Graphs and Uncertainty

Silviu Maniu

Graphes et incertitude

Graphs and Uncertainty

Habilitation à diriger des recherches de l'Université Paris-Saclay

présentée et soutenue à Gif-sur-Yvette,
le 01/06/2022, par

Silviu MANIU

Composition du jury

Sihem AMER-YAHIA Directrice recherche, CNRS & LIG	Rapportrice
Francesco BONCHI Research Director, ISI Foundation	Rapporteur
Michalis VAZIRGIANNIS Professeur, LIX, École Polytechnique	Rapporteur
Sarah COHEN-BOULAKIA Professeure, LISN, Université Paris-Saclay	Examinatrice
Michael BENEDIKT Professeur, Université d'Oxford	Examinateur

Contents

1 A Brief Overview of Uncertainty in Graphs

We discuss in this manuscript some research problems which are situated at the intersection between *graphs* as a data model and *uncertainty*. Uncertainty on graphs can occur either when *data* of the model is error-prone or incomplete, or when the *model* that generates behavior is not a deterministic one.

Indeed, graphs have emerged as one of the most intuitive ways of representing data, especially for application domains that are more and more prevalent in recent years; their properties are the basis of *network science* research [Easley and Kleinberg, 2010, Barabási and Pósfai, 2016]. Some – non exhaustive – examples include:

- *Online social networks* [Domingos and Richardson, 2001]: nodes in such graphs represent humans or *online application users* and the edges are their relationships – also potentially having different semantics.
- *Infrastructure networks* [Añez et al., 1996]: graphs encode *physical connections* – road networks, power-line networks – where nodes can be physical features and the edges are infrastructure links between them .
- *IoT networks* [Ghosh et al., 2007]: for example, cellular connections between mobile devices.
- *Biological networks* [Asthana et al., 2004]: in biological networks, how proteins interact with each other can be represented as a graph where nodes are proteins and the edges are interactions .

The principal task on graph data is to *query* it, where the input can be either a node, a pair of nodes or even graph motifs. Efficiency is a major challenge and various algorithms have been studied in the last few decades, e.g.: reachability analysis [Cohen et al., 2003], shortest path computation [Dijkstra, 1959], frequent subgraph mining [Kuramochi and Karypis, 2001], or graph pattern mining [Barceló et al., 2014]. Importantly, a large subset of queries allowing to match regular expressions over graphs can be solved in polynomial data complexity [Barceló, 2013].

Other tasks on graphs can be classified as *combinatorial*. Usually, this means selecting a subset of nodes or edges in the graph in order to optimize some function on the graph. Classical examples include vertex cover, spanning trees, or maximum cut; in this manuscript we will consider the problem where a subset of nodes has to be chosen so as to maximize a submodular function over those nodes – the particular instance we consider is *influence maximization* [Kempe et al., 2003]. Most interesting combinatorial problems on graphs are

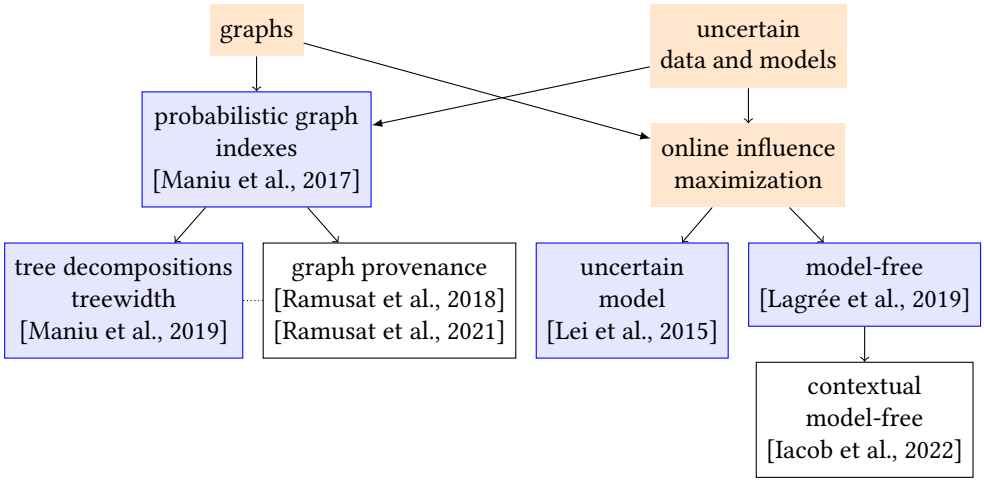


Figure 1.1: Diagram illustrating the context of the research presented in this manuscript.

NP-hard, some allowing polynomial approximation (e.g., the influence maximization problem allows one, via the greedy algorithm). Another avenue is to exploit the structure of the graphs. Generally, problems are easier on trees than on general graphs. The notion of treewidth [Robertson and Seymour, 1984] captures how "close" a graph is to a tree, and the most important result is that some hard problems become linear on graphs having bounded treewidth [Arnborg and Proskurowski, 1989]. We will discuss later whether real-world graphs have a reasonably low treewidth and how to best exploit their structure.

An orthogonal issue in graph data in the real world is how reliable they are. For instance, *data uncertainty* is inherent in the applications described above. Viral marketing techniques [Liben-Nowell and Kleinberg, 2007] study the purchase behavior of users in a social network. An edge between two users encodes the *probability* that one's purchasing behavior is influenced by the other – generally, this is not a deterministic behavior. In IoT networks, the connections between any two devices may or may not be established, as factors such as signal interference and antenna power may affect it. Determining whether a protein has interacted with another is usually done via several scientific studies, that do not always give the same result. In transport networks, traversal time is not constant, and is affected by deviations due to traffic patterns and unforeseen events [Hua and Pei, 2010]. Another example of uncertainty can be inherent to the *process* or the function to be optimized. For instance, in the influence maximization task, the influence process is not a deterministic one: influence may succeed at some time, and fail at another. This is an example where, for a combinatorial task on a graph, the function to be optimized is uncertain. Hence, solving tasks on such graphs without considering the above uncertainty issues can lead to *incorrect* answers.

We will now introduce the research challenges for the two types of uncertainty on graphs.

> This manuscript is intended to show the main strands of the research I have worked on since my PhD graduation in 2012. It is intended to give a high-level view of the research and collaborations that have occurred since then. The main technical details such as proofs and

extensive experiments are skipped, and can be referred to in their corresponding references.

A general overview of my research is presented in Figure 1.1. Two main directions are visible, corresponding to the main two chapters of the manuscript. The first is my research on probabilistic graphs indexes and evaluation of treewidth and partial tree decomposition in real-world graphs. The second is the general area of *online influence maximization* in which I mainly discuss solutions based on the multi-armed bandit framework. These correspond to the light blue boxes in the above figure. White boxes represents research that is linked, but has not been presented in detail here.

The current manuscript mainly covers contributions resulting from the following research collaborations and supervisions:

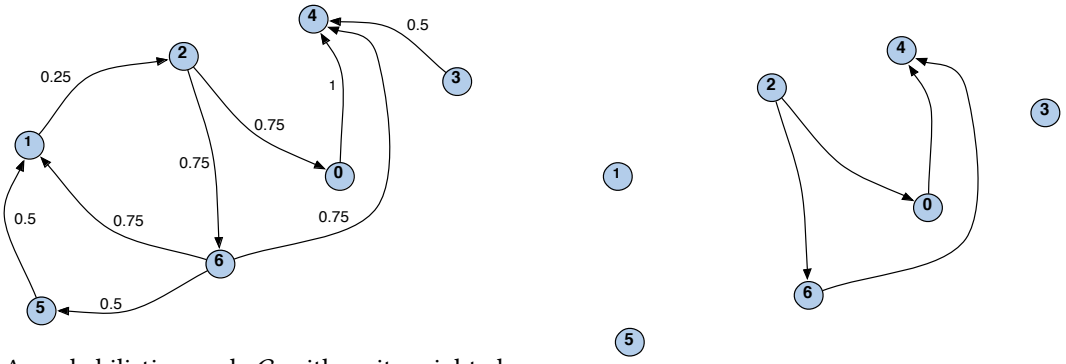
- Siyu Ray Lei, MSc student at the University of Hong Kong in 2013–2014, under my and Reynold Cheng's supervision.
- Paul Lagrée, PhD student at Univ. Paris-Saclay in 2015–2018, under Olivier Cappé and Bogdan Cautis'supervision. I collaborated intensively with Paul on one of his papers.
- Alexandra Iacob, PhD student at Univ. Paris-Saclay since 2019, under my and Bogdan Cautis' supervision.
- Yann Ramusat, PhD student at École Normale Supérieure since 2018, under my and Pierre Senellart's supervision.
- Stratis Ioannidis, associate professor at Northeastern University, visiting Université Paris-Saclay in summer of 2019.

In order to keep this manuscript relatively coherent in terms of research interests, it does not cover an important part of my research since 2012: the subject of dimensionality reduction for classifiers on data streams, with Maroua Bahri, PhD student at Télécom Paris in 2017–2020, under my and Albert Bifet's supervision. An overview survey of her research can be found in [Bahri et al., 2020].

1.1 Data Uncertainty on Graphs: Probabilistic Graphs

The following subsection is an adaptation of the introduction in [Maniu et al., 2017], dealing with indexes on probabilistic graphs.

A natural way to capture data uncertainty in graphs is to represent them as *probabilistic (or uncertain) graphs* [Valiant, 1979, Ball, 1986, Fishman, 1986, Jin et al., 2011, Hua and Pei, 2010], where each edge is annotated with some uncertainty value. There exist two main representations of edge uncertainty in probabilistic graphs. In the *edge-existential model*, each edge is augmented with a probability value, which indicates the chance that the edge exists (Fig. 1.2a). This model captures reliability and failure in computer network connections [Fishman, 1986, Jin et al., 2011], and it can also represent uncertainty in social and biological networks. In the *weight-distribution model*, each edge is associated with a probability



(a) A probabilistic graph $\mathcal{G}$ with unit-weighted edges

(b) A possible world of $\mathcal{G}$

Figure 1.2: Illustrating (a) a probabilistic graph and (b) a possible world. [Maniu et al., 2017]

distribution of weight values [Hua and Pei, 2010]. For example, the traveling time between two vertices in a road network can be represented by a normal distribution.

The problem of evaluating queries on large probabilistic graphs has been studied for a variety of tasks: reliability estimation [Fishman, 1986, Jin et al., 2011], searching nearest neighbors [Potamias et al., 2010], and mining frequent subgraphs [Zou et al., 2010]. Here, we study the evaluation of an important query class, known as the *source-to-target* queries, or *ST-queries*, which are defined over a source vertex s and a target vertex t in a probabilistic graph. Example ST-queries include reachability queries (RQ) and shortest distance queries (SDQ). These queries, when posed on probabilistic graphs, provide answers having probabilistic interpretations. For example, the answer of a reachability query tells us the chance that s can reach t; the shortest distance query returns the probability distribution of the distance between s and t.

Evaluating a probabilistic source-to-target query is computationally expensive. This is because probabilistic graphs have *possible world semantics* [Dalvi and Suciu, 2007]. Conceptually, $\mathcal{G}$ encodes a *distribution* of *possible worlds*, each of which is a definite (non-probabilistic) graph itself, as the example below explains.

Example 1.1. *Fig. 1.2b shows a possible world of the probabilistic graph in Fig. 1.2a. Each possible world is given a probability of existence derived from edge probabilities. For example, the graph in Fig. 1.2b exists only if edges* $0 \rightarrow 4$, $2 \rightarrow 0$, $2 \rightarrow 6$, *and* $6 \rightarrow 4$ *exist, with a probability of approximately 0.1, which is the product of the probabilities that edges in Fig. 1.2b exist, and the probabilities that other edges do not, i.e.,* $1 \times 0.75 \times 0.75 \times 0.75 \times (1 - 0.5) \times (1 - 0.25) \times (1 - 0.75) \times (1 - 0.5) \times (1 - 0.5)$.

Evaluating a query q (e.g., an SDQ) on $\mathcal{G}$ amounts to running the deterministic version of q (e.g., computing the shortest distance between two vertices) on every possible world. This approach is intractable, due to the exponential number of possible worlds; and indeed the problem has been proved to be #P-hard [Valiant, 1979, Ball, 1986, Dalvi and Suciu, 2007].

To improve ST-query efficiency, *sampling* is usually employed [Fishman, 1986, Jin et al., 2011, Potamias et al., 2010, Zou et al., 2010], where possible worlds with high existential probabilities

are sampled and the algorithm of choice is used on each resulting graph. These algorithms, which examine fewer possible worlds than the possible world semantics, are more efficient. However, they suffer from two major downsides, which can hamper query efficiency significantly:

Issue 1 A possible world can be very large. Some of the probabilistic graphs used in our experiments, for example, have millions of vertices and edges. If we want to run an SDQ on a probabilistic graph, a shortest path algorithm needs to be executed once for each sampled possible world. Since a possible world can be a very big graph, query efficiency can be affected.

Issue 2 To achieve high accuracy, a large number of possible world samples may need to be generated. In our experiments, around 1,000 samples are required to converge to acceptable approximate values.

We discuss in Chapter 2 our main contribution in this area: tree decompositions for source-to-target queries on probabilistic graphs. The main idea of using tree decompositions on graphs is to efficiently create a tree-shaped *index* on the graph. This index can then be used to extract – for a given query parameter, e.g., a pair of nodes – an *equivalent graph*. In case where this equivalent graph is much smaller than the original graph, any sampling algorithm can also function much more efficiently. As we will also discuss in Chapter 2, tree decompositions are linked directly to the notion of having a low *treewidth*; we study how realistic this assumption of low treewidth is in real-world graphs.

We will also briefly discuss two other contributions. The first is the extension of decompositions to kNN queries. The second is acknowledging that probabilistic queries are a particular case of *provenance on graphs*.

1.2 Process Uncertainty on Graphs: Online Social Influence

The following subsection is an adaptation of the introduction in [Lagrée et al., 2019] (an extension to [Lagrée et al., 2017]), which discusses the problem of *online social influence maximization*.

Viral advertising based on word-of-mouth diffusion in social media has become very important in the digital marketing landscape. Nowadays, social value and social influence are some of the most used concepts in the area of Web advertising and most companies that advertise in the Web space must have a "social" strategy. For example, on widely used platforms such as Facebook or Twitter, promoted posts are interleaved with normal posts on user feeds. Users interact with these posts by actions such as "likes" (adoption), "shares" or "reposts" (network diffusion) [Watts, 2003, Watts and Dodds, 2007].

Motivated by the need for effective viral social strategies, *influence estimation* and *influence maximization* (IM) have become important research problems, at the intersection of data mining and social sciences [Easley and Kleinberg, 2010]. In short, IM is the problem of selecting a set

of nodes from a given diffusion graph, maximizing the expected spread under an underlying diffusion model. This problem was introduced in 2003 by the seminal work of [Kempe et al., 2003], through two stochastic, discrete-time diffusion models, *Linear Threshold* (LT) and *Independent Cascade* (IC). These models rely on diffusion graphs whose edges are weighted by a score of influence. Both models above are *stochastic processes*: the process occurs on a graph whose edges are annotated with influence probabilities, i.e., a probabilistic graph. They proceed in time-discrete steps, where at each step an attempt is made to influence (in the IC model) or be influenced by (in the LT model) the neighbor nodes, depending on their edge influence probabilities. Moreover the IM problem of [Kempe et al., 2003] – selecting a set of nodes to maximize the *expected* spread – is NP-hard for both models. The greedy algorithm is proven to give a constant-factor approximation due to the sub-modularity property of the influence spread, but does not scale to large graphs.

A rich literature followed, focusing on computationally efficient and scalable algorithms to solve IM. The benchmarking study of [Arora et al., 2017] summarizes state-of-the-art techniques and also debunks many IM myths. In particular, it shows that, depending on the underlying diffusion model and the choice of parameters, each algorithm's behavior can vary significantly, from very efficient to prohibitively slow.

Importantly, all the IM studies discussed in [Arora et al., 2017] have as starting point a specific model (IC or LT), whose graph topology and parameters – basically the edge weights – are known. However, this assumption is unrealistic, and, recently, we have witnessed a trend towards bridging the gap between theory and practical relevance in the IM framework, along several dimensions.

In particular, one such dimension is the one of *offline, model-specific methods*, which can *infer* the diffusion parameters or the underlying graph structure (if unknown, or, as often the case, implicitly overlaying the existing social graph), or both, starting from observed information cascades [Saito et al., 2008, Goyal et al., 2010, Du et al., 2013, Gomez-Rodriguez et al., 2011, Gomez-Rodriguez et al., 2012, Gomez-Rodriguez et al., 2013]. In short, information cascades are time-ordered sequences of records indicating when a specific user adopted a specific item.

There are however many situations where it is unreasonable or counter-productive to assume the existence of relevant historical data in the form of cascades. For such settings, *online approaches*, which can learn the underlying diffusion parameters *while running diffusion campaigns*, have been proposed. Bridging IM and inference, this is done by balancing between *exploration* steps (of yet uncertain model aspects) and *exploitation* ones (of the best solution so far), usually by *multi-armed bandits* techniques, where an agent interacts with the network to infer influence probabilities [Vaswani et al., 2015, Chen et al., 2016, Wen et al., 2016]. The learning agent sequentially selects seeds from which diffusion processes are initiated in the network; the obtained feedback is used to update the agent's knowledge of the model.

Nevertheless, all these studies on inferring diffusion networks, whether offline or online, rely on parametric diffusion models, i.e., assume that the actual diffusion dynamics are well captured by such a model (e.g., IC). This is still a significant limitation for practical purposes. First, the more complex the model, the harder to learn, especially in campaigns that have a relatively short timespan, making model inference and parameter estimation very challenging within a small horizon (typically tens or hundreds of spreads). Second, it is commonly agreed that the

aforementioned diffusion models represent elegant yet coarse interpretations of a reality that is much more complex and often hard to observe fully.

Chapter 3 presents my contributions in the general area of *online influence maximization* (OIM). I present first an overview of the initial paper about the OIM problem, in which the assumption was that the process (IM in this case) and graph topology are known, but the influence probabilities had to be learned. The second contribution is the most recent article on the subject, in which the assumption that the process and the graph topology are known is significantly relaxed and in which a form of *persistence* is assumed. Both articles use the *multi-armed bandit* approach.

Other contributions that are discussed are work in progress to extend the online influence maximization to the *contextual* case, and how influence can be used to improve social recommendation algorithms.

2 Tree Decompositions for Probabilistic Graphs

In this chapter, we discuss some algorithmic challenges when dealing with uncertainty of graph edges. We focus on source-to-target queries in probabilistic graphs, and we discuss how tree indexes can be used efficiently for query answering. We also establish the link between our tree indexes and the notion of treewidth on graphs.

2.1 Probabilistic Graphs, Queries, and Indexing Systems

> The following section is adapted from [Maniu et al., 2017], work published in ACM Transactions on Database Systems.

We present in this section the notion of probabilistic graphs and indexing systems. These notions will be useful for discussing the ProbTree, a tree-based indexing system for source-to-target queries in probabilistic graphs.

