COMP 6591: Introduction to Knowledge Base System

Instructor's Name: Dr. Nematollaah SHIRI V.

Project Report on

An Implementation of Core Decomposition in Uncertain Graphs

Group Members

Ananta Purohit	26594530
Hazar Snoussi	27794711
Lokesh Paramppurath	27299680

Table of Contents

1.1 ABSTRACT

Core decomposition has proven to be a useful primitive for a wide range of graph analysis. This report presents a work on the computation of core decomposition with deterministic and uncertain graphs, as presented in the paper [1]. The paper presents two algorithms to compute the k-cores in deterministic and uncertain graphs; the former takes linear time for the computation but the latter takes polynomial time and could go more computationally intensive. The motivation behind the paper is to answer the question that "Can the core decomposition for uncertain graphs be performed efficiently?" The algorithm concerned with the deterministic graph is expressed to be more appealing as it can be computed linearly in size of the input graph. So, it is extended for the core decomposition in uncertain graphs and finely observed w.r.t the run time complexity and the behavior of the algorithm as the size of the graph increases.

1.2 INTRODUCTION

Extraction of highly connected parts of complex networks or communities and finding a relationship between these substructures is an issue of topical interest in the network research.

Decomposition helps to describe complex topologies of real world networks. The k-core decomposition is a well-established metric which partitions a graph into layers from external to more central vertices.

To perform many graph analysis tasks, the principle is to find dense subgraphs. 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 it can be computed in linear time and it is related to other definitions of dense subgraph.

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.

Analysing the graph structure has been highly beneficial in many applications such as targeted advertising, fraud detection, missing link prediction and one of the most important tasks is detecting communities in graph nodes having strong ties with each other.

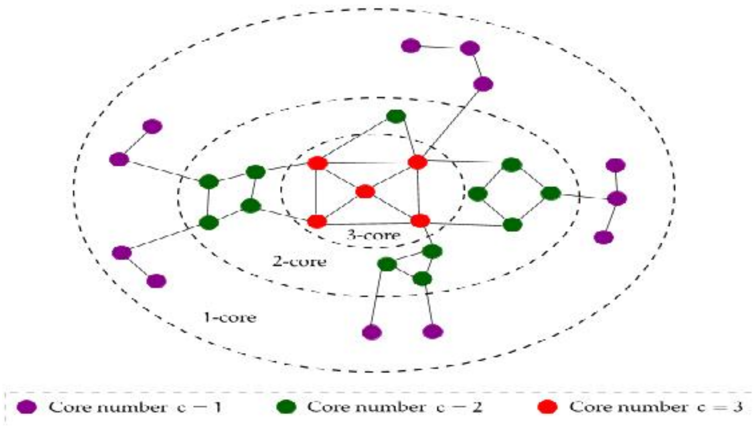


Image-(i)K-cores of a given graph

The graph above clarifies the definition of k-cores by representing the inner shell where every vertex is connected to three other vertices, expressed as k=3, similarly, the middle shell has each vertex connected to two other vertices representing k=2 and the outer shell represents k=1, where every vertex is connected to just one other vertex.

