How to infinitely share a secret more efficiently

Anat Paskin-Cherniavsky*

November 19, 2016

Abstract

We device a general secret sharing scheme for evolving access structures (following [KNY16]). Our scheme has (sub)exponentially smaller share complexity (share of i'th party) for certain access structures compared to the general scheme in [4]. We stress that unlike [4]'s scheme, our scheme requires that the entire evolving access structure is known in advance. Revising, [4]'s scheme (in its most optimized form) is based on a representation of the access structure by an ordered (possibly infinite) oblivious, read once decision tree. Each node is associated with an output of the function (0 or 1). The tree is augmented to cut paths that reach a node where f evaluates to 1 at that node (works for evolving access structures, in which the descendants of all 1-nodes must be 1). Each party P_i receives a (single-bit) share for each edge exiting a node labeled by x_i.

Generally, the scheme of [4] has share complexity $O(w_T(i))$, where $w_T(i)$ is the width of layer i in a decision tree for the access structure (equivalently, monotone function). In general, this width can reach $\Omega(2^i)$. To get non trivial share complexity, $e^{n^{o(1)}}$, a *tree* of width $e^{n^{o(1)}}$ is required.

Our scheme is based on a generalized (infinite) tree representation of the access structure. The main difference is that vertices are labeled with sequences of variables, rather than a single variable. As a result, we often get smaller trees, and the edges e are labeled by more complex (non-evloving) monotone functions g_e of the variables in the sequence. The share associated with the edge is shared (among the parties in the relevant sequence). As a result, the tree is smaller, while the shares received for every edge in it are bigger. Still, the tradeoff is often on our side. Namely, for access structures with ordered read-once *branching programs* with relatively small width, $e^{O(i^c)}$ for $c < 0.25$, share complexity of $e^{n^{o(1)}}$ is achieved. More specifically, the resulting share complexity is $(iw_{BP}(i^2))^{O(\log i + \log w_{BP}(i^2))}$. In particular, for $w = \Omega(i)$, we get share complexity of $w_{BP}(i^2)^{O(\log w_{BP}(i^2))}$.

Finally, a further improved variant of our scheme for a special class of "counting" access structures yields polynomial share complexity. In particular, we obtain an evolving secret sharing scheme for *evolving majority* with share complexity $\tilde{O}(n^6)$, answering an open question of [4].

1 Introduction

In this paper, we continue the investigation of evolving secret sharing schemes (directly extending the work of [4]). In such schemes, we only a-priori know the access structure, but not the number of parties that will appear, or any bound on it. We assume that the access structure is such that at the time of arrival of the i-th party, the qualification of all subsets of $[i-1]$ remains as before, and only sets in $2^{[i]} \setminus 2^{[i-1]}$ are newly determined (so called *evolving*). Such access structures can be handled in our setting where parties can only join, but not leave, and every newly joined party P_i receives its share at the time of its arrival, which is never changed in the future. See [4] for more motivation and previous work on the problem of secret sharing for unbounded sets.

We study the share complexity of such access structures, that is, the size $|s_i|$ of the share received by party P_i in a valid secret sharing scheme. This key complexity measure for secret sharing schemes is far from understood for standard schemes as well, with upper bounds of $2^{\Omega(n)}$ share complexity (size of biggest share) for n-party access structures, and only $\tilde{O}(n/\log(n))$ lower bounds. Clearly, in the harder setting of evolving access, it is strictly harder to build schemes with

*Ariel university, Israel

Repository for this paper: https://github.com/anpc/infinitely-share

small share complexity. More precisely, a scheme for an evolving access structure $\mathcal{A}$ that equals a (standard) scheme $\mathcal{B}$ for n parties when restricted to sets in $2^{[n]}$, then a scheme with the same share complexity bound for $\mathcal{B}$ (for each party) trivially follows. The other direction is not clear. Given a sequence of access structuresă$\mathcal{A}_1, \ldots, \mathcal{A}_i, \ldots$ which are restrictions of a "proper" evolving access structure and corresponding sharing schemes, $S_1, \ldots, S_i, \ldots$ it is not clear how to combine them into a single scheme for the evolving access structure with related share complexity.[1]

We focus on improving the known upper bounds for share complexity of schemes for general evolving access structures (first devised in [4]). Unlike [4]'s scheme, our scheme requires that the entire evolving access structure is known in advance.

Revising [4]'s solution, Their scheme is essentially a reduction to evolving undirected st-conn, executed on an infinite tree computing $\mathcal{A}$ (augmented with a sink vertex t). Evolving ust-conn has an easy (ideal) evolving scheme, pointed out by [4]. Each party is assigned a (finite) set of edges, where the corresponding scheme hands a party a bit per edge. The evolving scheme takes care of assigning each party the shares of relevant edges upon their arrival, in a straightforward manner.

