Uncertain Graph Sparsification

Panos Parchas, Nikolaos Papailiou, Dimitris Papadias, Francesco Bonchi

Abstract—Uncertain graphs are prevalent in several applications including communications systems, biological databases and social networks. The ever increasing size of the underlying data renders both graph storage and query processing extremely expensive. Sparsification has often been used to reduce the size of deterministic graphs by maintaining only the important edges. However, adaptation of deterministic sparsification methods fails in the uncertain setting. To overcome this problem, we introduce the first sparsification techniques aimed explicitly at uncertain graphs. The proposed methods reduce the number of edges and redistribute their probabilities in order to decrease the graph size, while preserving its underlying structure. The resulting graph can be used to efficiently and accurately approximate any query and mining tasks on the original graph. An extensive experimental evaluation with real and synthetic datasets illustrates the effectiveness of our techniques on several common graph tasks, including clustering coefficient, page rank, reliability and shortest path distance.

1 Introduction

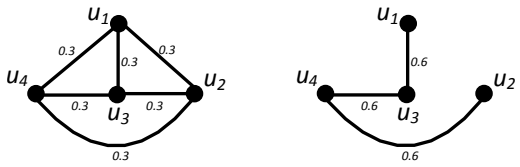
Uncertain graphs, where edges are associated with a probability of existence, have been used widely in numerous applications. For instance, in communication systems, each edge (u, v) is often associated with a reliability value that represents the probability that the channel from u to v will not fail. In biological databases, uncertain edges between vertices representing proteins are due to error-prone laboratory measurements. In social networks, edge probabilities can model the influence between friends, or the likelihood that two users will become friends in the future.

Several techniques have been proposed for diverse query processing and mining tasks on uncertain graphs (e.g. [9], [19], [29], [35]), most of which assume *possible-world* semantics. Specifically, let $\mathcal{G} = (V, E, p)$ be an uncertain (also called probabilistic) graph[1], where $p : E \rightarrow (0, 1]$ assigns a probability to each edge. $\mathcal{G}$ is interpreted as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ possible deterministic graphs, each defined on a subset of E. For example, since the uncertain graph of Figure 1(a) consists of 6 edges, there are 2^6 possible worlds. Under this interpretation, exact processing requires query evaluation on all possible worlds and aggregation of the partial results. In general, the probability of a query predicate Q is derived by the sum of probabilities of all possible worlds G for which $Q(G) = true$:

$$Q(\mathcal{G}) = \sum_{\substack{G \sqsubseteq \mathcal{G}, \\ Q(G)=true}} \Pr(G) \qquad (1)$$

Applying Equation (1), the probability that the uncertain graph of Figure 1(a) contains a single connected component is $\Pr[\mathcal{G}$ is connected$]=0.219$. This is obtained by generating the 2^6 deterministic graphs, and adding the probabilities of the connected ones.

Consequently, exact processing is prohibitive even for uncertain graphs of moderate size due to the exponential number of worlds. Thus, most techniques provide approximate results by applying Monte-Carlo (MC) sampling on a random subset of possible worlds. However, even MC may be very expensive for large uncertain graphs because generating a sample is time consuming as it involves sampling each edge. Moreover, due to the high entropy[2] of the uncertain graphs, there is significant variance among the possible worlds, which implies the need of numerous samples for accurate query estimation. This imposes huge overhead at query processing cost because the query must be executed at every sample.



(a) uncertain graph $\mathcal{G}$ (b) sparsified graph $\mathcal{G}'$

Fig. 1. Uncertain graph sparsification example.

In order to tackle the high cost, we develop techniques for *uncertain graph sparsification*. Specifically, given $\mathcal{G}$ and a parameter $\alpha \in (0, 1)$, the proposed methods generate a sparsified probabilistic subgraph $\mathcal{G}' = (V, E', p')$, which contains a fraction of the edges, i.e., $E' : E' \subset E$, $|E'| = \alpha|E|$. $\mathcal{G}'$ preserves the structural properties of $\mathcal{G}$, has less entropy, and can be used to approximate the result of a wide range of queries on $\mathcal{G}$. Sparsification yields significant benefits in execution time because the cost of sampling is linear to the number of edges. Moreover, the required number of samples is proportional to the graph's entropy ([13], [18]), which is lower in the sparsified graph. Finally, similar to the case of deterministic graphs, sparsification reduces the storage cost, and facilitates visualization of complex networks.

Figure 1(b) illustrates $\mathcal{G}'$, a sparsified subgraph of $\mathcal{G}$ that contains half the original edges. Observe that the edges of $\mathcal{G}'$ have higher probabilities than those in $\mathcal{G}$, in order to compensate for the missing ones. Assume, for instance, a query that asks for the probability that $\mathcal{G}$ consists of a single

- *P. Parchas and D. Papadias are with the Dept. of Computer Science and Engineering, Hong Kong University of Science and Technology. E-mail: {pparchas, dimitris}@cse.ust.hk*
- *N. Papailiou is with the National Technical University of Athens, Greece. E-mail: npapa@cslab.ece.ntua.gr*
- *F. Bonchi is with the ISI Foundation, Italy. E-mail: francesco.bonchi@isi.it*

1. $\mathcal{G}$ is assumed simple, unweighted, undirected and connected.

2. The entropy $H(\mathcal{G})$ of an uncertain graph $\mathcal{G}$ is defined as the joint entropy of its edges $H(e)$ for all $e \in E$. Due to the edge independence, $H(\mathcal{G}) = \sum_{e \in E} H(e) = \sum_{e \in E}(-p_e \log p_e - (1 - p_e) \log(1 - p_e))$.

connected component. Since $\Pr[\mathcal{G}'$ is connected]=0.216 and $\Pr[\mathcal{G}$ is connected]=0.219, $\mathcal{G}'$ can be used to effectively approximate the result of the query. Furthermore, the entropy decreases from 0.94 to 0.4. Because $\mathcal{G}'$ has fewer edges, it is more efficient to sample and store. Additionally, since it has less entropy, it requires fewer samples for accurate estimation. Our goal is to generate sparsified graphs that can be used for a variety of queries and tasks.

To the best of our knowledge, this is the first work on uncertain graph sparsification. On the contrary, sparsification has received considerable attention in the deterministic graph literature. In that context, most techniques aim at approximating all shortest path distances up to a multiplicative or additive factor, or preserving all cuts up to an arbitrarily small multiplicative error. As we demonstrate in our experimental evaluation, the adaptation of such methods to uncertain graphs yields poor results. On the other hand, our sparsification techniques achieve high accuracy and small variance for common graph tasks by capturing the *expected* node degrees, or the *expected* cut sizes up to a certain value. Summarizing, the contributions of the paper are:

- We propose a novel framework of uncertain graph sparsification with entropy reduction.
- We design algorithms that reduce the number of edges and tune the probability of the remaining ones to preserve crucial properties.
- We experimentally demonstrate that the sparsified graphs are effective for a variety of common tasks including *shortest path distance*, *reliability*, *page rank* and others.

The rest of the paper is organized as follows. Section 2 surveys the related work. Section 3 defines the problem, introduces our uncertain sparsification framework and presents baseline solutions motivated by the deterministic graph literature. Section 4 proposes sparsification algorithms that capture the expected vertex degrees. Section 5 analyzes rules to preserve cut sizes and modifies our algorithms for this case. Section 6 contains an extensive experimental evaluation on real and synthetic datasets, and Section 7 concludes the paper.

2 Related Work

Existing sparsification methods focus exclusively on deterministic graphs. Section 2.1 and 2.2 present sparsification techniques that preserve the shortest path distances and graph cuts respectively. Section 2.3 discusses other related methods.

2.1 Spanners

Sparsification aimed at preserving the shortest path distances between all node pairs is based on the concept of t-*spanner* [34]. A t-*spanner* of a weighted deterministic graph $G = (V, E, w)$ is a subgraph $G' = (V, E', w), E' \subseteq E$ such that, for any pair of vertices $u, v \in V$, their distance in G' is at most $t \in \mathbb{N}^+$ times their distance in G, i.e., $dist_{G'}(u, v) \leq t \cdot dist_G(u, v)$. The parameter t is the *stretch factor* of G'. Computing a t-*spanner* with the minimum number of edges has received considerable attention in the literature [37]. Peleg and Schäffer [34] prove that a spanner with stretch factor $2t-1$ must have at least $\Omega(n^{1+1/t})$ edges.

Based on this lower bound, a simple technique [4] to generate $(2t-1)$-$spanners$ processes the edges in increasing order of their weights. Initially, the spanner is $E' = \emptyset$. An edge $e = (u, v)$ is included to E', if the distance between u and v in E' exceeds $(2t-1)w_e$. The algorithm has $O(|E|n^{1+1/t})$ time complexity due to the shortest path distance computations. Roditty and Zwick [36] propose an improved $O(tn^{2+1/t})$ time method, which incrementally maintains a single source shortest path tree up to a given distance. Baswana et. all [5] design a randomized algorithm to compute a spanner of $(2t-1)$-stretch and $O(t|E|^{1+1/t})$ size in $O(t|E|)$ expected time, which uses a novel clustering approach and avoids distance computations. An adaptation of this method provides a benchmark for our experimental evaluation. Finally, [28] improves spanner size and algorithmic complexity by sacrificing constant factors in stretch.

2.2 Cut-based sparsifiers

Consider a deterministic, undirected, weighted graph $G = (V, E, w)$, where $|V| = n$. Given a vertex set $S \subseteq V$, the cut $E_G(S)$ is the set of edges with exactly one endpoint in S, i.e, $E_G(S) = \{(u, v) \in E | (u \in S, v \notin S)\}$. The size $C_G(S)$ of the cut is the sum of weights of the edges in $E_G(S)$, i.e., $C_G(S) = \sum_{e \in E_G(S)} w_e$. Cut-based sparsification preserves the size of *all* cuts of the graph within a multiplicative error. Formally, given a dense weighted graph ($|E| = \Theta(n^2)$) and an approximation error $\epsilon \in (0, 1)$, the output is $G' = (V, E', w')$, where $E' \subseteq E$ and $|E'| = O(n \log n/\epsilon^2)$, such that for any set $S \subseteq V$, $C_{G'}(S)$ is within ϵ of the original cut size $C_G(S)$, i.e., $C_{G'}(S) \in (1 \pm \epsilon)C_G(S)$ with high probability.

Algorithms for cut-based sparsification follow a two step approach. The first assigns a probability p_e to each edge based on the topology of the graph. The second step samples each edge with probability p_e, and assigns weight $w'_e \propto \frac{1}{p_e}$ to the sampled edges. Intuitively, sparsification through this framework relies on the following observations:

- Edges in dense areas are not crucial for maintaining the graph connectivity. Thus, they have low sampling probability p_e [15].
- Edges sampled with low p_e are assigned large weights so as to compensate for their missing neighbouring edges.

The existing methods assume integer weights and differ mainly in the first step, i.e., that of choosing p_e for each edge $e = (u, v)$. Benczúr and Karger [7] assign probabilities inversely proportional to the k-strong connectivity[3] of u, v. Fung et al. [15] simplify analysis by utilizing sampling probabilities inversely proportional to the size of the minimum cut separating u and v. Nagamochi and Ibaraki [30] estimate the edge connectivities using the NI index λ_e of an edge e. The NI index is generated by iteratively constructing spanning forests of the initial graph, while reducing the weights of the selected edges. In essence, the NI index is the last spanning forest that contains e, given that an edge with weight w_e needs to participate in w_e contiguous forests. To ensure that the sparse graph has $O(n \log n/\epsilon^2)$ edges in expectation, the sampling probability is set to $p_e = \frac{\rho}{\lambda_e}$, where $\rho = O(\log n/\epsilon^2)$. To see this, let $\mathbb{E}(|E'|)$ denote the expected number of edges of G'. According

3. A maximal k-strong connected component is a maximal induced subgraph of V that remains connected after removing up to k edges. The k-strong connectivity of an edge (u, v) is the maximum k such that u and v belong to the same k-strong connected component.

to [8], $\sum_e 1/\lambda_e = O(n \log n)$; thus, $\mathbb{E}(|E'|) = \sum_e p_e = \rho \sum_e 1/\lambda_e = O(n \log^2 n/\epsilon^2)$. A more refined analysis [15] reduces this to $O(n \log n/\epsilon^2)$. Section 3.3 modifies [30] as a benchmark for our experiments.