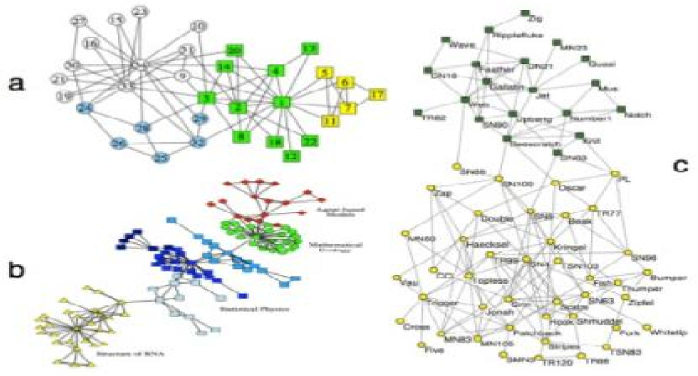


Image-(ii) Different communities in real world network

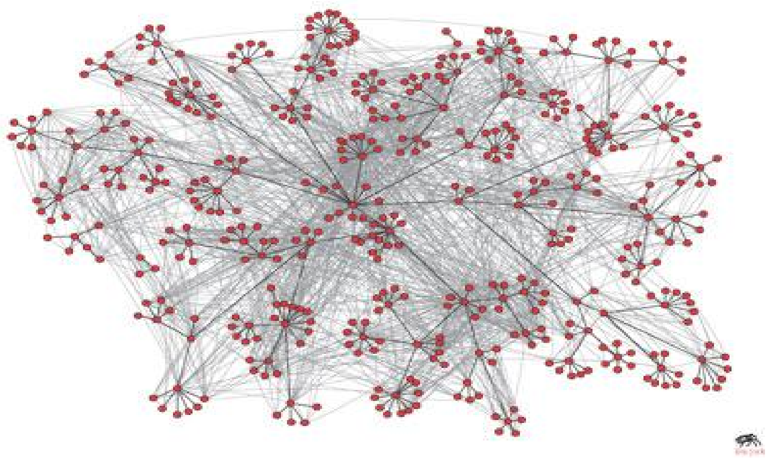
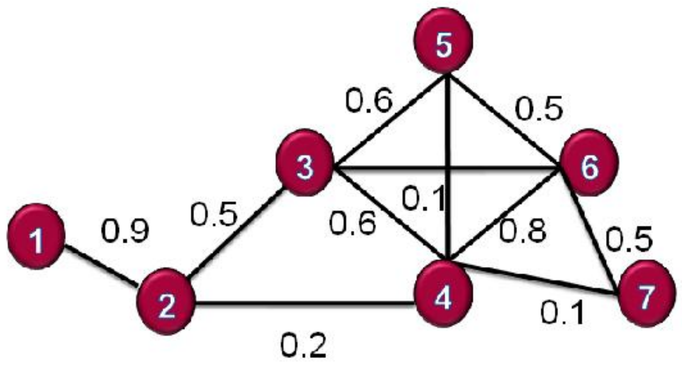


Image-(iii) frequency of e-mails exchanged in a company

The Images (ii) and (iii) shows communities in real world network and frequency of e-mails exchanged between the employees of a software company respectively. Understanding the notion of k-cores deeply, one can easily analyze the above graphs and their substructures and present a relationship among them.

There are networks where the data is uncertain, imprecise or might have an existence of probabilistic trust between the nodes i.e. Uncertain Graphs. The graphs, whose edges are assigned a probability of existence, arise in several emerging applications, for instance, in biological networks and protein-interaction networks, in which 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. In the social networks, uncertainty arises for various reasons, such as (i) Edge probabilities may represent the outcome of link prediction task or (ii) Influence of one person on another, like in viral marketing or (iii) Uncertainty can also be intentionally injected for privacy purposes. Incorporating the notion of k-cores in uncertain graphs, provide a better understanding of interactions among nodes of complex graphs.



Graph 1- As depicted in the paper [1]

The graph above shows the seven nodes with their probability of existence. Other direct applications of core decomposition of uncertain graphs include influence maximization and task-driven team formation [1].

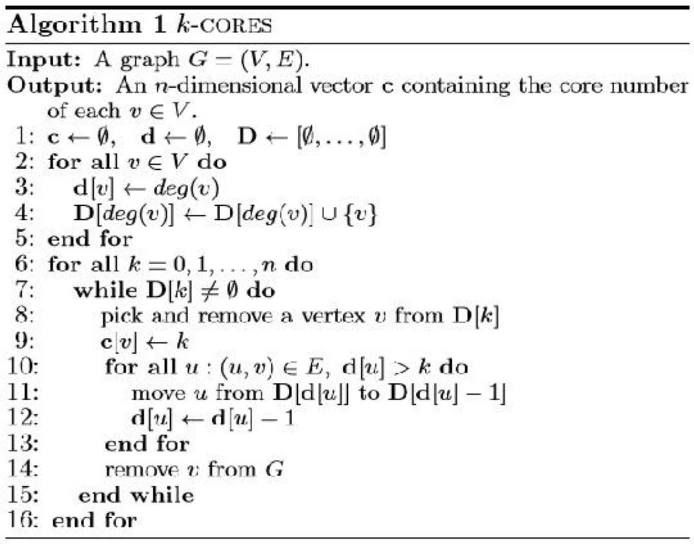
The paper [1] studies the problem of core decomposition of uncertain graphs. It introduced the notion of (k,η)- core as a maximal subgraph whose vertices have at least k neighbors in that subgraph with probability no less than η; here η ∈ [0, 1] is a threshold defining the desired level of certainty 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. The paper [1] presents an algorithm for finding a (k, η)-core decomposition that iteratively removes the vertex having the smallest η-degree and prove its correctness. The proposed algorithm resembles the traditional algorithm for computing the core decomposition of a deterministic graph; 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. We are presenting the implementation of both deterministic and uncertain graphs for computing the k-cores and shown our observations in following sections.

