Hierarchical In-Network Attribute Compression via Importance Sampling

Arlei Silva [#], Petko Bogdanov [*], Ambuj K. Singh [#]

[#] *Computer Science Department, University of California, Santa Barbara, CA, USA*
{arlei,ambuj}@cs.ucsb.edu
[*] *Computer Science Department, University at Albany, SUNY, NY, USA*
pbogdanov@albany.edu

***Abstract*—Many real-world complex systems can be modeled as dynamic networks with real-valued vertex/edge attributes. Examples include users' opinions in social networks and average speeds in a road system. When managing these large dynamic networks, compressing attribute values becomes a key requirement, since it enables the answering of attribute-based queries regarding a node/edge or network region based on a compact representation of the data. To address this problem, we introduce a lossy network compression scheme called *Slice Tree (ST)*, which partitions a network into smooth regions with respect to node/edge values and compresses each value as the average of its region. ST applies a compact representation for network partitions, called *slices*, that are defined as a center node and radius distance. We propose an importance sampling algorithm to efficiently prune the search space of candidate slices in the ST construction by biasing the sampling process towards the node values that most affect the compression error. The effectiveness of ST in terms of compression error, compression rate, and running time is demonstrated using synthetic and real datasets. ST scales to million-node instances and removes up to 87% of the error in attribute values with a 10^3 compression ratio. We also illustrate how ST captures relevant phenomena in real networks, such as research collaboration patterns and traffic congestions.**

I. Introduction

Managing large dynamic networks that represent data-rich complex systems is a challenge in terms of processing time, communication, and storage. In social networks, for instance, users' opinions regarding different topics along time generate a large amount of data that describes the network opinion dynamics. Similarly, sensors spread over a traffic network provide massive, real-time data about the traffic conditions across multiple regions (see Figure 1). In these scenarios, each node of the network has an attribute (e.g., opinion, speed) and these attributes describe properties of a subnetwork (e.g. a community's opinion on a topic) and the overall network, such as major traffic jams affecting several neighborhoods of a city. Therefore, compressing vertex/edge attributes is a fundamental problem in the management of these large dynamic networks.

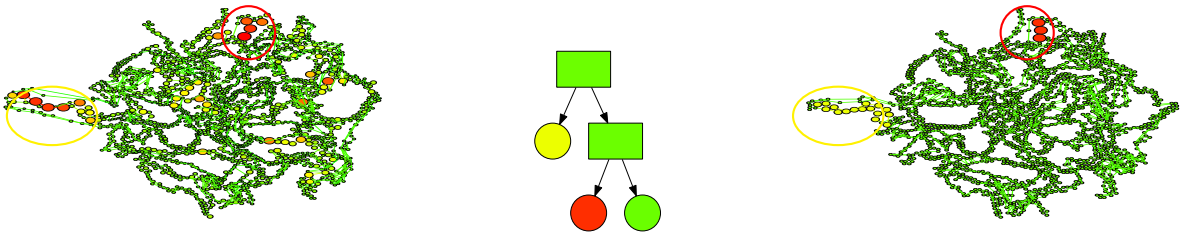
An important step towards compressing real-world network attributes is understanding the processes that generate attribute values. In social networks, opinion dynamics is affected by word-of-mouth dissemination [1]. Similarly, bad traffic conditions in a given location are likely to affect nearby locations. These are examples of network processes, which change node/edge values through connections in the network [2]. Network processes arise in many real domains, thus we can exploit them to better compress network attributes.

We can view network attributes as a signal over the network, where each node/edge has a value. Signals generated by network processes are expected to be smooth with respect to the network structure (i.e. values are similar at neighboring nodes) [3]. As a consequence, attributes can be expressed in terms of smooth regions, which are network regions with similar values. Examples of smooth regions in real networks include communities whose members share a similar opinion and neighborhoods affected by the same traffic conditions. In this paper, we study the problem of compressing network attributes by decomposing the network into smooth regions.

While there is a rich body of research on compressing network topologies [4], [5], compressing network attributes is an emerging problem [3]. In a traffic network, for instance, the structure is mostly fixed but traffic conditions change along time due to seasonal patterns (e.g. rush hours) and events (e.g. accidents). In this context, the compressed data provides, in a compact form, key information for answering queries regarding the traffic conditions in a particular location or region. In a hypothetical case where traffic conditions are random with respect to the network, accurately answering such queries would require keeping a large fraction of node values. However, if conditions are smooth over the structure, we can exploit such locality and represent network attributes in terms of average values of few regions (e.g., streets, neighborhoods).

We propose a network compression strategy called *Slice Tree (ST)*, which is a hierarchical decomposition of a snapshot of the network in terms of smooth regions. ST regions are represented in a compact form as a *center* and *radius*. A *slice* partitions a set of nodes into two regions: one composed of nodes within *radius* distance from a *center* node and the other containing the remaining nodes. An ST is constructed by recursive applications of such operations. Leaves of an ST define regions whose values are approximated as their average. To enable the reconstruction of regions, ST intermediate nodes retain information about each slice applied in the decomposition. Since ST is a lossy compression, we compute its error as the sum of squared errors (SSE) of the recovered values with respect to the original node values in the network.

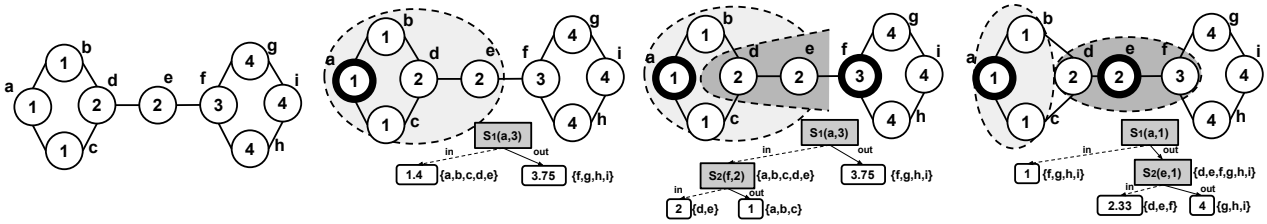
Figure 1 shows the ST compression of a real traffic network



(a) Traffic network (b) Slice tree (c) Reconstructed state values

Fig. 1: Slice Tree (ST) compression of a real traffic network (a). We set colors and sizes of nodes according to the average speed in the locations (large/red: very slow, medium/yellow: slow, small/green: fast). The network attributes are compressed as an ST with 2 slices (b) that decomposes the network into three smooth regions. The reconstruction of speed values (c) demonstrates that the compression captures the two major congestions in the network.

.



(a) Network, $SSE = 14.22$ (b) Slice 1, $SSE = 1.95$ (c) Slice 2, $SSE = 0.75$ (d) Opt. ST, $SSE = 0.65$

Fig. 2: An illustrative example of the ST compression. A network with node values and error (SSE) w.r.t. the average (a). First (b) and second (c) slice STs with respective errors. A slice divides nodes into two regions, one with nodes within *radius* distance from a *center* node and other with the remaining nodes. STs are constructed by choosing slices recursively and leaves of the ST define regions. Node values are compressed as the average of their region. Optimal ST with 2 slices (d).

(Figure 1a), where node values represent average speeds. Node colors and sizes emphasize low-speed locations. An ST (Figure 1b) decomposes the network into two congested regions and one non-congested region. Reconstructed speed values from the ST (Figure 1c) show that it captures the two major congestions (see more details in Section VII-D).

Figure 2 illustrates the ST compression of an example network (Figure 2a) with one (Figure 2b) and two slices (Figures 2c and 2d). We define the error of the network (14.22) w.r.t. its average (2.44). The slice S_1 (Figure 2b), which is centered at node a and has radius 3, separates the nodes into regions $\{a, b, c, d, e\}$ and $\{f, g, h, i\}$ with averages 1.4 and 3.75, respectively. The error of the resulting ST (1.9) is the total SSE of the region values w.r.t. the original node values. The ST given in Figure 2b can be extended by a new slice S_2 (center f and radius 2) that separates $\{a, b, c, d, e\}$ into two new regions $\{d, e\}$ (average 2) and $\{a, b, c\}$ (average 1). Figure 2d shows an alternative ST with 2 slices which is optimal, i.e. it minimizes the SSE of the region values.

Computing an optimal ST with k slices is an NP-hard problem. We propose a greedy scheme for ST construction which recursively selects the next slice that minimizes the compression error. While this solution achieves good results in practice, it is prohibitive for large networks due to the cost of searching over the set of possible slices. Therefore, we devise an efficient probabilistic approximation algorithm that prunes suboptimal slices with theoretical guarantees. In particular, we design an *importance sampling* strategy that biases the sampling process towards nodes for which the values are more likely to affect the error of the compression. We show that importance sampling requires less samples than uniform sampling for computing slices with the same error guarantees. Our main contributions are outlined as follows:

- We introduce Slice Tree as a novel network attribute compression strategy, show that computing an ST that minimizes the compression error is NP-hard and present a greedy heuristic for ST construction.
- We propose an importance sampling algorithm, with associated approximation guarantees, which enables the application of ST to million-node networks by effectively pruning the search space of candidate slices.
- We show that ST is an effective compression scheme, removing up to 87% of the error in node values with a 10^3 compression ratio. Case studies illustrate how STs model relevant properties of real networks, such as collaboration patterns and traffic congestions.

