Core Decomposition of Uncertain Graphs

Francesco Bonchi[1] Francesco Gullo[1] Andreas Kaltenbrunner[2] Yana Volkovich[2]

[1]Yahoo Labs, Spain
{bonchi,gullo}@yahoo-inc.com

[2]Barcelona Media - Innovation Centre, Spain
{andreas.kaltenbrunner,yana.volkovich}@barcelonamedia.org

ABSTRACT

Core decomposition has proven to be a useful primitive for a wide range of graph analyses. One of its most appealing features is that, unlike other notions of dense subgraphs, it can be computed linearly in the size of the input graph.

In this paper we provide an analogous tool for uncertain graphs, i.e., graphs whose edges are assigned a probability of existence. The fact that core decomposition can be computed efficiently in deterministic graphs does not guarantee efficiency in uncertain graphs, where even the simplest graph operations may become computationally intensive. Here we show that core decomposition of uncertain graphs can be carried out efficiently as well.

We extensively evaluate our definitions and methods on a number of real-world datasets and applications, such as *influence maximization* and *task-driven team formation*.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: [Database Applications-Data Mining]; G.2.2 [**Discrete Mathematics**]: [Graph Theory-Graph Algorithms]

Keywords

uncertain graphs; dense subgraph; core decomposition

1. INTRODUCTION

Uncertain graphs, i.e., graphs whose edges are assigned a probability of existence (see an example in Figure 1), arise in several emerging applications [24, 14, 15]. For instance, in *biological networks* and *protein-interaction networks* vertices represent genes and/or proteins, while edges represent interactions among them. Since the interactions are derived through noisy and error-prone laboratory experiments, the existence of each edge is uncertain [4, 26, 24]. In *social networks* uncertainty arises for various reasons [1]. Edge probabilities may represent the outcome of a *link-prediction* task [20] or the influence of one person on another, like in *viral marketing* [11]. Uncertainty can also be intentionally injected for privacy purposes [7].

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623655.

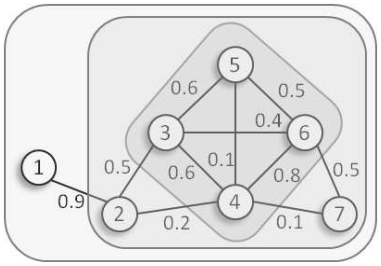

Figure 1: An uncertain graph and its (k, η)-*core decomposition* for $\eta = 0.04$. Vertex 1 has core number 1, vertices 2 and 7 have core number 2, and vertices 3, 4, 5 and 6 have core number 3.

Finding dense subgraphs is a fundamental primitive in many graph-analysis tasks [21]. There exist many different definitions of what a dense subgraph is, e.g., *cliques, n-cliques, n-clans, k-plexes, f-groups, n-clubs, lambda sets*, most of which are **NP**-hard to compute or at least quadratic in the size of the input graph. In this respect, the notion of *core decomposition* is particularly appealing as (i) it can be computed in linear time [5], and (ii) it is related to many other definitions of a dense subgraph (as discussed later).

The *k-core* of a graph is defined as a maximal subgraph in which every vertex is connected to at least k other vertices within that subgraph. The set of all k-cores of a graph G forms the *core decomposition* of G [25]. The fact that core decomposition can be performed in linear time in deterministic graphs does not guarantee efficiency in uncertain graphs. Indeed, in such graphs even the simplest tasks may become hard. As an example, consider the *two-terminal-reachability* problem, which asks whether two query vertices are connected. In a deterministic graph the solution to this problem requires a simple scan of the graph. Instead, in uncertain graphs, computing the probability that two vertices are connected is a #**P**-complete problem [28].

Thus, a major question we aim at answering in this paper is: *can the core decomposition of an uncertain graph be computed efficiently?*

Related work and applications. Existing research on uncertain graphs has mainly focused on querying [15, 33, 24, 31] and mining, particularly on extracting frequent subgraphs [34] or subgraphs that are connected with high probability [14], and clustering [22, 18].

Core decomposition of deterministic graphs has been exploited to analyse the nature of a network and discover dense

substructures [2, 17]. It has been applied in many different domains, such as bioinformatics [30], software engineering [32], and social networks [17]. Core decomposition has been also used to speed-up the computation of more complex definitions of a dense subgraph. For instance, it serves to find maximal cliques more efficiently [10], and it is at the basis of linear-time approximation algorithms for the *densest-subgraph* problem [19] and the *densest at-least-k-subgraph* problem [3]. It is also used to approximate *betweenness centrality* [13]. A core-decomposition tool for uncertain graphs would thus provide a natural extension of all these applications to the context of uncertain graphs. Other direct applications of core decomposition of uncertain graphs include *influence maximization* and *task-driven team formation*, which we showcase in Section 6 and 7, respectively.

In *influence maximization* [16], the probability of an edge (u, v) represents the influence that u exerts on v, i.e., the likelihood that some action/information propagates from u to v. The greedy algorithm [12] traditionally used to find the users that maximize the information spread over the network requires a number of Monte Carlo simulations that largely limit its efficiency. In Section 6 we show how our probabilistic core-decomposition tool can be used to speed-up the influence-maximization process.

In *task-driven team formation*, the input is a collaboration graph $G = (V, E, \tau)$, where vertices are individuals and edges exhibit a probabilistic topic model τ representing the topic(s) of past collaborations. A query is a pair $\langle T, Q\rangle$, where T is a set of terms describing a new task, and Q is a set of vertices. The goal is to find an answer set of vertices A, such that $A \supseteq Q$ is a good team for the task described by T. The given query task T, along with the topic model τ, induces a (single) probability value p^T for each edge $(u, v) \in E$, such that $p^T(u, v)$ represents the likelihood that u and v collaborate on T. This gives rise to an uncertain graph to which one can naturally apply core decomposition in order to find the desired team (Section 7).

Challenges and contributions. In this paper we study the problem of core decomposition of uncertain graphs, which, to the best of our knowledge, has never been considered so far. We introduce (Section 2) the notion of (k, η)-*core* as a maximal subgraph whose vertices have at least k neigbours in that subgraph with probability no less than η; here $\eta \in [0, 1]$ is a threshold defining the desired level of certainty of the output cores.

Let the η-degree of a vertex v be the maximum degree such that the probability for v to have that degree is no less than η. We design an algorithm for finding a (k, η)-core decomposition that iteratively removes the vertex having the smallest η-degree and prove its correctness (Section 3). The proposed algorithm resembles the traditional algorithm for computing the core decomposition of a deterministic graph [5]; however, as usual when the attention is shifted from the deterministic context to uncertain graphs, the adaptation of that algorithm is non-trivial. A major challenge is the capability of handling large graphs.

Two main critical steps affect our algorithm: computing initial η-degrees and updating η-degrees whenever a vertex is removed from the graph. While the corresponding steps in the deterministic case (i.e., computing and updating the degree of a vertex) are straightforward, performing them efficiently in uncertain graphs needs a great deal of attention; approaching them naïvely, indeed, may even lead to intractable (exponential) time complexity. We show how to overcome the exponential-time complexity by devising a novel yet efficient dynamic-programming method to compute η-degrees from scratch. We also exploit the same intuition underlying the dynamic-programming algorithm so as to efficiently update η-degrees after a vertex removal. As a result, we show that computing a (k, η)-core decomposition takes $\mathcal{O}(m\Delta)$ time, where m is the number of edges in the input uncertain graph and Δ is the maximum η-degree.

As a further contribution, we devise a novel method to improve the efficiency of the proposed (k, η)-core-decomposition algorithm (Section 4). The idea is to exploit a fast-to-compute lower bound on the η-degree that can be used as a placeholder during the first iterations while being replaced with the actual η-degree only when the vertex at hand is selected and the graph has become smaller.

Finally, we report experiments on efficiency and numerical stability on real-world graphs (Section 5) and show our proposal at work in two real-life applications (Sections 6–7).

2. PROBLEM DEFINITION

Cores of deterministic graphs. Before focusing on uncertain graphs, we briefly recall the problem of computing cores of deterministic graphs. Let $G = (V, E)$ be an undirected graph, where V is a set of n vertices and $E \subseteq V \times V$ is a set of m edges. For every vertex $v \in V$, let $deg(v)$ and $deg_H(v)$ denote the degree of v in G and in a subgraph H of G, respectively. Also, given a set of vertices $C \subseteq V$, let $E|C$ denote the subset of edges induced by C, i.e., $E|C = \{(u, v) \in E \mid u \in C, v \in C\}$.