On a high level, our scheme constructs an oblivious, ordered, generalized (augmented) tree as above. The main generalization is that vertices are labeled by a sequence $\overline{x} = x_i, x_{i+1}, \ldots, x_j$ of variables, rather then a single one. Each outgoing edge e is labeled by a monotone function $g_e(\overline{x})$. Our scheme also manages the share distribution for the evolving ust-conn. Namely, the share s_e assigned to an edge labeled by a given function value $g_e(x_i, x_{i+1}, \ldots, x_j) = l$, is obtained by sharing s_e among the parties $P_i, \ldots, P_j$, and giving each a share upon arrival. The share s_e and its sharing is only generated upon arrival of P_i. [2]

The main efficiency gain is that for individual edges we apply standard secret sharing schemes, which are often quite efficient. The longer we make the sequences, the smaller is the tree, but the share sizes in the individual schemes grow, so there is a tradeoff. Our choice of tree parameters is such, that the resulting access structure has share complexity $e^{i^{o(1)}}$ if it has a *branching program* of width $e^{O(n^c)}$ for constant $c < 0.25$. We also point out a useful class of access structures, for which we are able to further imporve our construction, so the resulting share complexity is polynomial. This class includes the *evolving majority* function, answering an open problem by [4]. In this access structure, a set X is qualified if $|X \cap [t]| > t/2$ for some t.

Road map. In section 2 we recall the definitions of evolving access structures and evolving secret sharing schemes. We also refer to several evolving and standard secret sharing schemes form the literature that will be useful to us. In Section 3 we present our main result. It includes our definition of monotone infinite branching programs (BP) (and trees as a special case), and present our general secret sharing scheme based on a BP representation. It achieves non-trivial share complexity of $e^{n^{o(1)}}$ for BP's with non-trivially small (slightly smaller) width $e^{O(n^c)}$ for $c < 0.25$. In Section 4 we devise a further improvement (of the analysis) of our scheme, to obtain a construction of an interesting class of functions including *evolving majority*. We conclude with some open questions in Section 5.

2 Preliminaries

Standard secret sharing. (See [2], for example.) An access structure is a monotone set $\mathcal{A} \subseteq 2^{[n]}$ (that is, if $A \subseteq B, A \in \mathcal{A}$, then $B \subseteq \mathcal{A}$). We identify $\mathcal{A}$ with its characteristic function $f_{\mathcal{A}}$, and use the terms interchangeably. A secret sharing scheme for $\mathcal{A}$ for sharing secrets in a (finite) domain X consists of a sharing algorithm Sh, and reconstruction algorithm Rec. Sh is a randomized mapping $\text{Sh} : X \to S_1 \times S_2 \ldots \times S_n$, and Rec is defined via $\text{Rec} : \{(T, S_T) | T \subseteq [n]\} \to X$. The scheme is private in the sense that for all pairs of secrets $x_1, x_2 \in X$, and all $T \notin \mathcal{A}$, the distributions $S_T(x_1), S_T(x_2)$ are identical. The scheme is correct in the sense that for all $A \in \mathcal{A}, x \in X$, $Pr[\text{Rec}(A, Sh(x)_A) = x] = 1$.

[1] A "strict" hardness follows from a lower bound in [4] for the evolving 2-threshold access structure, which is greater than $\log i + \log\log i$. For standard access structures $\lceil \log n \rceil$ is achieved by Shamir's scheme for (k, n)-threshold for any $k \leq n$.

[2] The idea of grouping parties into long subsequences is inspired by the (ad-hoc) efficient scheme of [4] for k-threshold evolving access structures, where parties are divided into "generations". One key difference between the two approaches, is that the [4] construction relies on any evolving scheme for a related functionality in a black-box way, while our evolving scheme is specifically tailored to the access structure.

The *share complexity* of a secret sharing scheme is defined as the maximal possible (bit) length of a share per secret bit: $max_{x\in X, i\in[n]}\text{support}(|\text{Sh}(x))_i|)/\log|X|$. We say the share complexity of an access structure $\mathcal{A}$ (denoted $\text{sc}(f_{\mathcal{A}})$) is bounded by some $g(n)$, if there exists a scheme for $\mathcal{A}$ with share complexity $O(g(n))$. Unless stated otherwise, in this work we focus on $X = \{0,1\}$, which is wlog., as one can share each bit via a fresh application of Sh to every bit of the secret. There exists a secret sharing scheme for all $\mathcal{A}$, with share complexity $O(MF(f_{\mathcal{A}}))$ [2], where MF denotes the monotone formula complexity of $f_{\mathcal{A}}$. Thus (by simple counting arguments), the approach yields exponential (in n) share complexity for some access structures. In the sequel, we refer to this construction as the ISN scheme.