2.1.1 Probabilistic Graphs and Queries

We start by defining probabilistic graphs, by extending standard graphs with a probability function on their edges:

Definition 2.1 (Probabilistic graph)**.** *A* probabilistic graph *is a triple* $\mathcal{G} = (V, E, p)$ *where:*

(i) V *is a set of* vertices*;*

(ii) $E \subseteq V \times V$ *is a set of* edges*;*

(iii) $p : E \to 2^{\mathbb{Q}^+ \times (0,1]}$ *is a function that assigns to each edge a* finite probability distribution of edge weights*, i.e., each edge* e *is associated with a partial mapping* $p(e) : \mathbb{Q}^+ \to (0, 1]$ *with finite support* $\text{supp}(p(e))$ *such that* $\sum_{w \in \text{supp}(p(e))} p(e)(w) \leq 1$.

We denote $V(\mathcal{G})$, $E(\mathcal{G})$, $p_{\mathcal{G}}$ *the vertex set, the edge set, and the probability assignment function of* $\mathcal{G}$ *respectively.*

Note that the probability that an edge e does not exist in $\mathcal{G}$ is $1 - \sum_{w \in \text{supp}(p(e))} p(e)(w)$. Definition 2.1 is essentially the *weight-distribution model* [Hua and Pei, 2010], where each edge is associated with a finite probability distribution of weights. This definition also captures the *edge-existential model* [Fishman, 1986, Jin et al., 2011], where an edge with existential probability p can be represented as a weight distribution $\{(1, p)\}$. Likewise, we assume that the probability distributions on different edges are independent.

Definition 2.2 (Possible world)**.** *Let $\mathcal{G} = (V, E, p)$ be a probabilistic graph. The (weighted) graph $G = (V, E_G, \omega)$ with $E_G \subseteq V \times V$ and $\omega : E_G \to \mathbb{Q}^+$ is a possible world of $\mathcal{G}$ if $E_G \subseteq E$ and for every edge $e \in E_G$, $\omega(e) \in \mathrm{supp}(p(e))$. We write $G \sqsubseteq \mathcal{G}$. The* probability *of the possible world G is:*

$$\Pr(G) := \prod_{e \in E_G} p(e)(\omega(e)) \times \prod_{e \in E \setminus E_G} \left(1 - \sum_{w' \in \mathrm{supp}(p(e))} p(e)(w')\right).$$

Figure 1.2 shows a probabilistic graph and a possible world.

A probabilistic graph has an exponentially large number of possible worlds:

Proposition 2.3. *Let $\mathcal{G}$ be a probabilistic graph. Let $\mathrm{PW}(\mathcal{G})$ denote the set of non-zero probability possible worlds of $\mathcal{G} = (V, E, p)$; formally, $\mathrm{PW}(\mathcal{G}) = \{ G \mid G \sqsubseteq \mathcal{G}, \Pr(G) > 0 \}$.*

Then $\prod_{e \in E} |\mathrm{supp}(p(e))| \leq |\mathrm{PW}(\mathcal{G})| \leq \prod_{e \in E}(|\mathrm{supp}(p(e))| + 1)$, and $\sum_{G \in \mathrm{PW}(\mathcal{G})} \Pr[G] = 1$.

Source-to-target queries In the following, we focus on *source-to-target distance* queries (or *ST-queries*), a common query class for probabilistic graphs. This kind of query requires two inputs: source vertex s and target vertex t, where $s, t \in V$. Typical example ST-queries include:

Reachability (RQ) [Cohen et al., 2003] Probability that t is reachable from s.

Distance-constraint reachability (d-RQ) [Jin et al., 2011] Probability that t is reachable from s within distance d.

Expected shortest distance (SDQ) [Ball, 1986] The expected value of the *distance distribution* $p(s \to t)$ between s and t. Formally, $p(s \to t)$ is a set of tuples (d_i, p_i), where p_i is the probability that the shortest distance between s and t is d_i.

To evaluate these queries, we generally need to obtain $p(s \to t)$, from which the result of any of these queries can be derived. This type of queries are hard to evaluate, as stated multiple times in the literature:

Theorem 2.4 ([Valiant, 1979, Ball, 1986])**.** *Evaluating RQ, d-RQ (for $d \geq 2$), and SDQ is* #P*-complete.*

Without loss of generality, in the following we assume that the answer of an ST-query is $p(s \to t)$, where any ST-query is answered from it. In fact, our solution can deal with *any* ST-query that depends only on $p(s \to t)$.

2.1.2 Graph Indexing Frameworks

We now propose an indexing framework for probabilistic graphs. First, we define the notion of *transformation system*.

Definition 2.5 (Transformation system)**.** *A* probabilistic graph transformation system *is a pair* (index, retrieve) *where:*

(a) $\mathcal{G}(q)$ for $q(0, 4)$ (b) $\mathcal{G}(q)$ for $q(1, 4)$

Figure 2.1: Illustrating two equivalent probabilistic graphs for the graphs in Figure 1.2, for two ST-queries. [Maniu et al., 2017]

- index *is a function that takes as input a probabilistic graph* $\mathcal{G}$ *and outputs an object* $\mathcal{I} = \mathsf{index}(\mathcal{G})$ *called an* index*;*
- retrieve *is an operator that, given an ST-query* $q(s, t)$ *in* $\mathcal{G}$*, and the index* $\mathcal{I}$*, produces a probabilistic graph* $\mathcal{G}(q) = \mathsf{retrieve}_q(\mathcal{I})$*, such that* $\{s, t\} \subseteq V(\mathcal{G}(q))$*.*

Essentially, a transformation encodes a probabilistic graph $\mathcal{G}$ into an index structure, which can generate another probabilistic graph $\mathcal{G}(q)$ for a given pair of vertices (s, t), representing the query q. Since s and t can be found in $\mathcal{G}(q)$, $q(s, t)$ can be evaluated on $\mathcal{G}(q)$. Two examples for the graph in Figure 1.2 are given in Figure 2.1.

We consider two important properties for queries evaluated on the transformed graph $\mathcal{G}(q)$:

(i) the *loss* – the difference between the result of q evaluated on $\mathcal{G}(q)$ and $\mathcal{G}$; and

(ii) the *efficiency* – the cost of evaluating q on $\mathcal{G}(q)$.

There are multiple definitions possible for *loss*; one of the most used ones is the *mean squared loss* – the square of the difference between real and estimated query results. If this loss is zero we call the transformation *lossless*; *lossy* otherwise.

A transformation system is called an *indexing system*, if it is efficient for answering a given kind of query:

Definition 2.6 (Indexing system)**.** *A transformation system* (index, retrieve) *is said to be an* indexing system for query class $\mathcal{Q}$ *if the following hold:*

(i) index *is a polynomial-time function;*

(ii) for every probabilistic graph $\mathcal{G}$*,* $|\mathsf{index}(\mathcal{G})| = O(|\mathcal{G}|)$ *(i.e., the space occupied by the index is bounded by a linear function of the space occupied by the original graph);*

(iii) for every query $q \in \mathcal{Q}$*,* $\mathsf{retrieve}_q$ *is linear-time computable.*

Let us give an example transformation system that is not an indexing system. Given query class $\mathcal{Q}$, consider a system that pre-computes all pairwise results, i.e., the index operator. This system satisfies Property (iii), since retrieve_q builds a trivial two-vertex graph. Evaluating q over the resulting graph is very efficient, since it just involves looking up the distance probability distribution on the edges of this graph, in $O(1)$ time. However, neither Property (i) nor (ii) holds, since indexing is intractable unless #P is tractable (which would imply P = NP), and since the index is at least quadratic in size.

We aim for indexing systems that allow efficient query evaluation (for a query class $\mathcal{Q}$) on the transformed graph: for every probabilistic graph $\mathcal{G}$ and query $q \in \mathcal{Q}$, the running time of retrieve_q on $\text{index}(\mathcal{G})$, together with the running time of q on $\mathcal{G}(q)$, should be less than evaluating q on $\mathcal{G}$.

Independent subgraphs Can we obtain an efficient index for probabilistic graphs, with zero or limited loss? We show that we can do this via a *tree decomposition* of the probabilistic graph $\mathcal{G}$, where *independent subgraphs* of $\mathcal{G}$ are identified and reduced.

Recall that each edge in a probabilistic graph, along with its associated probability distribution, is independent of probability distributions of the other edges. Thus, one way to derive a lossless indexing system is to collapse larger subgraphs to edges, such that *independence* is maintained:

Definition 2.7 (Independent subgraph)**.** *We define an* independent subgraph *of a probabilistic graph $\mathcal{G}$ as a (weakly) connected induced subgraph $S \subseteq \mathcal{G}$ with arbitrarily many* internal vertices *and at most two* endpoint vertices v_1, v_2 *such that each internal vertex has edges only to/from other internal vertices of S, or to/from the endpoint vertices.*

We can use these independent subgraphs to reduce the graph to an equivalent subgraph by replacing S with edges $v_1 \to v_2$ and $v_2 \to v_1$, with corresponding probability distributions $p(v_1 \to v_2)$ and $p(v_2 \to v_1)$ computed from S. We obtained a fundamental (and non-trivial) result: in the case of *undirected* graphs, independent subgraphs are exactly those that can be removed from the graph while preserving *joint distance probability distributions* for non-removed vertices.

Theorem 2.8. *[Maniu et al., 2017] Let $\mathcal{G} = (V, E, p)$ be a probabilistic graph with $(u, v) \in E \Leftrightarrow (v, u) \in E$ and V' a non-empty subset of vertices of V that are connected in $\mathcal{G}$. We assume for each $e \in E$, $\sum_{w \in \text{supp}(p(e))} p(e)(w) < 1$.*

There exists a probabilistic graph $\mathcal{G}' = (V \backslash V', E', p')$ such that the joint distance distributions for $V \backslash V'$ is the same in $\mathcal{G}'$ as in $\mathcal{G}$ if and only if *V' is the set of internal vertices of an independent subgraph of $\mathcal{G}$.*

In other words, Theorem 2.8 states that the independent subgraph approach is the *unique* manner in which a lossless indexing system can be obtained for a probabilistic graph, at least for undirected graphs[1].

[1]The case of directed graphs is more complex; the tools that we use (in particular, tree decompositions) are more robust and better understood in the setting of undirected graphs [Robertson and Seymour, 1984] than in that of directed ones [Safari, 2005]

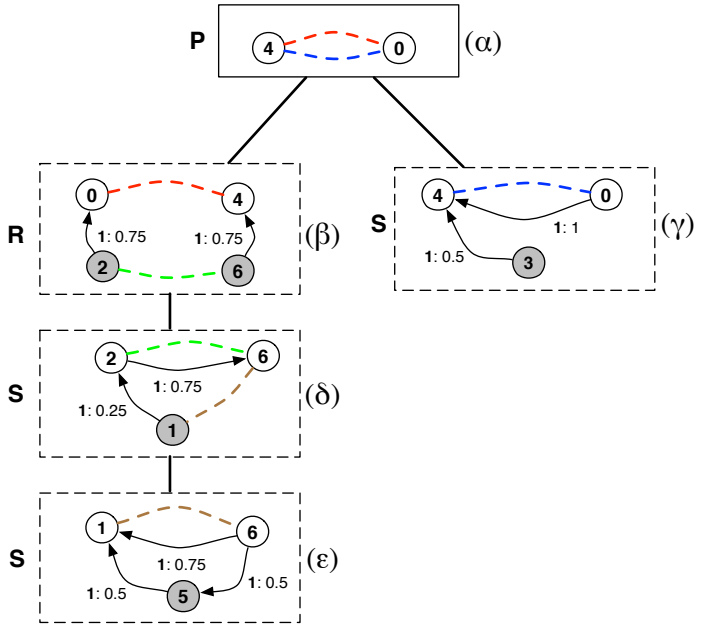


Figure 2.2: Decomposition into independent graphs of the graph in Fig. 1.2. [Maniu et al., 2017]

ProbTree Our definition of independent subgraphs relies on vertices in the graphs which separate the graph into two independent components. We can *decompose* the graphs into the corresponding independent subgraphs in a recursive way, by repeatedly identifying endpoints and sub-dividing the subgraphs until it is not longer possible to do so. It is easy to verify that such a recursive decomposition – our desired index $\mathcal{I} = \text{index}(\mathcal{G})$ – results in a tree where nodes are independent subgraphs and edges appear between subgraphs having common endpoints. We call such a tree decomposition a *ProbTree*:

Definition 2.9 (ProbTree). *Let $\mathcal{G} = (V, E, p)$ be a probabilistic graph. A* ProbTree *for V is a pair $(\mathcal{T}, \mathcal{B})$ where $\mathcal{T}$ is a tree (i.e., a connected, acyclic, undirected graph) and $\mathcal{B}$ is a function mapping each node of $\mathcal{T}$ to a* probabilistic graph *(called the* internal graph *or* bag *of n) with vertex set a subset of V. We further require that for every subtree $\mathcal{T}'$ of $\mathcal{T}$, the set of vertices in bags of nodes of $\mathcal{T}'$ induces an independent subgraph of $\mathcal{G}$.*

A first method for indexing probabilistic graphs into a ProbTree are SPQR trees. For a graph G, a vertex set $S \subseteq V(G)$ is called a *separator* for G if the graph induced by $V(G) \backslash S$ is disconnected. Given an integer k, a graph G is called *k-connected* if $V(G) \backslash S$ is connected for all $S \subseteq V(G), |S| < k$, i.e., there exists no separator for G of size less than k. 0-connected graphs are connected graphs in the usual sense, 1-connected graphs contain *cut vertices* which disconnect the graph into *biconnected* components, and 2-connected graphs have *separation pairs* of vertices which separate the graph into *triconnected* components. These definitions link directly to our desired properties for independent subgraphs. Connected, biconnected, and triconnected components are exactly independent subgraphs of 0, 1 and 2 endpoints, and we

aim to decompose the graph into a tree containing them.

Tutte [Tutte, 1966] studied the structure of the triconnected components of a graph, and Hopcroft and Tarjan [Hopcroft and Tarjan, 1973b, Hopcroft and Tarjan, 1973a] gave optimal algorithms for decomposition. They showed that the triconnected components of a graph are unique:

Theorem 2.10. *[Tutte, 1966, Hopcroft and Tarjan, 1973b, Hopcroft and Tarjan, 1973a] The biconnected and triconnected components of a graph G are unique and the inclusion relationships among them forms a tree.*

Using Theorem 2.8 and Theorem 2.10, we can derive the corollary that decomposing the graph into its biconnected and triconnected components is the unique manner in which we can obtain a lossless indexing of a probabilistic (undirected) graph – since biconnected and triconnected components are independent subgraphs and they are unique.

Hopcroft and Tarjan's algorithms were refined, via SPQR trees [Di Battista and Tamassia, 1990] and [Gutwenger and Mutzel, 2000]'s linear implementation. Our approach was to use these algorithms initially to obtain the decomposition; however, their practical performance was not sufficient.

Example 2.11. *To understand what a ProbTree looks like, consider the tree depicted in Fig. 2.2 which is a ProbTree for the graph of Fig. 1.2 (it is more precisely an SPQR tree). The tree $\mathcal{T}$ is defined by the black lines between bags; a Greek letter identifier is given on the right of each bag. The vertices in bags of any subtree of $\mathcal{T}$ induce an independent subgraph in the original graph: for example, the subtree rooted at node (δ) contains vertices 1, 2, 5, and 6, which indeed form an independent subgraph with endpoints 2 and 6. The nodes in white represent the endpoints of the independent subgraphs induced by the bag's respective subtree: here, for node (δ), these are 2 and 6.*

Our best performing ProbTree variant is not the SPQR one, but one which performs a *partial decomposition*. Before we can present it, we need to take a detour to discuss tree decompositions, their link to the notion of *treewidth*, and the notion of *partial tree decomposition*.

2.2 Tree Decompositions and Treewidth

> The following section is adapted from [Maniu et al., 2019], work presented at ICDT.

Following the original definitions in [Robertson and Seymour, 1984], we first define a tree decomposition:

Definition 2.12 (Tree Decomposition)**.** *Given an* undirected graph $G = (V, E)$*, where V represents the set of vertices (or nodes) and $E \subseteq V \times V$ the set of edges, a* tree decomposition *is a pair (T, B) where $T = (I, F)$ is a tree and $B : I \to 2^V$ is a labeling of the nodes of T by subsets of V (called* bags*), with the following properties:*

(i) $\bigcup_{i \in I} B(i) = V$;

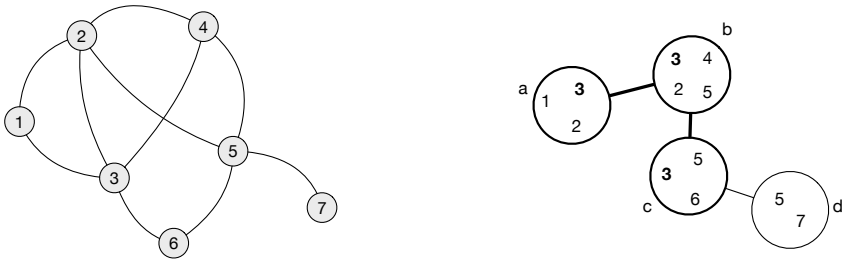


Figure 2.3: Example undirected, unlabeled, graph (left) and decomposition of width 3 (right). [Maniu et al., 2019]

(ii) $\forall (u,v) \in E, \exists i \in I$ *s.t.* $\{u,v\} \subseteq B(i)$*; and*

(iii) $\forall v \in V, \{i \in I \mid v \in B(i)\}$ *induces a subtree of* T*.*

Intuitively, a tree decomposition groups the vertices of a graph into bags so that they form a tree-like structure, where a link between bags is established when there exists common vertices in both bags.

Example 2.13. *Figure 2.3 illustrates such a decomposition. The resulting decomposition is formed of 4 bags, each containing a subset of the nodes in the graph. The bags containing node* 3 *(in bold) form a connected subtree of the tree decomposition.*

Based on the number of vertices in a bag, we can define the concept of *treewidth*:

Definition 2.14 (Treewidth)**.** *Given a graph* $G = (V, E)$ *the* width *of a tree decomposition* (T, B) *is equal to* $\max_{i \in I}(|B(i)| - 1)$*. The* treewidth *of* G*,* $w(G)$*, is equal to the minimal width of all tree decompositions of* G*.*

It is easy to see that an isolated point has treewidth 0, a tree treewidth 1, a cycle treewidth 2, and a $(k+1)$-clique (a complete graph of k nodes) treewidth k.

Example 2.15. *The width of the decomposition in Figure 2.3 is 3. This tells us the graph has a treewidth of at most 3. The treewidth of this graph is actually exactly 3: indeed, the 4-clique, which has treewidth 3, is a* minor *of the graph in Figure 2.3 (it is obtained by removing nodes* 1 *and* 7*, and by contracting the edges between* 3 *and* 6 *and* 5 *and* 6*), and treewidth never increases when taking a minor (see, for instance, [Harvey, 2014]).*

Intuitively, a tree decomposition groups the vertices of a graph into *bags* so that they form a tree-like structure, where a link between bags is established when there exists common vertices in both bags – in other words, exactly what we need for our ProbTree. However, note that algorithms which solve hard problems in linear time when restricted to graphs of bounded treewidth – including k-terminal reliability – have been proposed by [Arnborg and Proskurowski, 1989]. They have two main disadvantages:

(i) they use bottom-up dynamic programming for the computation of optimal values, but they retain an exponential dependence on the treewidth w; and

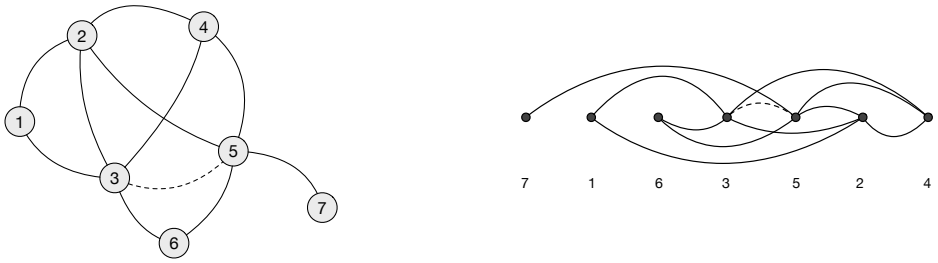


Figure 2.4: Graph triangulation for the graph of Figure 2.3 (left) and its elimination ordering (right). [Maniu et al., 2019]

(ii) the practical appeal is limited, as the computation of the query answers is made at the same time as the construction of the decompositions.