II. Related Work

The majority of existing work on network summarization and compression focuses on providing compact representations of a network structure [4], [5]. Prior work on network compression has proposed information-theoretic approaches for encoding structural data [4], [6]. More recent studies applied OLAP-like node/edge aggregations to networks, providing a multi-resolution summarization of nodes and their relationships [7], [5]. Our work is complementary to these efforts, since we focus on compressing attribute values in networks.

The compression of network attributes is related to a recent effort to generalize signal processing techniques to network data [8], [9], [3]. More specifically, although harmonic analysis has been successfully applied to problems that range from image compression [10] to query processing [11], applying these techniques to networks remains a challenge. A natural solution would be first to embed the network structure into an Euclidean space and then process the embedded values using existing signal processing approaches. However, Bourgain [12] has shown that such an embedding requires, in the worst case, a logarithmic number of dimensions w.r.t. to the size of the graph. An existing alternative, which is known as graph Fourier, constructs an orthonormal basis for a network using spectral analysis [13], [3]. The first eigenvectors of the network Laplacian matrix define a structure-aware basis that is expected to capture the smoothness of node values over the network.

Similarly to the traditional Fourier analysis in Euclidean spaces, graph Fourier is unable to localize signals in both time (space) and frequency domains [8], [14], [3]. Recent efforts attempt to address this limitation by generalizing the wavelet analysis [10] to networks. Examples of such efforts include graph wavelets [15], [14], diffusion wavelets [16], and Haar trees [8]. Similar to ST, graph wavelets [15] also average values within a radius distance from a center node. However, these wavelets are based on deviations in values on a disc and a surrounding ring, which is suitable for summarizing communication data but not for compressing network attributes under budget constraints. In [17], the authors propose a lossy compression of attributes over a grid structure. However, their approaches do not assume budget constraints and depend on either a rigid topology or the ordering of node identifiers. We show that ST achieves better compression accuracy than graph Fourier and Haar trees under a fixed budget. The main advantage of ST compared to these existing techniques is that ST performs an efficient value-sensitive search for slices that maximize the compression accuracy using sampling.

ST applies the network as a space where attributes are embedded and can be represented in a compact form using slices. Due to these properties, it differs from existing work on traditional clustering [18], which does not consider the network structure, and community detection based on attributes, which has no constraints on the representation cost [19]. Finally, while compression is a key problem in sensor networks, existing approaches [20] are focused on power consumption and routing, which is out of the scope of this work.

III. Problem Definition

We define a network as a triple $G(V, E, W)$, where V and E are the sets of vertices and edges, while W is a function over the vertices ($W : V \rightarrow \mathbb{R}$) that gives the vertex attributes. The value of a vertex v is denoted as $w(v)$ and $d(u, v)$ is the shortest path distance between u and v. The network compression problem is posed as follows.

Definition 1: **Network attribute compression.** Given a budget b and a network $G(V, E, W)$, compute a compression $\Gamma : V \rightarrow \mathbb{R}$, such that Γ can be encoded using b bits and $\Gamma(v)$ gives an approximate value for $w(v)$.

The function Γ is a lossy compression of W. We quantify the compression error of Γ as a sum of squared errors: $\sum_{v \in V}(w(v) - \Gamma(v))^2$. A good compression recovers the values given by W with small error. Moreover, Γ must be computed efficiently for large networks. While we define the problem in terms of SSE minimization and node attributes, our general framework can be adapted to other error metrics including those based on edge attributes. Also, we focus on the problem of compressing a single snapshot of the network at a time.

IV. Slice Tree

Our approach for the network attribute compression problem is Slice Tree (ST), which decomposes a fixed network structure into hierarchical regular regions that are smooth with respect to the attribute values. ST encodes information for reconstructing the regions and their averages, thus providing a compact representation for attribute values based on the network structure.

A slice divides a set of nodes into two subsets, one containing nodes within radius distance from a center node and the other composed of the remaining nodes in the set.

Definition 2: **Slice.** Given a network G, nodes $X \subseteq V$, a center $c \in V$, and a radius $r \geq 0$, a slice $s(c, r, X)$ partitions X into $P = \{u \in X | d(c, u) \leq r\}$ and $X \setminus P$.

Slices are compact representations for regular partitions. A slice encodes a separation of a set of nodes X into two subsets (P and $X \setminus P$) using only two parameters (center and radius). Figure 2b, shows a slice $S_1(a, 3, V)$ applied to the set of nodes from the network shown in Figure 2a. New slices can be applied to the resulting partitions P and $X \setminus P$, producing a hierarchical data structure. The sequence of slices is a compressed representation for the final regions in the ST.

Definition 3: **Slice Tree (ST).** A slice tree $ST(G, k)$ is a binary tree that encodes k recursive slices in G. The first slice is applied to V and subsequent slices are applied (recursively) to the resulting partitions (P and $X \setminus P$).

Leaves of an ST define partitions (or regions) in the network and the representation cost of an ST is linear in the number of slices k. To compress the network attribute values W, we assign to each leaf node of the ST a value $\mu(P)$ that is the average value of the nodes in P ($\mu(P) = 1/||P|| \sum_{v \in P} w(v)$). Values of nodes $v \in P_i$ are compressed as $\mu(P_i)$.

An ST with two slices (i.e. $k = 2$) for the network from Figure 2a is shown in Figure 2c. The partitions $\{a, b, c\}$, $\{d, e\}$, and $\{f, g, h, i\}$ are produced by the slices S_1 and S_2. We compress the value $w(f)$ as 3.75, since this is the

average $\mu(\{f,g,h,i\})$. The error of an ST is computed as a sum of squared errors with respect to the original values. For instance, the error of the ST in Figure 2c is 0.75. An optimal Slice Tree $ST^*(G,k)$ minimizes the compression error $\sum_{P\in\{P_1,\ldots,P_{k+1}\}}\sum_{v\in P}(w(v)-\mu(P))^2$. Figure 2d shows an optimal ST (SSE= 0.65) with two slices for the network shown in Figure 2a. Computing $ST^*(G,k)$ is NP-hard.

Theorem 1: **Optimal Slice Tree.** The optimal slice tree compression problem is NP-hard.

Proof: Let $VC(\mathcal{G}(\mathcal{V},\mathcal{E}),q)$ be an instance of the (NP-complete) *Vertex Cover* problem, which asks whether there exists a set of vertices $\mathcal{V}'\in\mathcal{V}$ such that $|\mathcal{V}'|=q$ and, for each edge $(u,v)\in\mathcal{E}$, $u\in\mathcal{V}'$ or $v\in\mathcal{V}'$ (or both). There is a corresponding instance $ST(G(V,E,W),2q)$ of the ST problem, where G is a 4-partite graph defined as follows. The set of nodes is $V=V^1\cup V^2\cup V^3\cup V^4$, where for each node $v_i\in\mathcal{V}$ we create nodes $v_i^1\in V^1$ and $v_i^2\in V^2$, and for each edge $e_j\in\mathcal{E}$ we create nodes $e_{j,1}^3,e_{j,2}^3\in V^3$ and $e_{j,1}^4,e_{j,2}^4\in V^4$. The set E contains edges (v_i^1,v_i^2) for all nodes $v_i\in\mathcal{V}$, $(e_{j,1}^3,e_{j,1}^4)$ and $(e_{j,2}^3,e_{j,2}^4)$ for all edges $e_j\in\mathcal{E}$, and $(v_i^2,e_{j,1}^3)$ for all edges e_j adjacent to node v_i in $\mathcal{G}$. W is such that $w(v_i^h)=1$ if $v_i^h\in V^3$ and $w(v_i^h)=0$, otherwise.