1.3 IMPLEMENTATION

This section describes the algorithms, presented in the paper [1], and our implementation. The section consists of two subsections; first presents the core decomposition in deterministic graph, its implementation, and observation, second focuses on core decomposition in uncertain graphs along with the observations and implementation details.

A) Cores of deterministic graphs



Algorithm 1 k-CORES

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

```
$\mathbf{c} \leftarrow \emptyset$,   $\mathbf{d} \leftarrow \emptyset$,   $\mathbf{D} \leftarrow [\emptyset, \ldots, \emptyset]$
for all $v \in V$ do
    $\mathbf{d}[v] \leftarrow deg(v)$
    $\mathbf{D}[deg(v)] \leftarrow \mathbf{D}[deg(v)] \cup \{v\}$
end for
for all $k = 0, 1, \ldots, n$ do
    while $\mathbf{D}[k] \neq \emptyset$ do
        pick and remove a vertex $v$ from $\mathbf{D}[k]$
        $\mathbf{c}[v] \leftarrow k$
        for all $u : (u, v) \in E$, $\mathbf{d}[u] > k$ do
            move $u$ from $\mathbf{D}[\mathbf{d}[u]]$ to $\mathbf{D}[\mathbf{d}[u] - 1]$
            $\mathbf{d}[u] \leftarrow \mathbf{d}[u] - 1$
        end for
        remove $v$ from $G$
    end while
end for
```

Let G = (V, E) be an undirected graph, where V is a set of n vertices and E ⊆ V × V is a set of m edges. For every vertex v ∈ V, let deg(v) and degH(v) denote the degree of v in G and in a subgraph H of G, respectively. Also, given a set of vertices C ⊆ V, let E|C denote the subset of edges induced by C, i.e., E|C = {(u,v) ∈ E | u ∈ C,v ∈ C}.

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) ≥ k.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 O(n) and O(1) time, respectively. The idea is to employ an n-dimensional vector D whose single cells D[i] store all vertices having a degree equal to “i” in the current graph. The overall time complexity of the algorithm is hence O (n+m), where n is the number of vertices and m is the number of edges.

Another paper by Batagelj and Zaversnik [2] shows how to compute the core decomposition of a graph G in linear time.

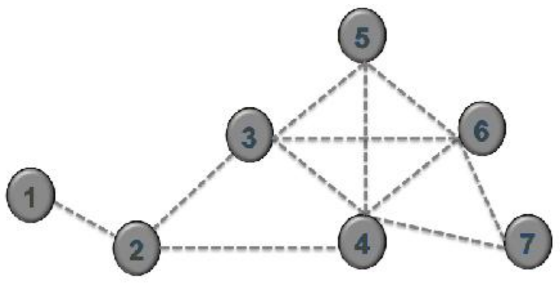
We used a similar structure as depicted in the paper [1], and used java as our programming language. To implement and understand the algorithm more clearly and to make it user interactive, we used an external library JGraphT.

We initiated the implementation by taking a small graph with 7 vertices and 11 edges, as an input, as depicted in the paper [1]. The image below shows the result of computation of k-cores for a given graph. It expresses that vertex 1 has core=1, vertex 2 has core=2, vertex 3 has core=3, vertex 4 has core=3, vertex 5 has core=3, vertex 6 has core=3 and vertex 7 has core=2. The run time observed was 10ms. The program was executed on the machine with the following configuration.

Processor:	Intel® Core(TM) i5-2410M CPU @ 2.30GHz
RAM:	4.00 GB
System Type	(Windows) 64- bit Operating System

The optimized version of the algorithm is presented in the paper [2], executed the algorithm on the machine with the below configuration.

Processor:	Intel® i7, CPU @ 2.20 GHz
RAM:	8.00 GB
System Type:	Ubuntu 14.03 (Linux).
The hard disk:	Seagate Barracuda ST31000524AS 1TB 7200 RPM.

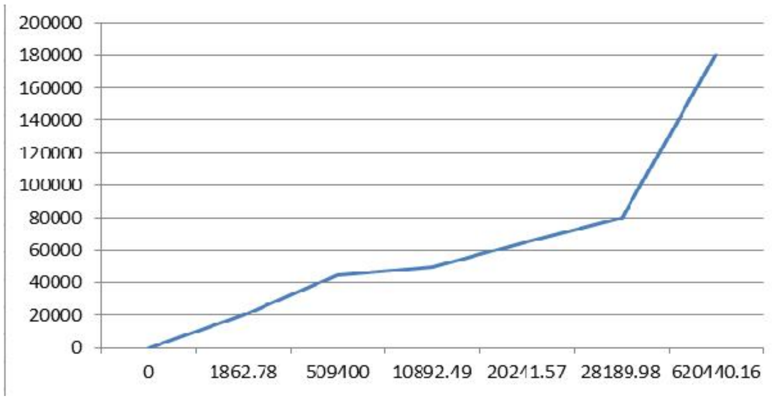


c[1]	c[2]	c[3]	c[4]	c[5]	c[6]	c[7]
1	2	3	3	3	3	2

Image-(iv) Showing the k-cores of a given graph

To observe the behavior of the algorithm with a large input, we increased the number of vertices and come up with following observation.