Secret sharing for evolving access structures. We precisely follow the definitions and notation of [4]. In particular, we define evolving access strcutures and secret sharing schemes for such structures as in definitions 2.6, 2.8 in [4] respectively. Repeating briefly, an *evolving* access structure $\mathcal{A} = \cup_{i\geq 1}\mathcal{A}_i$ where for each $i \geq 1$ $\mathcal{A}_i \subseteq 2^{[i]}$ is a (standard monotone) access structure. Additionally, for all $i \geq 1$ it holds that $\{\mathcal{A}_{i+1}(i)\} = \mathcal{A}_i$, where, for a (standard) access structure $\mathcal{B}$ on some $g \geq i$ parties $\mathcal{B}(i)$ is defined as $B(X \cap [i])$. In other words, $\mathcal{A}$ is defined by its characteristic function $f : S \to \{0,1\}$, where $S = \{(s_1, s_2, \ldots, s_i, \ldots)_{i\in\mathbb{N}^+} | \forall i\ s_i \in \{0,1\}, \text{finitely many } s_i's \text{ equal } 1\}$, which is monotone in the sense that $f(s^1) \leq f(s^2)$ iff. $s^1 \subseteq s^2$, where the s^i's are interpreted as subsets of $\mathbb{N}^+$ in the natural way. Note that not any function $f : \{0,1\}^* \to \{0,1\}$ monotone in the standard sense (that the restriction of f to $\{0,1\}^n$ for each n is monotone) corresponds to an $f : S \to \{0,1\}$ as above. In particular, reinterpreting f using the natural correspondence between $\{0,1\}^*$ and S, padding vectors $x \in \{0,1\}^*$ with 0's to the right does not generally work, as this mapping is not one-to-one. For evolving access structures, however, it is the case that $f_{|x|}(x) = f_{|y|}(y)$ for all $x, y \in \{0,1\}^*$ mapping to the same value $z \in S$. Thus, for such access structures, we will use $x \in \{0,1\}^*$ and its padded version $z \in S$ interchangeably. For a string $x \in \{0,1\}^*$ (or $x \in S$), let $\#_1(x)$ ($\#_0(x)$) denote the number of 1's (0's) in x. For indices i, j $(x_i, \ldots, x_j)$ is naturally defined to be the corresponding substring of x if $1 \leq i \leq j \leq |x|$. If $i > j$ we define it to be the empty string ϵ.

A secret sharing scheme for an evolving access structure $\mathcal{A}$ is comprised of a sharing and reconstruction algorithms Sh, Recon. For each newly arriving party P_i, the sharing algorithm is given the secret s, and the sequence of shares given to all previous parties $s_1, \ldots, s_{i-1}$, and gives party P_i its share s_i, so that the resulting share distribution is consistent with a (standard) secret sharing scheme's requirements for $\mathcal{A}_i$.[3] As in standard secret sharing schemes, the reconstruction algorithm is given $x \in \{0,1\}^*$ corresponding to a finite set of parties, and the share vector held by s_x.

Secret sharing for ust-conn and st-conn. In the standard st-conn (family) access structure, we are given a finite directed graph $G(V, E)$, and $s, t \in V$ is a pair of designated nodes. Each party P_j ($j \in [|E|]$) is assigned a distinct edge e_j in E. The access structure consists of all sets of parties T such that the sub-graph $G(V, E_T)$ contains an s-t path. The undirected version of the access structure, ust-conn, is the same, except that G is undirected. We consider a slightly modified version of the problem, where party P_i is assigned a non-empty subset $m(i) = E_i \subseteq E$. The E_i's may intersect. A set T is qualified if $G(V, \bigcup_{i\in T} m(i))$ satisfies s-t connectivity. We denote the corresponding access structures by $ext - st - conn_G$ and $ext - ust - conn_G$ respectively.

Consider an infinite graph G, nodes $s, t \in V$ and an injective mapping $m : \mathbb{N}^+ \to E$. As in the non-evolving case, we define the evolving $ust - conn_G$ access structure to contain all $T \in \{0,1\}^*$, for which $G_x = G(V, \bigcup_{i\in T}\{m(i)\})$ has an s-t path.

There exists a simple scheme for ust-conn-family structures, giving a single bit to each party [1]. It was observed in [4], that the [1] construction readily extends to the case of evolving ust-conn (not extended).

Claim 2.1 *For all parameters (G, m), evolving ust-conn has an evolving secret sharing scheme with share complexity $sc(i) = 1$.*

For completeness, let us review the evolving ust-conn construction outlined in [4].

[3] When describing concrete schemes, it will often be convenient to assume that the sharing algorithm is not only given the shares of previous parties, but that it also stores and passes on to the share function the (finite) randomness used to generate all previous shares. This is wlog., as it can be resampled to be uniform over the randomness consistent with the previous shares. In particular, this randomness is not counted towards the share complexity of any of the parties.

Proof sketch. Each node $v \neq t, s$ in the graph is assigned a random $b_v \in \mathbb{F}_2$, while t is assigned the secret bit s, and s is assigned 0. The assignment is done lazily. That is, when party P_i touching a new node arrives, that node is assigned its random bit b_v. The party, corresponding to edge $\{u, v\}$ is given $b_v - b_u$ as its share, upon arrival. Upon reconstruction, if parties forming a path from s to t are present, adding up their shares indeed results in $b_s + b_t = s$ (every b_v along the path for $v \neq s, t$ cancels out). □
We will also use the following scheme for standard directed ext-st-conn.