We show that $VC(\mathcal{G},q)$ is true iff $ST(G,2q)$ has 0-error. Given a vertex cover $\mathcal{V}'$, we generate a 0-error ST by placing a slice $s(v_i^1,2,X_i)$ followed by a slice $s(v_i^1,1,X_i')$ for every $v_i\in\mathcal{V}'$. The order in which these slices are placed and the sets X_i and X_i' involved do not matter. The resulting ST isolates nodes in V_3 from the remaining ones. Conversely, given a 0-error ST, we can generate a vertex cover by first applying some simple transformations to the ST and then selecting each vertex $v_i\in\mathcal{V}$ for which there is a slice $s(v_i^1,2,X)$ as part of the cover $\mathcal{V}'$. These transformations enforce all slices in the ST to be centered at nodes in V^1 and the cases to be considered are slices centered at nodes in V^2 and V^3. Slices centered in V^2 with radius 1 are replaced by slices centered in V^1 with radius 2. Moreover, each pair of slices separating nodes $e_{j,1}^3$ and $e_{j,2}^3$ from the rest are replaced by two slices with any center v_i^1 such that e_j is adjacent to v_i, one with radius 2 followed by another with radius 1. ■

Theorem 1 shows that finding the best ST from G with k slices might require searching over an $O(d(G)||V||^k)$ space of candidate STs, where $d(G)$ is the diameter of G. Candidate STs combine possible choices of centers and radii and are order-sensitive. In the rest of this paper, we devise efficient heuristics for computing accurate STs in large networks.

V. Fast Slice Tree Compression

We outline a greedy procedure that computes an ST by repeatedly selecting the slice that minimizes the compression error. To scale this solution to large networks, we introduce a probabilistic approximation algorithm that applies sampling and pruning in the ST construction.

A. Greedy Slice Tree Construction

While building an optimal ST is a computationally hard problem, a single optimal slice can be found in polynomial time by searching over all possible center-radius combinations. Therefore, a greedy algorithm, which builds an ST by selecting consecutive slices that minimize the compression error, is a natural approach for the ST problem. We can express the benefit of adding a slice to an ST as follows.

Definition 4: The **error reduction** $\phi(s)$ of a slice $s(c,r,X)$ is defined as:

$$\phi(s)=SSE(X)-SSE(P)-SSE(X\setminus P),$$

where P and $X\setminus P$ are the regions produced by the slice s and $SSE(Y)=\sum_{v\in Y}(w(v)-\mu(Y))^2$.

An important property of $\phi(s)$ is that it can be computed in terms of the mean values and the sizes of partitions X and P (or $X\setminus P$), as shown in Theorem 2.

Theorem 2: The error reduction $\phi(s)$ can be computed in terms of average values $\mu(X)$, $\mu(P)$ and $\mu(X\setminus P)$:

$$\begin{aligned}\phi(s)&=(\mu(X)-\mu(P))^2\frac{||P||||X||}{||X\setminus P||}\\&=(\mu(X)-\mu(X\setminus P))^2\frac{||X\setminus P||||X||}{||P||}\end{aligned}$$

Proof: The SSE of the average $\mu(Y)$ w.r.t. values $y\in Y$ can be expressed as:

$$\sum_{y\in Y}(y-\mu(Y))^2=\sum_{y\in Y}y^2-\frac{1}{||Y||}\left(\sum_{y\in Y}y\right)^2$$

Thus,

$$\begin{aligned}\phi(s)=&\sum_{v\in X}w(v)^2-\frac{1}{||X||}\left(\sum_{v\in X}w(v)\right)^2\\&-\sum_{v\in P}w(v)^2+\frac{1}{||P||}\left(\sum_{v\in P}w(v)\right)^2\\&-\sum_{v\in X\setminus P}w(v)^2+\frac{1}{||X\setminus P||}\left(\sum_{v\in X\setminus P}w(v)\right)^2\\=&-\frac{1}{||X||}\left(\sum_{v\in X}w(v)\right)^2+\frac{1}{||P||}\left(\sum_{v\in P}w(v)\right)^2\\&+\frac{1}{||X\setminus P||}\left(\sum_{v\in X\setminus P}w(v)\right)^2\\=&(\mu(X)-\mu(P))^2\frac{||P||||X||}{||X\setminus P||}\end{aligned}$$

We derive an expression w.r.t. $\mu(X\setminus P)$ by replacing $\mu(P)$ by $(\mu(X)||X||-\mu(X\setminus P)||X\setminus P||)/||P||$ in this equation. ■

Intuitively, a good slice produces new regions, P and $X\setminus P$, for which averages, $\mu(P)$ and $\mu(X\setminus P)$, deviate from the average value of X. Theorem 2 supports an efficient approach for computing the error reduction of concentric slices of increasing radii. Moreover, we apply it to compute sampling-based upper bounds on the error reduction of candidate slices, as described in Section V-B. Our greedy algorithm selects, at each step, the slice that maximizes the error reduction.

The *Slice* function (Algorithm 1), identifies the slice t with maximum error reduction $\phi(t)$ for a partition X. It

Algorithm 1: Slice

Require: Network G, set of nodes X
Ensure: Best slice t
1: **for** All centers $c \in X$ **do**
2: **for** All radii $r \in [0, radius(c, G, X)]$ **do**
3: $s \leftarrow (c, r, X)$
4: $\phi(s) \leftarrow (\mu(X) - \mu(P))^2 \frac{||P|| ||X||}{||X-P||}$
5: $t \leftarrow s$, if $\phi(s)$ is the largest observed reduction
6: **end for**
7: **end for**

Algorithm 2: Greedy Slice Tree

Require: Network G, budget k
Ensure: ST
1: Slice $ST \leftarrow$ empty Slice Tree
2: Candidate slices $C \leftarrow \emptyset$
3: Add best slice $s(c, r) \leftarrow Slice(G, V)$ to C
4: **while** Number of slices less than k **do**
5: Retrieve best slice $t(c, r)$ from C
6: Add $t(c, r)$ to ST
7: Let P and $X \setminus P$ be the partitions produced by $t(c, r)$
8: Add best slice $s(c, r) \leftarrow Slice(G, P)$ to C
9: Add best slice $s(c, r) \leftarrow Slice(G, X \setminus P)$ to C
10: **end while**

iterates over all possible centers and increasing radii up to a maximum $radius(c, G, X)$ that splits X into non-empty regions. A center-radius pair (c, r) defines a candidate slice s in X (step 3). The error reduction of a candidate slice s is computed according to Theorem 2 (step 4) and the slice that maximizes the error reduction is selected (step 5). Iterating over all possible radii for a given center c is equivalent to performing a breadth first search (BFS) starting from c.

Algorithm 2 describes our greedy strategy for computing an ST. It takes an input network G and a budget k and computes a slice tree $ST(G, k)$ by consecutively selecting the next slice with highest error reduction. The greedy ST algorithm proceeds in k iterations. In each iteration (steps 4-10), it selects the slice incurring highest error reduction, removes it from the priority queue C and inserts it in the corresponding branch of the ST (steps 4,5). Next, we compute the best slice for the new regions, P and $X \setminus P$, and add them to C (steps 8-9).

In our running example (Figure 2), the first slice S_1 (Figure 2b) selected by our greedy algorithm has center a, radius 3 and an error reduction of $\phi(S_1) = 12.27$ (the error of the network with respect to its global mean $\mu(V)$ is 14.22). The next best slice S_2 (Figure 2c) further divides the partition $\{a, b, c, d, e\}$ into $\{a, b, c\}$ and $\{d, e\}$ using center f and radius 3. The error reduction $\phi(S_2)$ of S_2 is 1.25. The following Lemma gives the complexity of our greedy ST algorithm.

Lemma 5.1: The greedy slice tree algorithm (Algorithm 2) runs in time $O(k||V||||E||)$.

Proof: A slice s over X produces regions X_1 and X_2. Let $E(X) = \{(v_i, v_j) \in E | v_i, v_j \in X\}$. *Slice* applies $||X||$ BFSs, in time $O(||E(X)||)$ each. In the worst case, s generates X_1 and X_2 such that $||X_1|| = ||E(X_1)|| = 1$, $||X_2|| = ||X|| - 1$ and $||E(X_2)|| = ||E(X)|| - 1$, resulting in a complexity:

$$O\left(\sum_{i=0}^{k-1}(||V|| - i)(O(||E|| - 1))\right) = O(k||V||||E||)$$

■

This complexity imposes a challenge to the application of our greedy algorithm to large networks. The ST from Figure 2c was built using the greedy algorithm described in this section and we show that it is not optimal. An optimal ST with two slices for the same network is shown in Figure 2d. While the proposed algorithm achieves good results in practice (see Section VII), whether there is a constant-factor polynomial approximation for the ST problem remains an open question.

B. Error Reduction via Sampling

The challenge in selecting the next best slice in ST construction stems from the need to explore all the node values at a given radius from a center to compute its error reduction. The idea behind our sampling schemes is that we do not need to observe all nodes within "good" slices to identify them, but instead can utilize a small sample of nodes. We propose two sampling schemes for computing upper bounds on the error reduction of slices: one based on uniform sampling and one based on importance sampling. The resulting bounds enable a scalable and accurate slice search algorithm.

As part of our *uniform sampling* approach, we generate a uniform node sample U with replacement from a set of nodes X. For a given slice $s(c, r, X)$, we estimate the average values of its regions $\mu(P)$ and $\mu(X \setminus P)$ based on values of nodes in $U \cap P$ and $U \cap (X \setminus P)$, respectively. We employ the Hoeffding inequality [21] to compute probabilistic errors for these estimates. The average estimates and their errors are then used to obtain probabilistic upper bounds on the error reduction of a slice according to Theorem 2.