Moreover, given a width, a tree decomposition can be constructed in linear time [Bodlaender, 1996]. However, determining the treewidth of a given graph is NP-complete [Arnborg et al., 1987]. This means that determining if a graph has a bounded treewidth, and thus being able to create its tree decomposition, cannot be reasonably performed on large-scale graphs.

It is possible, nevertheless, to obtain estimations of treewidth – along with their associated decompositions – via estimations of the range of possible treewidths, between a *lower bound* and an *upper bound*. We refer the reader to [Bodlaender and Koster, 2010] and [Bodlaender and Koster, 2011], respectively, for a more complete survey of treewidth upper and lower bound estimation algorithms on synthetic data, and we cover here only the subset of methods that fit our requirements.

Treewidth Upper Bounds The treewidth is the smallest width among all possible tree decompositions. In other words, the width of any decomposition of a graph is an upper bound of the actual treewidth of that graph. A treewidth upper bound estimation algorithm can thus be seen as an algorithm to find a decomposition whose width is as close as possible to the treewidth of the graph. To understand how one can do that, we need to introduce the classical concept of *elimination ordering* and to explain its connection to treewidth.

We start by introducing *triangulations* of graphs, which transform a graph G into a graph G^Δ that is *chordal*:

Definition 2.16 (Chordal graph, Triangulation)**.** *A* chordal *graph is a graph G such that every cycle in G of at least four vertices has a* chord *– an edge between two non-successive vertices in the cycle.*

A triangulation *(or* chordal completion*) of a graph G is a minimal chordal supergraph G^Δ of G: a graph obtained from G by adding a minimal set of edges to obtain a chordal graph.*

Example 2.17. *The graph in Figure 2.3 is not chordal, since, for example, the cycle* 3–4–5–6–3 *does not have a chord. If one adds an edge between* 3 *and* 5*, as in Figure 2.4 (left), one can verify that the resulting graph is chordal, and thus a triangulation of the graph of Figure 2.3.*

One way to obtain triangulations of graphs is *elimination orderings*. An elimination ordering ω of a graph $G = (V, E)$ of n nodes is an ordering of the vertices of G, i.e., it can be seen as a

bijection from V onto $\{1, \ldots, n\}$. From this ordering, one obtains a triangulation by applying sequentially the following *elimination procedure* for each vertex v: first, edges are added between remaining neighbors of v as needed so that they form a clique, then v is *eliminated* (removed) from the graph. For every elimination ordering ω, G along with all edges added to G in the elimination procedure forms a graph, denoted G^{Δ}_{ω}. This graph is chordal (indeed, we know that the two neighbors of the first node of any cycle we encounter in the elimination ordering have been connected by a chord by the elimination procedure). It is also a supergraph of G, and it can be shown it is a minimal chordal supergraph, i.e., a triangulation of G.

Example 2.18. *Figure 2.4 (right) shows a possible elimination ordering* $(7, 1, 6, 3, 5, 2, 4)$ *of the graph of Figure 2.3. The elimination procedure adds a single edge, when processing node* 6*, between nodes* 3 *and* 5*. The resulting triangulation is the graph on the left of Figure 2.4.*

Elimination orderings are connected to treewidth by the following result:

Theorem 2.19. [Bodlaender and Koster, 2010] *Let $G = (V, E)$ a graph, and $k \leq n$. The following are equivalent:*

(i) G has treewidth k.

(ii) G has a triangulation G^{Δ}, such that the maximum clique in G^{Δ} has size $k + 1$.

(iii) There exists an elimination ordering ω such that the maximum clique size in G^{Δ}_{ω} is $k + 1$.

Obtaining the treewidth of the graph is thus equivalent to finding an optimal elimination ordering. Moreover, constructing a tree decomposition from an elimination ordering is a natural process: each time a vertex is processed, a new bag is created containing the vertex and its neighbors. Note that, in practice, we do not need to compute the full elimination ordering: we can simply stop when we know that the number of remaining vertices is lower that the largest clique found thus far.

Example 2.20. *In the triangulation of Figure 2.4 (left), corresponding to the elimination ordering on the right, the maximum clique has size 4: it is induced by the vertices* 2, 3, 4, 5*. This proves the existence of a tree decomposition of width* 3*. Indeed, it is exactly the tree decomposition in Figure 2.3 (right): bag* d *is constructed when* 7 *is eliminated, bag* a *when* 1 *is eliminated, bag* c *when* 6 *is eliminated, and finally bag* b *when* 3 *is eliminated.*

Finding a "good" upper bound on the treewidth can thus be done by finding a "good" elimination ordering. This is still an intractable problem, naturally, but there are various heuristics for generating elimination orderings leading to good treewidth upper bounds. One important class of such elimination ordering heuristics are the greedy heuristics. Intuitively, the elimination ordering is generated in an incremental manner: each time a new node has to be chosen in the elimination procedure, it is chosen using a criterion based on its neighborhood. We have implemented[2] the following greedy criteria (with ties broken arbitrarily):

- Degree. The node with the minimum degree is chosen. [Markowitz, 1957, Berry et al., 2003]

[2] `https://github.com/smaniu/treewidth`

- FillIn. The node with the minimum needed "fill-in" (i.e., the minimum number of missing edges for its neighbors to form a clique) is chosen. [Bodlaender and Koster, 2010]
- Degree+FillIn. The node with the minimum sum of degree and fill-in is chosen.

Example 2.21. *The elimination ordering of Figure 2.4 (right) is an example of the use of Degree+FillIn. Indeed,* 7 *is first chosen, with value 1, then* 1 *with value 2, then* 6 *with value 2 + 1 = 3. After that, the order is arbitrary since* 2, 3, 4, *and* 5 *form a clique (and thus have initial value 3).*

Previous studies [van Dijk et al., 2006, Koller and Friedman, 2009, Bodlaender and Koster, 2010] have found these greedy criteria give the closest estimations of the real treewidth. An alternative way of generating an elimination ordering is based on maximum cardinality search [Koller and Friedman, 2009, Bodlaender and Koster, 2010]; however, it is both less precise than the greedy algorithms – due to its reliance on how the first node in the ordering is chosen – and slower to run.

In contrast to upper bounds, obtaining treewidth lower bounds is not constructive. In other words, lower bounds do not generate decompositions; instead, the estimation of a lower bound is made by computing other measures on a graph, that are a proxy for treewidth.

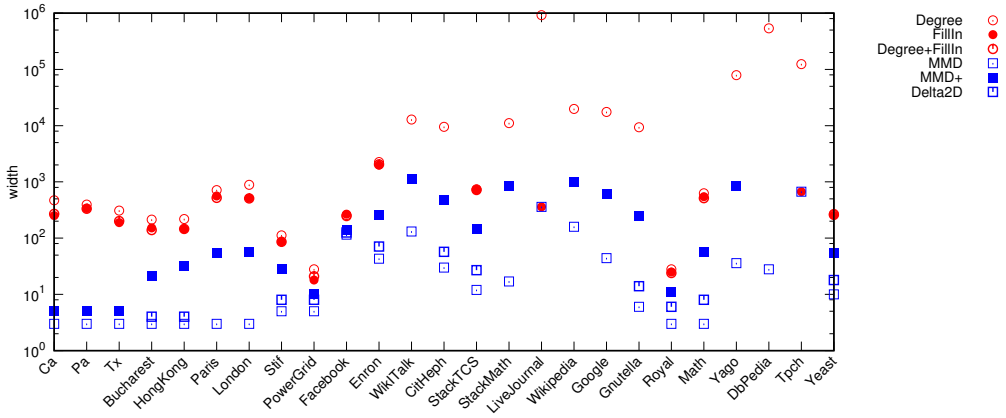
How high is the treewidth of real graphs? We want to answer the following question: *are tree decompositions based on treewidth estimations feasible in practice?*

Figure 2.5 shows the results of our estimation algorithms over 25 datasets from 4 domains:

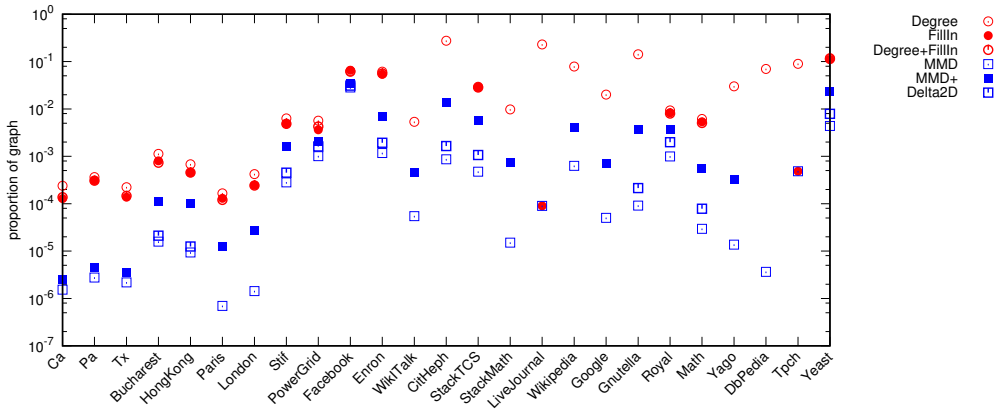
- infrastructure: road networks of states (Ca, Pa, Tx) and cities (Bucharest, HongKong, Paris), the Parisian tranport network Stif, and the US PowerGrid;
- social (Enron, Facebook, WikiTalk) and Web (Gnutella, Google) networks;
- knowledge graphs (Yago and DbPedia);
- hierarchical networks (Royal trees, Math generalogies).

Lower values mean better treewidth estimations. Focusing on the upper bounds only (red circular points), we notice that, in general, FillIn does give the smallest upper bound of treewidth, in line with previous findings [Bodlaender and Koster, 2011]. Interestingly, the Degree heuristic is quite competitive with the other heuristics. This fact, coupled with its lower running time, means that it can be used more reliably in large graphs. Indeed, as can be seen in the figure, on some large graphs only the Degree heuristic actually finished at all; this means that, as a general rule, Degree seems the best fit for a quick and relatively reliable estimation of treewidth.

We plot both the absolute values of the estimations in Figure 2.5a, but also their relative values (in Figure 2.5b, representing the ratio of the estimation over the number of nodes in the graph), to allow for an easier comparison between networks. The absolute value, while interesting, does not yield an intuition on how the bounds can differ between network types. If we look at the relative values of treewidth, it becomes clear that infrastructure networks have a treewidth that is much lower than other networks; in general they seem to be consistently under one thousandth of the original size of the graph. This suggests that, indeed, this type of



(a) absolute values



(b) relative values

Figure 2.5: Treewidth estimation of different algorithms (logarithmic scale). [Maniu et al., 2019]

network may have properties that make them have a lower treewidth. For the other types of networks, the estimations can vary considerably: they can go from one hundredth (e.g., Math) to one tenth (e.g., WikiTalk – the network of interactions on Wikipedia talk pages) of the size of the graph.

These results show us that, unfortunately, estimating a tree decomposition of the graphs leads to treewidth that is too high. The exponential dependency on the treewidth of the algorithms means that they are not usable in practice. Even so, there exists another possibility: stopping the decomposition early and hence obtaining a *partial tree decomposition*.

2.2.1 Partial Tree Decompositions

The manner in which treewidth decomposition can be used starts from a simple observation made in studies on complex graphs, that is, that they tend to exhibit a *tree-like fringe* and a *densely connected core* [Newman et al., 2001, Newman et al., 2002]. The tree-like fringe precisely corresponds to bounded-treewidth parts of the network. This yields an easy adaptation of the

upper bound algorithms based on node ordering: given a parameter w representing the highest treewidth the fringe can be, we can run any greedy decomposition algorithm (Degree, FillIn, DegreeFillIn) until we only find nodes of degree $w + 1$, at which point the algorithm stops. At termination, we obtain a data structure formed of a set of treewidth w elements (w-trees) interfacing through cliques that have size at most $w + 1$ with a *core graph*. The core graph contains all the nodes not removed in the bag creation process, and has unknown treewidth. Figure 2.6 illustrates the notion of partial decompositions.

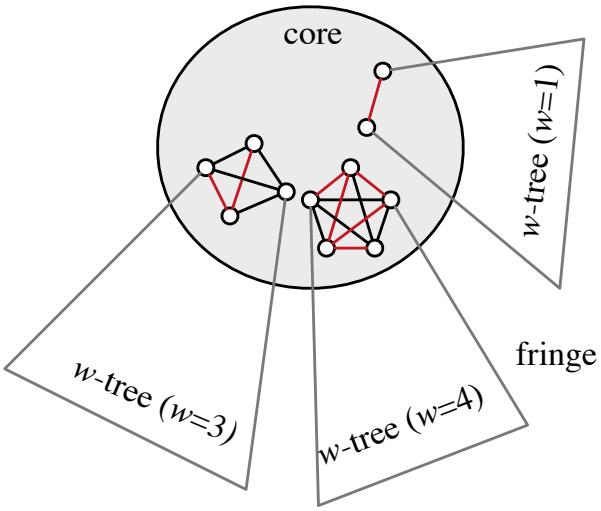


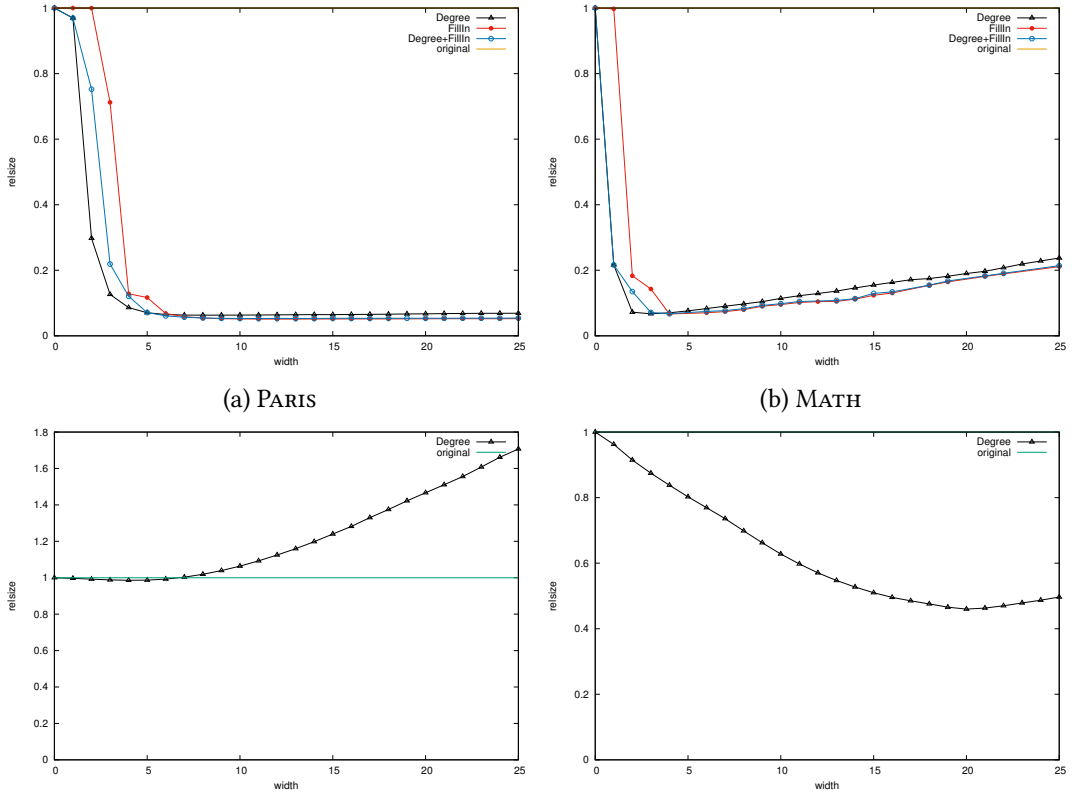
Figure 2.6: An abstract view of partial decompositions. Partial decompositions are formed of a *core* graph, which interfaces with w-trees through w-cliques (the *fringe*). [Maniu et al., 2019]

The resulting structure can be thought of as a *partial decomposition* (or *relaxed decomposition*), a concept introduced in [Wei, 2010, Akiba et al., 2012] in the context of answering shortest path queries – we will also show in the next section how to use them for probabilistic queries. A partial decomposition can be extremely useful: the tree-like fringe can be used to quickly precompute answers to partial queries (e.g., precompute distances in the graph). Once the precomputation is done, these (partial) answers are added to the core graph, where queries can be answered directly. If the resulting core graph is much smaller than the original graph, the gains in running time can be considerable, as shown in [Wei, 2010, Akiba et al., 2012] and here.

An interesting aspect of greedy upper bound algorithms is that they generate at any point a partial decomposition, with width equal to the highest degree encountered in the greedy ordering so far. The algorithm can then be stopped at any width, and the size of the resulting partial decomposition can be measured.

Hence, the objective of our experimental study is to check how feasible partial decompositions are. The measure of interest is how large the core (or root) bag is. Indeed, the smaller this core is, the faster *any* algorithm used on this decomposition is.

Figure 2.7 shows the relative size of the core in partial decompositions up to a width of 25, for a selection of graphs (the full results are available in [Maniu et al., 2019]). Immediately apparent



(a) Paris (b) Math

(c) CitHeph (d) Google

Figure 2.7: Zoomed view of decompositions up to width 25. [Maniu et al., 2019]

is the fact that the minimal size of the core graph occurs at very low widths: in almost all cases, it occurs around a width between 5 and 10. This is low enough that running algorithms even doubly exponential in the width on the resulting fringe graphs can be performed. In terms of actual size, it can vary greatly. In road networks (Paris), it even reaches 10%, compared to around 50% for other graphs. The exception to this are denser networks – CitHeph (a physics citation network) where almost no benefit is visible for partial decompositions of small width, and Google (the network of URLs crawled by Google) where the minimal size occurs much later ($w = 20$).

We show next that the fact that we can create a fringe of very small treewidth and, at the same time, have a very small core can be exploited efficiently for queries on probabilistic graphs.

2.3 Tree Decompositions for Probabilistic Graphs: ProbTree

> The following section is adapted from [Maniu et al., 2017], work published in ACM TODS.

In the following – supported by our main result in Theorem 2.8 – we study the suitability of partial decompositions for indexing probabilistic graphs[3] We will present the principle of the index and retrieve operators, and we show a selection of the experimental results on ProbTree.

2.3.1 Indexing and Retrieval

The principle behind the ProbTree is to first generate a partial tree decomposition for index, where the parameter is the maximum width of the non-root bags in the tree, w. The decomposition is then used to generate equivalent graphs using retrieve.

Indexing We now present in Algorithm 1 the index operator. It consists of three stages: the main decomposition, the building of the ProbTree and the pre-computation of probability distributions.