DEFINITION 1 (k-CORE). *The* k-core *(or* core of order k*) of G is a* maximal *subgraph $H = (C, E|C)$ such that $\forall v \in C : deg_H(v) \geq k$. The* core number *(or* core index*) of a vertex v, denoted $c(v)$, is the highest order of a core that contains v. The set of all k-cores of G, for all k, is the* core decomposition *of G.* □

The notion of k-core is strictly related to the notion of k-*shell*, that is the subgraph induced by the set of all vertices having core number equal to k. Note that neither k-cores nor k-shells are necessarily connected subgraphs. Also, while these two notions usually refer to subgraphs of the input graph, in the remainder we slightly abuse of notation and denote by k-core (or k-shell) both the subgraph $H = (C, E|C)$ itself and the vertex set C that induces H.

All k-shells of a graph G form a partition of the vertex set V, while all k-cores are nested into each other: $G = C_0 \supseteq C_1 \supseteq \cdots \supseteq C_{k^*}$ ($k^* = \max_{v \in V} c(v)$). As a result, the core decomposition of G is *unique* and fully determined by the core number $c(v)$ of all vertices v in G: the k-core of G simply corresponds to (the subgraph induced by) the set of all vertices v having core number $c(v) \geq k$.

Batagelj and Zaveršnik [5] show how to compute the core decomposition of a graph G in linear time (Algorithm 1). The algorithm iteratively removes the smallest-degree vertex and sets the core number of the removed vertex accordingly. Vertices are thus required to be ordered based on their degree. Defining the initial vertex ordering and keeping vertices ordered during the execution of the algorithm take $\mathcal{O}(n)$ and $\mathcal{O}(1)$ time, respectively. The idea is to employ an n-dimensional vector $\mathbf{D}$ whose single cells $\mathbf{D}[i]$ store all vertices having degree equal to i in the current graph. The overall time complexity of the algorithm is hence $\mathcal{O}(n + m)$.

Algorithm 1 k-CORES

Input: A graph $G = (V, E)$.
Output: An n-dimensional vector $\mathbf{c}$ containing the core number of each $v \in V$.

```
1: c ← ∅,  d ← ∅,  D ← [∅, ..., ∅]
2: for all v ∈ V do
3:    d[v] ← deg(v)
4:    D[deg(v)] ← D[deg(v)] ∪ {v}
5: end for
6: for all k = 0, 1, ..., n do
7:    while D[k] ≠ ∅ do
8:       pick and remove a vertex v from D[k]
9:       c[v] ← k
10:      for all u : (u, v) ∈ E, d[u] > k do
11:         move u from D[d[u]] to D[d[u] − 1]
12:         d[u] ← d[u] − 1
13:      end for
14:      remove v from G
15:   end while
16: end for
```

Cores of uncertain graphs. Let $\mathcal{G} = (V, E, p)$ be an uncertain graph, where $p : E \rightarrow (0, 1]$ is a function that assigns a probability of existence to each edge.[1] For the sake of brevity, we hereinafter denote the probabilities $p(e)$ with p_e. For every vertex $v \in V$, let $N_v = \{(u, v) \in E\}$ denote the set of edges incident to v, and $d_v = |N_v|$ its size.

To define our notion of core decomposition of an uncertain graph, we resort to the well-known *possible-world semantics*, which has been recognized as a sound principle to define queries on probabilistic data [9]. Broadly, such a principle interprets the probabilistic data as a set of deterministic instantiations, called *possible worlds*, each of which associated with its probability of being observed. In the context of uncertain graphs, the bulk of the literature assumes the probabilities of existence of the edges independent from one another [24, 14, 15]. Under this assumption, the possible-world semantics interprets an uncertain graph $\mathcal{G}$ with m edges as a set of 2^m possible deterministic graphs (worlds), each of which containing a subset of the edges in E. More precisely, an uncertain graph $\mathcal{G} = (V, E, p)$ yields a set of possible graphs $\{G = (V, E_G)\}_{E_G \subseteq E}$, and the probability of observing a possible graph $G = (V, E_G) \sqsubseteq \mathcal{G}$ is:

$$\Pr(G) = \prod_{e \in E_G} p_e \prod_{e \in E \setminus E_G} (1 - p_e). \quad (1)$$

According to the possible-world semantics, answering a probabilistic query q means to derive a probability distribution over all possible deterministic answers a to the query q, where the probability of an answer a corresponds to the sum of the probabilities of all worlds where a is the answer to q. As this answer distribution is usually too large and sparse to be explicitly interpreted or computed/stored, the general turnaround adopted is to assign a score to each domain object based on its probability of being part of an answer to the probabilistic query q, and return the objects having highest scores as a final answer to q [9].

We cast such a general framework to our context by defining the score of each vertex v to be part of a k-core $\mathcal{H}$ as the probability that v has degree no less than k in $\mathcal{H}$, i.e., $\Pr[deg_{\mathcal{H}}(v) \geq k]$. Then, we employ a classic threshold-based approach to decide which vertices should actually form a k-core based on their scores. As a result, the notion of *probabilistic (k, η)-core* we come up with is the following:

DEFINITION 2 (PROBABILISTIC (k,η)-CORES). *Given an uncertain graph $\mathcal{G} = (V, E, p)$, and a threshold $\eta \in [0, 1]$, the* probabilistic (k, η)-core *of $\mathcal{G}$ is a maximal subgraph $\mathcal{H} = (C, E|C, p)$ such that the probability that each vertex $v \in C$ has degree no less than k in $\mathcal{H}$ is greater than or equal to η, i.e., $\forall v \in C : \Pr[deg_{\mathcal{H}}(v) \geq k] \geq \eta$.* □

The notion of *η-core number* immediately follows from the definition of (k,η)-core and is defined as the highest order k of a (k,η)-core containing v.

The problem we address in this work is the following.

PROBLEM 1 (PROBCORES). *Given an uncertain graph $\mathcal{G}$ and a probability threshold $\eta \in [0, 1]$, find the* (k, η)-core decomposition *of $\mathcal{G}$, that is the set of all (k,η)-cores of $\mathcal{G}$.* □

Our definition of core decomposition of an uncertain graph, has the desirable feature of being unique, as formally shown in the next theorem.

THEOREM 1. *Given an uncertain graph $\mathcal{G}$ and a probability threshold η, the (k, η)-core decomposition of $\mathcal{G}$ is unique.*

PROOF. We prove the theorem by showing that $\mathcal{G}$ cannot have more than one (k, η)-core, for all k. Assume that $\mathcal{G}$ has two (k, η)-cores and denote them by $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. According to Definition 2, it holds that $\mathcal{H}_1$ is a maximal subgraph of $\mathcal{G}$ such that $\forall v \in \mathcal{H}_1 : \Pr[deg_{\mathcal{H}_1}(v) \geq k] \geq \eta$, and the same happens for $\mathcal{H}_2$. Combining the (k, η)-core conditions of $\mathcal{H}_1$ and $\mathcal{H}_2$ leads to the subgraph $\mathcal{H}_1 \cup \mathcal{H}_2$ to satisfy the (k, η)-core condition too, as $\forall v \in \mathcal{H}_1 : \Pr[deg_{\mathcal{H}_1}(v) \geq k] \geq \eta \wedge \forall v \in \mathcal{H}_2 : \Pr[deg_{\mathcal{H}_2}(v) \geq k] \geq \eta$ clearly implies that $\forall v \in \mathcal{H}_1 \cup \mathcal{H}_2 : \Pr[deg_{\mathcal{H}_1 \cup \mathcal{H}_2}(v) \geq k] \geq \eta$. This means that neither $\mathcal{H}_1$ nor $\mathcal{H}_2$ are maximal, thus contradicting the hypothesis. The theorem follows. □

An example of *(k, η)-core decomposition* of an uncertain graph is provided in Figure 1.

3. COMPUTING PROBABILISTIC CORES

For a vertex v of the input uncertain graph $\mathcal{G}$, the probability $\Pr[deg(v) \geq k]$ can be expressed as:

$$\Pr[deg(v) \geq k] = \textstyle\sum_{G \sqsubseteq \mathcal{G}_v^{\geq k}} \Pr(G), \quad (2)$$

where $\mathcal{G}_v^{\geq k}$ is the set of all possible graphs drawn from $\mathcal{G}$ where v has degree $\geq k$, i.e., $\mathcal{G}_v^{\geq k} = \{G \sqsubseteq \mathcal{G} \mid deg_G(v) \geq k\}$.

It is easy to see that such a probability value is monotonically non-increasing with k, i.e., $\Pr[deg(v) \geq 0] \geq \Pr[deg(v) \geq 1] \geq \ldots \geq \Pr[deg(v) \geq d_v]$. Then, given a threshold η, for every vertex v in the graph, there exists a value $\hat{k} \in [0..d_v]$ such that $\Pr[deg(v) \geq h] \geq \eta$, for all $h \leq \hat{k}$, and $\Pr[deg(v) \geq h] < \eta$, for all $h > \hat{k}$. We call this value the *η-degree* of vertex v.