In the uniform sampling scheme, nodes are sampled with equal probability regardless of whether they are part of a region with outlier values. Theorem 2 shows that high error reduction slices create regions P and $X \setminus P$ with values deviating from the average $\mu(X)$. To increase the likelihood of nodes from high error reduction regions to be sampled, we apply *importance sampling* [22]. Importance sampling is a statistical technique that biases the sampling procedure towards values that are more relevant for computing a given estimate. In our case, we bias the samples based on how much their values deviate from the average $\mu(X)$.

Definition 5: An **importance sample** from X is a multiset of nodes B where each node $v \in X$ is selected (with replacement) with probability:

$$p(v) = \frac{|w(v) - \mu(X)|}{\lambda}$$

where $\lambda = \sum_{v \in X} |w(v) - \mu(X)|$

The importance sample B is biased and hence, in order to obtain unbiased estimates for the average values in resulting regions $\mu(P)$ and $\mu(X \setminus P)$ we need to correct the sampled attribute values using a weighted average.

Definition 6: The **weighted average** $\mu(B_P)$ is defined as:

$$\mu(B_P) = \frac{1}{\Psi(B_P)} \sum_{v \in B_P} \frac{\lambda}{|w(v) - \mu(X)|} w(v)$$

where $B_P = B \cap P$ and $\Psi(B_P) = \sum_{v \in B_P} \frac{\lambda}{|w(v) - \mu(X)|}$.

We can compute a weighted average value $\mu(B_{X\setminus P})$ as an estimate of $\mu(X \setminus P)$ in a similar fashion. In the following lemma, we show that the weighting scheme produces an unbiased estimate for the average values of regions.

Lemma 5.2: If $\Psi(B_P)$ and $\sum_{v\in B_P} \frac{\lambda}{|w(v)-\mu(X)||} w(v)$ are independent, then $\mu(B_P)$ is an unbiased estimate for $\mu(P)$.

Proof: The expected value of $\mu(B_P)$ can be written as:

$$\mathbb{E}[\mu(B_P)] = \frac{1}{\mathbb{E}[\Psi(B_P)]}\mathbb{E}[\sum_{v\in B_P} \frac{\lambda}{|w(v)-\mu(X)|} w(v)]$$

Simplifying the two expectations:

$$\begin{aligned}\mathbb{E}[\Psi(B_P)] &= ||B_P||\mathbb{E}[\frac{\lambda}{|w(v)-\mu(X)|}] \\ &= ||B_P|| \sum_{v\in P} \frac{|w(v)-\mu(X)|}{\lambda} \times \frac{\lambda}{|w(v)-\mu(X)|} \\ &= ||B_P|||P||\end{aligned}$$

$$\begin{aligned}\mathbb{E}[\sum_{v\in B_P} \frac{\lambda}{|w(v)-\mu(X)||} w(v)] &= ||B_P||\mathbb{E}[\frac{\lambda}{|w(v)-\mu(X)|} w(v)] \\ &= ||B_P|| \sum_{v\in P} \frac{|w(v)-\mu(X)|}{\lambda} \times \frac{\lambda}{|w(v)-\mu(X)|} w(v) \\ &= ||B_P|| \sum_{v\in P} w(v)\end{aligned}$$

Combining the two:

$$\mathbb{E}[\mu(B_P)] = \frac{||B_P||}{||B_P||||P||} \sum_{v\in P} w(v) = \frac{1}{||P||} \sum_{v\in P} w(v) = \mu(P)$$

■

Lemma 5.2 provides an unbiased average estimate $\mu(B_P)$ that can be used as a uniform sampling-based estimate $\mu(U \cap P)$. In case the independence assumption does not hold, the covariance between the two variables must be considered. We omit this discussion here for simplicity. Next, we derive upper bounds on the error reduction of a slice based on unbiased estimates for region average values.

Theorem 3: Given a (uniform or importance) sample S, a confidence parameter δ ($0 \leq \delta < 1$) and a slice s, the error reduction $\phi(s)$ can be bounded as:

$$Pr[\phi(s) < \max\{\phi_1(s), \phi_2(s)\}] > 1 - \delta$$

where:

$$\begin{aligned}\phi_1(s) &= (\mu(X) - \mu(S_P) - \epsilon)^2 \frac{||P||||X||}{||X \setminus P||} \\ \phi_2(s) &= (\mu(X) - \mu(S_P) + \epsilon)^2 \frac{||P||||X||}{||X \setminus P||} \\ \epsilon &= \sqrt{\frac{-\theta^2}{2||S_P||} \log(\frac{\delta}{2})} \\ \theta &= \max_{u,v\in X} |w(u) - w(v)| \\ S_P &= S \cap P\end{aligned}$$

Proof: According to the Hoeffiding inequality [21]:

$$Pr[|\mu(S_P) - \mu(P)| \geq \epsilon] \leq \delta$$

where $\epsilon = \sqrt{\frac{-\theta^2}{2||S_P||} \log(\frac{\delta}{2})}$

Thus,

$$\begin{aligned}Pr[\phi(s) < \max\{\phi_1(s), \phi_2(s)\}] &= Pr[|\mu(X) - \mu(P)| < \\ &\quad \max\{|\mu(X) - \mu(P) - \epsilon|, |\mu(X) - \mu(P) + \epsilon|\}] \\ &= Pr[\mu(S_P) - \epsilon < \mu(P) < \mu(S_P) + \epsilon] > 1 - \delta\end{aligned}$$

■

A similar upper bound with the same confidence exists based on the samples from $X \setminus P$, and hence we can apply either of them with the same probabilistic guarantees. The theorem applies to both uniform and importance sampling as long as an unbiased estimate of the average is used.

While the expected number of sampled values $||U \cap P||$ from a region P is proportional to $||P||$ in a uniform sample U, regions that include many node values deviating from the average value $\mu(X)$ will be oversampled in the importance sample. For instance, consider the case of a small region P' with many values deviating from $\mu(X)$. Although a high error reduction slice might separate P' from X, only importance sampling is likely to produce enough samples from P' to induce the selection of such slice. In Theorem 4, we relate the size of $||P \cap B||$ to the error reduction $\phi(s)$ of the slice s, enabling a second upper bound for importance sampling.

Theorem 4: For an importance sample B, confidence δ and slice s, the error reduction $\phi(s)$ can be bounded as:

$$Pr[\phi(s) < \frac{(p' + \epsilon)^2\lambda^2}{||P||} \frac{||X||}{||X \setminus P||}] > 1 - \delta$$

where $p' = \frac{||P\cap B||}{||B||}$ and $\epsilon = \sqrt{\frac{-1}{2||B||} \log(\delta)}$.

Proof: The probability p of selecting a node from P is:

$$\begin{aligned}p = \sum_{v\in P} \frac{|w(v) - \mu(X)|}{\lambda} &\geq \frac{1}{\lambda}|\sum_{v\in P} w(v) - \mu(X)| \\ &= \frac{||P||||\mu(P) - \mu(X)|}{\lambda}\end{aligned}$$

The Hoeffiding inequality gives that:

$$Pr[p < p' + \epsilon] > 1 - \delta$$

Thus,

$$\begin{aligned}&Pr[p' + \epsilon > \frac{||P||||\mu(P) - \mu(X)|}{\lambda}] \\ &= Pr[|\mu(P) - \mu(X)| < \frac{(p' + \epsilon)\lambda}{||P||}] \\ &= Pr[\phi(s) < \frac{(p' + \epsilon)^2\lambda^2}{||P||} \frac{||X||}{||X - P||}] \qquad > 1 - \delta\end{aligned}$$

■

According to this bound, regions that get few samples will likely have a low error reduction. Also, different from Theorem 3, this bound does not depend on the range (θ) of node values in the network. This is a desired property when dealing with outliers and skewed value distributions.

In the next section, we introduce a sampling-based algorithm for identifying approximate optimal slices using uniform or importance sampling. This algorithm employs Theorems 3 and 4 to prune the search space of candidate slices.