The first stage of Algorithm 1 (lines 1–14) is the adaptation of the algorithms in [Wei, 2010, Akiba et al., 2012], which build the decomposition tree by essentially applying the Degree heuristic presented in Section 2.2. At each step, a vertex having a degree at most w is chosen, marked as *covered*, and its neighbors are added into the bag, along with the probabilistic edges from $\mathcal{G}$. Then, the covered vertex is removed from the undirected graph G and a clique between the neighbors is created. This process repeats until there are no such vertices left. Finally, the rest of the uncovered vertices and the remaining edges are copied in the root graph $\mathcal{R}$.

The second stage is the creation of the tree $\mathcal{T}$. We visit in creation order each bag and define as their parent the bag whose vertex set contains all uncovered vertices of the visited bag. If no such bag exists, the parent of the bag will be the root graph.

Finally, in each bag B, and for each pair (v_1, v_2), we need to compute $p(v_1 \rightarrow v_2)$ by using the information about the link configuration between v_1, v_2 and the covered vertex v . These can be computed *exactly* by a composition of min- (for parallel paths) and sum-convolutions (for paths that are sequential) of distance distributions – denoted as $\odot$ and $\oplus$ here and illustrated in Figure 2.8. For more details on the computation of convolutions of probability distributions, we refer the reader to [Ash and Doléans, 1999].

More precisely, the probability distribution can be computed as: $p(v_1 \rightarrow v_2) = p(v_1 \rightarrow v_2) \odot (p(v_1 \rightarrow v) \oplus p(v \rightarrow v_2))$. This is followed by the bottom-up propagation of computed probabilities: at each step pairwise probabilities are computed among the vertices which are not the covered vertex v of the respective bag – in precompute-propagate. In order to compute these probabilities, for each bag B, the first step is to "collect" the computed edges from B's children and combine them using the $\odot$ operator. Then, for each pair (v_1, v_2) we compute distances using the $\oplus$ operator between the edges $v_1 \rightarrow v$ and $v \rightarrow v_2$. Finally, the direct edge $v_1 \rightarrow v_2$

[3]Partial decompositions were called *fixed-width decompositions* in [Maniu et al., 2017]. There is no functional difference.

Algorithm 1 index($\mathcal{G}$)

Require: a probabilistic graph $\mathcal{G}$, width parameter w

▷ decompose the graph into bags of size $\leq w$

$G \leftarrow$ undirected, unweighted graph of $\mathcal{G}$
$\mathcal{S} = \emptyset$, $\mathcal{T} = \emptyset$
for $d \leftarrow 1$ **to** w **do**
  while there exists a vertex v with degree d in G **do**
    create new bag B
    $V(B) \leftarrow v$ and all its neighbors
    for all unmarked edge e in $\mathcal{G}$ between vertices of $V(B)$ **do**
      $E(B) \leftarrow E(B) \cup \{e\}$ and mark e
    end for
    covered$(B) \leftarrow \{v\}$
    remove v from G and add to G a $(d-1)$-clique between v's neighbors
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{B\}$
  end while
end for

▷ create the root graph and the bag tree

$V(\mathcal{R}) \leftarrow$ all vertices in $\mathcal{G}$ not in covered$(\mathcal{B})$
$E(\mathcal{R}) \leftarrow$ all unmarked edges in $\mathcal{G}$
for bag B in $\mathcal{S}$ **do**
  mark B
  if $\exists$ an unmarked bag B' s.t. $V(B) \backslash \text{covered}(B) \subseteq B'$ **then**
    update $(\mathcal{T}, \mathcal{B})$ so that B' is parent of B
  else
    update $(\mathcal{T}, \mathcal{B})$ so that $\mathcal{R}$ is parent of B
  end if
end for

▷ compute edges between uncovered vertices and propagate up

for $h \leftarrow$ height$(\mathcal{T})$ to 0 **do**
  for bag B s.t. level$(B) = h$ **do**
    `precompute – propagate`(B)
  end for
end for
root $\mathcal{T}$ at $\mathcal{R}$
  return $(\mathcal{T}, \mathcal{B})$

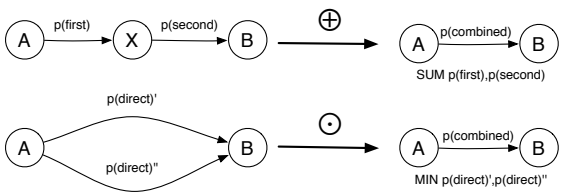


Figure 2.8: Probability compositions of simple paths. [Maniu et al., 2017]

is combined to get the final probability distribution. At the final level – the root bag $\mathcal{R}$ – the computed pairwise distance distributions are simply copied to the edge set of $\mathcal{R}$. Note that we do not compute the distance distributions by using other possible paths between endpoints, and we restrict the computations only between the direct endpoint edge and the unique path going through the covered node. We do this to allow tractability of the convolution computations and allow the same semantics of the edges in $\mathcal{R}$, i.e., each resulting edge between endpoints can be independently sampled.

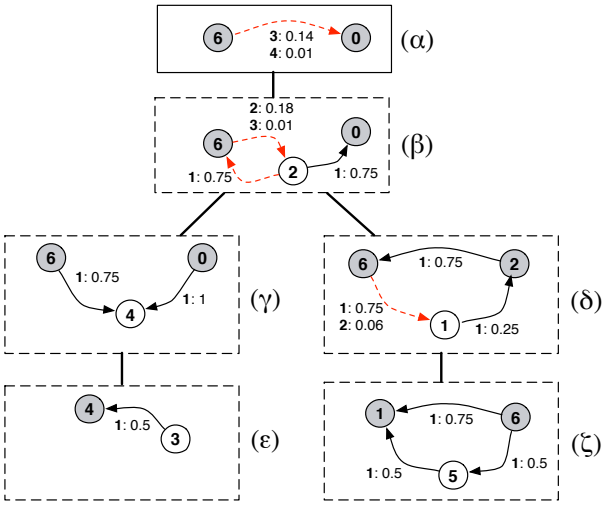


Figure 2.9: The $w = 2$ decomposition of the example graph. Vertices in white are the vertices covered by each bag, and dashed red edges are edges which are computed from children. Each edge has a distribution of distance probabilities associated to it. [Maniu et al., 2017]

Note that we cannot compute bi-directional edges, at least for $w > 2$. Hence, only a single bottom-up propagation is made, from the leaves to $\mathcal{R}$.

Example 2.22. *We give in Fig. 2.9 the result of applying Algorithm 1 on the graph in Fig. 1.2, for $w = 2$. The resulting decomposition consists of five bags in $\mathcal{B}$ and a root graph of two vertices, 6 and 0. Originally, the root graph does not contain any original edges, but it will have computed edges resulting from the bottom-up propagation. In the figure, the dashed red edges represent the edges which have been computed from the children.*

On the left-hand side of the tree, bags (γ) and (ϵ) do not propagate any edges up the tree, as they either do not have 2 endpoints, as is the case of bag (ϵ), or there exist no paths between the endpoints, as for (γ). On the right-hand side, bag (ζ) will provide a $6 \rightarrow 1$ edge to bag (δ). Bag (δ) also propagates edges $6 \rightarrow 2$ and $2 \rightarrow 6$ to bag (β). Finally, bag (β) propagates edge $6 \rightarrow 0$ to the root bag (α).

In terms of time complexity, we know that computing the tree decomposition itself is linear in the number of vertices in the graph [Wei, 2010, Akiba et al., 2012]. The computation of pairwise probability distributions, for each bag, is quadratic in w:

Proposition 2.23. *The complexity of precompute-propagate is $O(w^2 d)$, where d is the maximum distance having non-zero probability in the graph.*

While it is conceivable that a possible world exists in which a shortest distance path between two vertices visits all edges in a graph thus having $d = \Omega(|E|)$, this does not occur in practice. Moreover, for $w \leqslant 2$ (a case which we will explore in more detail), there are only two pairs to

Algorithm 2 retrieve$(\mathcal{T}, \mathcal{B}, s, t)$

Require: ProbTree $(\mathcal{T}, \mathcal{B})$, source s, target t
1: root the tree at one of the bags containing t
▷ propagate edges up the new tree
2: **for** $h \leftarrow$ height$\mathcal{T}$) **to** 0 **do**
3: **for** node n of $\mathcal{T}$ s.t. level$(n) = h$ **do**
4: $B \leftarrow \mathcal{B}(n)$
5: **if** $V(B) \cap \{s\} \neq \emptyset$ **then**
6: delete p^c in parent(B) resulting from B
7: $E(\text{parent}(B)) \leftarrow E(\text{parent}(B)) \cup E(B)$
8: $V(\text{parent}(B)) \leftarrow V(\text{parent}(B)) \cup V(B)$
9: **end if**
10: **end for**
11: **end for**
return $\mathcal{B}(\text{root}(\mathcal{T}))$

generate, and each bag is visited only once by Algorithm 1. Hence, in this setting, the complexity of propagating computations is linear in the number of vertices in the graph.

Retrieval When answering (s, t) queries on the ProbTree we have two main cases. First, when both s and t are present in the root node, we only need to query the root bag with no need to look in the decomposition. The second case is the most interesting one: when at least one of s, t are not in the root, but are vertices in the decomposition bags. In this case, the query vertices need to be propagated to the root node.

The original edges in ancestors of the bags containing the query vertices are propagated up, all the way to the root, in a bottom-up manner. The previous pre-computations of edges in areas of the graphs not containing the query vertices and in the subtree of the bags containing the query vertices are not affected by this change. Recomputing the edges on these parts of the tree is not necessary, and this ensures that only a fraction of the bags in the tree is affected by the retrieval. Algorithm 2 details this operation.

Each retrieve will output a graph that is at most as big as the original graph, and hence the standard shortest-path algorithms [Dijkstra, 1959] would execute in less time for each sample. Moreover, the retrieval is linear in the number of tree bags, which is itself linear in the size of $\mathcal{G}$, verifying Property (iii).

Analysis for $w \leq 2$ In this special case, the computations performed by precompute-propagate are correct:

Proposition 2.24. *precompute-propagate computes correct probability distributions, i.e., does not induce any error, for decompositions of $w \leqslant 2$.*

In addition, for $w \leqslant 2$ the decomposition defines a tree of independent subgraphs, i.e., a ProbTree. It follows that every computed edge in the root graph $\mathcal{R}$ corresponds to an independent subgraph:

Proposition 2.25. *Let $(\mathcal{T}, \mathcal{B})$ be a tree decomposition ProbTree of $w \leqslant 2$. Then every bag B in $\mathcal{B}(\mathcal{T})$ defines an independent subgraph, having as endpoints its uncovered vertices and as internal vertices all covered vertices in the subtree of $\mathcal{T}$ rooted at B.*

Combining the previous results with the complexity bounds on the index and retrieve operators established in the previous section, we obtain that, for $w \leq 2$, (index, retrieve) is a lossless indexing system. On the other hand, since few datasets have treewidth $\leqslant 2$, it is generally not an optimal decomposition into independent subgraphs.

This lossless indexing system provide gains in efficiency close to those of the lossy SPQR indexes. In some cases – such as denser networks – their efficiency is still not fully satisfactory.

Analysis for $w > 2$ Unfortunately, decompositions for $w > 2$ are not lossless, due to the correlations induced by pre-computing the distributions in bags, as witnessed by the following counter-example. Imagine a bag resulting from a $w > 2$ decomposition with covered vertex v and neighbor vertices $v_1, v_2, v_3, \ldots, v_w$ and the following edges: $v_1 \to v$ and $v \to v_2, \ldots, v \to v_w$. In this case, the computable edges would be $v_1 \to v_2, \ldots, v_1 \to v_w$. For every $1 < i \leqslant w$, $p(v_1 \to v_i) = p(v_1 \to v) \oplus p(v \to v_i)$. $p(v_1 \to v)$ appears in all equations, meaning that the computed edges would not be maintaining their independence, hence leading to lossy indexing. No guarantees can be obtained for them either, unlike in the case of SPQR.

As we show in the next section, we can use the decomposition with $w > 2$ as a starting point to design an index structure that is lossless and improves on query time efficiency w.r.t. one having $w \leq 2$, at the cost of an increase in space requirements, by representing explicitly the correlations introduced with higher treewidths.

2.3.2 Lineage Trees

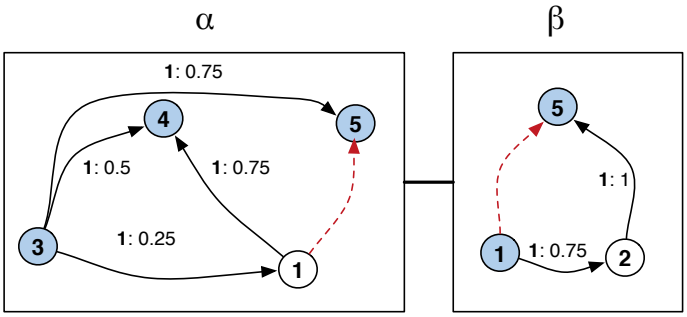
For $w > 2$, directly sampling the pre-computed edges is error-prone. Hence, sampling the pre-computed distance distributions directly from $\mathcal{R}$ or the graph returned by retrieve is not advisable.

Instead, we can compute the full *lineage* of the distance distributions between endpoints at pre-processing, and leave the handling of the correlations at query time. For this, we compute the lineage at tree decomposition time and build a *lineage tree*, i.e., a parse tree of the path between endpoints.

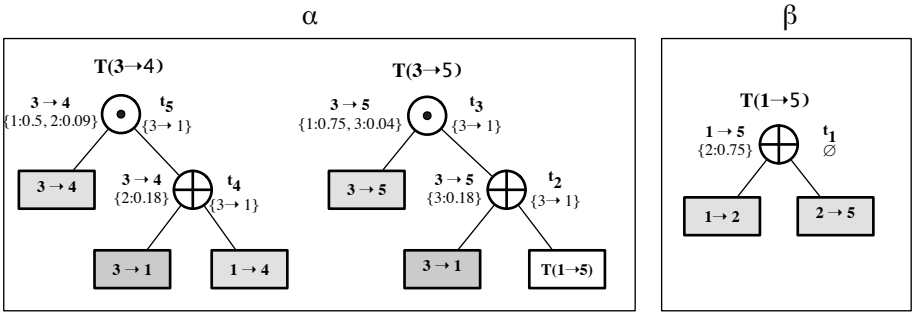
Definition 2.26 (Lineage tree)**.** *A* lineage tree *of a probabilistic graph $\mathcal{G}$ is a binary tree whose leaves are labeled with pairs of nodes of $\mathcal{G}$ and whose internal nodes are labeled with either $\odot$ or $\oplus$.*

The distance distribution *represented by a lineage tree is defined inductively given a distribution of distances on leaf nodes: the distributions of $\odot$- and $\oplus$-labeled internal nodes are given by min- or sum-convolutions of probability distributions of children, following Fig. 2.8.*

Such lineage trees, that will represent the actual distance distribution between two given nodes in a tree, can be efficiently computed at decomposition time by adding tree nodes "on top" of existing tree pointers, coming from previous bags. To enable efficient evaluation of edges which introduce correlations – hereby named *dependency edges* – we annotate each tree node T with a little information:



(a) Original decomposition



(b) Lineage trees

Figure 2.10: Example of dependent path lineages and annotated lineage trees. [Maniu et al., 2017]

(i) the set of dependency edges dependent(T), i.e., the edges which introduce correlations in the entire subtree T;

(ii) the pre-computed distance distribution, $T.dst$, i.e., the distance distribution computed as if dependent$(T) = \emptyset$; and

(iii) the edge being pre-computed, $T.edge$.

Both $T.edge$ and $T.dst$ can be computed directly at decomposition time, just as in the previous decompositions, SPQR and FWD.

The dependent(T) computation goes on as follows. We first compute T with the dependency annotations. For each subtree t that originates from a previous bag in the tree decompositions, we union its dependent(t) to the current dependent(T). Then, for each bag processed in precompute-propagate and for each distance distribution between endpoints, we keep the set of its lineage edges *only from the current bag*, linedges(T). Finally, for each pair of computed endpoint trees T_1 and T_2, we compute linedges$(T_1) \cap$ linedges(T_2) and add it to both dependent(T_1) and dependent(T_2), by set union. This ensures that each subtree will contain the correct set of dependency edges.

Example 2.27. *Let us take the FWD decomposition, with $w = 3$, in Fig. 2.10a, composed of two bags α and β.*

We wish to precompute the edges $3 \longrightarrow 4$ and $3 \longrightarrow 5$ in α. For that, we start at β and precompute $1 \longrightarrow 5$ and its associated lineage tree, $T(1 \longrightarrow 5)$, in Fig. 2.10b. It contains only one convolution node, t_1, and does not have any dependent edges because it belongs to a leaf bag of width $\leqslant 2$.

This lineage tree will be propagated to α and will help in the computation of $3 \longrightarrow 4$ and $3 \longrightarrow 5$ and their associated lineage trees, $T(3 \longrightarrow 4)$ and $T(3 \longrightarrow 5)$. $T(1 \longrightarrow 5)$ is involved in the computation of $T(3 \longrightarrow 5)$, as a pointer node that is a child of the convolution node t_2.

The edge $3 \longrightarrow 1$ introduces a dependency between $3 \longrightarrow 4$ and $3 \longrightarrow 5$ because it appears in the pre-computation of several edges in the same bag, and hence the dependent edge annotation of $T(3 \longrightarrow 4)$ and $T(3 \longrightarrow 5)$ is $\{3 \longrightarrow 1\}$.

The lineage trees described above can be added to computed edges of any partial tree decomposition. Instead of applying the convolution operators in lines 2, 8, and 9, we construct the lineage tree by adding the $\odot$ and $\oplus$ gates as needed. Hence, each edge for bags of $w > 2$ will be associated to a lineage tree. Then, at sampling time, each time a such an edge is encountered we evaluate the corresponding lineage tree, as detailed below.

Given a lineage tree on an edge, evaluating the distance distribution from a tree pointer T is at sample time for a tree pointer T. If the tree pointed by T does not contain any dependency edges, then we simply return the distance distribution $T.dst$. If, on the other hand, the pointer points to a leaf of the tree – which points to a graph edge – and this edge is a dependency edge, we need to sample it in this possible world. To ensure that we keep the correlation in all other possible trees which have this edge as a dependency edge, we need to ensure that the sampled distance is the same in all trees. For this we keep a map sampled which contains the sampled edges in the current possible world, ensuring no sampling of a dependency edge is repeated. If the edge is not a dependency edge, we can return its distance distribution. Finally, for intermediary tree nodes, we recursively evaluate the left and right branches and then compute the convolution indicated by the node, either $\oplus$ or $\odot$. The returned distance distribution d can be sampled by our sampler of choice.

Example 2.28. *Let us return to the tree $T(3 \longrightarrow 5)$ in Fig. 2.10b. At sampling time, edge $3 \longrightarrow 1$ needs to be sampled because it is a dependency edge, i.e., it introduces a correlation with tree $T(3 \longrightarrow 4)$. When $T(3 \longrightarrow 4)$ needs to be evaluated, we need to use this, previously sampled, distance for $3 \longrightarrow 1$. Edge $3 \longrightarrow 5$ and $T(1 \longrightarrow 5)$ can use their distributions without sampling, as they do not have correlations anywhere else in the decomposition.*

The problem with this lineage-based method is that it is not generally space-efficient. On each bag of width w, we only potentially remove $2w$ edges – two for each endpoint covered node pair –, while we can introduce $w(w-1)$ edges to the graph – one edge for all possible pairs of the w endpoints. For $w > 2$, this can add edges to the graph. We thus obtain a quadratic theoretical upper bound on the size of the resulting structure, though, as we will show, the blow-up is not nearly as bad in practice. Consequently, LIN does not satisfy our definition of an indexing system (Definition 2.6). Moreover, the number of computed edges added to $\mathcal{R}$ has a direct influence on query evaluation.