Spielman and Srivastava [40] generate the electrical equivalent of the graph by assuming resistors of 1Ω at each link. The sampling probability of edge (u, v) is proportional to the amount of current that flows through e when a unit voltage difference is applied to u, v. This approach also preserves every eigenvalue of the original graph, leading to the stronger notion of *spectral sparsification*[40]. [20], [23] employ spanners in the context of spectral sparsification. [20] utilizes spanners to approximate the "robust connectivity" $q_\kappa(u, v)$, i.e., the number of paths between vertices u and v with length at most κ. It is proven that $q_\kappa(u, v)$ serves as a good upper bound for the effective resistance of [40]. On the other hand, [23] extracts a collection of edge disjoint spanners and then samples the remaining edges with fixed probability, yielding a parallel algorithm that spectrally preserves the graph. Furthermore, spectral sparsification algorithms have been proposed for the streaming ([2], [3], [21]) and dynamic [1] graph settings. For a more detailed review of cut-based sparsification literature on deterministic graphs, we refer the reader to [6].

2.3 Other related techniques

Other than the theoretical papers presented above, there has been little work on implementing sparsification algorithms for deterministic graphs. Linder et al.[26] propose some heuristic methods aiming to preserve the structure of social networks. Their main method *Local Degree* keeps only the edges to *hubs*, i.e., vertices with high degree, claiming that they are crucial for the topology of complex social networks. However, the approach applies only on unweighted graphs and does not perform weight redistribution among the remaining edges; thus, it cannot be adapted to uncertain graphs.

Some deterministic graph sparsification techniques focus on approximating the result of particular queries, as opposed to structural graph properties. For instance, [10], [27] sparsify a directed graph, while preserving the influence logs among its vertices. Satuluri et al. [39] apply local sparsification in order to capture communities and facilitate clustering. These approaches are specific to the query in question, whereas we aim at generating sparsified graphs applicable to multiple, possibly diverse, query types. *Subgraph sampling* generates a subgraph of an input deterministic graph that contains a small fraction of the nodes (and edges), but has similar structural properties [17], [24]. This is different from sparsified graphs, which maintain the input set of nodes. In our experiments, we apply [24] to reduce the size of real graphs for expensive queries that cannot terminate in the original graphs.

[32], [33] propose algorithms for generating deterministic representative instances that approximate the expected node degrees of the uncertain graph. Queries can then be processed by applying conventional algorithms on these instances. Since the representatives have fewer edges than the original uncertain graph, this could be viewed as a special case of sparsification. However, a representative (i.e., deterministic graph) cannot be used to answer queries whose output is uncertain, e.g., return the probability that the graph consists of a single connected component, or the probability that two vertices are reachable from each other. On the other hand, our techniques generate uncertain graphs that can be used for the same query and mining tasks as the original graph. Moreover, the methods of [32], [33] do not provide control over the number of edges in the representative graphs. Intuitively, while [32], [33] aim at *eliminating* uncertainty by extracting a representative instance (i.e., zero entropy), in this work we aim at *decreasing* the uncertainty of the input graph (i.e., reducing its entropy).

3 Problem Definition and Framework

Let $\mathcal{G} = (V, E, p)$ be a probabilistic undirected graph, where $p : E \to (0, 1]$ is a function that assigns a probability $p(u, v)$ to each edge $(u, v) \in E$. Given a *sparsification ratio* $\alpha \in (0, 1)$, we extract from $\mathcal{G}$ a sparsified subgraph $\mathcal{G}' = (V, E', p')$ such that $E' \subset E$ and $|E'| = \alpha|E|$. $\mathcal{G}'$ should preserve the structural properties of $\mathcal{G}$, so that it can be used to accurately approximate the result of diverse queries on $\mathcal{G}$. Moreover, $\mathcal{G}'$ should reduce the entropy of $\mathcal{G}$ in order to decrease the variance of the queries. In addition to diminishing the storage overhead, sparsification yields significant benefits in terms of query processing because the cost of sampling is proportional to the number of edges[4]. Moreover, through entropy reduction, the resulting graph is less uncertain, thus it requires fewer samples for accurate query estimation.

3.1 Uncertain sparsification

As stated in Section 2, a prevalent goal of deterministic graph sparsification is preservation of the cut sizes. The notion of a cut can be extended naturally to uncertain graphs. In this case, due to the linearity of expectation, the *expected* size of a cut is the sum of the probabilities of the edges involved in the cut.

Definition 1 (Expected cut size). Given an uncertain graph $\mathcal{G} = (V, E, p)$ and a subset $S \subseteq V$, the expected cut size of S in $\mathcal{G}$ is the summation of the probabilities of the edges with exactly one endpoint in S:

$$C_{\mathcal{G}}(S) = \sum_{\substack{e=(u,v)\in E \\ u\in S, v\in V\setminus S}} p_e$$

We define the *absolute discrepancy* $\delta_A(S)$ of a vertex set S in a sparsified graph $\mathcal{G}'$ as the difference of S's expected cut size in $\mathcal{G}'$ to its expected cut size in $\mathcal{G}$, i.e.,

$$\delta_A(S) = C_{\mathcal{G}}(S) - C_{\mathcal{G}'}(S)$$

Accordingly, the *relative discrepancy* $\delta_R(S)$ is the absolute discrepancy of S divided by the original cut size:

$$\delta_R(S) = \frac{C_{\mathcal{G}}(S) - C_{\mathcal{G}'}(S)}{C_{\mathcal{G}}(S)}$$

To simplify notation, we collectively refer to δ_A and δ_R as δ, and we only differentiate when required. In addition, we use the term 'cut' to also refer to the expected cut size. Motivated by the work in deterministic sparsification, we aim at cut-preserving sparsified graphs, or, using our notation, at minimizing discrepancy δ. The exponential number

4. Sampling techniques have complexity $O(E)$ [25].

of cuts renders their exhaustive enumeration intractable. To overcome this, we target cuts of sets S with specific cardinality k.

Formally, given an integer k, we define the k-discrepancy Δ_k of a graph $\mathcal{G}'$ as the sum of the absolute values of the discrepancies for all sets with cardinality k:

$$\Delta_k(\mathcal{G}') = \sum_{S \subseteq V, |S|=k} |\delta(S)|$$

where $\delta(S)$ is the absolute or relative discrepancy of S. The absolute discrepancy emphasizes vertices with high degree because they are more likely to yield large absolute errors. On the other hand, the relative discrepancy targets all node degrees equally, by considering the relative error. We aim at minimizing the sum of Δ_i for $1 \leq i \leq k$, or equivalently at preserving the size of all cuts up to k. Accordingly, the problem we tackle in this work is:

Problem 1. Given an uncertain graph $\mathcal{G} = (V, E, p)$, and a sparsification ratio $\alpha \in (0, 1)$, find an uncertain graph $\mathcal{G}^* = (V, E^*, p^*)$, with $|E^*| = \alpha|E|$ that minimizes the sum of discrepancies $\sum_{i=1}^{k} \Delta_i(\mathcal{G}^*)$ up to a given $k \geq 1$.

In addition to discrepancy minimization, our methods aim at entropy reduction. Observe that the two objectives are not independent because, since the sparsified graph has fewer edges, it is likely to have lower entropy as well. Minimization of discrepancy refers to the *quality* of the sparsified graph, while entropy reduction relates to the *efficiency* of query processing. The proposed techniques apply a gradient descent framework that finds a local minimum in terms of discrepancy, but adjusts the gradient step with the aim of reducing entropy.

For the special case of $k = 1$, minimizing Δ_1 is equivalent to preserving the expected degrees for all vertices, which has been shown to be effective when generating deterministic representatives of uncertain graphs [32]. Next, we introduce two benchmark solutions, adapted from the deterministic sparsification literature.

3.2 Benchmark solutions

Spanners and cut based sparsifiers stem from theoretical papers, and to the best of our knowledge, they have not been implemented or evaluated in practice. Furthermore, our uncertain graph sparsification setting differs from that of deterministic sparsification. All spanners focus on selecting a subset of edges without changing their weights. On the other hand, we modify the probabilities of edges in the sparsified graph, in order to compensate for the eliminated edges. Moreover, probabilities of the uncertain graphs, unlike weights of deterministic graphs are bounded by 1, inhibiting the direct application of cut-based sparsification methods, all of which assume unbounded weights. Lastly, uncertain sparsification explicitly targets to reduce the entropy of the input graph so as to minimize the variance of query estimators. Deterministic sparsifiers do not consider entropy. Nevertheless, in the following, we extend two state-of-the-art methods, one based on *cut sparsifiers* and the other on *spanners*, to uncertain graphs. The resulting algorithms are used as benchmarks in our experiments.

As a representative of cut sparsifiers, we adopt the NI method[5] of [30] by transforming the uncertain graph $\mathcal{G}$ to a weighted deterministic G_w. Recall that NI requires integer weights and an approximation parameter $\epsilon \in (0, 1)$ to produce a sparsified graph G'_w with $O(n \log n/\epsilon^2)$ edges on expectation. Intuitively, the probabilities of $\mathcal{G}$ are directly analogous to the weights of G_w in terms of (expected) cut size. To maintain this analogy while ensuring $w_e \in \mathbb{N}^+$, our transformation divides each probability of $\mathcal{G}$ by the smallest value p_{min}, and rounds the result to the closest integer, i.e., $w_e = \lfloor p_e/p_{min} \rceil$. To relate ϵ to our sparsification ratio α, we set $\epsilon = \sqrt{n \log n/\alpha|E|}$. Next, we use NI as a black box to sparsify G_w into G'_w. However, since the number of edges of G'_w in [30] is given on expectation with asymptotic notation, it is not guaranteed to equal $\alpha|E|$. If the resulting graph has more (fewer) edges than $\alpha|E|$, we iteratively execute NI after increasing (decreasing) ϵ by a small factor θ, until the first (last) graph for which $|E'| \leq \alpha|E|$. The remaining $\alpha|E| - |E'|$ edges are randomly sampled from $E \setminus E'$ using the initial probabilities p. Intuitively, this calibration process approximates the minimum ϵ, which ensures $|E'| \leq \alpha|E|$. Finally, for each edge e, we convert w'_e into p'_e through the inverse transformation cupped by 1, i.e, $p'_e = \min\{w'_e \cdot p_{min}, 1\}$.

In order to apply spanners, we generate the weights of G_w using the formula $w_e = -\log(p_e)$ [35]. This transformation takes advantage of the logarithmic summation properties to preserve the *most probable paths* of $\mathcal{G}$. For the tuning of the stretch factor, we solve the equation $\alpha|E| = tn^{1+1/t}$ with respect to t in order to find the minimum spanner with the required number of edges $\alpha|E|$. Using the computed value of t, we run the spanner algorithm of [5] on top of G_w. As in the case of cuts, some calibration steps may be required to approximate the minimum t_{min} that ensures $|E'| \leq \alpha|E|$. This time however, at each iteration, t_{min} can only change by 1 because it is integer. After computing $G'_w = (V, E', w)$, we add its edges in the sparse probabilistic graph $\mathcal{G}'$ using the initial probabilities $p' = p$, since [5] retains the original edge weights. The appendix contains details and pseudocodes of the benchmark adaptations.

3.3 Framework

The proposed framework starts with an initialization step that generates a connected unweighted backbone graph G_b. Then two different techniques operate on G_b in order to produce the sparsified graph. *Gradient Descent Backbone* (GDB) assigns probabilities to the edges of G_b without altering its structure, i.e., the resulting graph has the same edges as G_b. On the other hand, *Expectation Maximization Degree* (EMD) updates both the structure of G_b and the edge probabilities. In the rest of this section we focus on the initialization step.

A simple approach could sample the edges of $\mathcal{G}$ in random order according to their probabilities, until it obtains $\alpha|E|$ edges[6]. However, this would not ensure the connectivity of G_b, especially for small α[7]. Disconnected graphs can introduce large errors on various queries such as shortest distances between vertices of different connected components. Moreover, individual vertices may become entirely disconnected, which would lead to high total discrepancy.

5. Any method of Section 2.2 can be applied similarly.

6. A similar approach has been applied in [26] for sparsification of deterministic graphs.

7. We assume $\alpha \geq \frac{|V|-1}{|E|}$, otherwise the sparsified graph cannot preserve the connectivity of the original.

To overcome this problem, we generate connected backbone graphs using the following method, which is inspired by the related work in deterministic sparsification [30][8]. We first compute a maximum spanning tree of $\mathcal{G}$, where the probabilities act as weights. Then, we remove the tree edges from $\mathcal{G}$ and insert them to G_b. This process is repeated until G_b contains $\alpha'|E|$ edges, where $\alpha' < \alpha$. Note that after the first application of the maximum spanning tree, $\mathcal{G}$ may become disconnected; thus, subsequent applications may return spanning forests instead of trees. Finally, the remaining $(\alpha - \alpha')|E|$ edges of G_b are generated by random sampling based on the edge probabilities. Algorithm 1 illustrates backbone graph generation.