Algorithm 3: Approx-Slice

Require: Network G, Set of nodes X, Confidence parameter δ, Approximation constant ρ, Sampling rate π
Ensure: Approximate best slice t

1: $L \leftarrow \emptyset$
2: **for** All centers $c \in X$ **do**
3: **for** All radii $r \in [0, radius(c, G, X)]$ **do**
4: $s \leftarrow (X, c, r)$
5: $L \leftarrow L \cup \{s\}$
6: **end for**
7: **end for**
8: $\phi(t) \leftarrow 0$
9: $S \leftarrow \emptyset$
10: **while** L is not empty **do**
11: Add $\lceil \pi.||X|| \rceil$ new samples from X to S
12: **for** All candidate slices $s \in L$ **do**
13: Compute $\phi_{max}(s), \phi'(s)$
14: **end for**
15: $q \leftarrow \max_{s \in L}\{\phi'(s)\}$
16: **if** $\phi'(q) \geq \phi(t)$ **then**
17: Compute $\phi(q)$
18: **if** $\phi(q) \geq \phi(t)$ **then**
19: $t \leftarrow q$
20: **end if**
21: **end if**
22: $L \leftarrow \{s \in L | \rho \times \phi_{max}(s) \geq \phi(t)\}$
23: **end while**

C. Approximate Slice Tree Construction

In Section V-A, we described a greedy algorithm for slice tree construction. It applies the *Slice* routine to choose the best slice in set of nodes X (see Algorithm 1). In order to speedup this exhaustive scheme, we next show how to find approximate optimal slices, with probabilistic guarantees using sampling.

Algorithm 3 (*Approx-Slice*) takes as input the network G, a set of nodes $X \subseteq V$, a confidence parameter δ $(0 < \delta \leq 1)$, an approximation constant ρ $(0 \leq \rho \leq 1)$, and a sampling rate π $(0 \leq \pi \leq 1)$. As its output, it returns a slice $t = (c, r, X)$ such that $c \in X$ and the error reduction $\phi(t)$ is at least $\rho.OPT$ with probability $1-\delta$, where OPT is the optimal error reduction for a slice in X. In particular, the algorithm finds the optimal slice with probability $1 - \delta$ if ρ is 1. We replace the previously defined *Slice* routine by *Approx-Slice* in Algorithm 2 after setting the additional parameters δ, ρ and π. This new version of our solution, which we call *Approximate Greedy Slice Tree*, returns an ST for which each slice is approximate optimal according to the given parameters.

Approx-Slice first generates the set of candidate slices L (steps 1-7) and then performs iterative sampling and pruning (steps 10-23) in the search for an approximate optimal slice. On each iteration, the set of samples S is increased by $\lceil \pi.||X|| \rceil$ samples from X (step 11). The sampling scheme applied can be either uniform sampling or importance sampling as long as proper upper bounds are considered. For each slice $s \in L$, we compute two values based only on the samples in S: an upper bound on the error reduction $\phi_{max}(s)$ and an estimate on the error reduction $\phi'(s)$ (step 13). The value of $\phi_{max}(s)$ is set according to Theorem 3 for uniform sampling and both Theorems 3 and 4 for importance sampling. Also, it follows from Definition 4 that the error reduction for any slice in X cannot be larger than the error of X ($SSE(X)$).

For a given slice s, we can estimate the error reduction $\phi(s)$ based on the estimate for the average attribute value of the regions P and $X \setminus P$ and by considering the number of samples in P, in case importance sampling is applied. We compute $\phi'(s)$ as the harmonic mean of these different estimates to enforce that the slice with highest $\phi'(s)$ has high error reduction estimates across different measures.

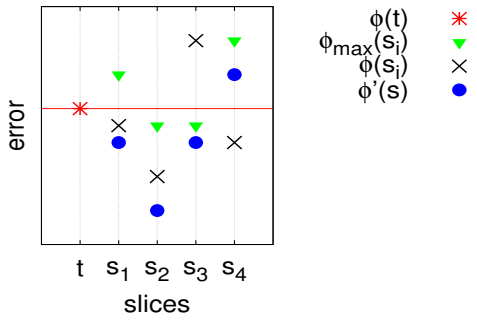


Fig. 3: Example of slice pruning: t is the current top slice and s_1-s_4 are candidate slices with their respective error reduction upper bounds $\phi_{max}(s_i)$, actual error reductions $\phi(s_i)$, and estimated error reductions $\phi'(s_i)$. Slices s_2 and s_3 are pruned and the next slice to be computed is s_4.

Figure 3 illustrates the *Approx-Slice* pruning strategy for ρ set to 1, a current best slice t and 4 candidate slices (s_1-s_4). The actual error reduction is known only for t ($\phi(t)$). For all candidate slices, the algorithm computes the error reduction upper bound $\phi_{max}(s_i)$ and the error reduction estimate $\phi'(s_i)$ based on the sample S. The actual error reductions for candidate slices ($\phi(s_i)$) are shown only for explanation. Slices s_2 and s_3 are pruned, since their upper bounds ($\phi_{max}(s_2)$ and $\phi_{max}(s_3)$, respectively) are lower than $\phi(t)$. Nevertheless, the actual error reduction of s_3 ($\phi(s_3)$) is greater than the error reduction of t, which means that s_3 should not be pruned. *Approx-Slice* is a probabilistic approximation algorithm in the sense that the probability of such an error is bounded by a small (user-defined) constant δ. The new set of candidate slices will contain only those slices with error reduction upper bound $\phi_{max}(s_i)$ greater than $\phi(t)$ (s_1 and s_4). Moreover, the slice s_4 is a candidate to replace t, since its error reduction estimate is greater then $\phi(t)$. After computing $\phi(s_4)$, the algorithm will not replace t because $\phi(s_4) < \phi(t)$.

In every iteration, new samples are added to S in order to improve the error reduction estimates and upper bounds of candidate slices until the candidate set L is empty. The execution time of *Approx-Slice* depends on how many iterations it takes to identify an approximate optimal slice. Some implementation decisions and associated complexity are discussed in Section VI. We show (see Section VII) that this algorithm can efficiently find a good slice s in X by pruning a large portion of the search space of candidate slices using a small sample S. It is important to notice that the constant factor approximation (ρ) given by our algorithm is for each single slice and not the resulting ST.

VI. Implementation details

This section covers some details of our implementation [1] of the the ST algorithm (Algorithm 2).

Estimating the sizes of regions: Theorems 3 and 4 depend on the sizes of regions P ($||P||$) and $X \setminus P$ ($||X \setminus P||$), which cannot be computed based only on a sample of nodes. Computing exact sizes of regions for a candidate slice $s(c, r, X)$ takes a prohibitive $O(||E||)$ time in the worst case, since it requires a BFS starting from c up to distance r. To avoid such a cost, we propose simple upper bounds $\lceil ||P|| \rceil$ and $\lceil ||X \setminus P|| \rceil$ and lower bounds $\lfloor ||P|| \rfloor$ and $\lfloor ||X \setminus P|| \rfloor$ on $||P||$ and $||X \setminus P||$, respectively, that depend on the network structure and are calculated in constant time based on a precomputed index. This index contains the number $z(v, r)$ of vertices at distance r from every vertex $v \in V$. For any slice s, $\lceil ||P|| \rceil = \max\{z(c, r), ||X||\}$ and $\lfloor ||X \setminus P|| \rfloor = \max\{||X|| - \lceil ||P|| \rceil, 0\}$. To compute the remaining bounds, we keep the maximum radius $r_f(v)$ for which a slice centered at v does not overlap with the border of any existing slice in the ST. We define $\lfloor ||P|| \rfloor = z(c, \min\{r, r_f(c)\})$ and $\lceil ||X \setminus P|| \rceil = \max\{||X|| - \lfloor ||P|| \rfloor, 1\}$. A region size is replaced by its upper bound if it appears as a numerator and by its lower bound if it appears as a denominator in Theorems 3 and 4.

Maximum radius: Our algorithm searches over all the slices centered at every center $c \in X$ to identify the best slice. However, we expect high error reduction slices to have small radius in real datasets, specially small-world networks [23]. Therefore, we set a maximum radius r_{max} for which slices are searched as an extra parameter of our algorithm.

In-memory distance structure for sampling: One of the advantages of using sampling in ST construction is that we can keep a distance structure D which gives the set $D(c, r) \subseteq S$ of distinct samples at distance $r (r \leq r_{max})$ from c in G. Such a data structure requires $O(||V||||S'||)$ space and can be computed in worst-case time $O(||S'||||E||)$, where S' is the set of distinct samples in S. Computing and maintaining a similar data structure with full data would not be feasible for large networks. For each center c, we can traverse $D(c, r_{max})$ in worst-case time $O(||S'||)$ in order to compute the error reduction of all slices centered in c. For this implementation, each iteration of Algorithm 3 (steps 10-23) runs in $O(||S'||(||E|| + ||X||))$ time.

VII. Experiments

We evaluate Slice Tree as a network attribute compression approach using real and synthetic networks. All algorithms are implemented in C++ and experiments were performed on a single core of a 2.67GHz Intel Core i7 with 12GB RAM.