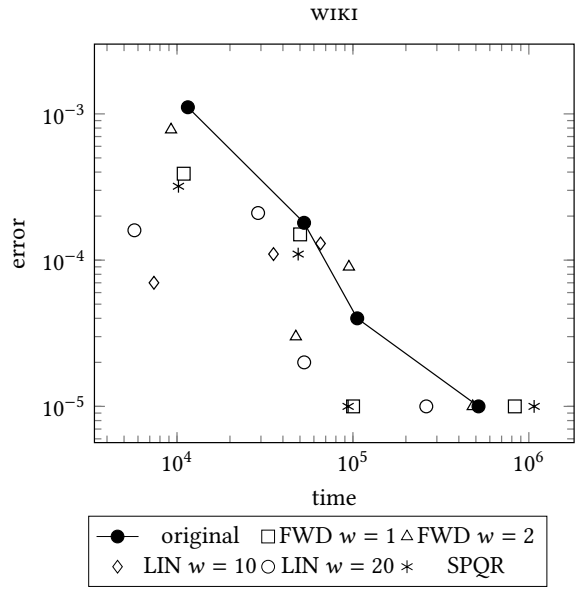


Figure 2.11: Relative error vs. time (log-log axis). [Maniu et al., 2017]

2.3.3 Experimental Results

Are the tree decomposition approaches better overall than sampling algorithms? That is, is the error vs. time trade-off enough to justify using our algorithms, and not simply use more sampling rounds?

We present in this section partial experimental results for the ProbTree variants: SPQR, the fixed-width tree decomposition (denoted as FWD), and the lineage approach (denoted as LIN). The full experimental results are available in [Maniu et al., 2017].

We have plotted the running time of applying sampling on ProbTree versus its error – expressed in terms of the mean squared error as compared to the ground truth results. For brevity, we only track the results for the reachability – or 2-terminal reliability – queries. As query answers are derived directly from the distance distribution, results for other types of queries have similar relative error results.

Fig. 2.11 (log-log axes) presents the results for the Wiki graph – the social network of Wikipedia collaborations. The black dots represent the results on sampling the original graph, for a number of sample rounds between 10 (top left) and 1,000 (bottom right). Intuitively, we want the points corresponding to ProbTree variants (drawn for the same amount of samples) to lie "below" the line induced by the black points, meaning that they yield a better time-accuracy trade-off. As seen before, the gains in execution time when using the decompositions are important. The results also show that the relative error can even be slightly improved when using ProbTree. For instance, note that the FWD and LIN errors in the Wiki graph are slightly lower than the corresponding black dots, i.e., the original graph, suggesting an increase in accuracy.

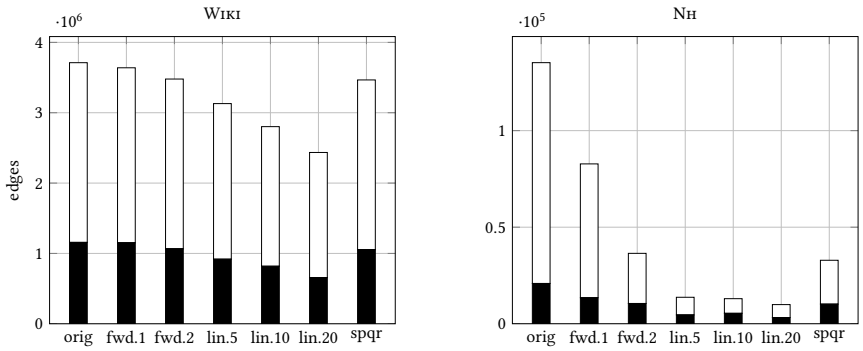


Figure 2.12: Independent graph size (white) versus sampled edges (black).[Maniu et al., 2017]

To understand more about the difference in running time savings of the decomposition, we plot, in Fig. 2.12, the size of the equivalent graph $\mathcal{G}(q)$ resulting from applying retrieve for each type of decomposition (white bars), along with the proportion of actual sampled edges in these graphs (black bars). The sizes represent average sizes over all 1,000 queries in the query load. In the case of Wiki, we observe that the $\mathcal{G}(q)$ graphs are relatively close in size to the original graphs. Generally, the proportion of actually sampled edges in the graph remains constant (around 40% in the case of Wiki, and lower in the case of Nh – the road network of the US state of New Hampshire); this means that, indeed, reducing the number of edges in the equivalent graphs can reduce query times. The large decrease in the query times for the transport networks – such as the illustrated Nh – can be directly attributed to their efficient decompositions, resulting in relatively more independent graphs, which in turn result in much smaller equivalent graphs.

2.4 Follow-up Contributions

We showed that tree decomposition can work well in evaluating source-to-target queries on probabilistic graphs. Interestingly, they also have high potential for other types of queries; we illustrate here kNN queries.

In general, probabilities on graphs can be thought of as a form of *provenance* on graphs. Depending on what type of provenance we use, the efficiency of source-to-target algorithms varies.

2.4.1 Tree Decompositions for Probabilistic kNN Queries

> The following is a brief outline of [Li et al., 2018] presented at EDBT.

Source-to-target graph queries are important; they are however not the only types of queries that could benefit from decomposition approaches. Finding the k-nearest neighbors from a source node in a graph is a query that has multiple applications: e.g., link prediction, community

detection, centrality analysis. For probabilistic graph, this distance is computed in *expectation* over the *distance distribution*.

The same decomposition process can be used for k-NN queries: decompose the graph into a tree and then extract an equivalent graph. Naturally, the decompositions that were presented in the previous section work with this approach – along with other decomposition approaches, such as ones based on core decompositions [Bonchi et al., 2014] and truss decomposition [Huang et al., 2016]. However, there are some particularities for kNN queries. First of all, there will be no clearly defined root of the decomposition, as it depends directly on the choice of the query source node. As a consequence, all the pairwise distances have to be computed in both directions, i.e., bottom-up and top-down. The U-Tree presented in [Li et al., 2018] fulfills the two criteria. In conjunction with some implementation optimization – such as pruning of un-needed bags in the decomposition — the U-Tree can exhibit high efficiency gains in query processing.

2.4.2 Semiring Provenance for Graph Queries

> The following is a brief outline of [Ramusat et al., 2018, Ramusat et al., 2021] presented at TaPP and EDBT.

Tracking *where* and *how* query results are computed – the concept known as *data provenance* – is important for query analysis, conditioning, and explainability. Provenance can be tracked by *annotating* data tuples with the elements of an algebraic structure. In relational databases, the most used structure is the *semiring*, represented by $(\mathbb{K}, \oplus, \otimes, \mathbb{0}, \mathbb{1})$, where $\mathbb{K}$ is some set, $\oplus$ and $\otimes$ are binary operators over $\mathbb{K}$, and $\mathbb{0}$ (annihilator element) and $\mathbb{1}$ (identity) are elements of $\mathbb{K}$, satisfying some axioms. Important examples of semiring are the tropical semiring $(\mathbb{R}, \min, +, \infty, 0)$, the counting semiring $(\mathbb{N}, +, \times, 0, 1)$, and the Boolean semiring $(\{\bot, \top\}, \vee, \wedge, \bot, \top)$. In relational databases and the SPUJ fragment of SQL, the $\oplus$ operation allows to aggregate annotations on tuples for projections, selections and unions, while $\otimes$ allows to aggregate annotations resulting from joins [Green et al., 2007].

In ProbTree's implementation, one important subtask is the pre-computation of probability distribution at the interface between bags in the decomposition. For this, two operations were introduced: the SUM-convolution over sequential paths, and the MIN-convolution over parallel paths.

The above operations are a particular case of the operations that must be performed when answering *any* source-to-target query in graphs: the answer to a query is the *aggregation* over all alternative paths. Our results in [Ramusat et al., 2018, Ramusat et al., 2021] show that this is precisely what we can do with semiring provenance. Formally, in a graph $G = (V, E)$, where each edge $e_i \in E$ has a weight $w[e_i]$ – represented as an element of a semiring –, the provenance of a source-to-target query is the (possibly infinite) sum over the set of all paths between x and y in G, $P_{xy}(G)$:

$$\mathrm{prov}_{\mathbb{K}}(G)(x,y) \coloneqq w\left[P_{xy}(G)\right] = \bigoplus_{\pi \in P_{xy}(G)} w[\pi] = \bigoplus_{\pi \in P_{xy}(G)} \bigotimes_{e_i \in \pi} w[e_i].$$

Depending on the semiring chosen, the semantics of the above sum change. In the *tropical semiring*, this is exactly the shortest distance between nodes x and y in G; in the *counting semiring*, it represents the number of paths between x and y. Finally, in the *Boolean semiring*, it encodes the truth value representing whether y is reachable from x, depending on the truth annotations on edges.

The *type of semiring* has a direct impact on what query processing algorithm one can use and, importantly, on their complexity. Our findings show that:

- The well known Dijkstra's algorithm, linear in the size of the graph, [Dijkstra, 1959] can be directly used with any semiring that is 0-closed and induces a total order over its elements. Intuitively, 0-closed semirings are those for which the above sum does not need to account for cycles in the graph.
- For 0-closed semirings for which elements form a distributive lattice, Dijkstra's algorithm can be used in parallel over the antichains in the lattice (our algorithm MultiDijkstra). The algorithm remains linear, but also depends on the *dimension* of the lattice.
- For k-closed semirings (cycles need to aggregated up to k times), an adaptation of Dikstra's algorithm, presented in [Mohri, 2002], can be used. The algorithm is exponential in the worst case, but much faster in practice.
- For all other semirings for which the above sum is finite, we presented the NodeElimination algorithm, which is cubic in the size of the graph in the worst case. The main cycle of the algorithm eliminate nodes in the graph and compute the provenance over their neighbors, until only the x and y nodes remain. Interestingly, the order in which the nodes are chosen has a direct link with the treewidth of the graph, and hence its performance.

The practical interest of our findings is to allow a *generic* graph processing system for rich queries, where, depending on the chosen provenance over a graph, the best algorithm can be chosen.

3 Maximizing Influence in Low Information Settings

We discuss in this chapter the problem of influence maximization when not all information about the influence graph is known. We start by considering the case of knowing the influence graph, but not the actual influence probabilities on *edges*. Then we relax the need to know the graph topology, by assuming that only the important *nodes* (influencers) are known. The solution to both approaches is to use the explore–exploit paradigm, and specifically *multi-armed bandit* algorithms.

3.1 Influence Maximization

> The following two subsections are an adaption of [Lei et al., 2015], presented at KDD.

Let $G = (V, E, p)$ be an *influence graph*, where $v \in V$ are *users* or nodes, and $e \in E$ are the links or edges between them. Each edge $e = (i, j)$ between users i and j is associated with an *influence probability* $p_{ij} \in [0, 1]$. This value represents the probability that user j is activated by user i at time $t + 1$, given that user i is activated at time t. We also suppose that time flows in discrete, equal steps. In the IM literature, p_{ij} is given for every i and j. Generally, obtaining p_{ij} requires the use of action logs [Goyal et al., 2010], which may not always be available.

In the independent cascade model, at a given timestamp t, every node is in either active (influenced) or inactive state, and the state of each node can be changed from inactive to active, but not vice-versa. When a node i becomes active in step t, the influence is independently propagated at $t + 1$ from node i to its currently inactive neighbors with probability p_{ij}. Node i is given one chance to activate its inactive neighbor. The process terminates when no more activations are possible. A node can be independently activated by any of its (active) incoming neighbors. Suppose that the activation process started from a set S of nodes. We call the expected number of activated nodes of S the *expected influence spread*, denoted $\sigma(S)$. Formally:

Definition 3.1 (Influence)**.** *Given a weighted graph $G = (V, E, p)$, let* infl *be the* immediate influence operator*, which is the random process that extends a set of nodes $X \subseteq V$ into a set of immediately influenced nodes* $\mathrm{infl}(X)$*, as follows:*

$$\Pr(v \in \mathrm{infl}(X)) = \begin{cases} 1 & \text{if } v \in X; \\ 1 - \prod_{\substack{(u,v) \in E \\ u \in X}} (1 - p_{uv}) & \text{otherwise.} \end{cases}$$

Given a seed set $S \subseteq V$, we define the set of influenced nodes $I(S) \subseteq V$ *as the random variable that is the fixpoint $I^{\infty}(S)$ of the following inflationary random process:*

$$\begin{cases} I^0(S) = \emptyset; \\ I^1(S) = S; \\ I^{n+2}(S) = I^{n+1}(S) \cup \text{infl}(I^{n+1}(S) \backslash I^n(S)) \quad \text{for } n \geq 0. \end{cases}$$

The influence spread $\sigma(S)$ is $\mathbb{E}[|I(S)|]$.

Based on the above definition, [Kempe et al., 2003] defines the *influence maximization problem* (IM) as follows.

Problem 3.2 (IM)**.** *Given a weighted graph $G = (V, E, p)$ and a number $1 \leq k \leq |V|$, the* influence maximization *(IM) problem finds a set $S \subseteq V$ such that $\sigma(S)$ is maximal subject to $|S| = k$.*

As discussed in [Kempe et al., 2003], evaluating the influence spread is difficult. Even when the spread values are known, obtaining an exact solution for the IM problem is computationally intractable. Next we outline the existing IM algorithms for this problem.

Influence maximization algorithms A typical IM algorithm evaluates the *score* of a node based on some metric, and inserts the k *best* nodes, which have the highest scores, into S. For example, the degree discount (DD) heuristic [Chen et al., 2009] selects the nodes with highest degree as S. Another classical example is *greedy*: at each step, the *next best node*, or the one that provides the largest marginal increase for σ, is inserted into S. This is repeated until $|S| = k$. The *greedy* algorithm provides an $(1 - 1/e)$-approximate solution for the IM problem. To compute the influence spread efficiently, *sampling-based* algorithms with theoretical guarantees were developed. For example, CELF [Leskovec et al., 2007] evaluates the expected spread of nodes with the seed nodes, and select the nodes with the largest marginal spread; TIM [Tang et al., 2014] counts the frequencies of the nodes appearing in the reversed reachable sets, and chooses the nodes with the highest frequencies; TIM+ [Tang et al., 2014] is an extension of TIM for large influence graphs.

We say that the above IM algorithms are *offline*, since they are executed on the influence graph once, assuming knowledge of p_{ij} for every i and j. If these values are not known, these algorithms cannot be executed. This problem can be addressed by *online* IM algorithms, as we will discuss next.

3.2 Online Influence Maximization

The goal of *online influence maximization* (or OIM) is to perform IM without knowing influence probabilities in advance. Given a number N of advertising campaigns (or *trials*), and an advertising budget of L units per trial, we would like to select up to L seed nodes in each trial. These chosen nodes are then advertised or *activated*, and their feedback is used to decide the seed nodes in the next trial:

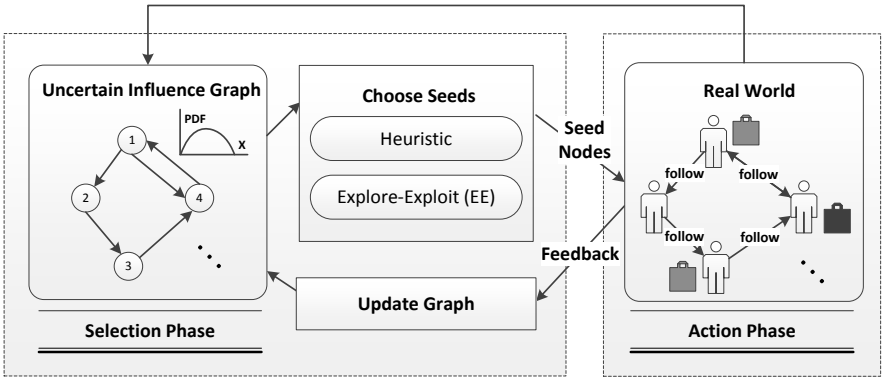


Figure 3.1: The OIM framework.

Problem 3.3 (OIM). *Given a weighted graph $G = (V, E, p)$ with unknown probabilities p_{uv}, and a budget consisting of N trials with $1 \leq L \leq |V|$ activated nodes per trial, the* online influence maximization *(OIM) problem is to find for each $1 \leq n \leq N$ a set S_n of nodes, with $|S_n| \leq L$, such that $\mathbb{E}\left[\left|\bigcup_{1 \leq n \leq N} I(S_n)\right|\right]$ is maximal.*

The IM problem is a special case of the OIM problem (by setting $N = 1$ and assuming the probabilities are known). Since solving the IM problem is computationally difficult, so is finding a solution for the OIM problem. We propose a solution that consists of multiple *trials*. In each trial, a *selection* (for choosing appropriate seed nodes) and an *action* (for activating the seed nodes chosen) is performed (Figure 3.1). The seed selection makes use of one of the offline IM algorithms.

The Uncertain Influence Graph We assume that a social network, which describes the relationships among social network users, is given. However, the exact influence probability on each edge is not known. We model this by using an *uncertain influence graph*, in which the influence probabilities of each edge are captured by probability density functions, or *pdf*. The pdf can be refined based on the feedback returned from a trial. Since influence activations are binary random variable, we capture the uncertainty over the influence as a *Beta distribution*. Specifically, the random variable of the influence probability from node i to node j, P_{ij} is modeled as a Beta distribution having probability density function:

$$f_{P_{ij}}(x) = \frac{x^{\alpha_{ij}-1}(1-x)^{\beta_{ij}-1}}{B(\alpha_{ij}, \beta_{ij})},$$

where $\mathrm{B}(\alpha_{ij}, \beta_{ij})$ is the Beta function, acting as a normalization constant to ensure that the total probability mass is 1, and α_{ij} and β_{ij} are the distribution parameters. For the Beta distribution, $\mathbb{E}[P_{ij}] = \frac{\alpha_{ij}}{\alpha_{ij}+\beta_{ij}}$ and $\sigma^2[P_{ij}] = \frac{\alpha_{ij}\beta_{ij}}{(\alpha_{ij}+\beta_{ij})^2(\alpha_{ij}+\beta_{ij}+1)}$. An advantage of using the Beta distribution is that it is a conjugate prior for Bernoulli distributions, or more generally, binomial distributions. This allows us to compute the posterior distributions easily when new evidence is provided.

At the time of the first trial, we assume no prior information about the influence graph, except global α and β parameters, shared by all edges, i.e., $P_{ij} \sim \mathrm{B}(\alpha, \beta)\ \forall (i, j) \in E$. These global α and

β parameters represent our global *prior belief* of the uncertain influence graph. In the absence of any better prior, we can set $\alpha = \beta = 1$, with $B(1, 1)$ being the uniform distribution.

Our model can be extended to handle various prior information about the influence graph. For example, if we have individual prior knowledge $(\alpha_{ij}, \beta_{ij})$ about an edge, we can set P_{ij} as $P_{ij} \sim \mathrm{B}(\alpha_{ij}, \beta_{ij})$. When we have access to only the mean and variance of the influence of an edge, we can derive α_{ij} and β_{ij} from the formulas of $\mathbb{E}[P_{ij}]$ and $\sigma^2[P_{ij}]$ given above. For the situation in which some action logs involving the social network users are available, algorithms for learning the influence probabilities from these logs [Goyal et al., 2010, Goyal et al., 2011] can be first applied, and the estimated influence probabilities can then be used as prior knowledge for the graph.