Claim 2.2 *For all (finite) G, and m, there exists a secret sharing scheme for $ext - st - conn$ with parameters (G, m) with share complexity $n^{O(\log(n))}$.*

Proof sketch. The above result for the special case of $st - conn$ follows simply by applying ISN to the monotone formula of size $n^{O(\log n)}$ for the problem of st-conn over G with $|E| = n$, which is known to exist [3]. Now, for $ext - st - conn$, we run the scheme of the standard variant for graph G, and give P_i the shares for all $e \in E_i$. The resulting share complexity is $ss_{ext} \leq |E_i| ss_{ext} \leq n \cdot n^{O(\log n)} = n^{O(\log n)}$. □

3 Our main result

Given a monotone function $f : \{0,1\}^* \to \{0,1\}$, we define a certain type of infinite branching programs (BP's) for evaluating the function.

Definition 3.1 *Let $f : S \to \{0,1\}$ denote a function corresponding to an evolving access structure $\mathcal{A}$, which is not identically 1. An inifinte monotone BP for evaluating f is an infinite directed acyclic graph (DAG) $G = (V = \mathbb{N}^+ \cup \{s,t\}, E)$, where s, t is a pair of designated source and sink nodes respectively.[4] Also $V/\{t\} = \cup_{i=0}^{\infty} V_i$, where $V_0 = \{s\}$, where the V_i's are all finite and disjoint. Every layer V_i has an associated a pair of indices $p_i < q_i$, so that $p_0 = 1$, and $p_{i+1} = q_i + 1$. We refer to the range (p_i, q_i) as* generation $i + 1$. *Every vertex $v \in V_i$ has a set of outgoing edges $E_v \subseteq V_{i+1} \cup \{t\}$. Every edge (v, u) is labeled by a monotone function $g_{v,u} : \{0,1\}^{q_i - p_i + 1} \to \{0,1\}$. For technical reasons, edges of the form $e = (v, t)$ are never labeled by $g_e \equiv 1$ [5]. For a vector $x \in S$, and $v \in V_i$, the evaluation of an edge (v, u) at x, $val(v, u, x)$ equals $g_{v,u}(x_{p_i}, x_{p_i+1}, \ldots, x_{q_i})$. Define a subgraph $G_x = (V, E_x)$ of G, where $E_x = \{(v, u) | val(v, u, x) = 1\}$. The function f_G computed by G maps x to 1 iff. there is a path between s and t in G_x.*

Throughout the paper, we refer to BP's as in Definition 3.1 as an "infinite monotone BP", or simply BP, when there is no confusion. We say a BP is simple, if $q_i = p_i + 1$ for all $i \geq 0$. In other words, every node is labeled by a single variable, as in the standard definition of BP. A bound on the width of a BP $w : \mathbb{N} \to \mathbb{N}$ is a functions satisfying $|V_i| \leq w(i)$. We say an *infinite monotone BP* G is an *infinite monotone tree*, if the undirected graph obtained from $G \setminus \{t\}$ by ignoring edge directions is cycle-free.[6]

By monotonicity of the $g_{u,v}$'s, infinite monotone BP's indeed compute only monotone functions $f : S \to \{0,1\}$.

A starting point - the work of [4]. Consider the simple infinite monotone tree where each node in layer V_i is labeled by a single variable x_i and has two outgoing edges labeled by "0" and "1" respectively, where some nodes output "1" and the rest output "0" (closer to the standard model of decision trees). The "1" nodes have no child nodes. Casting such trees in our framework, the tree is a *simple* one. The 0-edges are labeled by the function $g(x_i) \equiv 1$ (capturing the fact that 0-edges for monotone function are "for free"), and 1-edges are labeled by the function $g(x_i) = x_i$. Nodes labeled with 1 are all "collapsed" into t, and erased from the tree (in line with the requirement that no $g_{(v,t)}$ can equal 1).

The above tree $G_{\mathcal{A}}(V, E)$ is used in [4] for their secret sharing scheme for general evolving access structures ([4],theorem 3.1). Essentially (with an additional small optimization we don't get into here), they apply (a generalization) of the scheme for evolving ust-conn from Claim 2.1, $\mathcal{U}$ to G.

[4] The current definition corresponds to a variant of such trees which are oblivious, ordered, and read once. Possibly, more general definitions could be useful.

[5] All edges may be labeled by $g_{(v,u)} \equiv 0$. We could as well not include edges labeled by $g_{(v,u)} \equiv 0$, but this makes the definition a bit simpler.

[6] Technically, as G need not be connected, the resulting graph may be a forest, but it won't matter to us.