A. Datasets

Table I lists four real-world datasets applied in our experiments. The *Traffic* dataset is the highway network of Los Angeles, CA (from the *PeMS* website[2]) with node values corresponding to average speeds at highway locations along time. The *Human* dataset is a gene network for *Homo sapiens* with gene and protein interactions as edges and tissue expression as node values (114 tissues) [24]. *DBLP* is an academic co-authorship network in which author nodes are annotated with publication counts for 15 research areas[3]. *Twitter* is a combination of the (undirected) social graph provided in [25], the tweets from [26] and the tweet sentiments from [27]. We averaged the sentiment of the tweets for each user and weighted these sentiments with the users' number of retweets as means to compute popularity-aware sentiments.

name	vertex	edge	value	$\|\|V\|\|$	$\|\|E\|\|$
Traffic	sensor	highway	speed	2k	6k
Human	gene	interaction	expression	4k	9k
DBLP	author	collaboration	#papers	1.3m	5.2m
Twitter	user	following	sentiment	.7m	19.2m

TABLE I: Dataset description and statistics.

We also generate *synthetic* data by combining a network structure from the BA model [28] and node values defined by an ST. Values inside a region follow a Normal distribution with standard deviation set as to produce the desired SSE.

B. Scalability and Accuracy of Sampling

We start by evaluating our approximate algorithm applying different sampling strategies and under varied conditions using synthetic data. Figure 4 compares the greedy slice tree construction algorithm (*ST*) and three versions of the approximate algorithm: *STU*, *STI*, and *STIF*. *STU* applies uniform sampling with $\delta = 0.1, \rho = 0.6, \pi = 0.1$. *STI* applies importance sampling with the same parameters as *STU*. *STIF* also uses importance sampling, but with more relaxed parameters $\delta = 0.4, \rho = 0.1, \pi = 0.01$, thus being faster than STI. Maximum radius r_{max} is set to 2. We will make clear whenever these parameters are changed. Synthetic networks have 10^5 vertices, 5 edges for each new vertex, 32 regions (31 slices), slice radius 2, error of 2.10^5 and error reduction of 10^5, unless specified. The task for all algorithms is finding the first slice and we evaluate them in terms of average execution time and approximation (i.e. fraction of the optimal error reduction achieved). We also compute how often the pruning based on the number of samples (Theorem 4) is used by STI and STIF as means to measure the importance of this pruning strategy compared to the one based on mean estimates (Theorem 3).

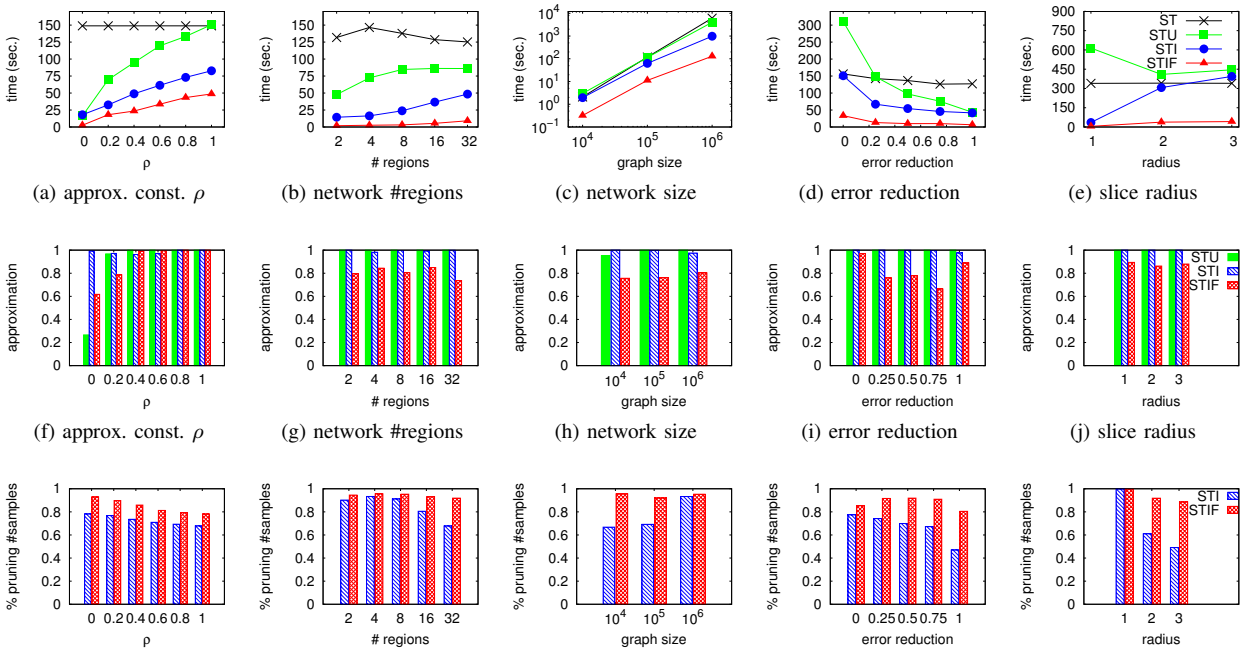
The constant ρ sets the compromise between the approximations given by *STU*, *STI*, and *STIF*, and their execution time (Figure 4a). However, the algorithms achieve much higher approximation than ρ in practice (Figure 4f). For instance, *STI* achieves at least a 0.95 approximation for any ρ and *STIF* obtains a 0.6 approximation in 2% of the time taken by *ST* ($\rho = 0$). Also, pruning based on the number of samples plays an important role for *STI* and *STIF* (Figure 4k).

Increasing the number of regions in the network has a small impact over the algorithms for all the evaluation metrics

[1] https://code.google.com/p/graph-compression

[2] http://pems.dot.ca.gov/

[3] Areas of venues were obtained from the classification in Wikipedia http://en.wikipedia.org/wiki/List_of_computer_science_conferences



(a) approx. const. ρ (b) network #regions (c) network size (d) error reduction (e) slice radius

(f) approx. const. ρ (g) network #regions (h) network size (i) error reduction (j) slice radius

(k) approx. const. ρ (l) network #regions (m) network size (n) error reduction (o) slice radius

Fig. 4: Execution time (a-e), approximation (f-j) and effectiveness of the pruning based on the number of samples (Theorem 4) (k-o) in finding the first slice ($k = 1$) in *synthetic* networks for greedy slice tree (*ST*), approximate slice tree with uniform sampling (*STU*), and two versions of slice tree with importance sampling using different parameters (*STI* and *STIF*) varying the approximation constant ρ and the number of regions, size ($||V||$), error reduction, and radius of slices of the input network. By applying importance sampling, we can efficiently discover accurate STs for large networks in various settings.

(Figures 4b, 4g and 4l). This shows that they can still quickly find a good slice in the presence of many candidates.

In terms of scalability with the network size (Figure 4c), *STU*, *STI*, and *STIF* achieve speedups up to $1.5\times$, $6\times$ and $47\times$ w.r.t. *ST* with at least 0.95, 0.97, and 0.75 approximation, respectively (Figure 4c). These results show how importance sampling enables the identification of approximate optimal slices using much less samples than the uniform sampling approach, which translates into effective pruning (Figure 4m).

The higher the input network error reduction, the better it matches an ST model, which leads to better performance for all the strategies except *ST* (Figure 4d) with no significant effect over approximation (Figure 4i). As the error reduction increases, uniform and importance sampling behave more similar and the pruning based on the number of samples becomes less effective for STI and STIF (Figure 4n). This analysis shows how a low-sampling version of the importance sampling algorithm enables users to assess whether ST is a suitable for the compression of a given network.

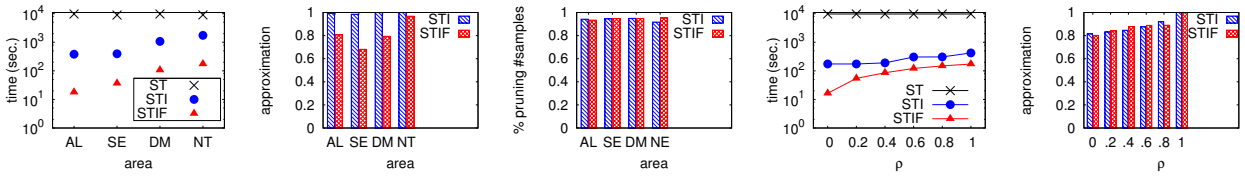
Because the uniform sampling approach is not value-sensitive, it needs more samples to detect slices with good approximation (Figures 4j and 4e)[4]. The use of the pruning reflects how importance sampling is more appropriate when slices are more condensed (Figure 4o).

[4]For this experiment, networks have $50k$ vertices and r_{max} is 3.

After understanding several aspects of different versions of our approximate algorithm using synthetic data, we next study its effectiveness on a real dataset. Figure 5 compares *ST*, *STI* and *STIF* in finding the first slice w.r.t. execution time, approximation, and use of the pruning based on number of samples for *DBLP*. *STU* was not considered in this evaluation because it does not scale to such a network. Default parameters for the importance sampling algorithm are set as follows: $\delta = 0.1, \rho = 0.9, , \pi = 0.02$ for *STI*; $\delta = 0.4, \rho = 0.1, \pi = 10^{-3}$ for *STIF* and $r_{max} = 2$ (*STI* and *STIF*).