DEFINITION 3 (η-DEGREE). *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and a threshold $\eta \in [0, 1]$, the* η-degree *η-deg(v) of a vertex $v \in V$ is defined as*

$$\eta\text{-}deg(v) = \max\{k \in [0..d_v] \mid \Pr[deg(v) \geq k] \geq \eta\}.$$

Let also η-deg$_{\mathcal{H}}(v)$ be the η-degree of v in a subgraph $\mathcal{H}$. □

[1] We consider undirected graphs for the sake of presentation and consistency with the literature on core decomposition. However, all our definitions/methods apply to directed graphs too, by simply replacing the notion of degree with either in-degree or out-degree. Indeed, in Section 6, where we focus on *influence maximization*, the graph is directed and we define probabilistic cores based on out-degree.

Algorithm 2 (k,η)-CORES

Input: An uncertain graph $\mathcal{G} = (V, E, p)$, a threshold $\eta \in [0, 1]$.
Output: An n-dimensional vector $\mathbf{c}$ containing the η-core number of each $v \in V$.

```
1: compute η-deg(v) for all v ∈ V
2: c ← ∅,  d ← ∅,  D ← [∅, ..., ∅]
3: for all v ∈ V do
4:    d[v] ← η-deg(v)
5:    D[η-deg(v)] ← D[η-deg(v)] ∪ {v}
6: end for
7: for all k = 0, 1, ..., n do
8:    while D[k] ≠ ∅ do
9:       pick and remove a vertex v from D[k]
10:      c[v] ← k
11:      for all u : (u, v) ∈ E, d[u] > k do
12:         recompute η-deg(u)
13:         move u from D[d[u]] to D[η-deg(u)]
14:         d[u] ← η-deg(u)
15:      end for
16:      remove v from G
17:   end while
18: end for
```

Intuitively, the notion of η-degree gives an idea of the degree of a vertex given a specific threshold η. We exploit the notion η-degree to adapt the k-CORES algorithm used for deterministic graphs to the context of uncertain graphs. The proposed algorithm, called (k,η)-CORES (Algorithm 2), follows the same scheme as in the deterministic case with the main difference of the use of the η-degree. The soundness of the proposed algorithm is shown in the following theorem.

THEOREM 2. *Given an uncertain graph $\mathcal{G}$ and a threshold η, Algorithm 2 provides the (k,η)-core decomposition of $\mathcal{G}$.*

PROOF. For every $v \in V$ and every subgraph $\mathcal{H} = (C, E_C, p|C)$ of $\mathcal{G}$, it is easy to see that $\eta\text{-}deg_{\mathcal{H}}(v) \leq \eta\text{-}deg(v)$, as the η-degree computation in $\mathcal{G}$ relies on more successful events than those encountered in $\mathcal{H}$. This implies that $\eta\text{-}deg_{\mathcal{H}}(v)$ is a *monotonic vertex property function* [5], where, for every $v \in V$ and $C \subseteq V$, a vertex property function on $\mathcal{G}$ is a function $\phi(v, C) : V \times 2^V \to \mathbb{R}$, and the monotonicity property holds if $\forall\ C_1, C_2 \subseteq V : C_1 \subseteq C_2$ implies that $\forall\ v \in V : \phi(v, C_1) \leq \phi(v, C_2)$. The proof is completed by the result by Batagelj and Zaveršnik [5], who show that, for a monotonic vertex property function $\phi(v, C)$, the algorithm that repeatedly removes a vertex with the smallest ϕ value gives the desired core decomposition.[2] □

Instead of computing/updating standard degrees, in the probabilistic case one thus needs to (i) compute all η-degrees at the beginning of the algorithm (Line 1), and (ii) update the η-degree of a neighbour of the currently being processed vertex v (Line 12). While computing/updating degrees in the deterministic case is straightforward, for the η-degrees such steps are non-trivial, as shown next.

Computing initial η-degrees

To show how to derive η-degrees from scratch, we first focus on the computation of $\Pr[deg(v) \geq k]$ for a vertex v, and note that $\Pr[deg(v) \geq k]$ is equal to the sum of the probabilities $\Pr[deg(v) = i]$ either for all $i \in [k..d_v]$ or, equivalently, for all $i \in [0..k-1]$:

$$\Pr[deg(v) \geq k] = \sum_{i=k}^{d_v} \Pr[deg(v) = i] = 1 - \sum_{i=0}^{k-1} \Pr[deg(v) = i]. \quad (3)$$

Furthermore, we observe that each individual $\Pr[deg(v) = i]$ can in turn be computed considering all subsets of edges $N \subseteq N_v$ of size i and summing over the probabilities that all and only the edges in these various N exist:

$$\Pr[\deg(v) = i] = \sum_{\substack{N \subseteq N_v, \\ |N| = i}} \prod_{e \in N} p_e \prod_{e \in N_v \setminus N} (1 - p_e). \quad (4)$$

The sum in the above formula is over all subsets $N \subseteq N_v$, $|N| = i$; thus, a naïve computation would lead to a time complexity exponential in the size of N_v. We can however manage this by rearranging the formula as

$$\Pr[deg(v) = i] = P_v R(i, N_v),$$

where $P_v = \prod_{e \in N_v}(1 - p_e)$, $R(i, N_v) = \sum_{\substack{N \subseteq N_v, \\ |N| = i}} \prod_{e \in N} \tilde{p}_e$, and $\tilde{p}_e = \frac{p_e}{1 - p_e}$. This rearrangement allows us to exploit the next recursive formula, which has originally been introduced in [8] for sampling from a finite population with unequal probabilities and without replacement:

$$R(i, N_v) = \frac{1}{i} \sum_{j=1}^{i} (-1)^{j+1} T(j, N_v) R(i - j, N_v), \quad (5)$$

where $T(j, N_v) = \sum_{e \in N_v} (\tilde{p}_e)^j$. Now, it is easy to see that Equation (5) allows for computing all individual $\Pr[\deg(v) = i]$ values, for all $i \in [0..k-1]$ (which, according to Equation (3), are needed to derive the desired $\Pr[\deg(v) \geq k]$) in polynomial time, precisely in $\mathcal{O}(kd_v)$ time.

A dynamic-programming method. Although the above way of computing $\Pr[\deg(v) = i]$ solves a seemingly exponential-time problem, it still has weaknesses due to the recursive formula in Equation (5). Firstly, as the formula involves both products and sums of $\tilde{p}_e$ values that can be either very large (when $p_e \to 1$) or very small (when $p_e \to 0$), it may incur numerical-stability issues, which might make the computation of $\Pr[\deg(v) = i]$ problematic when executed by a computer. Secondly, using such a formula, the η-degree of a vertex v when one of its incident edges is removed cannot be recomputed faster than a from-scratch computation.

For the above reasons, we propose here an alternative way of computing $\Pr[\deg(v) = i]$. Consider a vertex v and an edge e incident to v, and let $\mathcal{G}_{\setminus\{e\}}$ denote the subgraph of $\mathcal{G}$ where e is not present. The method is based on the following key observation: the event "v has degree k in $\mathcal{G}$" implies that either "e exists *and* v has degree $k-1$ in $\mathcal{G}_{\setminus\{e\}}$" or "$e$ does not exist *and* v has degree k in $\mathcal{G}_{\setminus\{e\}}$". This way, the probability for v to have degree k in the original graph $\mathcal{G}$ can be computed as a linear combination of the probabilities that v has degree either $k-1$ or k in the subgraph $\mathcal{G}_{\setminus\{e\}}$.

The above reasoning can be generalised to every subgraph of $\mathcal{G}$ and formally expressed in the next theorem (for which we omit a formal proof due to limited space).

THEOREM 3. *Given an uncertain graph $\mathcal{G} = (V, E, p)$ and a vertex $v \in V$, let $N_v = \{e_1, \ldots, e_{d_v}\}$ be the set of all edges incident to v ordered in some way. Also, given a subset*

[2] That work states that the time complexity of such an algorithm is $\mathcal{O}(m \times \max\{D, \log n\})$, where D is the maximum degree. But this is a general result for vertex property functions that can be updated linearly in the degree of a vertex. For any specific vertex property function, such as our η-degree, the complexity can be higher or lower.