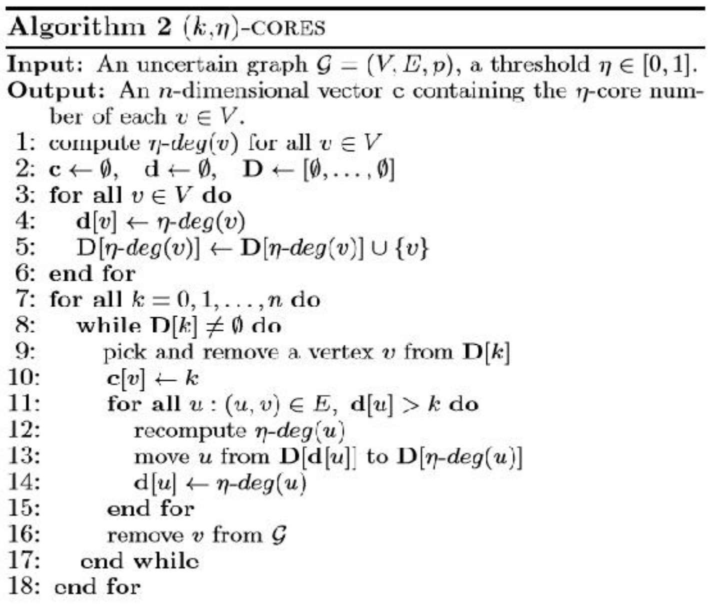
Observation



The above graph depicts y-axis, as number of nodes and x-axis, as run time in ms.If we compare it with the author's result; it is different, because of the machine and programming language limitation. To focus on the behavior of run time, we can see that after extending the number of vertices from 20,000 to 40,000 the run-time suddenly changes from 1.8 seconds to 8.49 minutes. We can improve the linearly of the graph by following the optimized version of the algorithm along with the similar or much improved machine configuration.

We concluded that the algorithm will be executed in linear time, with the size of the input graph.