Algorithm 1 Backbone Graph Initialization (BGI)

Input: uncertain graph $\mathcal{G} = (V, E, p)$, sparsification ratio α, spanning ratio α'
Output: backbone graph $G_b = (V, E_b)$

```
E_b ← maximum spanning tree of E
E ← E \ E_b
while |E_b| < α'|E| do
    F ← maximum spanning forest of E
    E_b ← E_b ∪ F
    E ← E \ F
while |E_b| < α|E| do
    sample edge e ∈ E with probability p_e
    if e is selected then
        E_b ← E_b ∪ {e}
        E ← E \ {e}
```

Parameter α' tunes the number of edges obtained through spanning forests. Intuitively, generating all $\alpha|E|$ edges using spanning forests is not desirable because all vertices would be treated equally, independently of their degrees. In our experiments, we set the value of α' so that it is the minimum of 0.5α and the number of edges in the first six maximum spanning forests.

Given the initial graph G_b, Section 4 proposes algorithms that aim at minimizing the degree discrepancy Δ_1, while Section 5 focuses on preserving the expected cuts i.e., Δ_k for $k > 1$. In both cases probability values that would incur high entropy are avoided.

4 Preserving Expected Degrees

We first focus on Problem 1 for $k = 1$, and describe methods to generate a sparsified graph $\mathcal{G}'$ that preserves the expected degrees of all vertices in $\mathcal{G}$. Section 4.1 describes probability assignment that minimizes Δ_1 using linear programming (LP). Due to the inefficiency of LP on large graphs, Sections 4.2 and 4.3 propose GDB and EMD, which assign probabilities inspired by gradient descent and expectation maximization, respectively.

4.1 Optimal probability assignement for minimizing Δ_1

Given the backbone graph $G_b = (V, E_b)$, we wish to compute the edge probabilities that minimize the discrepancy for $k = 1$. Let $\mathbf{d}$ be a vector of size $|V|$ that contains the expected degrees of the original graph $\mathcal{G}$. We represent G_b by an incidence matrix $\mathbf{A_b}$ of size $|V| \times |E_b|$. Using this notation, an equivalent formulation for minimizing the sum of absolute discrepancies δ^A is

$$\begin{aligned} \min_{\mathbf{p}'} \quad & |\mathbf{d} - \mathbf{A_b}\mathbf{p}'| \\ s.t. \quad & \mathbf{p}' \in (0,1]^{|E_b|} \end{aligned} \tag{2}$$

Lemma 1. For an incidence matrix $\mathbf{A_b}$, there is a probability assignment $\mathbf{p}^*$ that minimizes Δ_1 for which the expected degree $d_u^* \leq d_u, \forall u \in V$.

Proof: Consider a probability assignment $\mathbf{p}^*$ that minimizes $|\mathbf{d} - \mathbf{A_b}\mathbf{p}^*|$ and let $\mathbf{d}^* = \mathbf{A_b}\mathbf{p}^*$ be the new vector of expected degrees. For the sake of contradiction, assume also that $\mathbf{d}^*$ contains *illegal* vertices, i.e., vertices with $d_u^* > d_u$. Each illegal vertex u is adjacent to at least one *legal* vertex v, with $d_v^* \leq d_v$, otherwise the assignment is not optimal (setting $p^*(u,v) = p(u,v) - \epsilon$, for a small $\epsilon > 0$ yields a better result) . Let $\theta = d_u^* - d_u > 0$. We prove that, if we subtract θ from $p(u,v)$, then the resulting probability assignment $\mathbf{p}'$:

$$p'_e = \begin{cases} p_e - \theta, & \text{if } e = (u,v), \\ p_e, & \text{otherwise} \end{cases}$$

is also optimal. Let $\mathbf{d}' = \mathbf{A_b}\mathbf{p}'$ be the corresponding vector of expected degrees. Then,

$$d'_i = \begin{cases} d_i^* - \theta = d_i, & \text{if } i = u, \\ d_i^* - \theta, & \text{if } i = v, \\ d_i^*, & \text{otherwise} \end{cases} \tag{3}$$

Since $\mathbf{p}^*$ is optimal, $\sum_{i \in V} |d_i - d_i^*|$ is minimum. Then,

$$\begin{aligned} \sum_{i \in V} |d_i - d_i^*| &= \sum_{i \in V \setminus \{u,v\}} |d_i - d_i^*| + |d_u - d_u^*| + |d_v - d_v^*| \\ &= \sum_{i \in V \setminus \{u,v\}} |d_i - d'_i| + |d_u - d_u - \theta| + |d_v - d'_v - \theta| \end{aligned} \tag{4}$$

Given that v is *legal*, $d_v - d_v^* \geq 0 \overset{(3)}{\Longrightarrow} d_v - d'_v - \theta \geq 0$. Thus, Equation (4) becomes:

$$\sum_{i \in V} |d_i - d_i^*| = \sum_{i \in V \setminus \{u,v\}} |d_i - d'_i| + \theta + |d_v - d'_v| - \theta = \sum_{i \in V} |d_i - d'_i|$$

Since $\sum_{i \in V} |d_i - d_i^*|$ is minimum, $\sum_{i \in V} |d_i - d'_i|$ is also minimum. Thus, $\mathbf{p}'$ is optimal. By applying the above argument to all *illegal* vertices we construct an optimal instance that contains only legal vertices. □

Through Lemma 1 we prove the following theorem.

Theorem 1. Given a backbone incidence matrix $\mathbf{A_b}$ and an expected degree vector $\mathbf{d}$, the optimal probability distribution $\mathbf{p}^*$ for degree discrepancy δ_A, is the solution of the following LP:

$$\begin{aligned} \max_{\mathbf{p}'} \quad & |\mathbf{p}'| \\ s.t. \quad & \mathbf{A_b}\mathbf{p}' \leq \mathbf{d} \\ & \mathbf{p}' \in (0,1]^{|E|} \end{aligned} \tag{5}$$

Proof: According to Lemma 1, there exists an optimal assignment $\mathbf{p}^*$ for which $\mathbf{A_b}\mathbf{p}^* \leq \mathbf{d}$. We only need to prove that the objective function is equivalent to Equation (2). Let $\mathbf{A_b} = [\mathbf{a}_{u_1}\ \mathbf{a}_{u_2}\ \dots\ \mathbf{a}_{u_n}]^T$, i.e., the row vectors of matrix $\mathbf{A_b}$. Then

8. Other deterministic sparsification methods such as t-bundle [23] or *Local Degree* [26] could also be used.

$$\min |\mathbf{d} - \mathbf{A_b}\mathbf{p}'| = \min \sum_{u \in V} |d_u - \mathbf{a}_u\mathbf{p}'| \overset{(5)}{=} \min \sum_{u \in V} (d_u - \mathbf{a}_u\mathbf{p}')$$
$$= \max \sum_{u \in V} \mathbf{a}_u\mathbf{p}' = \max \sum_{u \in V} p_u = \max |\mathbf{p}'| \quad \square$$

Theorem 1 states that the probability assignment step can be performed optimally by any linear programming solver, e.g. simplex [11]. However, the running time of such solvers is prohibitive for large graphs, which is also confirmed in our experimental evaluation. Moreover, LP does not explicitly reduce entropy. The following method closely approximates the optimal probability assignment at a small fraction of the time, while also decreasing the entropy.

4.2 Gradient descent backbone

Given the backbone graph $G_b = (V, E_b)$, *Gradient Descent Backbone* (GDB) initially generates a seed uncertain graph $\hat{\mathcal{G}} = (V, E_b, \hat{p})$, $\hat{p} = p$, and proceeds in iterations. Let $\hat{\mathbf{p}}$ and $\mathbf{p}'$ be the probabilities of the previous and the current iteration, respectively. At each iteration, GDB optimizes the probability p'_e of each edge $e = (u_0, v_0)$, considering the remaining probabilities fixed. To this end, we need to calculate the derivative of the objective function $\sum_{u \in V} |\delta(u)|$. Since $|\delta(u)|$ is not differentiable at 0, we utilize the squared discrepancy[9] $\delta^2(u)$. Accordingly, the objective function of GDB is $D_1 = \sum_{u \in V} \delta^2(u)$. Its derivative with respect to p'_e is:

$$\frac{\partial D'_1}{\partial p'_e} = \sum_{u \in V} \frac{\partial \delta'^2(u)}{\partial p'_e} = -2 \cdot \delta'(u_0) - 2 \cdot \delta'(v_0) \tag{6}$$

Therefore, changing the probability of edge e from $\hat{p}_e$ to p'_e the discrepancy $\delta'(u)$ becomes:

$$\delta'(u) = \frac{\hat{\delta}_A(u) + (\hat{p}_e - p'_e)}{\pi(u)} \tag{7}$$

where: $$\pi(u) = \begin{cases} 1 & \text{if } \delta(u) = \delta_A(u) \\ C_{\mathcal{G}}(u) & \text{if } \delta(u) = \delta_R(u) \end{cases}$$

and δ_A and δ_R correspond to absolute and relative discrepancy, respectively. Substituting Equation (7) to Equation (6), the probability that sets the first derivative to zero, is:

$$p'_e = \hat{p}_e + stp, \text{ where } stp = \frac{\pi(v_0)\hat{\delta}_A(u_0) + \pi(u_0)\hat{\delta}_A(v_0)}{\pi(u_0) + \pi(v_0)} \tag{8}$$

Equation (8) raises two concerns: First, probability p'_e may fall outside the range [0,1]. In this case, D_1 is monotonic in [0,1] because it is convex (i.e., $\frac{\partial^2 D_1}{\partial p_e^2} > 0$). Second, the probability increase *stp* may result to higher entropy for e, which is not desirable. GDB overcomes these concerns by assigning probabilities using the following rule:

$$p'_e = \left\lfloor \hat{p}_e + h \frac{\pi(v_0)\hat{\delta}_A(u_0) + \pi(u_0)\hat{\delta}_A(v_0)}{\pi(u_0) + \pi(v_0)} \right\rceil_0^1 \tag{9}$$

where ${}_0\lfloor x \rceil^1 = \max\{0, \min\{x, 1\}\}$ and $h \in [0, 1]$.

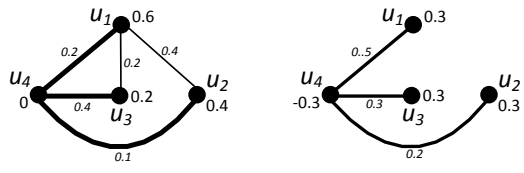
In essence, GDB performs gradient descent which is guaranteed to reach a local minimum of the objective function [12]. Parameter h relates the step size of gradient decent to the entropy. Intuitively, if the optimal assignment of Equation (8), results in entropy increase, GDB adds only a fraction h of *stp* to p_e, attenuating the negative side effect. This is a common practice in gradient decent techniques: instead of moving directly to the goal, move in smaller steps towards the correct direction [12]. In addition to limiting entropy increase, this allows other neighbouring edges to update their probabilities, decoupling the local minimum from the edge ordering.

Algorithm 2 contains the pseudocode of GDB. Lines 1-3 initialize $\mathcal{G}'$ with all edges of the backbone graph G_b using their corresponding probabilities in $\mathcal{G}$. Then, the algorithm iteratively examines every edge of E' and decides its probability: if the optimal assignment of Equation (8) leads to entropy increase, then stp is capped by parameter h (line 10). Otherwise, both the discrepancy and the entropy are reduced by the optimal assignment, and no limit is applied. The algorithm terminates when the improvement of the objective function is below a threshold τ. Each iteration of the for loop requires constant time. Since there are $\alpha|E|$ iterations within each repeat loop, the overall complexity of GDB is $O(M_{steps} \cdot \alpha|E|)$, where M_{steps} is the number of steps of gradient descent (lines 4-11)[10].

Algorithm 2 Gradient Descent Backbone (GDB)

Input: uncertain graph $\mathcal{G} = (V, E, p)$, backbone graph $G_b = (V, E_b)$, entropy parameter h
Output: sparse uncertain graph $\mathcal{G}' = (V, E', p')$
1: $E' \leftarrow \emptyset$
2: **for each** edge $e = (u, v) \in E_b$ **do**
3: $\quad E' \leftarrow E' \cup \{e\}$; $p'_e \leftarrow p_e$
4: **repeat**
5: $\quad \hat{D}_1 = D_1(\mathcal{G}')$
6: $\quad$ **for each** edge $e \in E'$ **do**
7: $\quad\quad stp \leftarrow \frac{\pi(v_0)\hat{\delta}_A(u_0) + \pi(u_0)\hat{\delta}_A(v_0)}{\pi(u_0) + \pi(v_0)}$; $p'_e \leftarrow p_e + stp$
8: $\quad\quad$ **if** $p'_e < 0$ **then** $p'_e \leftarrow 0$
9: $\quad\quad$ **else if** $p'_e > 1$ **then** $p'_e \leftarrow 1$
10: $\quad\quad$ **else if** $H(p'_e) > H(p_e)$ **then** $p'_e \leftarrow p_e + h \cdot stp$
11: **until** $|\hat{D}_1 - D_1(\mathcal{G}')| \leq \tau$

Figure 2 illustrates the execution of GDB for δ^A with $\alpha = 0.6$ and $h = 1$ in the uncertain graph of Figure 2(a). The edges of the backbone graph are depicted with bold, and the absolute discrepancies are shown next to each vertex. At each iteration, GDB examines edges (u_1, u_4), (u_2, u_4), (u_3, u_4) and decides their best probabilities. For example, for edge (u_1, u_4), $p'_{(u_1,u_4)} = p_{(u_1,u_4)} + \frac{\delta(u_1)+\delta(u_4)}{2} = 0.2 + \frac{0.6+0}{2} = 0.5$. Figure 2(b) contains the output of GDB. Note that at this point, Equation (9) cannot further modify the probability of any edge in the output graph. GDB improved the objective function $D_1 = \sum_{u \in V} \delta^2(u)$ from to 0.56 to 0.36 and reduced the entropy from 3.85 to 2.60.