$N \subseteq N_v$, let $deg(v|N)$ denote the degree of v in the subgraph $\hat{\mathcal{G}} = (V, E \setminus (N_v \setminus N), p)$. For all $h \in [1..d_v - 1]$ it holds that:

$$\begin{aligned}&\Pr[deg(v|\{e_1,\ldots,e_{h+1}\}) = i] = \\ &= \; p_{e_{h+1}} \Pr[deg(v|\{e_1,\ldots,e_h\}) = i-1] + \\ &\quad +(1-p_{e_{h+1}}) \Pr[deg(v|\{e_1,\ldots,e_h)\} = i]. \quad \square\end{aligned} \tag{6}$$

Theorem 3 provides a principled way to efficiently compute $\Pr[deg(v) = i]$ based on the dynamic-programming paradigm. Particularly, we take an arbitrary ordering of the edges incident to the vertex v being currently under consideration and define a proper recursive formula that allows for computing partial solutions relying only on the first i edges. The ultimate score (i.e., the actual value of $\Pr[deg(v) = i]$) is available only when all the edges have been considered; this makes the overall computation independent from the specific ordering of the edges. Formally, let $X(h, j) = \Pr[deg(v|\{e_1, \ldots, e_h\}) = j]$, for all $h \in [0..d_v]$, $j \in [-1..i]$. We set the following base cases:

$$\begin{cases} X(0,0) &= 1, \\ X(h,-1) &= 0, \quad \text{for all } h \in [0..d_v], \\ X(h,j) &= 0, \quad \text{for all } h \in [0..d_v], j \in [h+1..i], \end{cases}$$

while we exploit Equation (6) to compute the generic dynamic-programming recursive step as

$$X(h,j) = p_{e_h} X(h-1, j-1) + (1-p_{e_h}) X(h-1, j),$$

for all $h \in [1..d_v]$, $j \in [0..h]$. We need to compute all $X(\cdot,\cdot)$ values so as to get to $X(d_v, i)$, which corresponds to the desired probability $\Pr[deg(v) = i]$. This requires $\mathcal{O}(id_v)$ time.

Moreover, one can notice that the values of the *entire set* $\{X(d_v, j)\}_{j=0}^{i}$ (not just $X(d_v, i)$) correspond to the actual probability values $\{\Pr[deg(v) = j]\}_{j=0}^{i}$. Thus, employing the proposed dynamic-programming method and setting $i = k-1$, the probability values $\Pr[deg(v) = 0], \ldots, \Pr[deg(v) = k-1]$, which are required for computing $\Pr[deg(v) \geq k]$ according to Equation (3), can all be derived in $\mathcal{O}(kd_v)$ time.

Thus, the dynamic-programming method just described has the same complexity as the method based on Equation (4). But, at the same time, it (*i*) alleviates the numerical-stability shortcomings, as the numbers involved into Equation (6) are all probabilities ≤ 1 (unlike the numbers $\tilde{p}_e$ which range from $[0, \infty)$), and (*ii*) can easily be employed for efficiently updating η-degrees when an edge is removed from the graph, as described next.

Time complexity. The η-degree of a vertex v can be computed incrementally. We start with $k = 0$ and $\Pr[deg(v) \geq 0] = 1$. Then, we increase k one by one and compute $\Pr[deg(v) \geq k]$ as $\Pr[deg(v) \geq k-1] - \Pr[deg(v) = k-1]$. We stop once $\Pr[deg(v) \geq k] < \eta$, and we set $\eta\text{-}deg(v) = k-1$. This way, we need to compute probabilities $\Pr[deg(v) = k]$ only for $k = 0, ..., \eta\text{-}deg(v) + 1$, which, according to the findings reported above, leads to a time complexity of $\mathcal{O}(\eta\text{-}deg(v) \times d_v)$. Clearly, in the worst case, such a complexity equals $\mathcal{O}(d_v^2)$, but we expect in practice $\eta\text{-}deg(v)$ reasonably lower than d_v, especially for those vertices having very large d_v and/or large enough η values.

Computing all η-degrees hence takes $\mathcal{O}(\sum_{v \in V} \eta\text{-}deg(v) \times d_v)$. Denoting by Δ the maximum η-degree over all vertices in the graph, i.e., $\Delta = \max_{v \in V} \eta\text{-}deg(v)$, the complexity can be more compactly expressed as $\mathcal{O}(\sum_{v \in V} d_v \Delta) = \mathcal{O}(m\Delta)$.

Updating η-degrees

We now consider the case where the η-degree of a vertex v needs to be updated because an edge incident to v has been removed. We recall that this is the other crucial step of our (k,η)-CORES algorithm (Algorithm 2, Line 12).

As anticipated, we can exploit Theorem 3 to avoid from-scratch recomputations. The problem can be reduced to (efficiently) updating the probabilities $\Pr[deg(v) = 0], \ldots, \Pr[deg(v) = \eta\text{-}deg(v)]$, whose earlier values are available because of the computation of the earlier η-degree. Once all these new probabilities are computed, the new η-degree can be derived by the same incremental process described in the previous paragraph "Time complexity".

Let e denote the edge to be removed and let $\Pr[deg(v|\neg e) = i]$, for all $i \in [0, \ldots, \eta\text{-}deg(v)]$, be a shorthand for the new probabilities $\Pr[deg(v|N_v \setminus \{e\}) = i]$ to be computed. Such $\Pr[deg(v|\neg e) = i]$ values can be derived by rearranging Equation (6) as follows:

$$\Pr[deg(v|\neg e) = i] = \frac{\Pr[deg(v) = i] - p_e \Pr[deg(v|\neg e) = i-1]}{1 - p_e}. \tag{7}$$

This way, one can set $\Pr[deg(v|\neg e) = 0] = \frac{1}{1-p_e} \Pr[deg(v) = 0]$, and apply Equation (7) to compute the remaining $\Pr[deg(v|\neg e) = i]$ values, for all $i \in [1..\eta\text{-}deg(v)]$. Each probability $\Pr[deg(v|\neg e) = i]$ takes constant time. Computing all the new probabilities, and, hence, updating the η-degree of v, globally takes $\mathcal{O}(\eta\text{-}deg(v))$ time, thus improving upon the $\mathcal{O}(\eta\text{-}deg(v) \times d_v)$ time of a from-scratch recomputation.

Overall running time of (k,η)-CORES

We analyse now the overall time complexity of our (k,η)-CORES algorithm. The initialisation phase (Lines 1–6) is dominated by the computation of the initial η-degree for all vertices, which takes $\mathcal{O}(m\Delta)$ time (Δ is the maximum η-degree over all vertices). In the main cycle (Lines 7–18), like the deterministic case, each vertex is visited only once and then removed from the graph. For each visited vertex v, the η-degree of all its neighbours has to be updated. As reported above, for a single neighbour u, this takes $\mathcal{O}(\eta\text{-}deg(u))$. Thus, the main cycle globally takes $\mathcal{O}(\sum_{v \in V} \sum_{u:(u,v) \in N_v} \eta\text{-}deg(u)) = \mathcal{O}(\sum_{v \in V} d_v \Delta) = \mathcal{O}(m\Delta)$. In conclusion, the running time of the (k,η)-CORES algorithm is therefore $\mathcal{O}(m\Delta)$.

4. SPEEDING-UP (k,η)-CORES

In this section we show how to further speed-up our (k,η)-CORES algorithm. Our key observation is that the main bottleneck of (k,η)-CORES is the computation of initial η-degrees (experimentally confirmed in Section 5): although this step is asymptotically as fast as updating η-degrees after a vertex removal, the latter is in practice faster as it is performed on a graph that gets progressively smaller. In this regard, we derive a fast-to-compute lower bound on the η-degree and use it as a placeholder during the first iterations, while replacing it with the actual η-degree only when the vertex at hand is going to be processed. This way, the initial η-degrees can be computed only when actually needed and on a smaller graph, thus leading to the desired speed-up.

In the following we provide the details of our lower bound on the η-degree and show how to efficiently update this bound after vertex removals. Then, we describe how to incorporate such findings into the enhanced algorithm.

Lower bound on the η-degree. We define our lower bound on the η-degree in terms of the *regularised beta function*. Given a real number $z \in [0,1]$ and two integers a and b, the regularized beta function $I_z(a,b)$ is defined as the ratio between the *incomplete beta function* $B(z;a,b)$ and the *beta function* $B(a,b)$ [29]:

$$I_z(a,b) = \frac{B(z;a,b)}{B(a,b)} = \sum_{i=a}^{a+b-1} \binom{a+b-1}{i} z^i (1-z)^{a+b-1-i}.$$

Given a vertex v in the input graph, let $p_{min}(v)$ denote the minimum probability on the edges incident to v, i.e., $p_{min}(v) = \min_{e \in N_v} p_e$. The next lemma shows how the probability for v to have degree no less than k can be lower-bounded by using the regularised beta function I.