This access structure generalizes evolving ust-conn in the natural way, with $m : \mathbb{N} \to 2^E$ mapping only to *finite* sets. The assignment function $m(i)$ outputs $E_i = \{(v,u) \in E | v \in V_{i-1}\} \cup \{(v,u) | v \in V_j$ where $j \leq i, g_{(v,u)} \equiv 1\}$. It is a subset of edges in G_{e_i} (where e_i is the characteristic vector of $\{i\}$).

It is crucial here that all E_i are finite, to achieve finite share complexity. Finiteness is achieved by the small trick of not giving P_i edges with $g_{(v,u)} \equiv 1$ that lie beyond layer V_{i-1}, which does not hinder correctness. It is easy to see that indeed x is in $ext - ust - conn_{G,m}$ iff. $G'_x = (V, \bigcup_{\{i|x_i=1\}} m(i))$ has an s-t path for all $x \in \{0,1\}^*$. Here G'_x is an undirected graph (so, edge directions of the $m(i)$'s are ignored). The (almost) tree structure of G, allows us to model acceptance by T by (evolving) *ust-conn* rather than by (evolving) st-conn.

A modification of the scheme $\mathcal{U}$ in Claim 2.1 working for the extended variant, proceed as follows. Upon arrival of P_i, it gets all the (sub)shares for edges contained in previous shares ($e \in E_j$ for some $j < i$). New edges (v,u) it should receive, are assigned new shares generated as in the non-extended variant $\mathcal{U}$.

The share complexity of the resulting scheme is $O(2^i)$. Major gains can be sometimes made when the complete, binary sub-tree of height i gets "pruned" due to strings $x \in 0,1^i$ for which $f_{\mathcal{A}}(x) = 1$. For instance, consider the evolving k-threshold scheme, where k is constant. Then, in the corresponding tree, $\binom{i}{k-1}$ edges (v,u) exit V_{i-1}, with at most i edges leading to each such v. Thus, the share complexity of the scheme satisfies $sc_i \leq \binom{i}{k-1}(1+i) = O(i^k)$. As another example, the scheme for evolving majority resulting from [4]'s general scheme, P_i receives shares for at least $\Omega(2^{n/2})$ edges (even with the optimization in [4]).

Our Construction. Our goal is to devise an approach for constructing schemes with better share complexity for interesting classes of functions (while general share complexity remains exponential). In particular, we obtain polynomial share complexity for a class containing evolving majority. The idea is very simple: [4] have simple $g_{(v,u)}$'s (1 or x_i for some i) with very small share complexity (1) per edge. On the other hand, the decision tree itself (when truncated up to the layer including some party P_i) is large.

Our idea is to decrease the size of the tree, at the expense of labeling each layer with a large sequence of bits, such that the degree of each vertex is relatively small. As a result, the share complexity of the $g_{(u,v)}$'s, each now depending on $q_i - p_i + 1$ bits, increases. We still use the scheme $\mathcal{U}$ for ust-conn in Claim 2.1 as the high level evolving scheme at the basis of our construction, but we need to further generalize the access structure of ext-ust-conn. By the special structure of the tree used in [4]'s construction above, we could indeed say that individual parties hold certain subsets of the edges, and model the problem as a ext-ust-conn problem.

For more general trees, we have a mapping between sets of parties that are allowed to learn certain sets of edges - according to the various $g_{(v,u)}$'s. The idea is to share the resulting shares $s_{(v,u)}$ among the parties which satisfy $g_{(v,u)} = 1$. Following are details of the basic construction.

Lemma 3.1 *Let $\mathcal{A}$ denote an evolving access structure where $f_{\mathcal{A}} \not\equiv 1$. Then the scheme in Figure 1 is a correct evolving secret sharing scheme for $\mathcal{A}$.*

Proof Sketch. The lemma follows straightforwardly from the construction and our definition of infinite monotone trees. That is, the parties learn the shares corresponding to edges in G_x, iff. $x \in \mathcal{A}$. To see this, we make several observations.

Observation 1 *G_x has a directed s-t path iff. the underlying undirected graph G'_x has an s-t path.*

This is the case because $G \setminus \{t\}$ is a tree, so a simple s-t path in G'_x is in fact a path in G_x.

Next, we observe that the parties in x learn only the $\mathcal{U}$-shares $s_{(v,u)}$ where $g_{(v,u)}(x_{p_{gen(v)}}, \ldots, x_{q_{gen(v)}}) = 1$, namely those in G_x. Otherwise, they learn nothing about the share. This is by correctness of the (standard) schemes for the $g_{(v,u)}$'s. Also, by construction, some of the shares s_e for $e \in G_x$ are not learned by x. Namely, let l_x denote the last index of x at which x equals 1. The parties in x learn shares for all of the edges in G_x except for those with $g_{(v,u)} \equiv 1$ where $v \in V_j$ for $j \geq gen(l_x)$. Clearly, if $x \notin \mathcal{A}$, then G_x does not contain an s-t path, the set of edges for which the parties can reconstruct the $\mathcal{U}$-shares is a subset of G_x, so it definitely does not contain an s-t path. If $x \in \mathcal{A}$, we need to show that the subset of $s_{v,u}$'s learned contains an s-t path. By construction, in this case G'_x (and thus G_x) has an s-t path. Removing the edges above from G_x still leaves an s-t path. Assume the contrary, and consider a simple s-t path in G_x. As edges labeled by $g_{(v,t)}$ can not be