(a) backbone graph G_b (b) output of GDB

Fig. 2. GDB example.

9. Alternative approaches applicable to L1-norm, such as transformation to LP or usage of cutting plane methods [11], require a solver (e.g., simplex), whose running time is prohibitive for the size of our problem.

10. M_{steps} depends on the initial seed, the local minimum at convergence and the gradient step h, which is not fixed and not linear [16].

4.3 Expectation maximization degree

Since GDB only updates the edge probabilities of the backbone graph $G_b = (V, E_b)$ (without inserting or removing edges), it is sensitive to the choice of G_b. On the other hand, *Expectation-Maximization Degree* (EMD) modifies both E_b and the edge probabilities. EMD is inspired by *Expectation-Maximization* [14], which is an iterative optimization framework that estimates two sets of interdependent unknown parameters. In our case, EMD estimates the following sets of parameters: i) the set of edges in the sparsified graph and ii) their probabilities.

Similarly to GDB, the objective function is $D_1 = \sum_{u \in V} \delta^2(u)$. EMD starts with the input backbone graph, and the corresponding probabilities p of $\mathcal{G}$. Then, it enters the iterative process, which consists of two phases. E-phase replaces edges of E_b with edges from $E \setminus E_b$ considering the edge probabilities fixed. The new graph is denoted by $G'_b = (V, E'_b)$. M-phase calls GDB to optimize the edge probabilities considering $G'_b = (V, E'_b)$ as fixed. We denote with $\hat{p}$ and p' the probabilities of the previous and the current iteration respectively.

Equation (9) provides a rule to estimate the values of $\mathbf{p}'$ with respect to some backbone graph G_b and entropy parameter h. Accordingly, we need a rule to generate the graph G'_b that minimizes the objective function with respect to a fixed set of probabilities $\hat{\mathbf{p}}$. To this end, we define the *gain* of an edge $e = (u_0, v_0)$ as follows:

$$g(e)\big|_{p'_e} = \hat{\delta}^2(u_0)\big|_0 - \hat{\delta}^2(u_0)\big|_{p'_e} + \hat{\delta}^2(v_0)\big|_0 - \hat{\delta}^2(v_0)\big|_{p'_e} \quad (10)$$

where p'_e is the probability of Equation (9) and $\hat{\delta}(u_0)\big|_w$ is the degree discrepancy of vertex u_0 for $\hat{p}_e = w$. Intuitively, $g(e)$ quantifies the maximum improvement to D_1, incurred by including e with the best probability p'_e.

Our goal is to swap the edges of the current backbone E_b, with edges from $E \setminus E_b$ that have higher gain. An intuitive approach stores the edges of $E \setminus E_b$ in a dynamic max-heap $\mathcal{H}$ based on $g(e)$. At every iteration, it removes each edge $e = (u, v) \in E_b$ from E_b, and adds it in $\mathcal{H}$. Consequently, it recomputes the gains of all edges incident to u or v that have been affected by e's removal, and updates $\mathcal{H}$ with the new gains. Then, it includes the top of $\mathcal{H}$, e_H, to the new backbone E'_b. Lastly, it updates $\mathcal{H}$ after recalculating the gain of edges incident to e_H. Unfortunately, this approach yields high computational cost for the following reasons:

- The size of $\mathcal{H}$ is $O\big((1-\alpha)|E|\big)$. For small sparsification ratio, $\mathcal{H}$ is prohibitively large.
- For every edge, $O(|E|/|V|)$ heap operations must be performed because each edge affects on average $2 \cdot (1-\alpha)|E|/|V|$ other edges of $\mathcal{H}$. Thus, the total heap overhead of every E-phase is $O\big(\alpha(1-\alpha)|E|^2 \log|V|/|V|\big)$.

EMD alleviates this overhead by maintaining a max-heap $\mathcal{H}_v$ of the vertices V, based on their discrepancy δ. The method follows an approach similar to the above framework. This time however, an edge $e \in E_b$ is swapped with the edge e_s that is incident to the top of $\mathcal{H}_v$, and has the highest gain. This greatly reduces the running time of EMD compared to the above framework:

- $\mathcal{H}_v$ has size $O(|V|)$, which is much smaller than $\mathcal{H}$.
- For every edge, EMD performs only $O(1)$ heap operations because each edge affects the discrepancy of only two vertices. Thus, at every E-phase, the total heap overhead is $O(\alpha|E|\log|V|)$.

Algorithm 3 illustrates EMD. Lines 1-5 initialize δ_A, E' and $\mathbf{p}$. Lines 6-20 contain E-phase. Initially, EMD builds a max-heap $\mathcal{H}_v$ of the vertices V based on δ_A. At the iterative step, for each edge $e = (u, v) \in E_b$, EMD excludes e from E_b, calculates its gain $g(e)$ and updates the corresponding entries of $\mathcal{H}_v$ for the affected vertices u and v (line 12). Then, it retrieves the top of $\mathcal{H}_v$, namely v_H, without removing it from the heap. For all edges $e_r \in E \setminus E_b$ adjacent to v_H, EMD calculates their best probability according to Equation (9). Using this probability, it computes the gain $g(e_r)$. Let $e_{max} = \arg\max\{g(e_r), g(e)\}$ be the edge with the maximum gain among edges e_r and the original edge e. EMD includes e_{max} to the backbone graph and updates the incident vertices u_{max}, v_{max} in $\mathcal{H}_v$ (lines 19-20). This process repeats until all edges have been examined. Based on the new backbone graph, M-phase further optimizes the probability assignment of G'_b by calling GDB(line 21). The procedure terminates when the improvement of an iteration is below a threshold τ. Each iteration of the repeat loop takes $O(\alpha|E|\log|V|)$ and $O(M_{steps} \cdot \alpha|E|)$ for the *E-phase* and *M-phase* respectively, where M_{steps} is the number of steps required for the convergence of GDB. Thus the overall complexity of EMD is $O(E_{steps} \cdot \alpha|E|(\log|V| + M_{steps}))$, where E_{steps} is the number of iterations of lines 6-22.

Algorithm 3 Expectation-Maximization Degree (EMD)

Input: uncertain graph $\mathcal{G} = (V, E, p)$, backbone graph $G_b = (V, E_b)$, entropy parameter h
Output: sparse uncertain graph $\mathcal{G}' = (V, E', p')$

1: $E' \leftarrow \emptyset$
2: initialize δ_A with expected degrees
3: **for each** edge $e = (u, v) \in E_b$ **do**
4: $\quad E' \leftarrow E' \cup \{e\}$; $p'_e \leftarrow p_e$
5: $\quad \delta_A(u) \leftarrow \delta_A(u) - p_e$
6: **repeat**
7: $\quad \hat{D}_1 = D_1(\mathcal{G}')$ // E-phase
8: $\quad$ initialize max-heap $\mathcal{H}_v$ of vertices V based on $|\delta_A|$
9: $\quad$ **for each** $e = (u, v) \in E'$ **do**
10: $\quad\quad \delta_A(u) \leftarrow \delta_A(u) + p'_e$;$\delta_A(v) \leftarrow \delta_A(v) + p'_e$
11: $\quad\quad E_b$.remove(e); $p_e \leftarrow 0$
12: $\quad\quad \mathcal{H}_v$.update(u, v);
13: $\quad\quad v_H \leftarrow \mathcal{H}_v$.top()
14: $\quad\quad$ **for each** edge $e_r \in E \setminus E_b$ adjacent to v_H **do**
15: $\quad\quad\quad w \leftarrow$ probability of Equation (9)
16: $\quad\quad\quad g(e_r)|_w \leftarrow$ gain of Equation (10)
17: $\quad\quad e_{max} = (u_{max}, v_{max}) \leftarrow$ edge of max gain
18: $\quad\quad p_{max} \leftarrow$ probability of e_{max}
19: $\quad\quad E'_b$.add(e_{max}); $p_{e_{max}} \leftarrow p_{max}$
20: $\quad\quad \mathcal{H}_v$.update(u_{max}, v_{max})
21: $\quad \mathcal{G}' = \text{GDB}(\mathcal{G}, G'_b, h)$ // M-phase
22: **until** $|\hat{D}_1 - D_1(\mathcal{G}')| \leq \tau$

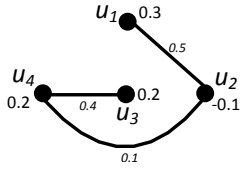
Figure 3 illustrates the execution of E-phase of EMD, for δ^A, in the uncertain graph of Figure 2(a), with the same backbone E_b (in bold) and entropy parameter $h = 1$. At the iterative phase, edge (u_1, u_4) is removed from E_b and vertices u_1, u_4 update their discrepancy to the values of the left table 3(a). The top of $\mathcal{H}_v$ is vertex u_1 and its adjacent

edges are $(u_1, u_4), (u_1, u_2)$ and (u_1, u_3). Equations (9) and (10) compute their best probability and gain respectively (right table of Figure 2(a)). Edge (u_1, u_2) has the highest gain 0.78, therefore it is included in the backbone E'_b. Figure 2(b) demonstrates E'_b with the corresponding probabilities (discrepancies) next to the edges (vertices).

vertex	δ_A
u_1	0.8
u_2	0.4
u_3	0.2
u_4	0.2

$e=(u_1,u_4)$:		
edge	p'	g
(u_1,u_4)	0.4	0.5
(u_1,u_2)	0.5	**0.8**
(u_1,u_3)	0.5	0.5

(a) $\mathcal{H}_v$ and relevant edges at first iteration

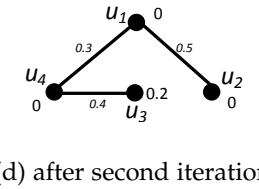


(b) after first iteration of E-phase

vertex	δ_A
u_1	0.3
u_4	0.3
u_3	0.2
u_4	0

$e=(u_2,u_4)$:		
edge	p'	g
(u_1,u_4)	0.3	**0.17**
(u_1,u_3)	0.3	0.08
(u_2,u_4)	0.2	0.05

(c) $\mathcal{H}_v$ and relevant edges at second iteration



(d) after second iteration of E-phase

Fig. 3. EMD example.

Let (u_2, u_4) be the second edge to be examined. The left table of Figure 2(c) shows $\mathcal{H}_v$ after the exclusion of (u_2, u_4). Again, the top of $\mathcal{H}_v$ is vertex u_1. This time the relevant edges are (u_1, u_4) and (u_1, u_3), along with the edge under consideration (u_2, u_4). The right table contains their corresponding best probability and gain. Edge (u_1, u_4) has the highest gain, thus EMD includes it to E'_b. Figure 2(d) contains E'_b. It can be easily verified that the remaining edge (u_3, u_4) does not affect the backbone graph structure (it is removed and reinserted), thus the E-phase is complete. A subsequent M-phase on the updated backbone graph, calculates the new $\mathbf{p}'$ probabilities for the respective edges $(u_1, u_2), 0.55$; $(u_1, u_4), 0.2$; and $(u_3, u_4), 0.55$. Note that the resulting discrepancy $D_1 = 0.01$ has improved considerably. The original objective function Δ_1 has also decreased to 0.2, from 1.2 in the backbone graph of Figure 2(a). Similarly, entropy has decreased to 2.7 from 3.85 in the original graph.