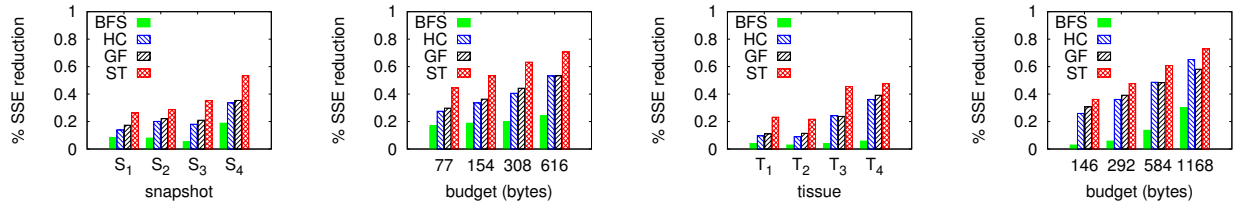
Figures 5a, 5b, and 5c show the performance of the algorithms for four different research areas: *algorithms*(AL), *security* (SE), *data management* (DM) and *networks* (NT). *STI* achieves speedups between $5\times$ (NT) and $25\times$ (AL) with average approximations between 0.79 (DM) and 1.0 (NT). *STIF* achieves speedups between $56\times$ (NT) and $544\times$ (DM) with approximations between 0.68 (SE) and 0.96 (NT). For all research areas, the number of samples gives a tighter bound on the error reduction in more than 93% of the time.

In Figures 5d and 5d, we evaluate the performance of the algorithms for the research area *algorithms* (AL) varying the approximation constant ρ. *STI* and *STIF* obtain a similar approximation average of 0.80 with a speedup of $56\times$ and $596\times$, respectively, when $\rho = 0$. When the best slice is to be returned with high probability ($\rho = 1$), *STI* and *STIF* get speedups of $23\times$ and $56\times$, respectively.



(a) time, 4 areas (b) approx., 4 areas (c) pruning, 4 areas (d) time, different ρ (e) approx., different ρ

Fig. 5: Execution time (a,d), approximation (b,e) and effectiveness of pruning based on the number of samples (c) in finding the first slice in *DBLP* for greedy slice tree (*ST*) and two versions of approximate slice tree using different parameters (*STI* and *STIF*). Four research areas (*algorithms* (AL), *security* (SE), *data management* (DM), and *networks* (NT)) are considered in (a,b,c). Results in (d,e) are for AL. Importance sampling enables the computation of accurate STs in large real networks.



(a) Traffic, 4 snapshots (b) S_4, inc. budget (c) Human, 4 tissues (d) T_4, inc. budget

Fig. 6: Error reduction for the wavelets with BFS (*BFS*), Haar trees (*HC*), graph Fourier (*GF*), and slice tree (*ST*) in the *Traffic* (a,b) and *Human* (c,d) datasets. We selected four network attributes (tissues and snapshots) from each dataset (a,c) and also a fixed attribute for increasing budget (b,d). *ST* compresses attribute values with high accuracy in real datasets.

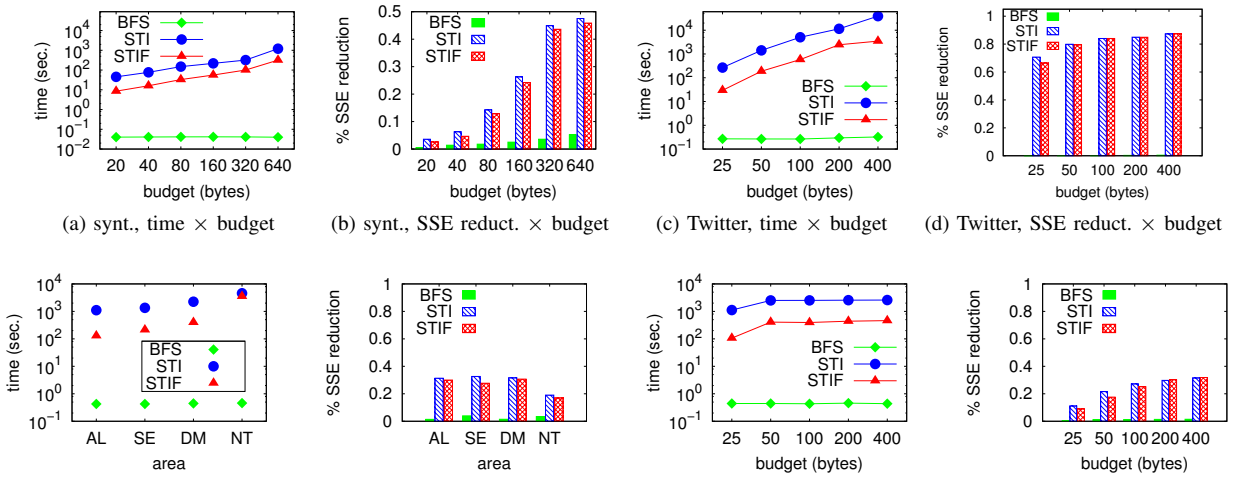
C. Slice Tree for Network Compression

We evaluate ST compression in terms of compression ratio, error, and running time using real and synthetic datasets. We also consider the following baseline approaches: *Graph Fourier (GF)* [13], *Wavelets over a BFS vector (BFS)*, and *Haar trees with Average Linkage (HC)* [18]. *GF* applies the eigenvectors of the Laplacian matrix of the network as a basis to represent the attribute values at different scales. Compression is achieved by selecting a small set of high-energy coefficients in the spectral domain. *BFS* maps node values to a vector by performing a BFS starting from an arbitrary node. This search tends to assign nodes that are close in the network structure to close positions in the vector and the values are further compressed using *Haar wavelets* [11]. *HC* is based on a tree representation of the network structure using hierarchical clustering [8]. Node values projected over this hierarchical structure are further compressed using a *Haar-like* wavelet basis. *Average Linkage* [18] and shortest path distances were applied in the hierarchical clustering.

Here, instead of computing the budget of an ST in number of slices (k), we convert this measure to *bytes* (see Definition 1). This enables a fair comparison between ST and the baselines. Figure 6 shows relative error reduction results for *ST*, *BFS* and *HC*. The relative SSE reduction is the ratio between the SSE of the dataset with respect to its overall average and the SSE of the compression. Since these datasets are small (up to $4k$ edges), we do not show the running time results. All the methods run in time in the order of seconds for both datasets. We first evaluate the compression approaches across different network attributes (Figures 6a and 6c). For *Traffic*, we selected four snapshots of the network that cover well the span of compression results. Similarly, we show results for four tissues[5] from the *Human* dataset. Budgets were set to 1% of the size of the dataset (154 and 292 $bytes$ for *Traffic* and *Human*, respectively). We also show results for a single attribute (S_4 and T_4) increasing the budget (Figures 6b and 6d). *ST* consistently outperforms the baselines for both datasets in all settings considered, achieving up to a 2-fold improvement over the best baseline (*GF*).

Next, we evaluate ST compression using large networks (*synthetic*, *DBLP* and *Twitter*). However, because *HC* requires the computation of a distance matrix and *GF* requires an spectral decomposition of the Laplacian matrix of the network, these methods cannot be applied to such large datasets. We restrict our results to our approximate algorithm using importance sampling and *BFS*. Figures 7a and 7b show the running time and *SSE* reduction as the budget is increased for two versions of the approximate algorithm using $\delta = 0.1, \rho = 0.6, \pi = 0.1$ (*STI*) and $\delta = 0.4, \rho = 0.1, \pi = 0.01$ (*STIF*) in synthetic networks. Maximum radius r_{max} is set to 2. The parameters used in the generation of the networks are the same ones described in Section VII-B and thus the optimal relative *SSE* reduction achievable using an ST with 32 regions (350 $bytes$) is 0.5. *BFS* is the most efficient algorithm, but it obtains poor error reduction results. For a budget of 320 $bytes$, *BFS*, *STI* and *STIF* achieve average error reductions of 0.04,

[5] T_1: frontal cortex, T_2 : stomach pylorus, T_3 : placenta and T_4 : tonsil.



(a) synt., time × budget (b) synt., SSE reduct. × budget (c) Twitter, time × budget (d) Twitter, SSE reduct. × budget

(e) DBLP, time × 4 areas (f) DBLP, SSE reduction × 4 areas (g) DBLP, time × budget (h) DBLP., SSE reduct. × budget

Fig. 7: Execution time (a,c,e,g) and error reduction (b,d,f,h) for *BFS* and two versions of approximate slice tree with importance sampling using different parameters (*STI* and *STIF*) in *synthetic* datasets (a,b), *Twitter* (c,d) and *DBLP* (e-g). For *synthetic*, *Twitter*, and *DBLP* (DM), we evaluate how the execution time and error reduction increase with the budget (a-d, g,h). We also show time and error reduction results for a fixed budget and four research areas in *DBLP* (e,f). *BFS* is more efficient than *STI* and *STIF* but achieves much poorer performance in terms of compression error.