- Let $\mathcal{A}$ denote an evolving access structure where $f_{\mathcal{A}}$ is not identically 1. Let T denote a tree for $\mathcal{A}$.[a] Let m order the tree edges in some order where n edge (v_i, u) always preceeds an edge (v_{i+1}, w) (that is, by order of tree layers, starting from s). Let $\mathcal{U}$ denote the evolving ust-conn scheme for T (as in Claim 2.1) for (G, m).
- The parties are partitioned into generations as in T.
- $\mathrm{Sh}(\mathrm{s}_1, \ldots, \mathrm{s}_{\mathrm{i}-1})$:
  1. Assume $P_i = P_{p_g}$ is the first party in generation g in T. Assign new shares to all edges outgoing a vertex in layer V_g according to $\mathcal{U}$ (by the order in m). Use previous $\mathcal{U}$-shares from previous s_j's for that matter. For $v \in V_g$, let s_{v,u_j} denote the share assigned to edge $\{v, u_j\}$. Share s_{v,u_j} among $P_i, \ldots, P_{q_g}$ according to some standard secret sharing scheme for g_{v,u_j}. Give P_i each of its shares s^i_{v,u_j} as part of s^i. The shares of the following parties in generation g are stored by Sh for future use. [b]
  2. Otherwise, P_i is not the first party in its generation g. It gets the shares for all edges (v, u) for $v \in V_g$, generated when the first party in generation g arrived, and stored in $s^i_{(v,u)}$.
  3. In any case, for j with $g_{v,u} \equiv 1$, where $v \in V_{g'}$ for some $g' < g$, $s_{(v,u)}$ is stored as part of s_i. [c]
- $\mathrm{Recon}(x, \overline{s}_x)$: Here $\overline{s}_x$ is a vector of shares held by the (finite) set of parties represented by $x \in \{0,1\}^*$. If there exists a path $p = (v_0 = s, v_1, v_2, \ldots, v_\ell, v_{\ell+1} = t)$ in G_x, recover all shares $s_{v_i,v_{i+1}}$, using the $g(v_i, v_{i+1})$-shares $s^j_{v_i,v_{i+1}}$ held by the parties.

[a] Such a tree always exists, the simple tree used in [4] for example.

[b] Again, for convenience of presentation, the dealer stores the shares for the other parties for future iterations. We could as well resample based on previous parties' shares.

[c] We could also modify $\mathcal{U}$ a little, to make $b_{u_j} = b_v$ in case g_{v,u_j} is identically 1, as done in [4]. This way, nothing would need to be given to any party for that node, as the share s_{v,u_j} is always 0. However, that would make the construction slightly less modular.

Figure 1: Main construction

the constant 1 function, the last edge (v, t) on the path satisfies $g_{(v,t)}(x) = 1$, and thus it must be the case that $gen(v) \leq gen(l_x)$. By structure of G, the path is of the form $\overline{p} = (v_0, \ldots, v_k = v, t)$, where each v_i is in layer V_i is an s-t path in G_x. The set x holds the shares of all of the edges in $\overline{p}$, as all of these edges exit levels $g' \leq i - 1$. □

We derive our main theorem by constructing trees resulting in $e^{n^{o(1)}}$ share complexity of Construction 1 in case $f_{\mathcal{A}}$ has branching programs of sufficiently small $e^{n^{o(1)}}$ size (e.g $e^{O(n^{0.24})}$). Our result is in fact stronger, providing a bound on achievable share complexity based on a bound on the width of a BP computing $f_{\mathcal{A}}$.

Theorem 3.2 *(Main theorem) Let $\mathcal{A}$ denote an evolving access structure, and $B(V^B, E^B, s_B, t_B)$ denote a simple monotone infinite BP computing $\mathcal{A}$. Let $w(i)$ denote a bound on the width of B (assume w is a monotone function). Then $\mathcal{A}$ has an evolving secret sharing scheme with share complexity $max((w(i^2)i)^{O(\log w(i^2) + \log i)}$.*

Proof sketch We construct a suitable infinite monotone tree G evaluating $\mathcal{A}$, and plug it into Construction 1. The main idea is to replicate layer $V^B_{q_i}$ as the set of children of every vertex (u, z) in layer V_i in our graph. Here z is the name of the vertex in $V^B_{q_{i-1}}$ that (u, z) replicates. details follow.

Lemma 3.3 *Assume $f_{\mathcal{A}} \not\equiv$ is an evolving access structure. Then the tree on Figure 2 correctly evaluates $f_{\mathcal{A}}$.*

Now, an edge from (u, z) to $((u, z), w)$ evaluates to 1 iff. $(x_{p_i}, \ldots, x_{q_i})$ leads from z to w in $(V^B, E^B_{0^{p_i - 1}x})$. Edges to t are handled similarly. Details follow in Figure 2.