5 Preserving Expected Cuts

EMD cannot be applied for $k > 1$ because our gain definition requires the computation of the discrepancy for all k-cuts that contain an edge, whose number is exponential. In the following we design a new rule that enables the application of GDB to arbitrary values of $k \geq 1$. Since $|\delta(S)|$ is not differentiable at 0, we again utilize the squared discrepancies $\delta^2(S)$, focusing on the absolute discrepancy δ_A. Accordingly, the objective function is:

$$D_k = \sum_{i=1}^{k} \sum_{S \subseteq V, |S|=k} \delta_A^2(S)$$

Its derivative with respect to p'_e for an edge $e = (u_0, v_0)$ is:

$$\frac{\partial D'_k}{\partial p'_e} = -2 \sum_{i=1}^{k} \sum_{\substack{S \subseteq V, |S|=k \\ u_0 \in S, v_0 \notin S}} \delta'_A(S) - 2 \sum_{i=1}^{k} \sum_{\substack{S \subseteq V, |S|=k \\ v_0 \in S, u_0 \notin S}} \delta'_A(S)$$

Changing the probability of edge e from $\hat{p}_e$ to p'_e the discrepancies of all cuts that contain e become:

$$\delta'_A(S) = \hat{\delta}_A(S) + \hat{p}_e - p'_e \tag{11}$$

Setting the derivative equal to zero and solving with respect to p'_e, using Equation (11), we obtain the best probability for e:

$$p'_e = \hat{p}_e + \frac{\sum_{i=1}^{k} \Big(\sum_{\substack{S \subseteq V, |S|=k \\ u_0 \in S, v_0 \notin S}} \hat{\delta}_A(S) + \sum_{\substack{S \subseteq V, |S|=k \\ v_0 \in S, u_0 \notin S}} \hat{\delta}_A(S) \Big)}{\sum_{i=1}^{k} \Big(\sum_{\substack{S \subseteq V, |S|=k \\ u_0 \in S, v_0 \notin S}} 1 + \sum_{\substack{S \subseteq V, |S|=k \\ v_0 \in S, u_0 \notin S}} 1 \Big)} \tag{12}$$

The computation of the above equation is not tractable due to the fact that we need to enumerate all k-cuts that contain edge $e = (u_0, v_0)$. To avoid this we introduce the following enumeration function:

$$\binom{n}{k}_{\Sigma} = \begin{cases} 0 & \text{if } k < 0 \\ \sum_{i=0}^{k} \binom{n}{i} & \text{if } k > 0 \end{cases}$$

We count how many times the discrepancy of an edge is present in the sum of the numerator. If the edge is incident to u_0 or v_0, it will be counted $\binom{n-3}{k-1}_{\Sigma}$ times because we restrict its other vertex from entering S. All other edges, not incident to u_0 or v_0, will be counted $2\binom{n-4}{k-2}_{\Sigma}$ times in the sum, since only one of their incident vertices belongs to S.

$$\sum_{i=1}^{k} \sum_{\substack{S \subseteq V, |S|=k \\ v_0 \in S, u_0 \notin S}} \hat{\delta}_A(S) = \binom{n-3}{k-1}_{\Sigma} \hat{\delta}_A(u_0) + 2\binom{n-4}{k-2}_{\Sigma} \hat{\Delta}(e)$$

where: $$\hat{\Delta}(e) = \sum_{\substack{(u_1, v_1) \in E, \\ u_1 \neq u_0, v_1 \neq v_0}} p_{u_1 v_1} - \hat{p}_{u_1 v_1}$$

Therefore Equation (12) reduces to:

$$p'_e = \hat{p}_e + \frac{\binom{n-3}{k-1}_{\Sigma} \Big(\hat{\delta}_A(u_0) + \hat{\delta}_A(v_0) \Big) + 4\binom{n-4}{k-2}_{\Sigma} \hat{\Delta}(e)}{2\binom{n-2}{k-1}_{\Sigma}} \tag{13}$$

Equation (13) proposes a probability change that weights the degree discrepancies $\hat{\delta}(u_0)$ and $\hat{\delta}(v_0)$ versus the discrepancy of the edges that are not incident to u_0 and v_0, $\hat{\Delta}(e)$. The best probability p'_e can exceed [0,1]. However, since the objective function is again convex (i.e., $\frac{\partial^2 D_k}{\partial {p_e}^2} > 0$), the optimal probability is:

$$p'_e = \Bigg\lfloor \hat{p}_e + h \frac{\binom{n-3}{k-1}_{\Sigma} \Big(\hat{\delta}_A(u_0) + \hat{\delta}_A(v_0) \Big) + 4\binom{n-4}{k-2}_{\Sigma} \hat{\Delta}(e)}{2\binom{n-2}{k-1}_{\Sigma}} \Bigg\rceil_0^1 \tag{14}$$

where the entropy parameter $h \in [0, 1]$ tunes the step size of gradient decent.

The only modification of GDB, is that in line 7 of Algorithm 2, the optimal step *stp* is replaced by the ratio in Equation (13). Special cases of the general rule include $k = 1$, $k = 2$ and $k = n$. For $k = 1$ and absolute discrepancies δ_A, the above equation reduces to Equation (9), and thus takes into account only the degree discrepancies. For $k = 2$, the best probability is:

$${p^2}'_e = \Bigg\lfloor \hat{p}_e + h \frac{(n-2) \Big(\hat{\delta}_A(u_0) + \hat{\delta}_A(v_0) \Big) + 4\hat{\Delta}^e}{(2n-2)} \Bigg\rceil_0^1 \tag{15}$$

For $k = n$, the update rule changes to the following formula, which distributes the cumulative probability of eliminated edges to all the remaining ones. This corresponds to random probability reassignment.

$$p^{n\prime}{}_e = \left\lfloor \hat{p}_e + h \sum_{e_1 \in E' \setminus \{e\}} (p_{e_1} - \hat{p}_{e_1}) \right\rceil_0^1 \quad (16)$$

The general rule of Equation (14) is analytic and does not require any enumeration of cuts. Consequently, the running time of GDB is insensitive to k and depends only on the convergence speed of gradient descent.

6 Experiments

In our evaluation, we use two real undirected uncertain graphs with various sizes, densities, and edge probabilities, summarized in Table 1. Flickr [35] is a social network, where edge probabilities are based on the principle that similar interests indicate social ties. This is the densest dataset; a vertex has on average about 130 neighbours. Twitter [9] is extracted from the popular online micro-blogging service. Probabilities denote the influence that the associated users exert on each other. Although sparser than Flickr, Twitter has higher average probability on the edges.

dataset	vertices	edges	$\lvert E\rvert/\lvert V\rvert$	$\mathbb{E}[p_e]$	$\mathbb{E}[d_u]$
Flickr	78 322	10 171 509	129.89	0.09	22.93
Twitter	26 362	663 766	25.17	0.15	7.71
Synthetic	1 000	77 099	77.1	0.09	12.7
		147 565	147.5		24.3
		269 325	269.3		44.3
		435 336	435.3		71.5

TABLE 1
Characteristics of datasets.

In order to assess the behaviour of the methods in graphs with increasing density, we also use 4 synthetic undirected datasets, whose characteristics are summarized in Table 1. They all stem from an induced subgraph of Flickr with 1000 vertices, where edges have been added between random pairs of vertices, until the density becomes 15, 30, 50, 90 % of the complete graph. The additional edge probabilities follow the same distribution as the original Flickr.

All methods were implemented in C++ and executed in a single core of an Intel Xeon E5-2660 with 2.20GHz CPU and 96GB RAM. Section 6.1 assesses variants of the proposed methods on the objective function of Problem 1 for various values of k in order to identify the best ones depending on the problem characteristics. We refer to the graph characteristics preserved by these objective functions (degrees/cut sizes and entropy) as structural properties. Section 6.2 and 6.3 compare representative variants against the benchmarks on structural properties and common graph queries, respectively. Lastly, Section 6.4 examines the running time of all sparsification techniques and compares it against the query processing time.

6.1 Assessment of proposed techniques

We evaluate the proposed methods GDB (*Gradient Descent Backbone*) and EMD (*Expectation Maximization Degree*) on structural properties using absolute degree discrepancy[11]. As a benchmark, we use LP, the *linear programming* technique of Section 4.1, which, given a backbone graph, yields the optimal probability assignment that minimizes Δ_1 with entropy parameter $h = 1$. Because LP fails to terminate within reasonable time in the real datasets, the experiments are performed on an induced subgraph of Flickr that consists of 5,000 vertices and 655,275 edges, and was extracted using Forest Fire [24].

We use the following notation to differentiate among variants of each method:

- A and R superscripts denote the variants that aim at minimization of the absolute δ_A and relative δ_R discrepancy, respectively. The first type favors nodes with high degree by targeting the absolute error, whereas the second treats all degrees equally.
- t suffix signifies that the backbone graph is generated by Algorithm 1, which ensures connectivity. Absence of this suffix means that the backbone is created by Monte Carlo sampling on the original graph, until reaching of $\alpha|E|$ edges. We refer to this as the *random backbone*.
- $k = \{2, n\}$ subscript denotes GDB preserving k-cuts. Absence of this subscript implies that $k = 1$ (i.e., preservation of the expected degrees).

Table 2 shows the mean absolute error (MAE) of the absolute degree discrepancy δ_A of variants of GDB, EMD and LP for sparsification ratio α ranging from 8% to 64%. GDB_n^A has by far the worst performance as it randomly distributes the missing probabilities to all edges of $\mathcal{G}'$. The accuracy of GDB^A, GDB^R and GDB$_2^A$ is similar and comparable to LP for the corresponding backbone graph. The backbone graphs generated by Algorithm 1 benefit all variants (compared to random backbones). However, for $\alpha = 8\%$, the spanning nature of the graph increases the discrepancy of some vertices that would be otherwise disconnected. In this case, LP and GDB perform better using random backbones as input. In general, EMD improves the accuracy compared to the respective GDB versions by restructuring the backbone. Methods that preserve the relative discrepancy have similar behaviour to those aiming at absolute discrepancy. The variant with the best overall performance is EMD^R-t, which achieves the highest accuracy for all values of $\alpha > 8\%$, shown in bold in Table 2.

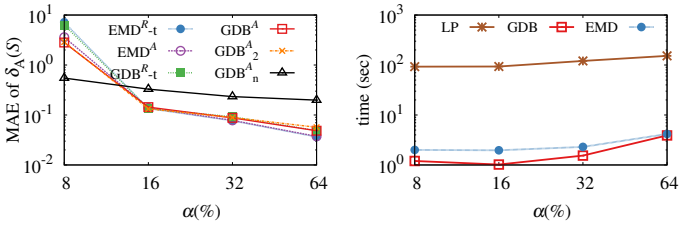
	8%	16%	32%	64%
LP	**1.04**	2.56E-02	4.22E-03	2.70E-04
GDB^A	1.19	2.68E-02	4.38E-03	3.92E-04
GDB^R	1.21	2.67E-02	4.38E-03	3.92E-04
GDB$_2^A$	1.73	4.25E-02	4.74E-03	6.04E-04
GDB$_n^A$	5.78	3.27	2.17	1.8
EMD^A	1.33	2.56E-02	1.22E-03	7.89E-13
EMD^R	1.35	2.18E-02	1.43E-03	1.79E-12
LP-t	2.27	2.95E-04	2.99E-05	2.62E-12
GDB^A-t	3.54	1.78E-03	1.82E-04	1.66E-04
GDB^R-t	2.47	4.11E-04	2.99E-05	2.62E-12
EMD^A-t	2.53	1.83E-04	4.81E-12	8.23E-13
EMD^R-t	2.55	**9.23E-05**	**8.17E-13**	**7.34E-13**

TABLE 2
Mean Absolute Error (MAE) of absolute degree discrepancy $\delta_A(u)$ (Flickr reduced).

Figure 4(a) shows the MAE of the cut discrepancies versus α. Since it is intractable to measure all cuts, we randomly

11. The respective relative discrepancy results are similar and omitted.

select 1000 k-cuts for $k = 1$ up to $|V|$ for each value of k, and we compute the average absolute discrepancy. LP is excluded because it explicitly aims at Δ_1. Similar to Table 2 and for the same reasons, GDB$_n^A$ under-performs the other variants for $\alpha > 8$. The superior performance of GDB$_n^A$ for $\alpha = 8\%$ is explained as follows. In Flickr (also in the reduced graph), the average probability is 0.09; thus, the expected number of edges is approximately $0.09|E|$. For $\alpha < 9\%$ the sparsified graph does not contain enough edges to reach the expected number under any probability assignment. GDB$_n^A$ assigns the maximum probability $p = 1$ to all available edges. On the contrary, the rest of the methods, respecting the constraints on degree and 2-cuts, do not entirely redistribute the missing probabilities. For $\alpha > 8\%$, this ceases to be the case and they yield high accuracy.



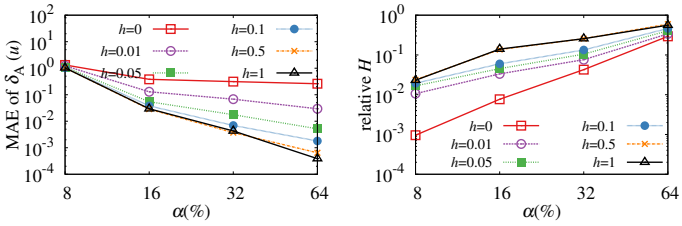
(a) MAE of $\delta_A(S)$ (b) Execution time

Fig. 4. (a) MAE of cut size discrepancy $\delta_A(S)$ and (b) execution time (Flickr reduced).