LEMMA 1. *Given an uncertain graph $\mathcal{G} = (V,E,p)$, for every vertex $v \in V$ and for all $k \in [0..d_v]$ it holds that*

$$\Pr[deg(v) \geq k] \;\geq\; I_{p_{min}(v)}(k, d_v - k + 1).$$

PROOF. Consider a vertex v' having as many incident edges as v, and assume that each edge incident to v' has probability $p_{min}(v)$. It is easy to see that $\Pr[\deg(v) = i] \geq \Pr[\deg(v') = i]$, for all i. Exploiting Equation (4) we get:

$$\begin{aligned}\Pr[\deg(v) = i] &\geq \Pr[\deg(v') = i] = \\ &= \sum_{\substack{N \subseteq N_{v'}, \\ |N| = i}} \prod_{e \in N} p_{min}(v) \prod_{e \in N_{v'} \setminus N} (1 - p_{min}(v)) = \\ &= \binom{d_v}{i} (p_{min}(v))^i (1 - p_{min}(v))^{d_v - i}.\end{aligned}$$

Combining such a result with Equation (3) we obtain:

$$\Pr[deg(v) \geq k] = \sum_{i=k}^{d_v} \Pr[deg(v) = i] \geq$$

$$\geq \sum_{i=k}^{d_v} \binom{d_v}{i} (p_{min}(v))^i (1 - p_{min}(v))^{d_v - i} = I_{p_{min}(v)}(k, d_v - k + 1).$$

The lemma follows. □

The desired lower bound on η-degree can now immediately be derived by exploiting Lemma 1. We denote such a lower bound by η-LB and formally state it in the next theorem.

THEOREM 4. *Given an uncertain graph $\mathcal{G} = (V,E,p)$, for every vertex $v \in V$ it holds that*

$$\eta\text{-}deg(v) \geq \eta\text{-LB}(v) = \max\{k \in [0..d_v] \,|\, I_{p_{min}(v)}(k, d_v - k + 1) \geq \eta\}.$$

The computation of the above lower bound is very fast. For a fixed z, the values $I_z(a,b)$ of the regularised beta function are monotonically non-increasing as a increases and/or b decreases. Therefore, the lower bounds on $\Pr[deg(v) \geq k]$ are monotonically non-increasing as k increases and one can thus perform binary search to derive the maximum k such that $I_{p_{min}}(k, d_v - k + 1) \geq \eta$, which, according to Theorem 4, corresponds to the lower bound η-LB(v). The computation of η-LB(v) requires a logarithmic (in the number of edges of v) number of evaluations of I_z. Each evaluation of I_z can be computed in constant time using tables [23]. Thus, computing η-LB(v) for a vertex v takes $\mathcal{O}(\log d_v)$ time.

A major feature of the lower bound η-LB is its fast from-scratch computation. Here we show that it can also be updated very efficiently (i.e., in constant time) when an edge is removed from the graph. To this end, we first need to report a couple of results. We start by showing that the η-degree of a vertex v can decrease at most by one when an edge incident to v is removed (Lemma 2).

LEMMA 2. *Given an uncertain graph $\mathcal{G} = (V,E,p)$ and a vertex $v \in V$, let $e \in N_v$ be an edge incident to v and let $\mathcal{H} = (V, E \setminus \{e\}, p)$ be the subgraph of $\mathcal{G}$ where e is missing. Also, let $\eta\text{-}deg_{\mathcal{H}}(v)$ be the η-degree of v in $\mathcal{H}$. It holds that $\eta\text{-}deg_{\mathcal{H}}(v) > \eta\text{-}deg(v) - 2$.*

PROOF.

$$\Pr[deg(v) \geq k] = \sum_{i=k}^{d_v - 1} \Pr[deg_{\mathcal{H}}(v) = i] + p_e \Pr[deg_{\mathcal{H}}(v) = k-1] =$$

$$= \underbrace{\sum_{i=k-1}^{d_v - 1} \Pr[deg_{\mathcal{H}}(v) = i]}_{\Pr[deg_{\mathcal{H}}(v) \geq k-1]} - (1 - p_e) \Pr[deg_{\mathcal{H}}(v) = k-1] \leq$$

$$\leq \Pr[deg_{\mathcal{H}}(v) \geq k-1].$$

By the definition of η-degree we know that $\Pr[deg_{\mathcal{H}}(v) \geq \eta\text{-}deg_{\mathcal{H}}(v) + 1] < \eta$; thus, setting $k = \eta\text{-}deg_{\mathcal{H}}(v) + 2$ in the above inequality, we get

$$\Pr[deg(v) \geq \eta\text{-}deg_{\mathcal{H}}(v) + 2] \leq \Pr[deg_{\mathcal{H}}(v) \geq \eta\text{-}deg_{\mathcal{H}}(v) + 1] < \eta.$$

Then, $\eta\text{-}deg(v) < \eta\text{-}deg_{\mathcal{H}}(v) + 2$, or, equivalently, $\eta\text{-}deg_{\mathcal{H}}(v) > \eta\text{-}deg(v) - 2$. The lemma follows. □

Based on the above lemma, we can also prove that the lower bound η-LB(v) of a vertex v can decrease at most by one when an edge incident to v is removed.

THEOREM 5. *Given an uncertain graph $\mathcal{G} = (V,E,p)$ and a vertex $v \in V$, let $e \in N_v$ be an edge incident to v and let $\mathcal{H} = (V, E \setminus \{e\}, p)$ be the subgraph of $\mathcal{G}$ where e is missing. Also, let $\eta\text{-LB}_{\mathcal{H}}(v)$ be the lower bound on the η-degree of v in $\mathcal{H}$. It holds that $\eta\text{-LB}_{\mathcal{H}}(v) > \eta\text{-LB}(v) - 2$.*

PROOF. Consider a vertex v' having as many incident edges as v, and assume that each edge incident to v' has probability $p_{min}(v)$. It is easy to see that the η-degree $\eta\text{-}deg(v')$ of v' equals the lower bound η-LB(v). Combining this with Lemma 1, we get $\eta\text{-LB}_{\mathcal{H}}(v) = \eta\text{-}deg_{\mathcal{H}}(v') > \eta\text{-}deg(v') - 2 = \eta\text{-LB}(v) - 2$. The theorem follows. □

Theorem 5 can be exploited for safely updating η-LB in constant time. Let e denote again the edge incident to v to be removed and let $\mathcal{H}$ be the subgraph of $\mathcal{G}$ where e is missing. Thus, η-LB(v) denotes the earlier lower bound of v, while η-LB$_{\mathcal{H}}(v)$ denotes the new lower bound to be computed after the e's removal. The idea is to compute (in constant time) just the value $I_{p_{min}(v)}(\eta\text{-LB}(v), (d_v - 1) - \eta\text{-LB}(v) + 1) = I_{p_{min}(v)}(\eta\text{-LB}(v), d_v - \eta\text{-LB}(v))$. Lemma 1 ensures that

$$\Pr[deg_{\mathcal{H}}(v) \geq \eta\text{-LB}(v)] \geq I_{p_{min}(v)}(\eta\text{-LB}(v), d_v - \eta\text{-LB}(v)).$$

Thus, if $I_{p_{min}(v)}(\eta\text{-LB}(v), d_v - \eta\text{-LB}(v))$ is still $\geq \eta$, then the lower bound has not changed, i.e., $\eta\text{-LB}_{\mathcal{H}}(v) = \eta\text{-LB}(v)$. Otherwise, it means that the lower bound has decreased. According to Theorem 5, this decreasing can be at most by one, hence we can safely set $\eta\text{-LB}_{\mathcal{H}}(v) = \eta\text{-LB}(v) - 1$.

A major shortcoming of updating η-LB as described above is that, for each vertex v, we need to load/keep-in-memory $\mathcal{O}(d_v^2)$ values of I_z (i.e., all values within $\{I_{p_{min}(v)}(k, h-k+1) \mid h \in [0..d_v], k \in [0..h]\}$). This would penalize too much both time and space complexity of the algorithm. However, this can be overcome by still relying on Theorem 5. The idea is to simply set $\eta\text{-LB}_{\mathcal{H}}(v) = \max\{0, \eta\text{-LB}(v) - 1\}$ *every time* an edge e incident to v is removed, no matter whether $I_{p_{min}(v)}(\eta\text{-LB}(v), d_v - \eta\text{-LB}(v)) \geq \eta$ or not. Indeed, Theorem 5 guarantees that η-LB$(v) - 1$ is still a lower-bound for η-$deg(v)$, even though possibly less tight. This way our algorithm would require only $\mathcal{O}(d_v)$ values of I_z for each vertex v, i.e., just the values $\{I_{p_{min}(v)}(k, d_v - k + 1) \mid k \in [0..d_v]\}$.

The E-(k,η)-cores algorithm. We now provide the details of our ENHANCED-(k,η)-CORES (for short, E-(k,η)-CORES) algorithm (pseudocode omitted for space reasons). The algorithm follows the scheme of the basic (k,η)-CORES algorithm (Algorithm 2). The main difference is that, for each vertex v, the lower bound η-LB(v) is computed in the initialisation phase, rather than the exact η-degree. A set $\widetilde{V}$ keeps trace of the vertices for which the exact η-degree has not been computed yet. Right after initialisation, $\widetilde{V}$ corresponds to the whole vertex set V. In the main cycle, vertices are processed based on their (lower bound on) η-degree. When a vertex v is being processed, it is primarily checked whether its exact η-degree is already available. If not, the exact η-degree of v is computed and v is moved to the proper set of the vector $\mathbf{D}$, so that it can be processed in the correct (possibly later) iteration. Otherwise, if the exact η-degree of v is available, the η-core number of v is set and the η-degrees (either the exact or the lower bounds) of all v's neighbours are updated.