- Let $\mathcal{A}$ denote a non-trivial evolving access structure. We construct an infinite monotone tree G evaluating it. Set $q_i = 2^{2^i}$ (this defines the sequence of p_i's as well).
- We define G recursively. Namely, we define a sequence $G_0 \subseteq G_1 \subseteq G_2 \ldots$ of (finite) graphs and set $G = (\bigcup_{i=0}^{\infty} V_i, \bigcup_{i=0}^{\infty} E_i)$ (that is, we add a tree layer V_i every step).
- Set $V_0 = \{(\perp, s), t\}$, and $G_0 = (V_0, \phi)$.
- Assume we have defined G_j for all $j < i$. To define G_i, we add the layer V_i, and all edges leaving V_{i-1} to G_{i-1}. For each $v = (u, z) \in V_{i-1}$:
  1. For each $w \in V^B_{q_i}$:
     - Consider the $ext - st - conn$ function $f_{v,w}$ on the graph G_v with $V_v = (\bigcup_{i=p_i-1}^{q_i}) V^B_i \cup \{t_B\} \subseteq V^B$, and E_v the set of edges induced in B by V_v. The s, t nodes are z, w respectively. The access structure is defined on $q_i - p_i + 1$ parties. Set $m(i) = E^B_{0^{p_{i-1}-1}{}_x} \cap E_v$. For each $w \in V^B_{q_i}$.
     - Add a vertex $u = (v, w)$ to V_i. Label the edge (v, w) by $g_{(v,w)} = f_{v,w}(x)$.
  2. Define f'_v similarly to the $f_{v,w}$'s above with the only exception that the t vertex now equals t_B. Add an edge (v, t) to E_i labeled by f'_v.

Figure 2: Tree construction

As to complexity, the main point is that $s - t$ connectivity has quite low share complexity. As we manage to create a rather small tree, and the complexity of evaluating the shares for various edges reduces to $ext - st - conn$ in a small graph when w is small, the increase in share complexity per edge is out weighted by the reduction in the number of edges in the (sub)tree relevant to P_i in Construction 1.

Let us make to precise calculations. First, we estimate the size of the subtree for which P_i may receive a share for an edge. Namely, it consists of the subgraph induced by $\bigcup_{j=0}^{gen(i)+1} V_j$. That is, $G_{gen(i)+1}$, as in the tree construction above. It is easy to see that the number of edges in the tree is bounded by

$$|E_i| = \Pi_{j=1}^{gen(i)} |V_j| = \Pi_{j=1}^{gen(i)} (|V_j| + 1) \leq |w(q_j) + 1|.$$

Note that $w(q_{gen(i)}) \leq w(i^2)$, as for our choice of parameters, $q_{gen(i)} = p^2_{gen(i)} \leq i^2$. Thus, it is not hard to see that

$$|E_i| \leq \Pi_{j=0}^{\infty} w(i^{2^{1-j}}) + 1 = (w(i^2))^{O(\log i)}. \tag{1}$$

Next, let us bound the share size given to P_i for a single edge. At worst, P_i receives a share for $ext - st - conn$ on a graph with $\leq |w(i^2)i^2 + 1|$ vertices (by monotonicity of w), and thus contains as most $O(w^2(i^2)i^4)$ edges. By Claim 2.2, the share complexity per edge is

$$(w(i^2)i)^{O(\log i + \log w(i^2))} \tag{2}$$

Multiplying the two, we obtain a bound of $(w(i^2)i)^{\log i + \log w(i^2)}$ on the resulting share complexity.

4 A further improvement for "counting" access structures

We have obtained a scheme with $e^{n^{o(1)}}$ share complexity for access structures that posses efficient BP's. More precisely, access structures with simple monotone infinite BP's with $w(i) = e^{O(n^c)}$ for $c < 0.25$ results in $e^{n^{o(1)}}$ share complexity. In particular, the evolving majority function has a (simple) BP with width $w(n) = n + 1$. Thus, Theorem 3.2 yields share complexity $n^{O(\log n)}$, which is a substantial improvement over $\Omega(2^{n^{0.5}})$ when simple trees are used, as in [4]. However, it turns out that for functions based on "counting" like evolving majority we can further improve the share complexity to $poly(n)$.

We define the class of *evolving counting* access structures as the set of non trivial monotone functions $f : S \to \{0, 1\}$ for which there exists a function $f_{cnt} : \mathbb{N} \times \mathbb{N} \times \{0, 1\} \to \{0, 1\}$ such that for all $x = (x_1, \ldots, x_t) \in \{0, 1\}^+$ for which $f(x) = 0$, $f(x) = f_{cnt}(|x|, \#_1(x_1, \ldots, x_{t-1}), x_t)$.