Figure 4(b) illustrates the running time as a function of α. For both GDB and EMD the running time is independent of the discrepancy (absolute or relative), and the structure of the backbone graph. In addition, GDB optimizes cut sizes using analytic equations; thus its performance is insensitive to k. Accordingly, the plots of GDB and EMD capture all variants of each method. Both techniques are significantly faster than LP, which cannot be applied for large graphs. EMD is slower than GDB since the latter is invoked as a module for the M-phase of the former. However, the overhead is small, which confirms the efficiency of the vertex (as opposed to edge) heap.

In order to compare against the benchmark methods, we select as representative variants of our techniques EMD^R-t and GDB^A. EMD^R-t has the most balanced performance for the settings of Table 2 and Figure 4(a). GDB^A is in general inferior, but it outperforms EMD^R-t for sparsification ratio $\alpha = 8\%$. Moreover, the two variants collectively cover all combinations of discrepancy type and backbone structure. Thus, in the following, the terms EMD and GDB refer to these variants. Since in the remaining we use the real graphs, LP is excluded due its high cost.

A final remark concerns the fine-tuning of entropy parameter $h \in [0, 1]$ in GDB (and EMD, since GDB constitutes one of its modules). Recall from Section 4.2 that h reduces the gradient descent step size when the optimal probability assignment increases entropy. Figure 5(a) plots the MAE of the absolute degree discrepancy δ_A versus the sparsification ratio, for various values of h. In the extreme case of $h = 0$, GDB yields poor performance for δ_A because it discards *any* probability assignment that increases the edge entropy. On the other hand, for $h = 1$ GDB yields the best result on δ_A, but the worst entropy values as shown in Figure 5(b), which plots the relative entropy $\frac{H(\mathcal{G}')}{H(\mathcal{G})}$ versus α. Intermediate values of h span between these two extremes. In the remaining, we set $h = 0.05$ as it has the most balanced performance.

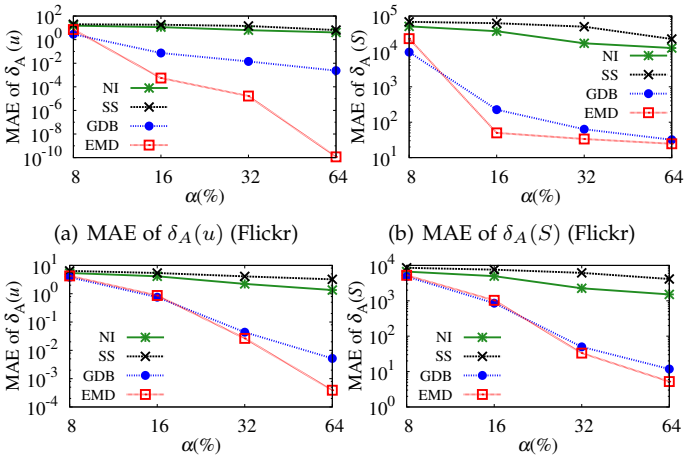


(a) MAE of $\delta_A(u)$ (b) Relative entropy $\frac{H(\mathcal{G}')}{H(\mathcal{G})}$

Fig. 5. Effect of entropy parameter h on GDB (Flickr reduced).

6.2 Comparison with benchmarks on structural properties

We compare EMD and GDB against the benchmarks NI and SS. Recall that NI constitutes the adaptation of a cut-based deterministic sparsification method, whereas SS extends a spanner-based technique to the uncertain setting. Figure 6 plots the MAE of the absolute degree discrepancy $\delta_A(u)$ and the MAE of the cut discrepancy $\delta_A(S)$ (objective function for $k = 1$ and $k \geq 1$, respectively) versus α. The proposed methods consistently outperform the benchmarks for both structural properties in all datasets. The low accuracy of SS can be explained by the fact that it was designed to capture shortest path distances, instead of cuts or degrees. NI is comparable to the proposed techniques for small values of α in Twitter that have high edge probabilities. In these cases, the backbone graph is almost deterministic (most edges have probability 1) and there is little space for improvement by EMD and GDB. For the other settings, NI fails because it assumes unbounded weights. Bounding the maximum weight to 1 seriously affects both its performance and its theoretical guarantees: NI yields a mild probability redistribution that fails to preserve degrees and cuts in practice. Moreover, NI is designed for dense graphs with $E = \Theta(n^2)$, whereas the evaluated datasets are much sparser. As expected from Table 2 and Figure 4(a), in most settings EMD outperforms GDB.



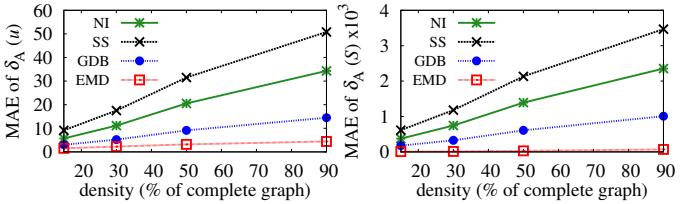
(a) MAE of $\delta_A(u)$ (Flickr) (b) MAE of $\delta_A(S)$ (Flickr)

(c) MAE of $\delta_A(u)$ (Twitter) (d) MAE of $\delta_A(S)$ (Twitter)

Fig. 6. MAE of absolute degree discrepancy $\delta_A(u)$ and absolute cut size discrepancy $\delta_A(S)$ (real datasets).

Figure 7 plots MAE of $\delta_A(u)$ and $\delta_A(S)$ as a function of the graph density (percentage of the complete graph), on the synthetic datasets, which are denser than the real

ones. The sparsification ratio α is fixed to 16%. As the graph density increases, all methods yield increasing error. To see why this happens, consider SS that does not perform probability redistribution. In the simplified case of uniform edge distribution on the vertices with mean probability $\tilde{p}$, MAE($\delta_A(u)$)= $\frac{\tilde{p}(1-\alpha)}{|V|}|E|$. Since all other factors are constant, MAE($\delta_A(u)$) increases linearly with $|E|$. NI that applies limited probability redistribution yields smaller error. Clearly the winner again is EMD with much smoother increase, verifying its robustness for dense graphs.



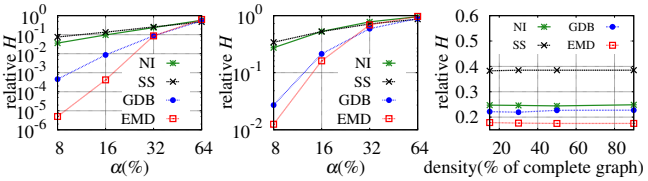
(a) MAE of $\delta_A(u)$ (Synthetic) (b) MAE of $\delta_A(S)$ (Synthetic)

Fig. 7. MAE of absolute degree discrepancy $\delta_A(u)$ and absolute cut size discrepancy $\delta_A(S)$ versus graph density (% of complete graph) (synthetic datasets).

Figure 8 plots the relative entropy of the sparsified graphs versus the sparsification ratio (real datasets) and the density (synthetic datasets). The relative entropy of a sparsified graph $\mathcal{G}'$ is the ratio $\frac{H(\mathcal{G}')}{H(\mathcal{G})}$, where $\mathcal{G}$ is the original graph. EMD and GDB have at least an order of magnitude less entropy for small α compared to NI and SS which overall perform similarly. This is expected since our methods aim at reducing entropy, unlike the competitors that are designed for deterministic graphs (zero entropy). Relative entropy increases with α, always remaining less than 1. Figure 8(c) plots the entropy of the synthetic graphs with $\alpha = 16\%$. The relative entropy is constant as the percentage of edges in the sparsified graph remains the same.

6.3 Comparison with benchmarks on queries

We evaluate the following graph queries. (i) *Pagerank* (PR) is a measure of the node's influence in the graph and has been widely used to rank Web page search results according to their links [31]. (ii) *Shortest path distance* (SP) is the average



(a) Flickr (b) Twitter (c) Synthetic

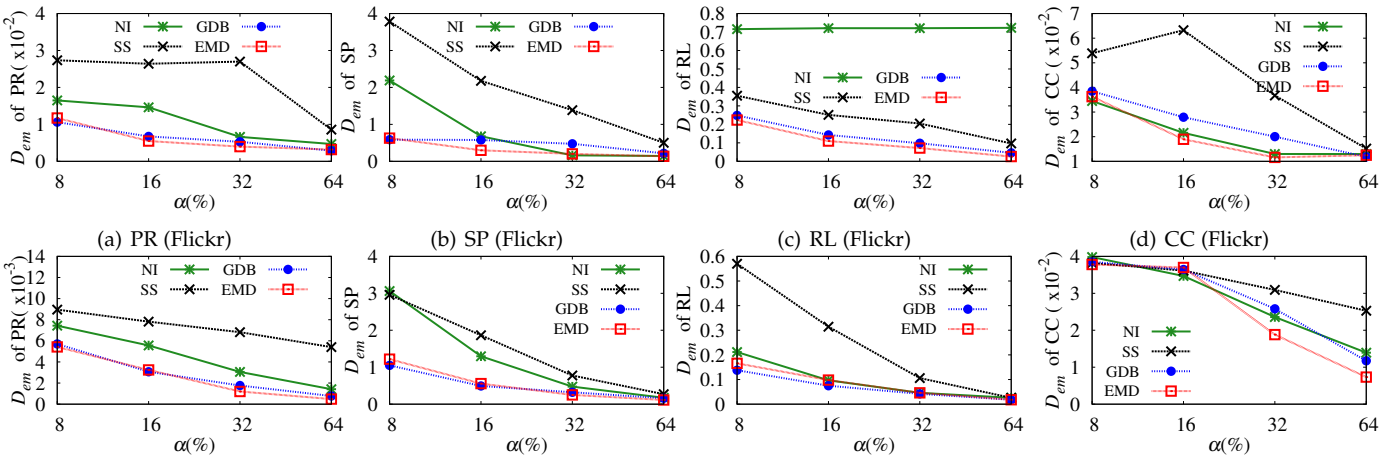
Fig. 8. Graph entropy H (real and synthetic datasets).

shortest distance between a pair of vertices in all worlds excluding the ones that disconnect them. SP is essential for any task involving shortest path computations [35]. (iii) *Reliability* (RL) is the probability that a vertex is reachable from another in the graph. It is a common metric for the resilience of router networks. (iv) *Clustering coefficient* (CC) is the ratio of the number of edges between the neighbours of a vertex to the maximum number of such links. CC constitutes an important metric for search strategies and social networks [22]. Although we evaluate CC and PR on *all* vertices of $\mathcal{G}$, we choose 1000 random vertex pairs for the evaluation of SP and RL because the evaluation on all pairs would be too expensive to terminate for our datasets.

Quality results. Let $\mathcal{G}'$ be a sparsified subgraph of $\mathcal{G}$, and a query Q. Q is evaluated through Monte-Carlo sampling on 500 possible worlds of both $\mathcal{G}'$ and $\mathcal{G}$. The various outcomes of Q in different samples of $\mathcal{G}'$ form a cumulative distribution $F_{\mathcal{G}',Q}(x)$ of results. To quantify the similarity of $\mathcal{G}'$ to $\mathcal{G}$ with respect to Q, we have to measure the difference between $F_{\mathcal{G}',Q}(x)$ and $F_{\mathcal{G},Q}(x)$. To this end, a robust metric is the *earth mover's distance* D_{em} [38]. Intuitively, D_{em} measures the minimum change that aligns the two distributions. Formally, let $\{x_0, x_1, \cdots, x_M\}$ be the ordered set of all observed results of Q in $\mathcal{G}$ and $\mathcal{G}'$. To compute D_{em} we apply the following equation:

$$D_{em}(\mathcal{G}, \mathcal{G}', Q) = \sum_{i=1}^{M} \left| F_{\mathcal{G},Q}(x_i) - F_{\mathcal{G}',Q}(x_i) \right| \cdot (x_i - x_{i-1})$$

Figure 9 plots D_{em} versus the sparsification ratio. Each row of diagrams corresponds to a dataset and each column to a query. With few exceptions, GDB and EMD outperform the benchmarks for all settings, usually by a wide margin. Moreover, the diagrams are consistent with those on struc-



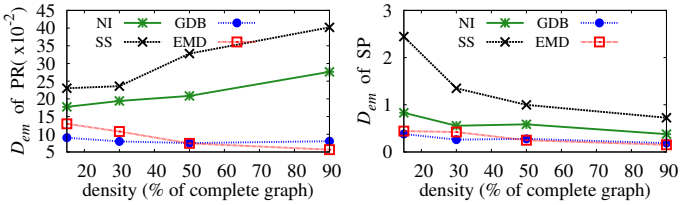
(a) PR (Flickr) (b) SP (Flickr) (c) RL (Flickr) (d) CC (Flickr)

(e) PR (Twitter) (f) SP (Twitter) (g) RL (Twitter) (h) CC (Twitter)

Fig. 9. Earth mover's distance D_{em} for Pagerank (PR), Shortest Path distance (SP), Reliability (RL) and Clustering Coefficient (CC) (real datasets).

tural properties, confirming the correlation of our objective functions with the performance of the sparsified graphs for diverse queries. SS yields the highest error even on the SP metric, which constitutes its focus. The main cause for its poor performance is that it does not involve any probability redistribution. Although NI achieves good approximation for CC, it usually introduces large error for the rest of the queries. EMD is the winner for high sparsification ratio, while GDB is preferable for small α in most queries. This is in accordance with Figure 6, where GDB preserves better the structural properties for $\alpha = 8\%$.