The worst-case time complexity of E-(k,η)-CORES is the same as the basic (k,η)-CORES algorithm, i.e., $\mathcal{O}(m\Delta)$. However, smaller running times are expected in practice due to the lazy computation/updating of η-degrees in reduced versions of the input graph.

5. EXPERIMENTS

In this section we report quantitative experiments on efficiency and numerical stability of our (k,η)-CORES and E-(k,η)-CORES algorithms (Sections 3 and 4).[3] For this task we use the following real-world uncertain graphs.

Flickr (`www.flickr.com`, $|V| = 24\,125$, $|E| = 300\,836$). We borrowed the dataset from [24], where the probability of an edge between two users is defined based on *homophily*, the principle that similar interests indicate social ties. Particularly, [24] uses as a measure of homophily the Jaccard coefficient of the interest groups shared by the two users.

DBLP (`www.informatik.uni-trier.de/~ley/db/`, $|V| = 684\,911$, $|E| = 2\,284\,991$). The dataset was borrowed from [24, 15]. Two authors are connected if they co-authored at least once, and the probability on an edge expresses the fact that the collaboration has not happened by chance: the more the collaborations, the larger the probability. Precisely, [24, 15] define the probability of each edge based on an exponential function to the number of collaborations.

BioMine (`biomine.org`, $|V| = 1\,008\,200$, $|E| = 6\,742\,939$). A snapshot of the database of the BioMine project [26] containing biological interactions. Edges inherently come with probabilities. The probability of any edge provides evidence that the interaction actually exists.

[3] We implemented our code in Java and run experiments on a 2.83GHz, 32GB Intel Xeon server.

Table 1: Times (secs) of the proposed methods for computing (k,η)-core decomposition (precision 64 bits). The column "*gain (%)*" reports the gain of the E-(k,η)-cores algorithm over the (k,η)-cores algorithm.

η	*initial* η-*degrees*	*main cycle*	*total*	*initial* η-*degrees*	*main cycle*	*total*	*gain (%)*
	Flickr, (k,η)-CORES			Flickr, E-(k,η)-CORES			
0.1	15.45	8.88	24.33	14.41	7.98	22.39	7.99%
0.3	13.73	7.89	21.61	12.90	7.22	20.12	6.89%
0.5	12.56	7.33	19.89	11.86	6.71	18.57	6.62%
0.7	11.45	6.64	18.09	10.82	6.14	16.96	6.25%
0.9	9.86	5.72	15.58	9.34	5.32	14.66	5.87%
	DBLP, (k,η)-CORES			DBLP, E-(k,η)-CORES			
0.1	53.81	36.92	90.73	38.23	26.45	64.68	28.71%
0.3	49.08	33.16	82.24	36.28	25.21	61.48	25.24%
0.5	44.74	31.14	75.88	33.98	24.45	58.43	23.00%
0.7	40.65	28.40	69.05	31.86	23.07	54.92	20.46%
0.9	35.54	24.42	59.96	28.40	21.06	49.46	17.51%
	BioMine, (k,η)-CORES			BioMine, E-(k,η)-CORES			
0.1	4801	1549	6350	4388	1404	5792	8.78%
0.3	4704	1542	6246	4333	1447	5780	7.46%
0.5	4645	1538	6183	4281	1404	5685	8.05%
0.7	4568	1523	6091	4240	1403	5643	7.35%
0.9	4498	1478	5977	4151	1423	5575	6.72%

Efficiency. Table 1 reports on the running times exhibited by our (k,η)-CORES (left) and E-(k,η)-CORES (right) algorithms on the selected datasets. Times are split by the main phases of computing initial η-degrees and running the main cycle. Both algorithms are very fast on Flickr and DBLP. They take on average around 20 and 60 seconds, respectively. On BioMine, which is much larger and denser, clearly the time increases. However, the time required by our algorithms on the latter dataset is in the order of one hour. This is reasonable for networks of such size and testifies the applicability of our methods to very large uncertain graphs.

As expected, E-(k,η)-CORES runs faster than the basic (k,η)-CORES algorithm, allowing a reduction of the total time up to around 30% (DBLP, $\eta = 0.1$). The gain is more evident on the larger datasets (i.e., DBLP and BioMine) and is generally increasing as η decreases. The latter finding is expected because the smaller η, the larger the η-degree of a vertex, and, thus, the better the chance for the lower-bound to be tighter and lead to better pruning. Larger η-degrees for smaller η is also the reason why times (for both phases and both algorithms) are increasing with smaller η.

Numerical stability. As discussed in Section 3, probabilities may lead to numerical instability. To prevent this, one can exploit native solutions provided by modern programming languages to enlarge range and/or precision of the numerical representation. As a side effect, this would slow down the overall computation as larger precision implies slower arithmetic computations. Thus, the goal is to minimise the number of critical operations that may lead to numerical instability, to avoid using a too large precision with the aim of achieving reasonable accuracy. As reported in Section 3, a major feature of the novel dynamic-programming method we employ in our algorithms to compute/update η-degrees is to alleviate such numerical issues. We next provide experimental evidence on this.

First, we report results by varying the precision used for representing numbers (we consider 32, 64, 128, and 256 bits

Table 2: Accuracy of (k,η)-core index for $\eta = 0$ w.r.t. deterministic core index (ground truth) for different values of precision (bits).

dataset	*pr=32*	*pr=64*	*pr=128*	*pr=256*
	avg absolute error			
Flickr	6.17	5.12	3.4	2.26
DBLP	0.27	0.1	0.03	0.01
BioMine	2.18	1.25	0.41	0.14
	% vertices with non-zero error			
Flickr	31.69%	18.91%	11.92%	6.00%
DBLP	17.48%	2.27%	0.51%	0.18%
BioMine	1.51%	1.11%	0.47%	0.09%

as precision levels).[4] We note that, for $\eta = 0$, the (k,η)-core decomposition of an uncertain graph $\mathcal{G}$ should ideally correspond to the core decomposition of the deterministic graph derived from $\mathcal{G}$ by ignoring probabilities. Thus, we measure accuracy by comparing, for each vertex, the 0-core number outputted by our algorithms with the core number returned by the standard k-CORE algorithm (Algorithm 1) on such a deterministic graph.

Tables 2 and 3 show accuracy results (in terms of per-vertex average absolute error and percentage of vertices with core number other than the exact one) and running times, respectively. We report times separately for (k,η)-CORES and E-(k,η)-CORES, while accuracy is the same for both. As expected, larger precision leads to better accuracy and worse efficiency. Particularly, the results show a linear trend: doubling the precision, time doubles while errors get halved.

We also compare the results of our algorithms when equipped with the proposed dynamic-programming method to the results of our algorithms equipped with the method that computes/updates η-degrees using the formula in Equation (5). We denote our proposed combination "(k,η)-CORES + dynamic-programming method" simply as (k,η)-CORES, while we refer to the "baseline" combination "(k,η)-CORES + Equation (5)-based method" as EQ5. These results are summarised in Table 4 (precision 64 bits). Our method outperforms EQ5 in terms of both average absolute error and percentage of vertices with non-zero error. Particularly, the average absolute error of the EQ5 method is reduced by 9% (Flickr), 41% (DBLP), and 40% (BioMine).

6. INFLUENCE MAXIMIZATION

The *influence-maximization* problem [16], has received a great deal of attention over the last decade. It requires to find a set of vertices S, with $|S| = s$, that maximizes the *expected spread*, i.e., the expected number of vertices that would be infected by a viral propagation started in S, under a certain probabilistic propagation model.

The *independent cascade model* [16] is a widely-used propagation model; under this model, the problem of finding a set S of s vertices that maximizes the expected spread $\sigma(S)$ is **NP**-hard. However, the submodularity of $\sigma(S)$ allows the GREEDY algorithm that iteratively adds to S the vertex bringing the largest marginal gain to the objective function to achieve a $(1 - \frac{1}{e})$ approximation guarantee. Unfortunately, finding the maximum-marginal-gain vertex requires to solve a **#P**-complete reliability problem. Hence, existing approaches usually apply sampling methods (e.g., Monte Carlo) to estimate the best seed vertex at each iteration of the algorithm. This drastically affects the efficiency of the algorithm, thus limiting its applicability only to moderately-sized networks (the time complexity of the algorithm is $\mathcal{O}(sTnm)$, where T is the number of Monte Carlo samples, with $T \in [1\,000, 10\,000]$, usually). Optimizations of the basic algorithm have been defined which exploit the submodularity of σ to avoid unneeded computations [12], but the improvement achieved is typically not enough to handle large graphs (in the experiment that we show below, on a moderately sized graph a state-of-the-art algorithm such as CELF++ [12] could not finish after several weeks).