0.45 and 0.44, respectively. As expected, once this budget is reached and there are no more new regions to be captured in the network, the benefit of adding extra budget is reduced.

Figures 7c and 7d show the compression results for the *Twitter* dataset. Default parameters for the versions of the importance sampling algorithm are set as: $\delta = 0.1, \rho = 0.9, \pi = 0.02$ for *STI*; $\delta = 0.4, \rho = 0.1, \pi = 0.001$ for *STIF* and $r_{max} = 2$ (*STI* and *STIF*). Maximum radius r_{max} is set to 1. As for the synthetic networks, *BFS* is fast but cannot produce accurate compression. While both *STI* and *STIF* achieve 87% relative *SSE* reduction with only 400 bytes (10^3 compression rate), *BFS* is not able to achieve any significant error reduction in any of the experiments. These results contrast with the performance of *BFS* for small datasets, where it was able to achieve up to 50% of the relative *SSE* reduction of *ST*.

Compression results for *DBLP* are given in Figures 7e-7h. Default parameters for the algorithms are set as for the experiments using *Twitter*. Maximum radius r_{max} is set to 2. Figures 7e and 7f show the execution time and error reduction of the algorithms for four research areas and a fixed budget of 400 *bytes* (31 slices). The compression results for *STI* and *STIF* vary according to the research area considered. For instance, the compression of *Algorithms* (AL) resulted in twice as much relative error reduction as achieved for *Networks* (NT) (Figure 7f). This is due to the fact that NT is a much more prolific area, with three times as many publications and active researchers compared to AL. Similarly, compressing NT takes 4 and 25 times longer than AL for *STI* and *STIF*, respectively. The gains in *SSE* reduction vanish quickly as we increase the budget for a fixed research area (AL). Figure 7h shows that both *STI* and *STIF* achieve a maximum relative *SSE* reduction of 30% with a 10^3 compression rate.

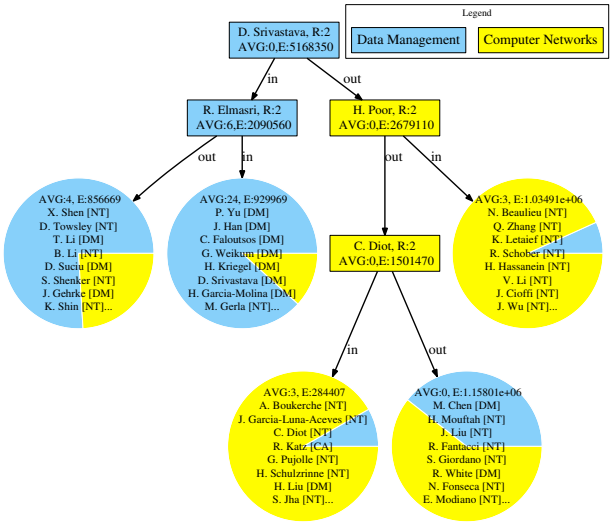


Fig. 8: First slices for *DBLP* combining paper counts for the areas Data Management and Networks. ST captures publications patterns across different research areas.

D. Slice Trees in Real-World Networks

In what follows, we illustrate ST compressions of real-world networks. We show that STs capture in a compact form important properties of networks, such as collaboration patterns in *DBLP* and major congestions in the *Traffic* network.

Figure 8 shows how authors from two areas (*Data Management (DM) and Networks (NT)*) are distributed among the first slices of an ST based only on combined publication counts.

Moreover, we assign authors to a single main area in which they are most prolific as means to characterize the regions discovered. We color-code both slice nodes (rectangles) and resulting regions (circles) based on the fraction of authors in DM and NT. The leaf region nodes list the 10 most prolific authors with their respective areas. The ST naturally separates the communities from the two areas as they are both distinct within the structure (researchers within areas are more likely to collaborate among each other) and also in terms of number of publications. The first slice separates the majority of the DM community within 2 hops from *Divesh Srivastava* from the NT community. Further slices separate authors based on how prolific they are (average paper count for each leaf ST node is reported in the first line of the label).

In Figure 1, we show how one snapshot of *Traffic* (Figure 1a) is compressed using 2 slices (Figure 1b). Nodes in the network have their sizes and colors set based on average speed values. The largest (red) node has an average speed of $10mph$ while the smallest (green) node has an average speed of $85mph$. Medium (yellow) nodes have average speed around $45mph$. Intermediate and leaf regions of the ST are also colored according to their average speeds (from left to right: 41, 15, and 65). The values reconstructed from the ST are projected back in the network (Figure 1c). ST captures two major congestions with smooth regions while the the remaining locations are covered by a third region. Further slices can isolate more congested regions.

VIII. Conclusions

We introduced Slice Tree as a network compression strategy for attribute values. ST is a hierarchical decomposition scheme using slices, which partition the network into compactly represented regions that are smooth w.r.t. attribute values. Computing an ST that minimizes the compression error is NP-hard, thus we proposed an efficient greedy heuristic for ST construction. In order to scale ST to large networks, we devised an importance sampling strategy that efficiently computes approximate optimal slices with high probability.

We evaluated our approach in terms of compression error and scalability using synthetic and real-world datasets. Results showed that ST produces accurate compression, achieving up to 87% error reduction in node values, with 10^3 compression ratio, and up to 2-fold improvements over the best baseline method considered. Moreover, importance sampling enables the efficient compression of million-node networks with speedups up to $47\times$ over our scheme using full data. We also demonstrated the effectiveness STs in capturing relevant phenomena in real networks, such as collaboration patterns in co-authorship networks and congestions in traffic networks.

This work opens promising directions for future research. A key question is how to update STs as the network changes over time. We also want to generalize our framework to other error metrics and more complex data (e.g. attribute vectors). Finally, distributed algorithms might enable the compression of even larger networks than those considered in this paper.

Acknowledgment

Research partially sponsored by NSF grant IIS-1219254 and the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (NSCTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] J. J. Brown and P. H. Reingen, "Social ties and word-of-mouth referral behavior." *Journal of Consumer research*, 1987.

[2] M. Newman, *Networks: An Introduction*. Oxford, 2010.

[3] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs," *IEEE Signal Processing Magazine*, 2013.

[4] M. Naor, "Succinct representation of general unlabeled graphs," *Discrete Applied Mathematics*, 1990.

[5] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *SIGMOD*, 2008.

[6] Y. Choi, "Fast algorithm for optimal compression of graphs." in *ANALCO*, 2010.

[7] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, "Graph olap: Towards online analytical processing on graphs," in *ICDM*, 2008.

[8] M. Gavish, B. Nadler, and R. R. Coifman, "Multiscale wavelets on trees, graphs and high dimensional data," in *ICML*, 2010.

[9] A. Sandryhaila and J. Moura, "Discrete signal processing on graphs," in *ICASSP*, 2013.

[10] S. Mallat, *A wavelet tour of signal processing*. Academic press, 1999.

[11] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim, "Approximate query processing using wavelets," *The VLDB Journal*, 2001.

[12] J. Bourgain, "On lipschitz embedding of finite metric spaces in hilbert space," *Israel Journal of Mathematics*, 1985.

[13] Z. Karni and C. Gotsman, "Spectral compression of mesh geometry," in *SIGGRAPH*, 2000.

[14] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Applied and Computational Harmonic Analysis*, 2011.

[15] M. Crovella and E. Kolaczyk, "Graph wavelets for spatial traffic analysis." in *INFOCOM*, 2003.

[16] M. Maggioni, J. C. BremerJr, R. R. Coifman, and A. D. Szlam, "Biorthogonal diffusion wavelets for multiscale representations on manifolds and graphs," in *SPIE*, 2005.

[17] J. Iverson, C. Kamath, and G. Karypis, "Fast and effective lossy compression algorithms for scientific datasets," in *Euro-Par*, 2012.

[18] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani, *The elements of statistical learning*. Springer, 2009.

[19] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," in *VLDB*, 2009.

[20] T. Srisooksai, K. Keamarungsi, P. Lamsrichan, and K. Araki, "Practical data compression in wireless sensor networks: A survey," *Journal of Network and Computer Applications*, 2012.

[21] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, 1963.

[22] S. Asmussen and P. Glynn, *Stochastic Simulation: Algorithms and Analysis*. Springer, 2007.

[23] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, 1998.

[24] F. Moser, R. Colak, A. Rafiey, and M. Ester, "Mining cohesive patterns from graphs with feature vectors." in *SDM*, 2009.

[25] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *WWW*, 2010.

[26] J. Yang and J. Leskovec, "Patterns of temporal variation in online media," in *WSDM*, 2011.

[27] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *Project Report*, 2009.

[28] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, 1999.