Figure 10(a) (resp. Figure 10(b)) illustrates D_{em} of PR (resp. SP) on the synthetic datasets, as a function of density for $\alpha = 16\%$. The proposed techniques clearly yield smaller error than the benchmarks. Observe that PR is node centric and highly correlated with the degree; thus, the diagram of Figure 10(a) is similar to that of Figure 7(a). The plots of CC are similar to those of PR and omitted. On the other hand, the error of SP decreases with increasing density because more alternative short paths are available, due to the abundance of edges. RL has practically zero error for all methods, since the dense graph has reliability almost 1 for all pairs of vertices.



(a) PR (Synthetic) (b) SP (Synthetic)

Fig. 10. Earth mover's distance D_{em} for Pagerank (PR) and Shortest Path distance (SP) (synthetic datasets).

Variance results. We assess the performance of the various sparsifiers with respect to the variance of an MC estimator on the above queries. Specifically, due to the randomized nature of MC estimators, different executions of the same experiment may yield different results. The variance quantifies the deviation of results from the mean. Let $\Phi(\mathcal{G})$ denote the result of an MC simulation on an uncertain graph $\mathcal{G}$.

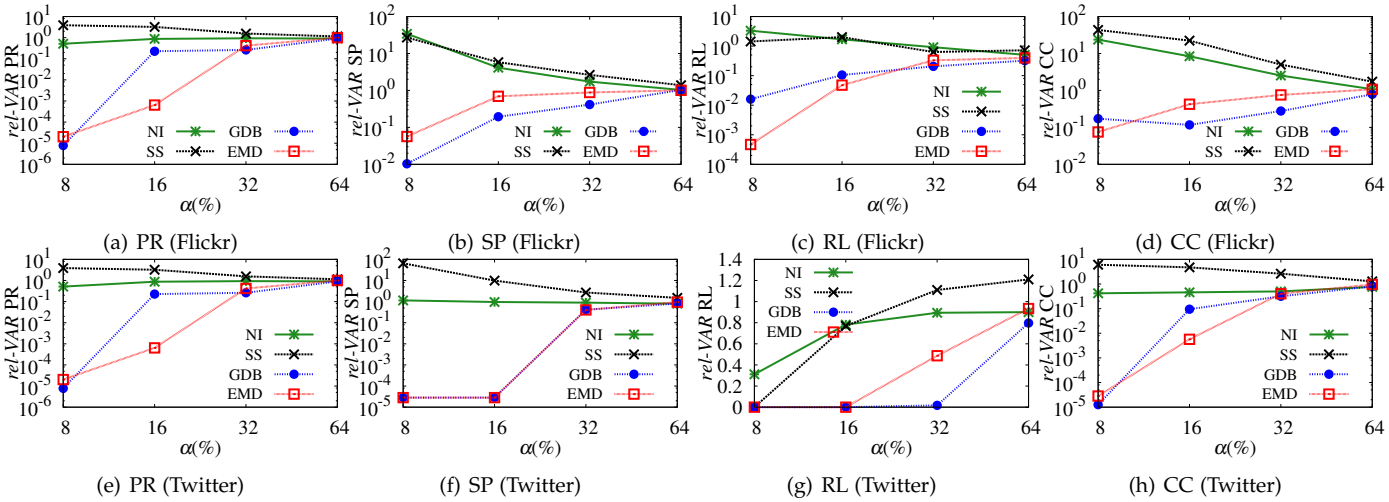
A sparsified graph with low variance on MC estimator[12] implies the need of fewer samples for accurate estimation. Specifically, according to the theory of MC simulations, the unknown expected value of a query in $\mathcal{G}$ belongs to the *confidence interval* $CI = [\Phi(\mathcal{G}) - 1.96\sigma(\mathcal{G})/\sqrt{N}, \Phi(\mathcal{G}) + 1.96\sigma(\mathcal{G})/\sqrt{N}]$ with probability 95%, where $\sigma(\mathcal{G})$ is the variance of the query in $\mathcal{G}$ and N is the number of samples. Let the *confidence width* CW be the length of the confidence interval, i.e., $CW = 2CI = 3.92\sigma(\mathcal{G})/\sqrt{N}$. In order to achieve the same level of accuracy CW between the original graph $\mathcal{G}$ and the sparsified $\mathcal{G}'$, we require

$$CW = CW' \rightarrow \sigma(\mathcal{G})/\sqrt{N} = \sigma(\mathcal{G}')/\sqrt{N'} \rightarrow N'/N = \left(\sigma(\mathcal{G}')/\sigma(\mathcal{G})\right)^2. \quad (17)$$

Intuitively, small relative variance $\frac{\sigma(\mathcal{G}')}{\sigma(\mathcal{G})}$ implies the need of fewer samples for accurate estimation. This is why variance is among the most important metrics for the quality of MC simulation (see [18], [25]). However, calculating the actual variance is intractable. Thus, we follow a strategy similar to [25] for an unbiased estimator of the variance of $\Phi(\mathcal{G})$, denoted as $\hat{\sigma}(\mathcal{G})$. Specifically, we run each estimator ($\Phi_i(\mathcal{G})$) 100 times. Then the unbiased estimator of the variance of $\Phi(\mathcal{G})$ is $\hat{\sigma}(\mathcal{G}) = \sum_{i=1}^{100}(\Phi_i(\mathcal{G}) - \bar{\Phi}(\mathcal{G}))^2/99$, where $\bar{\Phi}(\mathcal{G})$ denotes the mean of $\Phi_i(\mathcal{G})$ for $(i = 1, \cdots, 100)$.

Figure 11 illustrates the relative variance of the queries versus the sparsification ratio α. Let $\hat{\sigma}(\mathcal{G})$ and $\hat{\sigma}(\mathcal{G}')$ denote the variance of the MC estimator on the original and the sparsified graph respectively. The y-axis represents the relative variance, i.e., the ratio $\frac{\hat{\sigma}(\mathcal{G}')}{\hat{\sigma}(\mathcal{G})}$. Each row of diagrams corresponds to a dataset and each column to a query. Consistently, EMD and GDB drop the variance of the original graph up to several orders of magnitude. On the other hand, NI and SS have, in most cases, higher variance than the original graph. The justification of this result is based on the fact that NI and SS perform limited (if any) probability redistribution. Assume for instance the SP query. During evaluation we only consider *reliable* possible worlds for which the distance of the query points is less than infinite. Sparsification without probability redistribution reduces the

12. Slightly abusing notation, we use the term "variance of $\mathcal{G}$" to imply the variance of a query estimator on a graph $\mathcal{G}$.



(a) PR (Flickr) (b) SP (Flickr) (c) RL (Flickr) (d) CC (Flickr)

(e) PR (Twitter) (f) SP (Twitter) (g) RL (Twitter) (h) CC (Twitter)

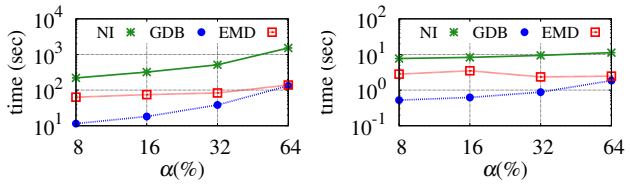
Fig. 11. Relative variance for Pagerank (PR), Shortest Path distance (SP), Reliability (RL) and Clustering Coefficient (CC).

number of possible worlds for which a path exists. Consequently, a bigger proportion of possible worlds is discarded, increasing the variance of the estimator, which now depends on fewer samples. Moreover, according to the small world phenomenon, the majority of shortest paths are small (i.e., less than 10). Thus, the removal of edges is more likely to affect the the probability of short paths. This in turn increases the probability of larger paths, inflating the variance.

Our techniques alleviate the above short-comings by applying aggressive probability redistribution that preserves the expected number of edges, reducing the entropy of the sparsified graphs. This results in many edges having probability one. For instance, in Twitter with $\alpha = 8\%$, 75% of the edges of GDB have probability 1. In comparison, in NI only 25% of the edges are deterministic. As α increases, fewer edges reach probability 1; thus, the variance of EMD and GDB increases. This result is very important and highlights one of the core differences of our methods compared to the competitors: the goal for entropy reduction.

6.4 Running time

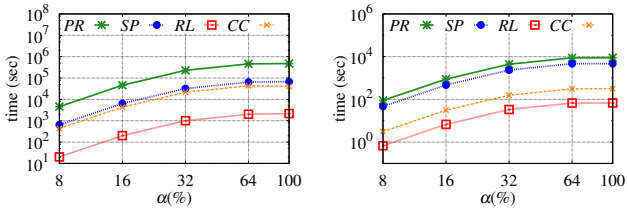
The next experiment measures the running time of GDB, EMD and NI in the real graphs. As shown in Figure 12, the proposed methods usually terminate within a minute, whereas NI is more than an order of magnitude slower. SS is omitted from the diagrams because it requires several hours to terminate. For GDB the running time grows with the sparsification ratio α because the cost of each step is proportional to $\alpha|E|$. For EMD the effect of increasing α is attenuated by the smaller number of E_{steps} that are required for convergence. This is expected because as the sparsified graph becomes denser, more edges can compensate towards the objective by altering their probabilities; thus, the structure of the graph is less important. For instance, in Twitter the number of E_{steps} is $\{4, 4, 2, 1\}$ for $\alpha = \{8, 16, 32, 64\}$ respectively.



(a) Flickr (b) Twitter

Fig. 12. Execution time of sparsification algorithms (real graphs).

Figures 13(a) and (b) plot the query execution time versus the sparsification ratio α, on Flickr and Twitter respectively[13]. Due to the smaller variance of the queries in the sparsified graphs (as illustrated in Figure 11), fewer samples are required for accurate Monte Carlo simulation, especially for smaller α. Based on Equation 17 and Figure 11, we have utilized $N = \{10, 100, 500, 1000, 1000\}$ samples for $\alpha = \{8, 16, 32, 64, 100\}$ respectively. Note that $\alpha = 100\%$ refers to the original, unsparsified graph. In all cases the query processing cost drops as the number of edges in the sparsified graph decreases. Compared with Figure 12, with the exception of *RL* that terminates very fast for $\alpha = 8$, graph sparsification is much faster than query execution, even for small α. For instance, in Flickr *PR*, *SP* and *CC* require one to four orders of magnitude more processing time than the corresponding sparsification through EMD, the slowest of our techniques. As expected, queries in Flickr are slower than the corresponding ones in Twitter due to Flickr's larger size.



(a) Flickr (b) Twitter

Fig. 13. Query processing time (real graphs).

Summarizing the experiments, as shown in Table 2 and Figures 4, 6 and 7, the proposed techniques capture well the structural properties of the input uncertain graph, even if they do not constitute their explicit optimization criterion. For instance, variants that aim at the relative discrepancy δ_R, e.g. EMD^R-t, also preserve the absolute one δ_A. According to Figures 9-10, the preservation of structural properties leads to accurate results for various queries with different characteristics. Moreover, by reducing the entropy of the uncertain graph (Figure 8), our methods decrease the variance of the MC estimator of all evaluated queries (Figure 11). This reduces the processing time, as considerably fewer samples are required for accurate query estimation (Figure 13). As opposed to the proposed methods, techniques based on deterministic sparsification usually fail, in terms of result quality, variance and execution time. Finally, our algorithms are efficient and applicable to large uncertain graphs.

7 Conclusion

Sparsification has often been used to reduce the size of deterministic graphs and facilitate efficient query processing. However, it has not been applied previously to uncertain graphs, although they incur significantly higher cost for common query and mining tasks. This paper introduces novel sparsification techniques that, given an uncertain graph $\mathcal{G} = (V, E, p)$ and a parameter $\alpha \in (0, 1)$, they return a subgraph $\mathcal{G}' = (V, E', p')$, such that $E' : E' \subset E, |E'| = \alpha|E|$. $\mathcal{G}'$ preserves the structural properties of $\mathcal{G}$, has less entropy than $\mathcal{G}$, and can approximate the result of various queries on $\mathcal{G}$.