[4] In our implementation, we use the *BigDecimal* Java API, which allows for representing numbers arbitrarily large and/or small, and with arbitrary user-defined precision (up to "unlimited" precision).

Table 3: Times (secs) of the two proposed methods for computing (k,η)-core decomposition, for $\eta = 0.1$, for different values of precision (bits).

prec. (bits)	*initial η-degrees*	*main cycle*	*total*	*initial η-degrees*	*main cycle*	*total*	*gain (%)*
	Flickr, (k,η)-CORES			Flickr, E-(k,η)-CORES			
32	6.96	3.83	10.79	6.63	3.73	10.36	3.94%
64	15.23	8.89	24.12	14.08	7.94	22.02	8.72%
128	25.55	14.48	40.03	23.69	12.92	36.62	8.53%
256	34.35	22.13	56.48	31.95	19.68	51.63	8.59%
	DBLP, (k,η)-CORES			DBLP, E-(k,η)-CORES			
32	26.71	20.22	46.93	19.46	15.51	34.97	25.48%
64	56.73	39.19	95.92	40.98	27.17	68.14	28.96%
128	86.65	59.81	146.5	62.84	40.40	103.2	29.51%
256	128.7	89.14	217.8	91.15	59.30	150.5	30.93%
	BioMine, (k,η)-CORES			BioMine, E-(k,η)-CORES			
32	2 376	704	3 080	2 021	659	2 681	12.97%
64	5 452	1 693	7 145	4 738	1 390	6 128	14.24%
128	9 815	3 146	12 961	8 153	2 607	10 760	16.98%
256	13 296	5 055	18 351	11 274	4 515	15 789	13.96%

Table 4: Accuracy of the proposed method in terms of error w.r.t. a ground truth (precision 64 bits).

	avg absolute error		*vertices w. non-zero error*	
dataset	(k,η)-CORES	EQ5	(k,η)-CORES	EQ5
Flickr	5.12	5.62	18.91%	19.91%
DBLP	0.1	0.17	2.27%	4.42%
BioMine	1.25	2.07	1.11%	1.36%

Within this view, a useful application of our (k, η)-core decomposition is to provide a way to speed-up the execution of the GREEDY algorithm. The idea is simple: just reduce the input graph $\mathcal{G}$ by keeping only the inner-most η-shells and run the (optimized version of the) GREEDY algorithm on such a reduced graph. The rationale here is that, as experimentally observed in [17], the core decomposition of the deterministic version of $\mathcal{G}$, is a direct indicator of the expected spread of a vertex: the higher the core index is, the more likely the vertex is an influential spreader. The finding in [17] however exploits cores derived from a deterministic version of the input graph, thus completely ignoring its probabilistic nature. We conjecture that exploiting a notion of core decomposition defined ad-hoc for uncertain graphs can only positively affect the behaviour observed in [17]. We next empirically show the correctness of our conjecture.

Experiments. We use a small *directed* graph from Twitter ($|V| = 21\,882, |E| = 372\,005$), and a set of propagations of URLs in the social graph, which we use as past evidence to learn the influence probabilities (we employ the traditional method described in [11] for this). Each edge (u, v) expresses the fact that v is a follower of u and the corresponding probability provides evidence that an action performed by u will be performed by v as well.

The objective here is to show that running the standard GREEDY influence-maximization algorithm on a reduced version of the graph given by the inner-most (k, η)-shells allows to achieve high-quality results while keeping the running time small. We test our method replacing the notion of de-

Table 5: Expected spread achieved by the proposed (k,η)-cores-based method vs. some baselines with varying the output set size $|S|$.

	$\|S\| = 10$	$\|S\| = 20$	$\|S\| = 30$
(k,η)-CORES	9 570	9 606	9 610
OUT-DEGREE	9 014	9 016	9 130
η-DEGREE	9 019	9 089	9 125
EXP-DEGREE	9 012	9 093	9 123
k-CORES	9 134	9 192	9 223

gree with out-degree (given that the graph is directed) and setting $\eta = 0.5$. We obtain 8 cores and keep the three innermost (k, η)-shells. This gives a reduced graph with 2 064 vertices and 86 142 edges. We run the optimized version of the GREEDY algorithm defined in [12], i.e., the CELF++ algorithm, on such a reduced graph and take the seed vertices S outputted as our result.

For accuracy evaluation, we compute the expected spread achieved by S on *the whole graph* (using Monte Carlo sampling with 10 000 samples). As criteria for comparison, we use the top-K vertices ranked according to the following baseline ranking functions: (*i*) maximum out-degree (ignoring probabilities, as suggested in the seminal work on influence maximization [16]), (*ii*) maximum η-degree, (*iii*) maximum expected degree (computed by summing the probabilities on the edges outgoing from a vertex), and (*iv*) vertices computed by running CELF++ on the graph reduced according to *deterministic* core decomposition (ignoring probabilities). Note that we could not use the results of the direct execution of CELF++ on the whole graph due to its excessive running time (it could not finish in several weeks).

The results reported in Table 5 (we vary $|S|$ from 10 to 30) show how our (k,η)-CORES-based method evidently outperforms all the baselines, allowing to increase the spread up to 590 (OUT-DEGREE), 551 (η-DEGREE), 558 (EXP-DEGREE), and 436 (k-CORES). As far as efficiency, we report runtimes in the order of 4–5 hours (with $|S| = 30$), which are times clearly affordable—contrast to the unaffordable runtime of the direct execution of CELF++ on the whole graph.

7. TASK-DRIVEN TEAM FORMATION

In *task-driven team formation* we are given a collaboration graph $G = (V, E, \tau)$, where vertices are individuals and edges are assigned a probabilistic topic model τ, representing (a distribution on) the topics exhibited by past collaborations. The topic model can be produced by standard methods, such as the popular *Latent Dirichlet Allocation* (LDA) [6]. The input of LDA (or any other similar method) is (*i*) a number Z of topics, and (*ii*) for each edge $(u, v) \in E$, a document $d(u, v)$ representing all the past collaborations between u and v. The document $d(u, v)$ is a bag of terms coming from a finite vocabulary Σ. The output is the topic model τ, that is:

- for each edge $(u, v) \in E$ and each topic $z \in [1..Z]$, the probability $p^z_{u,v} = (z|u, v)$ that the collaborations between u and v are on the topic z, with $\sum_{z=1}^{Z} p^z_{u,v} = 1$.
- for each term $t \in \Sigma$, a distribution over topics, i.e., for each topic $z \in [1, Z]$, the probability $\gamma^z_t = P(t|z)$ that the term t has been generated by the topic z, with $\sum_{t\in\Sigma} \gamma^z_t = 1$.

A task-driven team-formation query is a pair $\langle T, Q\rangle$, where $T \subset \Sigma$ is a set of terms describing a task, and $Q \subset V$ is a set of vertices (possibly even a single vertex). The goal is to find an answer vertex set A, with $Q \subseteq A$, which is a good team to perform the task described by the terms in T. Being a good team means having a good affinity among the team members with respect to the given task. We report more formal details on this in the following.

The query task T, together with the topic model τ, induce a single probability value $p^T(u, v)$ for each edge $(u, v) \in E$, such that $p^T(u, v)$ represents the likelihood that T has been generated by a collaboration between u and v:

$$p^T(u, v) = p(u, v|T) = \prod_{t\in T} \sum_{z=1}^{Z} \gamma^z_t p^z_{u,v}. \quad (8)$$

Hence, given a task T, the input collaboration graph G yields an uncertain graph $\mathcal{G}^T = (V, E, p^T)$. This way, given $\mathcal{G}^T$ and a set of query vertices $Q \subseteq V$, the task of finding a good team for the query at hand directly translates into finding a subgraph of $\mathcal{G}^T$ that represents a good community for Q. Formally, the goal is to find a *connected* subgraph $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ of $\mathcal{G}^T$ that (*i*) contains all query vertices ($Q \subseteq V_\mathcal{H}$), and (*ii*) maximizes a notion of density. Particularly, as far as the density measure, the *minimum degree* has been widely recognized as a principled choice for this kind of problem.[5] We therefore rely on this notion of density and ask for the subgraph $\mathcal{H}$ to maximize the minimum η-degree of a vertex in $\mathcal{H}$. The resulting problem statement is:

PROBLEM 2 (TASK-DRIVEN TEAM FORMATION). *Given a collaboration graph $G = (V, E, \tau)$ and a query $\langle T, Q\rangle$, let $\mathcal{G}^T$ be the uncertain graph derived from G and T as described in Equation (8). Given a threshold $\eta \in [0, 1]$, we want to find a* connected *subgraph $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ of $\mathcal{G}^T$ induced by a set of vertices $V_\mathcal{H}$ such that*

$$V_\mathcal{H} = \arg\max_{Q\subseteq S\subseteq V} \min_{u\in S} \eta\text{-}deg(u). \quad \square$$

Exploiting (k,η)-cores for team formation. We now show that Problem 2 can be optimally solved by resorting to our notion of (k,η)-core decomposition. This result is stated in the next theorem (we omit the proof for space reasons).