Algorithm 3 `Framework`(G, k, N)

Require: number trials N, budget k, uncertain influence graph G
1: $A \leftarrow \emptyset$
2: **for** $n = 1$ **to** N **do**
3: $\quad S_n \leftarrow$ `Choose`(G, k)
4: $\quad (A_n, F_n) \leftarrow$ `RealWorld`(S_n)
5: $\quad A \leftarrow A \cup A_n$
6: $\quad$ `Update`(G, F_n)
7: **end for**
return $\{S_n | n = 1 \ldots N\}, A$

Online Influence Maximization Framework Algorithm 3 depicts the solution framework of the OIM problem. In this algorithm, N trials are executed. Each trial involves selecting seed nodes, activating them, and consolidating feedback from them. In each trial n (where $n = 1, \ldots, N$), the following operations are performed on the uncertain influence graph G:

1. `Choose` (Line 3): A seed set S_n is chosen from G, by using an offline IM algorithm, and strategies for handling the uncertainty of G.

2. `RealWorld` (Lines 4–5): The selected seeds set is tested in the real world (e.g., sending advertisement messages to selected users in the social network). The feedback information from these users is then obtained. This is a tuple (A_n, F_n) comprised of: (i) the set of activated nodes A_n, and (ii) the set of edge activation attempts F_n, which is a list of edges having either a successful or an unsuccessful activation.

3. `Update` (Line 6): We refresh G based on (A_n, F_n).

One could also choose not to update G, and instead only run an offline IM based on the prior knowledge.

Algorithms for OIM The above framework lends itself naturally to the *explore-exploit* paradigm. When *exploring* the objective is to try different seeds so that the estimation of

the influence probability distributions P_{ij} is further refined. The *exploit* branch simply apply IM algorithms to the current expectation of the probabilities $\mathbb{E}[P_{ij}]$.

In this paradigm, several solutions are possible. The simplest one is the ϵ-greedy approach: with probability ϵ one chooses the exploit branch, and with $1-\epsilon$ the explore branch is chosen and random seeds are selected. In the current *multi-armed bandit* literature [Lattimore and Szepesvári, 2020], however, the exploration and exploitation are integrated into a single parameter, the *confidence bound*. Assuming that each edge in the graph is considered a *bandit* with distribution P_{ij}, then the confidence bound approach estimates p_{ij} as:

$$p_{ij} = \mathbb{E}[P_{ij}] + \theta_{ij}.$$

How θ_{ij} is chosen varies: from simple heuristics estimating the bound, to the UCB-like approaches using the *optimism in the face of uncertainty* principle: assuming that the edge distributions are always underestimated and hence $\theta > 0$. As the edge (i, j)'s estimation is improved, θ gets closer to zero. UCB algorithms in the bandit literature establish how to set the bound in relation to the number of rounds, to achieve regret that is sub-linear in the number of rounds. In [Lei et al., 2015], we used the classical ExponentiatedGradient (EG) algorithm to update the θ values.

As discussed before, the issue with this approach is that one assumes that the graph topology is available and represents the actual diffusion graph. Moreover, estimating each edge means one has to estimate too many variables.

We discuss now an approach in which the graph topology is not necessary, and the estimation is made at the seed (or *influencer*) level.

3.3 Online Influence Maximization with Persistence

> The following subsection is an adaptation of [Lagrée et al., 2019], published in ACM Transactions on Knowledge Discovery from Data.

In this approach, in contrast to [Lei et al., 2015], we do not try to estimate edge probabilities in a diffusion graph, but, instead, we assume the existence of a known set of spread seed candidates – in the following referred to as the *influencers* – who are the only access to the medium of diffusion. Formally, we let $[K] := \{1, \dots, K\}$ be a set of influencers up for selection; each influencer is connected to an *unknown* and potentially large base (the influencer's *support*) of basic nodes, each with an unknown activation probability. For illustration, we give in Figure 3.2 an example of this setting, with 3 influencers connected to 4, 5, and 4 basic nodes, respectively. Let $A_k \subseteq V$, for $k = 1, \dots, K$, denote the sets of basic nodes such that each influencer $k \in [K]$ is connected to each node in A_k. We further denote by $p_k(u)$ the probability for influencer k to activate the child node $u \in A_k$. In this context, the diffusion process can be abstracted as follow:

Definition 3.4 (Influence process)**.** *When an influencer* $k \in [K]$ *is selected, each basic node* $u \in A_k$ *is* sampled *for activation, according to its probability* $p_k(u)$*. The* feedback *for* k*'s selection consists of all the activated nodes, while the associated* reward *consists only of the* newly activated *ones.*

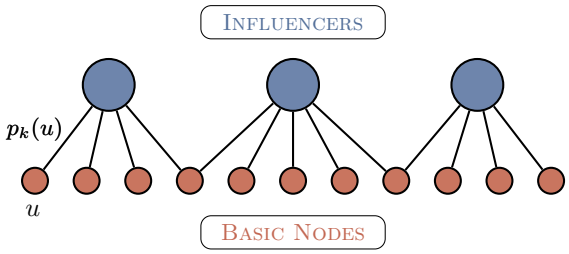


Figure 3.2: Three influencers with associated activation probabilities $p_k(u)$.

Limiting the influence maximization method to working with a small subset of the nodes allows to accurately estimate their value more rapidly, even in a highly uncertain environment, hence the algorithmic interest. At the same time, this is directly motivated by marketing scenarios involving marketers who only have access to a few influencers who can diffuse information. Moreover, despite the fact that we model the social reach of each influencer by 1-hop links to the to-be-influenced nodes, these edges are just an abstraction of the activation probability, and may represent in reality longer paths in an underlying unknown influence graph. Importantly, this approach assumes a form of *persistence*: each influencer can be activated multiple times, and the influence process can occur over the same nodes; however, only *new* nodes are to be counted.

We are now ready to define the *online influencer marketing with persistence* problem:

Problem 3.5 (OIMP). *Given a set of influencers* $[K] := \{1, \ldots, K\}$*, a* budget *of* N *trials, and a number* $1 \leq L \leq K$ *of influencers to be activated at each trial, the objective of the* online influencer marketing with persistence *(OIMP) is to solve the following optimization problem:*

$$\underset{I_n \subseteq [K], |I_n| = L, \forall 1 \leqslant n \leqslant N}{\arg\max} \mathbb{E}\left| \bigcup_{1 \leqslant n \leqslant N} S(I_n) \right|.$$

As noticed before, the offline influence maximization can be seen as a special instance of the online one, where the budget is $N = 1$ (single-trial campaigns).

Lemma 3.6. *The OIMP problem is NP-hard.*

Note that, in contrast to persistence-free online influence maximization – considered in [Vaswani et al., 2017, Wen et al., 2017] – the performance criterion used in OIMP displays the so-called *diminishing returns property*: the expected number of nodes activated by successive selections of a given seed is decreasing, due to the fact that nodes that have already been activated are discounted. We refer to the expected number of nodes remaining to be activated as the *remaining potential* of a seed. The diminishing returns property implies that there is no static best set of seeds to be selected, but that the algorithm must follow an adaptive policy, which can detect that the remaining potential of a seed is small and switch to another seed that has been less exploited. Our solution to this problem has to overcome challenges on two fronts: (1) it needs to estimate the potential of nodes at each round, without knowing the diffusion

model nor the activation probabilities, and (2) it needs to identify the currently best influencers, according to their estimated potentials.

In this approach, we work with parameters on *nodes*, instead of edges. More specifically, these parameters represent the potentials of remaining spreads from each of the influencer nodes. This way, we can go around the dependencies on specific diffusion models, and furthermore, we can address settings in which one does not have access to a detailed graph topology.

3.3.1 Remaining Potential and Good-Turing Estimator

A good algorithm for OIMP should aim at selecting the influencer k with the largest remaining potential for influencing its children A_k. However, the true potential value of an influencer is *a priori* unknown to the decision maker.

In the following, we index trials by t when referring to the time of the algorithm, and we index trials by n when referring to the number of selections of the influencer. For example, the t-th spread initiated by the algorithm is noted $S(t)$ whereas the n-th spread of influencer k is noted $S_{k,n}$.

Definition 3.7 (Remaining potential $R_k(t)$)**.** *Consider an influencer $k \in [K]$ connected to A_k basic nodes. Let $S(1), \dots, S(t)$ be the set of nodes that were activated during the first t trials by the seeded influencers. The* remaining potential $R_k(t)$ *is the expected number of* new *nodes that would be activated upon starting the $t+1$-th cascade from k:*

$$R_k(t) := \sum_{u \in A_k} \mathbb{1}\left\{u \notin \bigcup_{i=1}^{t} S(i)\right\} p_k(u),$$

where $\mathbb{1}\{\cdot\}$ denotes the indicator function.

Definition 3.7 provides a formal way to obtain the remaining potential of an influencer k at a given time. The optimal policy would simply select the influencer with the largest remaining potential at each time step. The difficulty is, however, that the probabilities $p_k(u)$ are unknown. Hence, we have to design a *remaining potential estimator* $\hat{R}_k(t)$ instead. It is important to stress that the remaining potential is a random quantity, because of the dependency on the spreads $S(1), \dots, S(t)$. Furthermore, due to the diminishing returns property, the sequence $(S_{k,n})_{n \geq 1}$ is stochastically decreasing.

Following ideas from [Good, 1953, Bubeck et al., 2013], we now introduce a version of the Good-Turing statistic, tailored to our problem of rapidly estimating the remaining potential. Denoting by $n_k(t)$ the number of times influencer k has been selected after t trials, we let $S_{k,1}, \dots, S_{k,n_k(t)}$ be the $n_k(t)$ cascades sampled independently from influencer k. We denote by $U_k(u,t)$ the binary function whose value is 1 if node u has been activated *exactly* once by influencer k – such occurrences are called *hapaxes* in linguistics – and $Z_k(u,t)$ the binary function whose value is 1 if node u has never been activated by influencer k. The principle of the Good-Turing estimator is to estimate the remaining potential as the proportion of hapaxes within the $n_k(t)$ sampled cascades:

$$\hat{R}_k(t) := \frac{1}{n_k(t)} \sum_{u \in A_k} U_k(u,t) \prod_{l \neq k} Z_l(u,t).$$

Albeit simple, this estimator turns out to be quite effective in practice. If an influencer is connected to a combination of both nodes having high activation probabilities and nodes having low activation probabilities, then successive traces sampled from this influencer will result in multiple activations of the high-probability nodes and few of the low-probability ones. Hence, after observing a few spreads, the influencer's potential will be low, a fact that will be captured by the low proportion of hapaxes. In contrast, estimators that try to estimate each activation probability independently will require a much larger number of trials to properly estimate the influencer's potential. This assumption was verified by an analysis of the empirical activation probabilities from a Twitter dataset [Lagrée et al., 2019].

While bearing similarities with the traditional missing mass concept in the bandit literature, we highlight one fundamental difference between the remaining potential and the traditional missing mass studied in [Bubeck et al., 2013], which impacts both the algorithmic solution and the analysis. Since at each step, *every* node connected to the selected influencer is sampled, the algorithm receives a larger feedback than in [Bubeck et al., 2013], whose feedback is in $\{0, 1\}$. However, contrary to [Bubeck et al., 2013], the hapaxes of an influencer $(U_k(u,t))_{u\in A_k}$ are independent. Interestingly, the quantity $\lambda_k := \sum_{u\in A_k} p(u)$, which corresponds to the expected number of basic nodes a influencer k activates in a cascade, will prove to be a crucial ingredient for our problem.

Upper confidence bounds Following principles from the bandit literature, the GT-UCB algorithm relies on *optimism in the face of uncertainty*. At each step (trial) t, the algorithm selects the highest upper-confidence bound on the remaining potential – denoted by $b_k(t)$ – and activates (plays) the corresponding influencer k. This algorithm achieves robustness against the stochastic nature of the cascades, by ensuring that influencers who "underperformed" with respect to their potential in previous trials may still be selected later on. Consequently, GT-UCB aims to maintain a degree of *exploration* of influencers, in addition to the *exploitation* of the best influencers as per the feedback gathered so far.

Algorithm 4 – GT-UCB ($L = 1$)

Require: Set of influencers $[K]$, time budget N
1: **initialization** play each influencer $k \in [K]$ once, observe the spread $S_{k,1}$, set $n_k = 1$
2: **for** $t = K + 1, \ldots, N$ **do**
3: Compute $b_k(t)$ for every influencer k
4: Choose $k(t) = \arg\max_{k\in[K]} b_k(t)$
5: Play influencer $k(t)$ and observe spread $S(t)$
6: Update statistics of influencer $k(t)$: $n_{k(t)}(t+1) = n_{k(t)}(t) + 1$ and $S_{k,n_{k(t)}} = S(t)$.
7: **end for**
return W

Algorithm 4 presents the main components of GT-UCB for the case $L = 1$, that is, when a single influencer is chosen at each step.

The algorithm starts by activating each influencer $k \in [K]$ once, in order to initialize its Good-Turing estimator. The main loop of GT-UCB occurs at lines 2-7. Let $S(t)$ be the observed spread at trial t, and let $S_{k,s}$ be the result of the s-th diffusion initiated at influencer k. At every

step $t > K$, we recompute for each influencer $k \in [K]$ its index $b_k(t)$, representing the upper confidence bound on the expected reward in the next trial. The computation of this index uses the previous samples $S_{k,1}, \ldots, S_{k,n_k(t)}$ and the number of times each influencer k has been activated up to trial t, $n_k(t)$. Based on the result of Theorem 3.9, the upper confidence bound is set as:

$$b_k(t) = \hat{R}_k(t) + \left(1 + \sqrt{2}\right) \sqrt{\frac{\hat{\lambda}_k(t) \log(4t)}{n_k(t)}} + \frac{\log(4t)}{3n_k(t)}, \tag{3.1}$$

where $\hat{R}_k(t)$ is the Good-Turing estimator and $\hat{\lambda}_k(t) := \sum_{s=1}^{n_k(t)} \frac{|S_{k,s}|}{n_k(t)}$ is an estimator for the expected spread from influencer k.

Then, in line 4, GT-UCB selects the influencer $k(t)$ with the largest index, and initiates a cascade from this node, yielding $S(t)$. We stress again that $S(t)$ provides only the Ids of the nodes that were activated, with no information on *how* this diffusion happened in the hidden diffusion medium. Finally, the statistics associated to the chosen influencer $k(t)$ are updated.

Extensions for the case $L > 1$ Algorithm 4 can be easily adapted to select $L > 1$ influencers at each round. Instead of choosing the influencer maximizing the Good-Turing UCB in line 4, we can select those having the L largest indices. Note that $k(t)$ then becomes a *set* of L influencers. In the beginning, the algorithm can select in some predefined (random) order L influencers at each round, for initialization. It is required that all influencers be activated at least once for initialization – before entering the main loop of the GT-UCB algorithm – in order for the estimators $\hat{R}_k(t)$ to be well defined. How this initial stage is done is not essential and may depend on the specific application scenario. At each round, a diffusion is initiated from the associated nodes and, at termination, all activations are observed. Similarly to [Vaswani et al., 2017], the algorithm requires feedback to include the influencer responsible for the activation of each node, in order to update the corresponding statistics accordingly.

3.3.2 Theoretical Analysis

In the following, to simplify the analysis and to allow for a comparison with the oracle strategy, we assume that the influencers have *non intersecting support*. This means that each influencer's remaining potential and corresponding Good-Turing estimator does not dependent on other influencers. Hence, for notational efficiency, we also omit the subscript denoting the influencer k. After selecting the influencer n times, the Good-Turing estimator is simply written $\hat{R}_n = \sum_{u \in A} \frac{U_n(u)}{n}$. We note that the non-intersecting assumption is for theoretical purposes only – our experiments are done with influencers that can have intersecting supports.

The classic Good-Turing estimator is known to be slightly biased (see for example Theorem 1 in [McAllester and Schapire, 2000]). We show in Lemma 3.8 that our remaining potential estimator adds an additional factor $\lambda = \sum_{u \in A} p(u)$ to this bias:

Lemma 3.8. *The bias of the remaining potential estimator is*

$$\mathbb{E}[R_n] - \mathbb{E}[\hat{R}_n] \in \left[-\frac{\lambda}{n}, 0\right].$$

Since λ is typically very small compared to $|A|$, in expectation, the estimation should be relatively accurate. However, in order to understand what may happen in the worst-case, we need to characterize the deviation of the Good-Turing estimator:

Theorem 3.9. *With probability at least* $1-\delta$*, for* $\lambda = \sum_{u \in A} p(u)$ *and* $\beta_n := \left(1+\sqrt{2}\right)\sqrt{\frac{\lambda \log(4/\delta)}{n}} + \frac{1}{3n}\log\frac{4}{\delta}$*, the following holds:*

$$-\beta_n - \frac{\lambda}{n} \leq R_n - \hat{R}_n \leq \beta_n.$$

Note that the additional term appearing in the left deviation corresponds to the bias of our estimator, which leads to a non-symmetrical interval.

We can now provide an analysis of the *waiting time* (defined below) of GT-UCB, by comparing it to the waiting time of an oracle policy, following ideas from [Bubeck et al., 2013]. Let $R_k(t)$ be the remaining potential of influencer k at trial number t. This differs from $R_{k,n}$, which is the remaining potential of influencer k once *it* has been played n times.

Definition 3.10 (Waiting time)**.** *Let* $\lambda_k = \sum_{u \in A_k} p(u)$ *denote the expected number of activations obtained by the first call to influencer* k*. For* $\alpha \in (0,1)$*, the* waiting time $T_{UCB}(\alpha)$ *of GT-UCB represents the round at which the remaining potential of* each *influencer* k *is smaller than* $\alpha\lambda_k$*. Formally,*

$$T_{UCB}(\alpha) := \min\{t : \forall k \in [K], R_k(t) \leq \alpha\lambda_k\}.$$

The above definition can be applied to any strategy for influencer selection and, in particular, to an oracle one that knows beforehand the α value that is targeted, the spreads $(S_{k,s})_{k\in[K],1\leq s\leq t}$ sampled up to the current time, and the individual activation probabilities $p_k(u), u \in A_k$. A policy having access to all these aspects will perform the fewest possible activations on each influencer. We denote by $T^*(\alpha)$ the waiting time of the oracle policy. We are now ready to state the main theoretical property of the GT-UCB algorithm.