The proposed methods, GDB (*Gradient Descent Backbone*) and EMD (*Expectation Maximization Degree*), involve an initialization and a probability assignment step. First, a backbone deterministic graph G_b with $\alpha|E|$ edges is generated. In order to obtain $\mathcal{G}'$, GDB assigns probabilities to the edges of G_b aiming at preserving the expected vertex degrees or cut sizes, while reducing the entropy. In addition to assigning probabilities, EMD also changes the structure of G_b by adding or removing edges. An extensive experimental evaluation with real and synthetic uncertain graphs confirms that GDB and EMD consistently outperform benchmarks adapted from the deterministic graph literature, on several graph queries and metrics. In the future, we intend to investigate sparsification of multigraphs and weighted uncertain graphs.

13. The graphs have been sparsified using EMD.

ACKNOWLEDGMENTS

This work was supported by GRF grants 16201615, 16205117 from Hong Kong RGC.

REFERENCES

[1] I. Abraham, D. Durfee, I. Koutis, S. Krinninger, and R. Peng. On fully dynamic graph sparsifiers. In *FOCS* 2016.
[2] K. J. Ahn, S. Guha and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *PODS* 2012.
[3] K. J. Ahn, S. Guha and A. McGregor. Spectral sparsification in dynamic graph streams. In *RANDOMAPPROX* 2013.
[4] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *DCG*, 9(1):81–100, 1993.
[5] S. Baswana and S. Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.
[6] J. Batson, D.A. Spielman, N. Srivastava and S.H. Teng. Spectral sparsification of graphs: theory and algorithms. *Communications of the ACM*, 56(8):87–94, 2013.
[7] A. A. Benczúr and D. R. Karger. Approximating st minimum cuts in $\tilde{o}$ (n^2) time. In *STOC*, 1996.
[8] A. A. Benczúr and D. R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.
[9] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *SIGKDD*, 2014.
[10] F. Bonchi, G. D. F. Morales, A. Gionis, and A. Ukkonen. Activity preserving graph simplification. *DMKD*, 27(3):321–343, 2013.
[11] S. Boyd and L. Vandenberghe. Convex optimization. *Cambridge university press*, 2004.
[12] J. B. Crockett, H. Chernoff, et al. Gradient methods of maximization. *Pacific J. Math*, 5:33–50, 1955.
[13] D. M. Dang, K. Jackson, and M. Mohammadi. Dimension and variance reduction for Monte Carlo methods for high-dimensional models in finance. *Applied Mathematical Finance*, 22:6, 2015.
[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc*, 1–38, 1977.
[15] W. S. Fung, R. Hariharan, N. J. Harvey, and D. Panigrahi. A general framework for graph sparsification. In *STOC*, 2011.
[16] G.N. Grapiglia, J. Yuan and Y. Yuan. On the worst-case complexity of nonlinear stepsize control algorithms for convex unconstrained optimization. *Optimization Methods and Software*, 31:591–604, 2016.
[17] C. Hübler, H.-P. Kriegel, K. Borgwardt, and Z. Ghahramani. Metropolis algorithms for representative subgraph sampling. In *ICDM*, 2008.
[18] A. W. M. H. Kahn. Methods of reducing sample size in monte carlo computations. *Journal of the Operations Research Society of America*, 1(5):263–278, 1953.
[19] P. Hintsanen and H. Toivonen. Finding reliable subgraphs from large probabilistic graphs. *DMKD*, 17(1):3–23, 2008.
[20] M. Kapralov and R. Panigrahy. Spectral sparsification via random spanners. *ITCS*, 2012.
[21] M. Kapralov, Y. T. Lee, C. N. Musco, C. P. Musco and A. Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1): 456–477, 2017.
[22] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *Science*, 311(5757):88–90, 2006.
[23] I. Koutis. Simple Parallel and Distributed Algorithms for Spectral Graph Sparsification. *SPAA*, 2014.
[24] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *SIGKDD*, 2006.
[25] R.-H. Li, J. X. Yu, R. Mao, and T. Jin. Efficient and accurate query evaluation on uncertain graphs via recursive stratified sampling. In *ICDE*, 2014.
[26] G. Lindner, C. L. Staudt, M. Hamann, H. Meyerhenke and D. Wagner. Structure-preserving sparsification of social networks. In *ASONAM*, 2015.
[27] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen. Sparsification of influence networks. In *SIGKDD*, 2011.
[28] G.L. Miller, R. Peng, A. Vladu and S.C. Xu. Improved Parallel Algorithms for Spanners and Hopsets In *SPAA*, 2015.
[29] A. Mukherjee, P. Xu, and S. Tirthapura. Mining maximal cliques from an uncertain graph. In *ICDE*, 2015.
[30] H. Nagamochi and T. Ibaraki. A linear-time algorithm for finding a sparsek-connected spanning subgraph of a k-connected graph. *Algorithmica*, 7(1-6):583–596, 1992.
[31] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: bringing order to the web. 1999.
[32] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The pursuit of a good possible world: Extracting representative instances of uncertain graphs. In *SIGMOD*, 2014.
[33] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. Uncertain graph processing through representative instances. *ACM Transactions on Database Systems (TODS)*, 40(3):20, 2015.
[34] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of graph theory*, 13(1):99–116, 1989.
[35] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *PVLDB*, 3(1-2):997–1008, 2010.
[36] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *ESA*, 2004.
[37] N. Ruan, R. Jin, and Y. Huang. Distance preserving graph simplification. In *ICDM*, 2011.
[38] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *IJCV*, 40(2):99–121, 2000.
[39] V. Satuluri, S. Parthasarathy, and Y. Ruan. Local graph sparsification for scalable clustering. In *SIGMOD*, 2011.
[40] D. A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM J. Comput.* 40(6): 1913-1926, 2011.

Panos Parchas is a Software Engineer at Amazon Web Services, Berlin, Germany. He received his Ph.D. from the Department of Computer Science and Engineering, HKUST in February 2017. He obtained a five-year diploma in Electrical and Computer Engineering from NTUA, Greece. His research interests include uncertain graph management and time-dependent road networks.

Nikolaos Papailiou received his Ph.D. from the Department of Electrical and Computer Engineering, NTUA, Greece in 2016. He was post-graduate researcher at HKUST. He obtained a five-year diploma from NTUA. His research interests include distributed graph databases, large-scale data analytics, application elasticity using cloud resources and uncertain graph algorithms.

Dimitris Papadias is a Professor of Computer Science and Engineering, HKUST. Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the UCSD (California), the Technical University of Vienna, the NTUA, Queen's University (Canada), and University of Patras (Greece). He has served in the editorial boards of the VLDBJ, IEEE TKDE, and Information Systems.

Francesco Bonchi is Research Leader at the ISI Foundation, Turin, Italy, where he leads the "Algorithmic Data Analytics" group. He is also Scientific Director for Data Mining at Eurecat (Technological Center of Catalunya), Barcelona. Before he was Director of Research at Yahoo Labs in Barcelona, Spain. His recent research interests include mining query-logs, social networks, and social media, as well as the privacy issues related to mining these kinds of sensible data.

APPENDIX

Algorithm 4 Nagamochi Ibaraki (NI)

Input: graph $G_w = (V, E, w)$, approximation parameter ϵ
Output: sparse graph $G'_w = (V, E', w')$

1: $E' \leftarrow \emptyset$; $E_c \leftarrow E$; $F_0 = \emptyset$
2: $r = 0$
3: **while** $E_c \neq \emptyset$ **do**
4: $r \leftarrow r + 1$
5: compute a spanning forest F_r of E_c such that $(F_r \setminus F_{r-1}) \cap E_c = \emptyset$
6: **for each** edge $e \in F_r$ **do**
7: $w_e \leftarrow w_e - 1$
8: **if** $w_e = 0$ **then**
9: sample e with probability $\ell_e = \min\{\log|V|/(\epsilon^2 \cdot r), 1\}$
10: **if** e is sampled **then**
11: $E' \leftarrow E' \cup \{e\}$ with $w'_e \leftarrow w_e/\ell_e$
12: **else**
13: discard e
14: $E_c \leftarrow E_c \setminus \{e\}$

This section provides details of the benchmark methods NI [30] and SS [5]. Algorithm 4 contains the core iterative process of NI. The method requires as input an approximation parameter ϵ, which is initially tuned depending on our parameter α; $\epsilon = \sqrt{|V| \log^2 |V|/\alpha|E|}$. The result set E' is initially empty and the set of available edges $E_c \leftarrow E$. At each iteration r, a spanning forest F_r is computed on the set E_c with the requirement that if an edge e appears in F_{r-1}, then it must also appear in F_r (contiguous spanning forests). Each time an edge e is covered by a spanning forest, w_e is reduced by one. When w_e becomes 0, e is sampled with probability $\ell_e = \min\{\log|V|/(\epsilon^2 \cdot r), 1\}$. If e is selected, then line 11 adds it to the result set. Otherwise, e is discarded. The iterative process stops when all edges of E have been examined. Intuitively, the sampling probability ℓ_e approximates the connectivity of edge e; if e belongs to a sparse subgraph, it is covered by spanning trees of early iterations, therefore it is sampled with high probability. On the other hand, an edge in a dense component is covered after several iterations r, thus it's sampling probability is significantly smaller.

Algorithm 5 presents the main process of SS [5], which computes a $(2t-1)$-*spanner* of $O(t \cdot m^{1+1/t})$ expected size. The algorithm performs $t-1$ iterations, incrementally forming clusters of vertices with the minimum edge that connects them. C_i maintains the set of vertex clusters for iteration i. Initially, C_i contains $|V|$ sets of individual vertices (line 3) and the spanner E' is empty. At each iteration, a set of clusters, R_i, is selected with probability $n^{-1/t}$ for each cluster. For each vertex $u \notin R_i$ that is not yet connected to the spanner, SS examines its neighbours. If none of them is in R_i, then for each adjacent cluster $c \in C_{i-1}$, SS adds the least weight edge to E', and updates the clusters (lines 22-25). If u has an adjacent node in R_i, the least weight edge $e = (u, v \in R_i)$ is added to the spanner (line 10). Then, SS iterates over all edges of u, adding for each adjacent cluster c, the minimum weight edge e', if $w(e') < w(e)$ (line 17). To ensure that the resulting spanner is connected, the algorithm joins all remaining clusters using the connecting edge with the minimum weight (lines 26 - 28).

Recall from Section 3.3 that, in both methods, the expected number of edges is given in O-notation, thus the sparsified graph is not guaranteed to reach $\alpha|E|$ edges. To ensure this, we run Algorithms 4 and 5 with modified parameters to approximate the smallest ϵ and t that contains $E' < \alpha|E|$. The remaining $\alpha|E| - |E'|$ edges are sampled using the original probabilities.

Algorithm 5 Spanner Sparsification (SS)

Input: uncertain graph $G = (V, E, w)$, t
Output: sparse spanner $G' = (V, E', w)$

1: $E' \leftarrow \emptyset$ spanner edges
2: $V_S \leftarrow \emptyset$ spanner vertices
3: $C_0 = \{\{u\} | u \in V\}$ clusters
4: **for** i=1 to t-1 **do**
5: $R_i \leftarrow$ sample C_{i-1} with probability $n^{-1/t}$
6: $C_i \leftarrow R_i$
7: **for each** $u \in V \setminus V_S$ and $u \notin R_i$ **do**
8: $N_R \leftarrow N(u) \cap R_i$ neighbours of u in R_i
9: **if** $N_R \neq \emptyset$ **then**
10: $e \leftarrow$ minimum weight edge $u, v \in N_R$
11: $E' \leftarrow E' \cup e$, $V_S \leftarrow V_S \cup u$
12: $E \leftarrow E \setminus E(u, N_R)$
13: merge clusters $C_i(v), C_i(u)$
14: **for each** cluster $c \in C_{i-1}$ and $c \notin R_i$ **do**
15: $e' \leftarrow$ minimum weight edge
16: **if** $w(e') < w(e)$ **then**
17: $E' \leftarrow E' \cup e'$
18: $E \leftarrow E \setminus E(u, c)$
19: merge clusters $c, C_i(u)$
20: **else**
21: **for each** cluster $c \in C_{i-1}$ **do**
22: $e \leftarrow$ minimum weight edge $u, N(u) \cap c$
23: $E' \leftarrow E' \cup e$, $V_S \leftarrow V_S \cup u$
24: $E \leftarrow E \setminus E(u, c)$
25: merge clusters $c, C_i(u)$
26: **for each** $c \in C_{t-1}$ **do**
27: $e \leftarrow$ minimum weight edge $c, v \in N(c)$
28: $E' \leftarrow E' \cup e$