THEOREM 6. *Given an uncertain graph $\mathcal{G}^T$ and a threshold $\eta \in [0, 1]$, let $\mathbf{C} = \{C_0, C_1, \ldots, C_{k^*}\}$ be the (k,η)-core decomposition of $\mathcal{G}^T$ (with $C_0 \supseteq C_1 \supseteq \cdots \supseteq C_{k^*}$), and, given a set of query vertices $Q \subseteq V$, let C^*_Q be the smallest-sized core in $\mathbf{C}$ such that every $q \in Q$ belongs to the same connected component of C^*_Q.*

Then, the solution to Problem 2 is given by the connected component of C^*_Q that contains Q. $\square$

Theorem 6 provides us with a principled way of solving Problem 2. The solution can be summarized as follows:

1. Given a collaboration graph $G = (V, E, \tau)$ and a task-driven team-formation query $\langle T, Q\rangle$, derive the uncertain graph $\mathcal{G}^T = (V, E, p^T)$ (Equation (8)).[6]
2. Compute the (k,η)-core decomposition $\mathbf{C}$ of $\mathcal{G}^T$;

[5] As argued in [27], maximizing the minimum degree provides a better evidence of the goodness of a community than, e.g., the maximization of the average degree, which is instead more suitable for dense-subgraph discovery.

[6] As Equation (8) can produce very small probabilities, in our implementation we prune $\mathcal{G}^T$ by removing edges with probability smaller than a threshold ϵ ($\epsilon = 10^{-16}$ in our experiments).

Table 6: Three examples of task-driven team-formation queries and corresponding results.

$T = \{gene, express\}$, $Q = \{H.V.Jagadish\}$	$T = \{xml, tree\}$, $Q = \{H.V.Jagadish, S.Muthukrishnan\}$	$T = \{auction, model\}$, $Q = \{S.Muthukrishnan\}$
Brian D. Athey, Giovanni Scardoni, Kathleen A. Stringer, Venkateshwar G. Keshamouni, Jing Gao, Terry E. Weymouth, Vasudeva Mahavisno, Charles F. Burant, Christopher W. Beecher, Maureen A. Sartor, Alla Karnovsky, Rork Kuick, Zach Wright, James D. Cavalcoli, Gilbert S. Omenn, **H. V. Jagadish**, Carlo Laudanna, Tim Hull, Barbara R. Mirel, V. Glenn Tarcea	**S. Muthukrishnan**, Panagiotis G. Ipeirotis, Lauri Pietarinen **H. V. Jagadish**, Divesh Srivastava, Nick Koudas	Uri Nadav, Noam Nisan, Jon Feldman, Vahab S. Mirrokni, Gagan Aggarwal, Tanmoy Chakraborty, Aranyak Mehta Evdokia Nikolova, **S. Muthukrishnan**, Martin Pal, Clifford Stein, Eyal Even-Dar Florin Constantin, Yishay Mansour

3. Visit the cores in $\mathbf{C}$ starting from the smallest-sized one (i.e., the inner-most core), until finding C^*_Q;
4. Return the connected component of C^*_Q containing Q as the solution to Problem 2.

Experiments. We consider task-driven team formation in the context of collaborations among computer-science researchers. We build a collaboration network from the DBLP database (`www.informatik.uni-trier.de/~ley/db/`): vertices are authors and an edge connects two authors if they co-authored at least once. The resulting graph has $|V| = 1\,089\,442$ and $|E| = 4\,144\,697$. For each edge, we take the bag of words of the titles of all papers coauthored by the two authors (words are stemmed and stop-words are removed), and apply LDA to infer the topic model τ (we set $Z = 100$).

In Table 6 we report the results of three task-driven team-formation queries. The first two queries share the query vertex `H. V. Jagadish`, but the first task is about *gene expression* while the second one is about *xml*: as expected the two proposed teams are very different. The third query shares with the second one the vertex `S. Muthukrishnan`; but, unlike the previous one that is about xml (a database topic), the third query is about *auction models* (an algorithm-theory topic): the different teams proposed correctly reflect the difference in the tasks. It is worth noticing that the extraction of these teams, following the process described above and exploiting our efficient (k,η)-core decomposition, takes approximately 2-3 seconds on a commodity laptop.

8. CONCLUSIONS

In this paper we extend the graph tool of core decomposition to the context of uncertain graphs. We define the (k, η)-core concept, and we devise efficient algorithms for computing a (k, η)-core decomposition. As a future work, we plan to investigate the relationship between (k, η)-cores and other definitions of (probabilistic) dense subgraphs, so as to exploit the former as a speeding-up preprocessing.

9. REFERENCES

[1] E. Adar and C. Re. Managing Uncertainty in Social Networks. *IEEE Data Eng. Bull.*, 30(2):15–22, 2007.

[2] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS*, 2005.

[3] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, 2009.

[4] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting Protein Complex Membership using Probabilistic Network Reliability. *Genome Res.*, 14:1170–1175, 2004.

[5] V. Batagelj and M. Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2):129–145, 2011.

[6] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[7] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting Uncertainty in Graphs for Identity Obfuscation. *PVLDB*, 5(11):1376–1387, 2012.

[8] X. H. Chen, A. P. Dempster, and J. S. Liu. Weighted finite population sampling to maximize entropy. *Biometrika*, 81:457–469, 1994.

[9] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[10] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *ISAAC*, 2010.

[11] A. Goyal, F. Bonchi, and L. V. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, 2010.

[12] A. Goyal, W. Lu, and L. V. Lakshmanan. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *WWW*, pages 47–48, 2011.

[13] J. Healy, J. Janssen, E. E. Milios, and W. Aiello. Characterization of graphs using degree cores. In *WAW*, 2006.

[14] R. Jin, L. Liu, and C. C. Aggarwal. Discovering Highly Reliable Subgraphs in Uncertain Graphs. In *KDD*, 2011.

[15] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-Constraint Reachability Computation in Uncertain Graphs. *PVLDB*, 4(9):551–562, 2011.

[16] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.

[17] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identifying influential spreaders in complex networks. *Nature Physics 6, 888*, 2010.

[18] G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *TKDE*, 25(2):325–336, 2013.

[19] G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *J. Algorithms*, 17(2):222–236, 1994.

[20] D. L.-Nowell and J. Kleinberg. The Link Prediction Problem for Social Networks. In *CIKM*, 2003.

[21] V. E. Lee, N. Ruan, R. Jin, and C. C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*. 2010.

[22] L. Liu, R. Jin, C. Aggrawal, and Y. Shen. Reliable clustering on uncertain graphs. In *ICDM*, 2012.

[23] K. Pearson. *Tables of the Incomplete Beta-Function*. Cambridge University Press, 1968.

[24] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-Nearest Neighbors in Uncertain Graphs. *PVLDB*, 3(1):997–1008, 2010.

[25] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[26] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link Discovery in Graphs Derived from Biological Databases. In *DILS*, 2006.

[27] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, 2010.

[28] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. on Computing*, 8(3):410–421, 1979.

[29] E. W. Weisstein. Binomial distribution. From MathWorld—A Wolfram Web Resource. Last visited on 16/5/2013, `http://mathworld.wolfram.com/BinomialDistribution.html`.

[30] S. Wuchty and E. Almaas. Peeling the yeast protein network. *Proteomics*, 5(2):444–449, Feb. 2005.

[31] Y. Yuan, G. Wang, L. Chen, and H. Wang. Efficient Subgraph Similarity Search on Large Probabilistic Graph Databases. *PVLDB*, 5(9):800–811, 2012.

[32] H. Zhang, H. Zhao, W. Cai, J. Liu, and W. Zhou. Using the k-core decomposition to analyze the static structure of large-scale software systems. *The Journal of Supercomputing*, 53(2):352–369, 2010.

[33] L. Zou, P. Peng, and D. Zhao. Top-K Possible Shortest Path Query over a Large Uncertain Graph. In *WISE*, 2011.

[34] Z. Zou, H. Gao, and J. Li. Discovering Frequent Subgraphs over Uncertain Graph Databases under Probabilistic Semantics. In *KDD*, 2010.