Theorem 3.11 (Waiting time)**.** *Let* $\lambda^{min} := \min_{k\in[K]} \lambda_k$ *and let* $\lambda^{max} := \max_{k\in[K]} \lambda_k$*. Assuming that* $\lambda^{min} \geq 13$*, for any* $\alpha \in \left[\frac{13}{\lambda^{min}}, 1\right]$*, if we define* $\tau^* := T^*\left(\alpha - \frac{13}{\lambda^{min}}\right)$*, with probability at least* $1 - \frac{2K}{\lambda^{max}}$ *the following holds:*

$$T_{UCB}(\alpha) \leq \tau^* + K\lambda^{max}\log(4\tau^* + 11K\lambda^{max}) + 2K.$$

3.3.3 Experimental Results

We conducted experiments on two types of datasets: (i.) two graphs, widely-used in the influence maximization literature, and (ii.) a crawled dataset from Twitter, consisting of tweets occurring during August 2012. All methods are implemented in C++ [1]

In [Lei et al., 2015] we compared the solutions on the Weighted Cascade (WC) instance of IC, where the influence probabilities on incoming edges sum up to 1. More precisely, every edge (u,v) has weight $1/d_v$ where d_v is the in-degree of node v. In [Lagrée et al., 2019] we added two

[1] The code is available at `https://github.com/smaniu/oim`.

other diffusion scenarios to the set of experiments. First, we included the tri-valency model (TV), which associates randomly a probability from $\{0.1, 0.01, 0.001\}$ to every edge and follows the IC propagation model. We also conducted experiments under the Linear Threshold (LT) model, where the edge probabilities are set like in the WC case and where thresholds on nodes are sampled uniformly from $[0, 1]$.

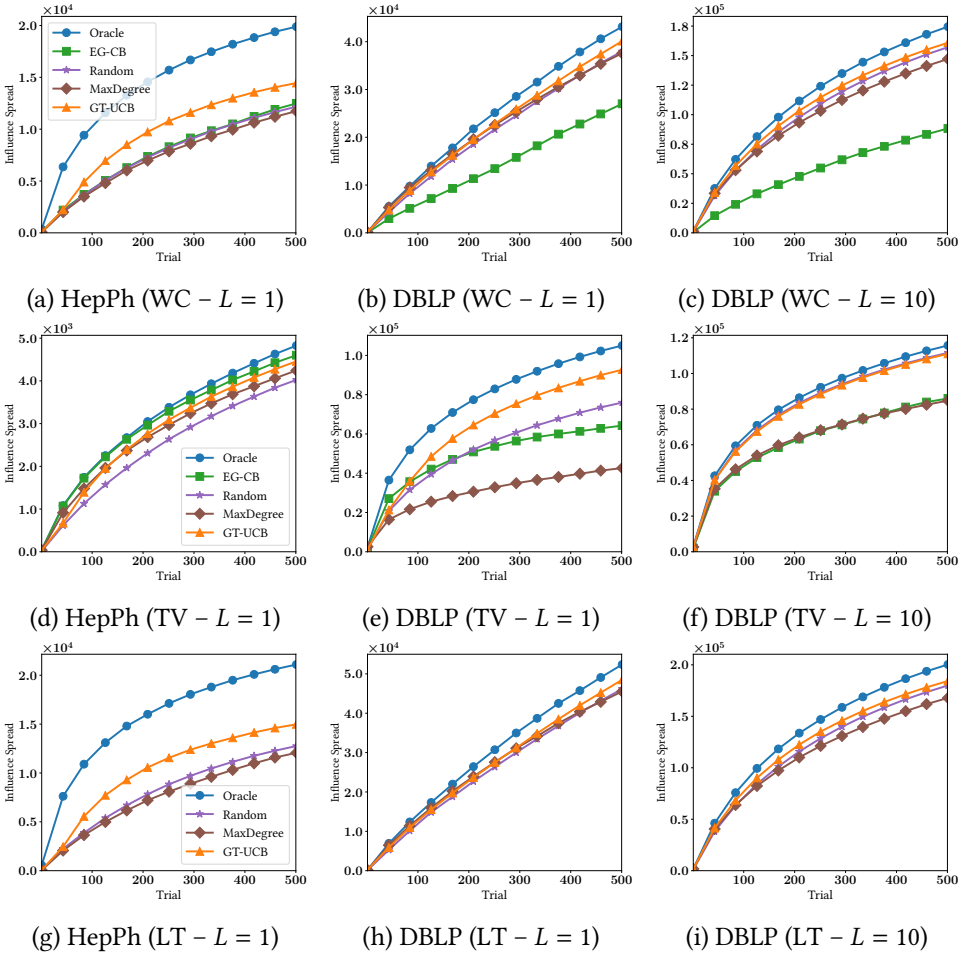
Algorithms We compare GT-UCB to several other algorithms. Random chooses a random influencer at each round. MaxDegree selects the node with the largest degree at each step i, where the degree does not include previously activated nodes. Finally, EG corresponds to the confidence-bound explore-exploit method with exponentiated gradient update from [Lei et al., 2015]. We use EG on WC and TV weighted graphs; note that EG learns parameters for the IC model, and hence is not applicable for LT. These baselines are compared to an Oracle that knows beforehand the diffusion model together with probabilities. At each round, it runs an influence maximization approximated algorithm – PMC for IC propagation, SSA for LT. Note that previously activated nodes are not counted when estimating the value of a node with PMC or SSA, thus, making Oracle an adaptive strategy.

All experiments are done by fixing the trial horizon $N = 500$, a setting that is in line with many real-world marketing campaigns, which are fairly short and do not aim to reach the entire population.

In Figure 3.3, we show the growth of the spread for the various approaches. For each experiment, GT-UCB uses $K = 50$ if $L = 1$ and $K = 100$ if $L = 10$. First, we can see that MaxDegree is quite a strong baseline in many cases, especially for WC and LT. GT-UCB results in good quality spreads across every combination of network and diffusion model. Interestingly, on the smaller graph HepPh, we observe an increase in the slope of spread after initialization, particularly visible at $t = 50$ with WC and LT. This corresponds to the step when GT-UCB starts to select influencers maximizing $b_k(t)$ in the main loop. It shows that our strategy adapts well to the previous activations, and chooses good influencers at each iteration. Interestingly, Random performs surprisingly well in many cases, especially under TV weight assignment. However, when certain influencers are significantly better than others, Random cannot adapt to this diversity to select the best influencers, unlike GT-UCB. EG performs well on HepPh, especially under TV weight assignment. However, it fails to provide competitive cumulative spreads on DBLP. We believe that EG tries to estimate too many parameters for a horizon $T = 500$. After reaching this time step, less than 10% of all nodes for WC, and 20% for TV, are activated. This implies that we have hardly any information regarding the majority of edge probabilities, as most nodes are located in parts of the graph that have never been explored.

Using real influence traces: Twitter The interest of this experiment is to observe actual spreads, instead of simulated ones, over data that does not provide an explicit influence graph.

From the retweeting logs, for each *active* user u – a user who posted more than 10 tweets – we select users having retweeted at least one of u's tweets. By doing so, we obtain the set of potentially influenceable users associated to active users. We then apply the greedy algorithm to select the users maximizing the corresponding set cover. These are the influencers of GT-UCB and Random. MaxDegree is given the entire reconstructed network, that is, the network



(a) HepPh (WC – $L = 1$) (b) DBLP (WC – $L = 1$) (c) DBLP (WC – $L = 10$)

(d) HepPh (TV – $L = 1$) (e) DBLP (TV – $L = 1$) (f) DBLP (TV – $L = 10$)

(g) HepPh (LT – $L = 1$) (h) DBLP (LT – $L = 1$) (i) DBLP (LT – $L = 10$)

Figure 3.3: Growth of spreads against the number of rounds.

connecting active users to re-tweeters.

To test realistic spreads, at each step, once an influencer is selected by GT-UCB, a random cascade initiated by that influencer is chosen from the logs and we record its spread. This provides realistic, model-free spread samples to the compared algorithms. Since Twitter only contains successful activations (re-tweets) and not the failed ones, we could not test against EG, which needs both kinds of feedback.

In Fig. 3.4, we show the growth of the diffusion spread of GT-UCB against MaxDegree and Random. Again, GT-UCB uses $K = 50$ if $L = 1$ and $K = 100$ if $L = 10$. We can see that GT-UCB outperforms all the baselines, especially when a single influencer is selected at each round. We can observe that MaxDegree performs surprisingly well in both experiments. We emphasize that MaxDegree relies on the knowledge of the entire network reconstructed from retweeting logs, whereas GT-UCB is only given a set of (few) fixed influencers.

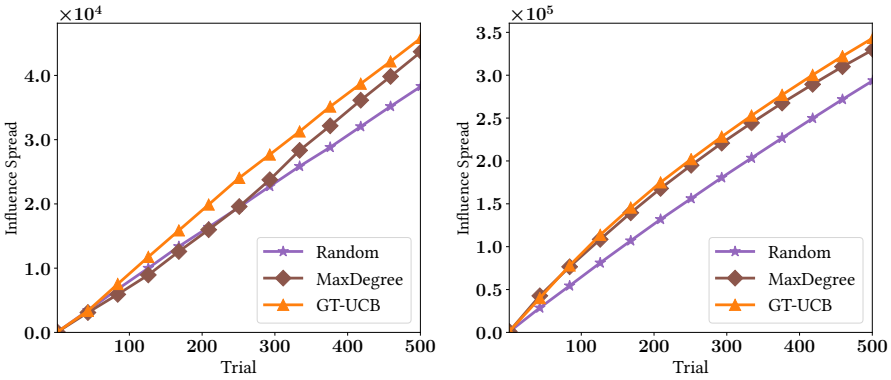


Figure 3.4: Twitter spread against rounds: (left) $L = 1$ (right) $L = 10$.

3.4 Follow-up Contributions

Considering influence dynamics in an online explore-exploit setting allows us to revisit some algorithms from the bandit literature. We considered two scenarios: when influence maximization depends on the *context* of the message, and how to design online recommender systems while taking into account user profile *influence* dynamics.

3.4.1 Contextual Online Influence Maximization

> The following is a brief outline of [Iacob et al., 2022], to be presented at SDM.

Within an influence maximization campaign, the manner in which the information may be formulated, presented, or diffused may vary from round to round, and the *context* variations will lead to different propagation dynamics. For example the focus of a campaign may be political, and individual messages may take different contexts (news, data analysis, multimedia content).

Adapting the online influence maximization problem to contexts directs us to the *contextual multi-armed bandits*, and specifically their linear formulation. In the contextual multi-armed bandit formulation, at each round t, a context is $Y_t \in \mathbb{R}^d$ received (adversarially). The reward is then a linear combination between the context and some unknown *profile* vector $\theta_k \in \mathbb{R}^d$, plus some noise:

$$r_t = \langle \theta_{k,t}, Y_t \rangle + \epsilon.$$

In the literature, UCB-like approaches use linear regression to estimate the profile vector $\hat{\theta}_{k,t}$, and the upper confidence bound depends on the covariance matrix. The algorithms that are generally used in the literature are some variation of LinUCB (Linear Upper Confidence Bound) [Chu et al., 2011]. The setting can be extended to generalized linear models [Li et al., 2017].

Applying to our case, each influencer k will change the base probabilities of each basic node j by a function α, depending on Y_t, θ_k, and the number of selections of influencer k, $n_{k,t}$:

$$p_{k,j}(t) = \alpha(\langle \theta_k, Y_t \rangle, n_{k,t}) p_j.$$

The function α is introduced to take into account the *influencer fatigue*, i.e., the fact that, with each activation, the number of *new* nodes influences tends to decrease.

In [Iacob et al., 2022], we designed two linear bandit algorithms – performing better than both non-contextual and simple linear bandit algorithms – working on two different assumptions:

1. The first is assuming that the *logarithm of rewards is normally distributed* – in other words, we are more interested in predicting the *scale* of the cascade rather than its size. The resulting algorithm, `LogNorm-LinUCB`, is an application of LinUCB on logarithms of rewards, and keeps the same regret ratio.

2. The second is an assumption that the *distribution of each node's cumulative count of activations* is Poisson with intensities $\lambda_j \sum_{s=1}^{t} \sum_{k \in I_s} \alpha(\langle \theta_k, Y_s \rangle,\ n_{k,s})$. We further assume that the remaining potential also follows a Poisson distribution with intensities $\alpha(\langle \theta_k, Y_t \rangle, n_k(t))\lambda_k$; in this case, the Good-Turing estimator can be adapted to account the distribution and used as an estimator into a generalized linear model (algorithm `GLM-GT-UCB`).

3.4.2 Using Influence for Item Recommendation

> The following is a brief outline of [Maniu et al., 2020], presented at ICDM.

The influence mechanics between users can also be a proxy for modeling user preference dynamics, and can be part of recommender systems which learn interests to recommend items (documents, videos, etc.) to users [Lu et al., 2014]. Sequential learning approaches, such as the multi-armed bandit setting briefly described above, are natural algorithmic toolboxes for this setting.

In this work, we modeled the rating users give as a linear relationship between their hidden interest and the item being recommended, in a time-discrete recommender system. At each $t \in \mathbb{N}$, users $i \in [n]$ have *user profiles* represented by d-dimensional vectors $\mathbf{u}_i(t) \in \mathbb{R}^d$. Then, if $\mathbf{v}_i(t) \in \mathcal{B}$ (some subset of $\mathbb{R}^d$) is the profile of the item recommended to i at time t, ratings satisfy:

$$r_i(t) = \langle \mathbf{u}_i(t), \mathbf{v}_i(t) \rangle + \varepsilon.$$

Importantly, at each time $t \in \mathbb{N}$, user profiles evolve according to:

$$\mathbf{u}_i(t) = \alpha \mathbf{u}_i^0 + (1 - \alpha) \sum_{j \in [n]} P_{i,j} \mathbf{u}_j(t-1),\ i \in [n],$$

where (a) $\mathbf{u}_i^0 \in \mathbb{R}^d$ is user i's inherent (static) profile, (b) $\alpha \in [0, 1]$ captures the probability that users act based on their inherent profiles, and (c) $P_{ij} \in [0, 1]$, $i, j \in [n]$, where $\sum_j P_{ij} = 1$, capture the probability user i is influenced by the profile of user j.

Item suggestions $\mathbf{v}_i(t)$, $i \in [n]$ are selected from a set $\mathcal{B} \subseteq \mathbb{R}^d$. We consider two possibilities:

- $\mathcal{B}$ is a finite subset of $\mathbb{R}^d$, i.e., it is a "catalog" of possible recommendations.

- $\mathcal{B}$ is an arbitrary convex subset of $\mathbb{R}^d$, e.g., the unit ball $\mathbb{B} \equiv \{\mathbf{v} \in \mathbb{R}^d : \|\mathbf{v}\|_2 \leq 1\}$.

We show that, in all cases, the aggregated expected reward (i.e., the sum of ratings over time) depends linearly on the matrix of initial user profiles U^0. This motivated our exploration of the classic linear bandit optimization problem, such as LinREL [Dani et al., 2008], Thompson Sampling [Agrawal and Goyal, 2014] and LinUCB. Approaches such as LinREL are similar to bandit approaches, with the difference that, instead of having a uni-dimensional confidence bound, the actual bound is obtained by an optimization on a L_1 or L_2 constraint ellipsoid around the estimation. Thompson Sampling samples $\mathbf{u}_i(t)$ from their posterior distributions, updated after the feedback has been obtained – in that sense, the exploration is inherent in the uncertainty of the distribution.

For LinREL and Thompson sampling we obtained the *same* regret bounds that the ones obtained in the static case, depending on the number of rounds T: a $\tilde{O}(\sqrt{T})$ regret bound for LinREL, and the same $\tilde{O}(\sqrt{T})$ bound on the Bayesian regret for Thompson Sampling.

In the general case, the optimization around the confidence ball is computationally expensive. Importantly, we show that we can obtain efficient polynomial algorithms[2] when the optimization is on L_1 constraints and $\mathcal{B}$ is in one of the two above cases (finite set and convex set). Unfortunately, while applying the classical LinUCB is a possibility, the resulting optimization is non-convex; SDP relaxations can be used, but without any theoretical guarantee.

[2]Implementation at `https://github.com/neu-spiral/OnlineSocialRecommendations`.

4 Research Perspectives

We presented in this manuscript algorithms for tree decomposition for answering queries on probabilistic graphs, and multi-armed bandit approaches to solve the influence maximization problems in the case where little is known about the underlying process and social network topology. The inherent *uncertainty* in the settings of both problems renders the classical algorithms unusable, as we have shown in the previous chapters. It is not the last word on the matter: there is exciting potential in extending decompositions to generic query processing on graphs and – in the influence maximization case – when using more powerful sequential learning approaches such as reinforcement learning coupled with representation learning.

Leveraging structure for graph problems We now know that *the structure of data matters*, at least when it comes to graphs. We noticed this first in our research on source-to-target reachability in uncertain graphs [Maniu et al., 2017]. There, we devised an index that allows faster (approximate) query answering of probabilistic reachability queries, by creating a *tree decomposition* of the data. How efficient such an index is is directly related to the *treewidth* of the graph – informally, a quantity measuring how "close" a graph is to a tree. Treewidth is a very important theoretical tool to establish fixed-parameter tractability: many problems that are non-polynomial become linear when data is of bounded treewidth [Courcelle, 1992].

In practice, however, data is not as well-behaved as we would like. As we saw before [Maniu et al., 2019], only some graphs have (relatively) low treewith – among them, transport graphs. But, for low-treewidth graphs, even polynomial algorithms can be considerably faster. For instance, all-pairs shortest paths algorithms become much faster in low-treewidth graphs [Planken et al., 2012], and using tree decompositions and exploiting on low treewith areas in part of the graphs can significantly increase single-source shortest path computations [Akiba et al., 2012]. Recently, a promising research direction has started to gain traction, called "FPT within P" – which aims for polynomial algorithms that become linear-time and have a polynomial dependency in a graph parameter [Giannopoulou et al., 2017]: some examples of such algorithms are Gaussian elimination [Fomin et al., 2018] and diameter and radius calculation [Abboud et al., 2016]. Treewidth is not the only graph parameter that may be of interest – for denser graphs, *clique-width* upper bounds are also established via decompositions [Courcelle and Olariu, 2000] and hence have potential to be applied here; note that however, for queries on probabilistic data, only treewidth is the measure of interest. This direction is extremely promising in today's massive data environment, where most algorithms that cost more than linear time become prohibitive. Moreover, no such research has started for the generic, semiring-based provenance.

To realize this promise, an important step in the *short term* is to reproduce the same experimental study as in [Maniu et al., 2019] – namely, decompositions for cliquewidth upper-bounds.

As in the case of treewidth, such decompositions would allow for more succinct representations of graphs, and the hope is that they allow processing of other types of queries and on denser graphs. Hence our more *long term* objective is to study the types of queries possible under other graph measures, such as cliquewidth.

Going beyond bandits for influence maximization On the other hand, in influence maximization we have established that good results on influence maximization can be obtained even when the structure is not known. In the *bandit* setting, however, the assumption is that previous actions do not have a consequence on future ones – put otherwise, bandits are without *state*. As we have seen previously, spread in online influence maximization have a tendency to decrease with the number and size of previous activations – the *diminishing returns property*. Adding this information to the decision process leads naturally into the domain of *reinforcement learning* [Szepesvári, 2010].

When the state space includes a potentially infinitely countable number of *contexts*, *influencers*, and *activations* usual tabular approaches do not work. Instead, when the state space is potentially very large the state transitions and rewards are approximated using *function approximation*. When assuming linear functions, linear Markov Decision Processes can use upper confidence bounds to minimize regret [Yang and Wang, 2019]. Designing the state space can still be challenging, but it can be aided by recent advances in *graph representation learning* when applied to combinatorial optimization over graphs – of which influence maximization is a particular case [Dai et al., 2017, Lattimore et al., 2020, Cappart et al., 2021]. Designing efficient and effective reinforcement learning approaches for the contextual influence maximization setting is still a sparse research topic with exciting potential.