Theorem 4.1 *Let $\mathcal{A}$ be an evolving* counting *access structure. Then it has share complexity of $\tilde{O}(n^6)$.*

The evolving majority function is of the type above, with $f_{cnt}(t,k,b) = 1$ iff. $2(k+b) > t+1$.

Corollary 4.2 *The evolving majority access structure has share complexity of $\tilde{O}(n^6)$.*

Proof Sketch of Theorem 4.1 Clearly, for this type of functions in each layer of a simple BP B, we need to store only the number of 1's encountered so far. That is, layer V_i^B contains vertices from $\{v_0^i, \ldots, v_i^i\}$ where the semantics of v_j^i is that G_x contains an s-v_j path iff. $\#_1(x_1, \ldots, x_{q_i}) \geq j$. An edge (v_j^i, v_k^{i+1}) is thus marked by 1 if $k \leq j$, by x_{i+1} if $k = j+1$ and 0 otherwise. Furthermore, v_k^{i+1} is included in V_{i+1}^B iff. $v_k^i \in V_i^B$ or $v_{k-1}^i \in V_i^B$ and $f_{cnt}(i+1, k-1, 1) = 0$. This is required because we do not want to include a vertex for which $g_{(v,t)} \equiv 1$. Also, for each $v_k^i \in V_i^B$, we add an edge (v,t) iff. $f_{cnt}(i+1, k, 1) = 1$. It is not hard to see that B is indeed a monotone infinite BP computing $\mathcal{A}$. See the full version for a proof.

Now, consider the proof of Theorem 3.2. It proceeds by constructing a tree G (Construction 2) for $\mathcal{A}$ based on B, and plugging it into Construction 1. In this construction, each party receives a share of a (fresh) sharing for edge (v,u) according to a secret sharing scheme for $g_{(v,u)}$ (in G) among $q_i - p_i + 1$ parties. By construction, each party P_i gets a share for at most $|E_i|$ edges, where $|E_i|$ is bounded as in Equation 1. As $w(i) \leq i+1$ is a bound on the width of B, we get $|E_i = O(i^4)|$.

Now, in general we bound the share size of every edge according to Equation 2. Here we used a general bound on ext-st-conn, as the $g_{(v,u)}$ can always be cast as an instance of this problem (on at most i^2 parties, and graphs of size $\leq i^4 w^2(i^2)$ edges). However, for our particular BP, we could do much better. Consider $e = ((w, v_i), ((w, v_i), v_j))$ in the tree Construction 2, where $(w, v_i) \in V_l$. If $j \leq i$, $g_e \equiv 1$. Otherwise $g_e(x) = 1$ iff. $\#_1(x) \geq j - i$. For this goal, we can use Shamir's scheme [5] for $(j-i, q_l - p_l + 1)$-threshold. As $q_l - p_l + 1 \leq i^2$ (to be relevant to P_i), the resulting share complexity is $O(\log i)$. For edges of the form $e = ((w, v_i), ((w, v_i), t_B))$, the predicate is of the form $\bigvee_{j=1}^{q_l - p_l + 1}(\#_1(x_1, \ldots, x_j) \geq k_j) \wedge x_j$. This type of secret sharing can use a variant of the ISN scheme, for a monotone formula where leaves are not only variables x_k, but can also be threshold functions. The only difference is that random values for the leafs corresponding to threshold functions are shares among the parties according to $(k_j, q_l - p_l + 1)$ threshold scheme (rather than given to the corresponding party) - see the full version for a detailed construction. Overall, the resulting share complexity of a relevant (v,t) edge in G is $O(i^2 \log i)$. We conclude that the share complexity of our scheme is $\tilde{O}(i^6)$, as required. □

5 Future work

We have obtained general schemes for evolving secret sharing schemes with low BP complexity for BP's where all layers are labeled by consecutive, non-intersecting ranges of variables. Can we sometimes do better for protocols based on more general classes of BP's? How about BP's that read each variable more than once?

One simple modification of our scheme that could work for some schemes is playing with the size of a generation.

References

[1] Josh Cohen Benaloh and Steven Rudich. *ideal secret sharing for undirected st-conn.* unpublished. 1989.

[2] Mitsuru Ito, Akira Saito, and Takao Nishizeki. "Multiple assignment scheme for secret sharing". In: *Journal of Cryptology* 6 (1993), pp. 115–20.

[3] Mauricio Karchmer and Avi Wigderson. "Monotone Circuits for Connectivity Require Superlogarithmic Depth". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by Janos Simon. ACM, 1988, pp. 539–550. ISBN: 0-89791-264-0. DOI: 10.1145/62212.62265. URL: http://doi.acm.org/10.1145/62212.62265.

[4] Ilan Komargodski, Moni Naor, and Eylon Yogev. "How to Share a Secret, Infinitely". In: *Electronic Colloquium on Computational Complexity (ECCC)* 23 (2016), p. 23. URL: http://eccc.hpi-web.de/report/2016/023.

[5] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.