP*, pages 721–732, 2004.

[Hoc97] D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, Boston, 1997.

[JR10] Madhav Jha and Sofya Raskhodnikova. Testing and reconstruction of lipschitz functions with applications to data privacy. Manuscript, 2010.

[Kor01] Guy Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001.

[LT79] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[Pel00] David Peleg. *Distributed computing: a locality-sensitive approach.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[PS89] David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[PU89a] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

[PU89b] David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *JACM*, 36(3):510–530, 1989.

[RS96] Ronitt Rubinfeld and Madhu Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[RTZ02] Liam Roditty, Mikkel Thorup, and Uri Zwick. Roundtrip spanners and roundtrip routing in directed graphs. In *SODA*, pages 844–851, 2002.

[Sei] Raimund Seidel. Understanding the inverse Ackermann function. Available at `http://cgi.di.uoa.gr/~ewcg06/invited/Seidel.pdf`.

[SFM07] Alfredo De Santis, Anna Lisa Ferrara, and Barbara Masucci. Efficient provably-secure hierarchical key assignment schemes. In *MFCS*, pages 371–382, 2007.

[SMCB10] David Garca Soriano, Arie Matsliah, Sourav Chakraborty, and Jop Briet. Monotonicity testing and shortest-path routing on the cube. ECCC Report TR10-048, 2010.

[SS08] M. E. Saks and C. Seshadhri. Parallel monotonicity reconstruction. In *Proceedings of the 19th Annual Symposium on Discrete Algorithms (SODA)*, pages 962–971, 2008.

[Tho92] Mikkel Thorup. On shortcutting digraphs. In *WG*, pages 205–211, 1992.

[Tho95] Mikkel Thorup. Shortcutting planar digraphs. *Combinatorics, Probability & Computing*, 4:287–315, 1995.

[Tho97] Mikkel Thorup. Parallel shortcutting of rooted trees. *J. Algorithms*, 23(1):139–159, 1997.

[Tro92] W. Trotter, editor. *Combinatorics and Partially Ordered Sets: Dimension Theory.* Johns Hopkins University Press, Baltimore, MD, 1992.

[TZ01] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001.

[TZ05] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *JACM*, 52(1):1–24, 2005.

[Yan82] Mihalis Yannakakis. The complexity of the partial order dimension problem. *JMAA*, 3(3):351–358, 1982.

[Yao82] Andrew Chi-Chih Yao. Space-time tradeoff for answering range queries (extended abstract). In *STOC*, pages 128–136, 1982.