In this context, we are currently working of extending the regret analysis of contextual online influence maximization, presented in [Iacob et al., 2022], to upper-confidence bounds for function approximation in linear Markov Decision Processes, especially as it pertains to algorithms based on Least Squares Value Iteration [Jin et al., 2020] and posterior sampling approaches [Osband et al., 2016]. In the more *long term*, we aim to investigate how graph representations such as vectorial representations (embeddings) can be merged with regret minimizing algorithms.

The research presented in this manuscript – along with the perspectives described above – can be thought of as being a particular case of *online optimization problems over graphs* with uncertain or incomplete information. Some examples of this kind of optimization where online feedback can be integrated include link recommendations, minimization of undesired information flow, or network vaccination. The two threads – succinct representations of graphs and sequential optimization – can be thought as complementary. For example, indexes on graph data may allow smaller models for sequential learning processes to converge quicker, while online feedback loops – sometimes with human-in-the-mix approaches – can help with learning heuristics for indexing graphs.

Self References

[Bahri et al., 2020] Bahri, M., Bifet, A., Maniu, S., and Gomes, H. M. (2020). Survey on feature transformation techniques for data streams. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4796–4802. Survey track.

[Iacob et al., 2022] Iacob, A., Cautis, B., and Maniu, S. (2022). Contextual bandits for advertising campaigns: A diffusion-model independent approach. In *SIAM International Conference on Data Mining (SDM)*.

[Lagrée et al., 2017] Lagrée, P., Cappé, O., Cautis, B., and Maniu, S. (2017). Effective large-scale online influence maximization. In *IEEE International Conference on Data Mining (ICDM)*, pages 937–942.

[Lagrée et al., 2019] Lagrée, P., Cappé, O., Cautis, B., and Maniu, S. (2019). Algorithms for online influencer marketing. *ACM Trans. Knowl. Discov. Data*, 13(1):3:1–3:30.

[Lei et al., 2015] Lei, S., Maniu, S., Mo, L., Cheng, R., and Senellart, P. (2015). Online influence maximization. In *21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 645–654.

[Li et al., 2018] Li, X., Cheng, R., Fang, Y., Hu, J., and Maniu, S. (2018). Scalable evaluation of k-nn queries on large uncertain graphs. In *21st International Conference on Extending Database Technology (EDBT)*, pages 181–192.

[Maniu et al., 2017] Maniu, S., Cheng, R., and Senellart, P. (2017). An indexing framework for queries on probabilistic graphs. *ACM Trans. Database Syst.*, 42(2):13:1–13:34.

[Maniu et al., 2020] Maniu, S., Ioannidis, S., and Cautis, B. (2020). Bandits under the influence. In *20th IEEE International Conference on Data Mining (ICDM)*, pages 1172–1177.

[Maniu et al., 2019] Maniu, S., Senellart, P., and Jog, S. (2019). An experimental study of the treewidth of real-world graph data. In *22nd International Conference on Database Theory (ICDT)*, volume 127 of *LIPIcs*, pages 12:1–12:18.

[Ramusat et al., 2018] Ramusat, Y., Maniu, S., and Senellart, P. (2018). Semiring provenance over graph databases. In *10th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*.

[Ramusat et al., 2021] Ramusat, Y., Maniu, S., and Senellart, P. (2021). Provenance-based algorithms for rich queries over graph databases. In *24th International Conference on Extending Database Technology (EDBT)*, pages 73–84.

Bibliography

[Añez et al., 1996] Añez, J., De La Barra, T., and Pérez, B. (1996). Dual graph representation of transport networks. *Transportation Research Part B: Methodological*, 30(3).

[Abboud et al., 2016] Abboud, A., Williams, V. V., and Wang, J. R. (2016). Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Krauthgamer, R., editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391.

[Agrawal and Goyal, 2014] Agrawal, S. and Goyal, N. (2014). Thompson sampling for contextual bandits with linear payoffs. In *JMLR*.

[Akiba et al., 2012] Akiba, T., Sommer, C., and Kawarabayashi, K.-i. (2012). Shortest-path queries for complex networks: Exploiting low tree-width outside the core. In *EDBT*.

[Arnborg et al., 1987] Arnborg, S., Corneil, D. G., and Proskuworski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8(2).

[Arnborg and Proskurowski, 1989] Arnborg, S. and Proskurowski, A. (1989). Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1).

[Arora et al., 2017] Arora, A., Galhotra, S., and Ranu, S. (2017). Debunking the myths of influence maximization: An in-depth benchmarking study. In *SIGMOD*, SIGMOD '17. ACM.

[Ash and Doléans, 1999] Ash, R. B. and Doléans, C. A. (1999). *Probability & Measure Theory*. Academic Press, 2nd edition.

[Asthana et al., 2004] Asthana, S., King, O. D., Gibbons, F. D., and Roth, F. P. (2004). Predicting protein complex membership using probabilistic network reliability. *Genome Research*, 14(6).

[Ball, 1986] Ball, M. O. (1986). Computational complexity of network reliability analysis: An overview. *IEEE Trans. Reliability*, 35(3).

[Barabási and Pósfai, 2016] Barabási, A.-L. and Pósfai, M. (2016). *Network science*. Cambridge University Press.

[Barceló, 2013] Barceló, P. (2013). Querying graph databases. In *PODS*, page 175–188.

[Barceló et al., 2014] Barceló, P., Libkin, L., and Reutter, J. L. (2014). Querying regular graph patterns. *J. ACM*, 61(1).

[Berry et al., 2003] Berry, A., Heggernes, P., and Simonet, G. (2003). The minimum degree heuristic and the minimal triangulation process. *Graph-Theoretic Concepts in Computer Science*, 2880(Chapter 6).

[Bodlaender, 1996] Bodlaender, H. L. (1996). A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6).

[Bodlaender and Koster, 2010] Bodlaender, H. L. and Koster, A. M. C. A. (2010). Treewidth computations I. Upper bounds. *Information and Computation*, 208(3).

[Bodlaender and Koster, 2011] Bodlaender, H. L. and Koster, A. M. C. A. (2011). Treewidth computations II. Lower bounds. *Information and Computation*, 209(7).

[Bonchi et al., 2014] Bonchi, F., Gullo, F., Kaltenbrunner, A., and Volkovich, Y. (2014). Core decomposition of uncertain graphs. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, page 1316–1325.

[Bubeck et al., 2013] Bubeck, S., Ernst, D., and Garivier, A. (2013). Optimal discovery with probabilistic expert advice: finite time analysis and macroscopic optimality. *Journal of Machine Learning Research*, 14(1):601–623.

[Cappart et al., 2021] Cappart, Q., Chételat, D., Khalil, E. B., Lodi, A., Morris, C., and Veličković, P. (2021). Combinatorial Optimization and Reasoning with Graph Neural Networks. *IJCAI Survey*, pages 4348–4355.

[Chen et al., 2009] Chen, W., Wang, Y., and Yang, S. (2009). Efficient influence maximization in social networks. In *KDD*.

[Chen et al., 2016] Chen, W., Wang, Y., Yuan, Y., and Wang, Q. (2016). Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *Journal of Machine Learning Research (JMLR)*, 17(1):1746–1778.

[Chu et al., 2011] Chu, W., Li, L., Reyzin, L., and Schapire, R. (2011). Contextual bandits with linear payoff functions. In *AISTATS*.

[Cohen et al., 2003] Cohen, E., Halperin, E., Kaplan, H., and Zwick, U. (2003). Reachability and distance queries via 2-hop labels. *SIAM J. Comput.*, 32(5).

[Courcelle, 1992] Courcelle, B. (1992). The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *ITA*, 26:257–286.

[Courcelle and Olariu, 2000] Courcelle, B. and Olariu, S. (2000). Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1):77–114.

[Dai et al., 2017] Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *NIPS*, pages 6351–6361.

[Dalvi and Suciu, 2007] Dalvi, N. N. and Suciu, D. (2007). Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4).

[Dani et al., 2008] Dani, V., Hayes, T. P., and Kakade, S. M. (2008). Stochastic linear optimization under bandit feedback. In *COLT*.

[Di Battista and Tamassia, 1990] Di Battista, G. and Tamassia, R. (1990). On-line graph algorithms with spqr-trees. In *ICALP*.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

[Domingos and Richardson, 2001] Domingos, P. and Richardson, M. (2001). Mining the network value of customers. In *KDD*.

[Du et al., 2013] Du, N., Song, L., Woo, H., and Zha, H. (2013). Uncover topic-sensitive information diffusion networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS, Scottsdale, AZ, USA, April 29 - May 1*, pages 229–237.

[Easley and Kleinberg, 2010] Easley, D. A. and Kleinberg, J. M. (2010). *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press.

[Fishman, 1986] Fishman, G. S. (1986). A comparison of four Monte Carlo methods for estimating the probability of s-t connectedness. *IEEE Trans. Reliability*, 35(2).

[Fomin et al., 2018] Fomin, F. V., Lokshtanov, D., Saurabh, S., Pilipczuk, M., and Wrochna, M. (2018). Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Trans. Algorithms*, 14(3).

[Ghosh et al., 2007] Ghosh, J., Ngo, H. Q., Yoon, S., and Qiao, C. (2007). On a routing problem within probabilistic graphs and its application to intermittently connected networks. In *INFOCOM*.

[Giannopoulou et al., 2017] Giannopoulou, A. C., Mertzios, G. B., and Niedermeier, R. (2017). Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 689:67 – 95.

[Gomez-Rodriguez et al., 2011] Gomez-Rodriguez, M., Balduzzi, D., and Schölkopf, B. (2011). Uncovering the temporal dynamics of diffusion networks. In *Proceedings of the 28th International Conference on Machine Learning, ICML, Bellevue, Washington, USA, June 28 - July 2*, pages 561–568.

[Gomez-Rodriguez et al., 2012] Gomez-Rodriguez, M., Leskovec, J., and Krause, A. (2012). Inferring networks of diffusion and influence. *ACM Trans. Knowl. Discov. Data*, 5(4):21:1–21:37.

[Gomez-Rodriguez et al., 2013] Gomez-Rodriguez, M., Leskovec, J., and Schölkopf, B. (2013). Structure and dynamics of information pathways in online media. In *Sixth ACM International Conference on Web Search and Data Mining, WSDM, Rome, Italy, February 4-8*, pages 23–32.

[Good, 1953] Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3-4):237.

[Goyal et al., 2011] Goyal, A., Bonchi, F., and Lakshmanan, L. V. (2011). A data-based approach to social influence maximization. *PVLDB*, 5(1).

[Goyal et al., 2010] Goyal, A., Bonchi, F., and Lakshmanan, L. V. S. (2010). Learning influence probabilities in social networks. In *Proceedings of the Third International Conference on Web Search and Web Data Mining (WSDM)*, pages 241–250.

[Green et al., 2007] Green, T. J., Karvounarakis, G., and Tannen, V. (2007). Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, page 31–40.

[Gutwenger and Mutzel, 2000] Gutwenger, C. and Mutzel, P. (2000). A linear time implementation of spqr-trees. In *Graph Drawing*.

[Harvey, 2014] Harvey, D. J. (2014). *On Treewidth and Graph Minors*. PhD thesis, The University of Melbourne.

[Hopcroft and Tarjan, 1973a] Hopcroft, J. E. and Tarjan, R. E. (1973a). Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3).

[Hopcroft and Tarjan, 1973b] Hopcroft, J. E. and Tarjan, R. E. (1973b). Efficient algorithms for graph manipulation [h] (algorithm 447). *Commun. ACM*, 16(6).

[Hua and Pei, 2010] Hua, M. and Pei, J. (2010). Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*.

[Huang et al., 2016] Huang, X., Lu, W., and Lakshmanan, L. V. (2016). Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD)*, page 77–90.

[Jin et al., 2020] Jin, C., Yang, Z., Wang, Z., and Jordan, M. I. (2020). Provably efficient reinforcement learning with linear function approximation. In Abernethy, J. and Agarwal, S., editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 2137–2143.

[Jin et al., 2011] Jin, R., Liu, L., Ding, B., and Wang, H. (2011). Distance-constraint reachability computation in uncertain graphs. *PVLDB*, 4(9).

[Kempe et al., 2003] Kempe, D., Kleinberg, J., and Tardos, E. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 137–146.

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

[Kuramochi and Karypis, 2001] Kuramochi, M. and Karypis, G. (2001). Frequent subgraph discovery. In *ICDM*.

[Lattimore and Szepesvári, 2020] Lattimore, T. and Szepesvári, C. (2020). *Bandit Algorithms*. Cambridge University Press.

[Lattimore et al., 2020] Lattimore, T., Szepesvári, C., and Weisz, G. (2020). Learning with Good Feature Representations in Bandits and in RL with a Generative Model. *ICML*.

[Lei et al., 2015] Lei, S., Maniu, S., Mo, L., Cheng, R., and Senellart, P. (2015). Online influence maximization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15.

[Leskovec et al., 2007] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N. (2007). Cost-effective outbreak detection in networks. In *KDD*.

[Li et al., 2017] Li, L., Lu, Y., and Zhou, D. (2017). Provably optimal algorithms for generalized linear contextual bandits. In *ICML*.

[Liben-Nowell and Kleinberg, 2007] Liben-Nowell, D. and Kleinberg, J. M. (2007). The link-prediction problem for social networks. *JASIST*, 58(7).

[Lu et al., 2014] Lu, W., Ioannidis, S., Bhagat, S., and Lakshmanan, L. V. (2014). Optimal recommendations under attraction, aversion, and social influence. In *KDD*, pages 811–820.

[Markowitz, 1957] Markowitz, H. M. (1957). The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3).

[McAllester and Schapire, 2000] McAllester, D. A. and Schapire, R. E. (2000). On the convergence rate of good-turing estimators. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (COLT 2000), June 28 - July 1, 2000, Palo Alto, California*, pages 1–6.

[Mohri, 2002] Mohri, M. (2002). Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7(3):321–350.

[Newman et al., 2001] Newman, M. E. J., Strogatz, S. H., and Watts, D. J. (2001). Random graphs with arbitrary degree distributions and their applications. *Phys. Rev. E*, 64.

[Newman et al., 2002] Newman, M. E. J., Watts, D. J., and Strogatz, S. H. (2002). Random graph models of social networks. In *Proceedings of the National Academy of Sciences*.

[Osband et al., 2016] Osband, I., Van Roy, B., and Wen, Z. (2016). Generalization and exploration via randomized value functions. In *Proceedings of the International Conferenceon Machine Learning (ICML)*, page 2377–2386.

[Planken et al., 2012] Planken, L., de Weerdt, M., and van der Krogt, R. (2012). Computing all-pairs shortest paths by leveraging low treewidth. *Journal of Artificial Intelligence Research*, 43.

[Potamias et al., 2010] Potamias, M., Bonchi, F., Gionis, A., and Kollios, G. (2010). k-nearest neighbors in uncertain graphs. *PVLDB*, 3(1).

[Robertson and Seymour, 1984] Robertson, N. and Seymour, P. D. (1984). Graph minors. III. Planar tree-width. *J. Comb. Theory, Ser. B*, 36(1).

[Safari, 2005] Safari, M. A. (2005). D-width: A more natural measure for directed tree width. In *MFCS*.

[Saito et al., 2008] Saito, K., Nakano, R., and Kimura, M. (2008). Prediction of information diffusion probabilities for independent cascade model. In *Knowledge-Based Intelligent Information and Engineering Systems, 12th International Conference, KES, Zagreb, Croatia, September 3-5, Proceedings, Part III*, pages 67–75.

[Szepesvári, 2010] Szepesvári, C. (2010). *Algorithms for Reinforcement Learning*. Morgan and Claypool Publishers.

[Tang et al., 2014] Tang, Y., Xiao, X., and Shi, Y. (2014). Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD*.

[Tutte, 1966] Tutte, W. T. (1966). *Connectivity in graphs*, volume 15 of *Mathematical Expositions*. University of Toronto Press.

[Valiant, 1979] Valiant, L. G. (1979). The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3).

[van Dijk et al., 2006] van Dijk, T., van den Heuvel, J.-P., and Slob, W. (2006). Computing treewidth with LibTW. Technical report, University of Utrecht.

[Vaswani et al., 2017] Vaswani, S., Kveton, B., Wen, Z., Ghavamzadeh, M., Lakshmanan, L., and Schmidt, M. (2017). Diffusion independent semi-bandit influence maximization. In *ICML*.

[Vaswani et al., 2015] Vaswani, S., Lakshmanan, V., and Schmidt, M. (2015). Influence maximization with bandits. In *Workshop NIPS*, NIPS '15.

[Watts and Dodds, 2007] Watts, D. and Dodds, P. (2007). Influentials, networks, and public opinion formation. *Journal of Consumer Research*, 34(4):441–458.

[Watts, 2003] Watts, D. J. (2003). *Six Degrees: The Science of a Connected Age*. W. W. Norton, New York.

[Wei, 2010] Wei, F. (2010). TEDI: efficient shortest path query answering on graphs. In *SIGMOD*.

[Wen et al., 2016] Wen, Z., Kveton, B., and Valko, M. (2016). Influence maximization with semi-bandit feedback. In *Working paper*.

[Wen et al., 2017] Wen, Z., Kveton, B., Valko, M., and Vaswani, S. (2017). Online influence maximization under independent cascade model with semi-bandit feedback. In *NIPS*.

[Yang and Wang, 2019] Yang, L. and Wang, M. (2019). Sample-Optimal Parametric Q-Learning Using Linearly Additive Features. *ICML*.

[Zou et al., 2010] Zou, Z., Gao, H., and Li, J. (2010). Discovering frequent subgraphs over uncertain graph databases under probabilistic semantics. In *KDD*.

Titre: Graphes et incertitude
Mots clés: fouille données graphe, données incertaines, bandit manchots

Résumé: Dans de nombreux domaines, les graphes sont l'un des moyens les plus intuitifs de représenter les données, et de nombreuses tâches importantes peuvent être traduites en requêtes de graphes et en problèmes combinatoires sur les graphes. Dans la plupart des scénarios du monde réel, les données ou les modèles de graphes comportent une certaine incertitude.

Dans ce manuscrit, nous abordons deux aspects des défis qui se présentent lorsque les données ou les modèles de graphes sont incertains. Tout d'abord, nous discutons de la façon dont les requêtes sur le graphe changent lorsque les bords deviennent incertains ; pour cela, nous introduisons le concept de décompositions d'arbres pour le traitement des ces requêtes. De plus, nous faisons le lien entre l'efficacité des requêtes et le concept de largeur d'arbre. Dans la deuxième partie du manuscrit, nous discutons de la manière de résoudre un problème de graphe bien connu - la maximisation de l'influence sociale - dans le cas où l'on sait peu de choses sur le graphe sous-jacent sur lequel l'influence est exercée. Nous discutons de la manière dont des approches telles que les bandits manchots peuvent être appliquées.

Title: Graphs and Uncertainty
Keywords: graph data mining, probabilistic data, multi-armed bandits

Abstract: In many domains, graphs are one of the most intuitive ways of representing data, and many important tasks can be translated into graph queries and combinatorial problems on graphs. In most real-world scenarios, graph data or models have some uncertainty attached.

In this manuscript, we discuss two aspects of challenges that occur when graph data or models are uncertain. First, we discuss how querying the graph changes when edges become uncertain; for this, we introduce the concept of tree decompositions for query processing on uncertain graphs. Moreover, we make the link between query efficiency and the concept of treewidth. In the second part of the manuscript, we discuss how to solve a well-known graph problem – social influence maximization – in the case where little is known about the underlying graph over which influence is performed. We discuss how approaches such as multi-armed-bandits can be applied.