B) Core decomposition in uncertain graphs



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 $\eta$-$deg(v)$ for all $v \in V$
2: $\mathbf{c} \leftarrow \emptyset$,   $\mathbf{d} \leftarrow \emptyset$,   $\mathbf{D} \leftarrow [\emptyset, \dots, \emptyset]$
3: for all $v \in V$ do
4:    $\mathbf{d}[v] \leftarrow \eta$-$deg(v)$
5:    $\mathbf{D}[\eta$-$deg(v)] \leftarrow \mathbf{D}[\eta$-$deg(v)] \cup \{v\}$
6: end for
7: for all $k = 0, 1, \dots, n$ do
8:    while $\mathbf{D}[k] \neq \emptyset$ do
9:       pick and remove a vertex $v$ from $\mathbf{D}[k]$
10:      $\mathbf{c}[v] \leftarrow k$
11:      for all $u : (u, v) \in E$,  $\mathbf{d}[u] > k$ do
12:         recompute $\eta$-$deg(u)$
13:         move $u$ from $\mathbf{D}[\mathbf{d}[u]]$ to $\mathbf{D}[\eta$-$deg(u)]$
14:         $\mathbf{d}[u] \leftarrow \eta$-$deg(u)$
15:      end for
16:      remove $v$ from $\mathcal{G}$
17:   end while
18: end for
```

Let G = (V,E,p) be an uncertain graph, where p : E → (0,1] is a function that assigns a probability of existence to each edge. For every vertex v ∈ V, let N (v) = {(u,v) ∈ E} denotes the set of edges incident to v, and d(v) = |N(v)| its size. To understand the notion of core decomposition of an uncertain graph, consider the following definition for Probabilistic (k,η)-cores

Given an uncertain graph G = (V,E,p), and a threshold η ∈ [0,1], the probabilistic (k,η)-core of G is a maximal subgraph H = (C,E|C,p) such that the probability that each vertex v ∈ C has degree no less than k in H is greater than or equal to η, i.e., ∀v ∈ C : Pr[degH(v) ≥ k] ≥ η.

The algorithm shown above follows the same scheme as in the deterministic case with the main difference of the use of the η-degree.

The η-degree of a vertex v is computed by using following equation

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

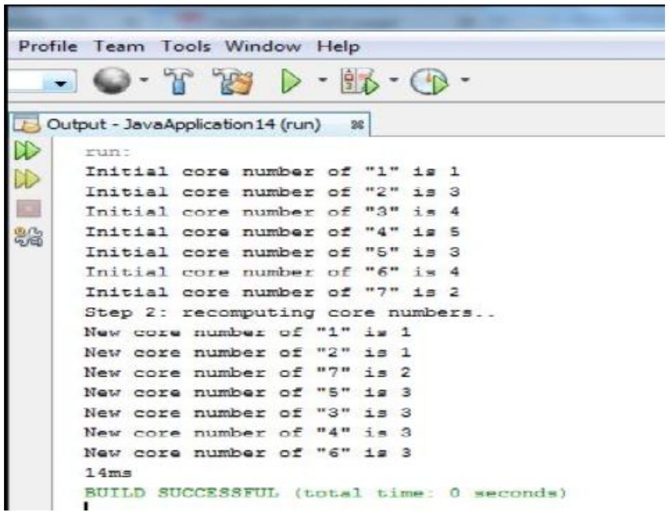
where, the probability Pr[deg(v) ≥ k] is expressed as:

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

The notion of η-degree gives an idea of the degree of a vertex given a specific threshold η.

The paper [1] presents all the necessary equations required to do the computations. Referring to one of those, each individual Pr[deg(v) = i] is computed considering all the subset of edges of and summing over the probabilities. This causes the evaluation to be performed with exponential time complexity. To overcome this intensive computation the paper proposes dynamic programming, Speed up techniques and enhanced version of the algorithm i.e. E(k,n)-core decomposition, and are presented in the Experimental analysis.

We implemented the algorithm on the same input as we used in the deterministic case. The output is shown below.

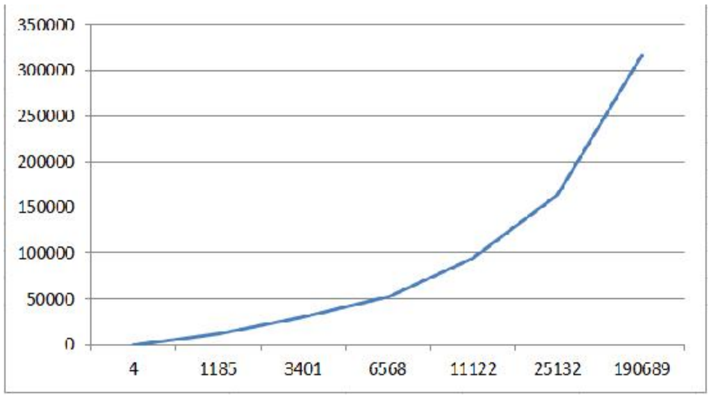

Profile Team Tools Window Help
Output - JavaApplication14 (run)

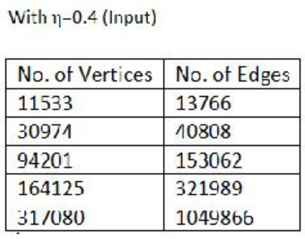
```
run:
Initial core number of "1" is 1
Initial core number of "2" is 3
Initial core number of "3" is 4
Initial core number of "4" is 5
Initial core number of "5" is 3
Initial core number of "6" is 4
Initial core number of "7" is 2
Step 2: recomputing core numbers..
New core number of "1" is 1
New core number of "2" is 1
New core number of "7" is 2
New core number of "5" is 3
New core number of "3" is 3
New core number of "4" is 3
New core number of "6" is 3
14ms
BUILD SUCCESSFUL (total time: 0 seconds)
```

The execution took 14 ms.

Our main concern is to observe the behaviour of the algorithm for large graphs. So, we used the dataset from [7] and executed the core decomposition program as shown in the graph below.

Observation




With η-0.4 (Input)

No. of Vertices	No. of Edges
11533	13766
30974	40808
94201	153062
164125	321989
317080	1049866

As shown in the graph above, y-axis holding the number of nodes and x-axis is showing the run time (in ms). The table above shows the number of edges associated with each set of vertices. In order to concentrate on the number of nodes which suddenly raise the run time of the algorithm, we can evaluate that as the graph is extended from 164125 nodes/vertices and 321989 edges to 317080 nodes and 1049866 edges, the run time suddenly rises from 25.13 seconds to 3.17 minutes. If we compare the result with the experiments performed by the authors, the difference is natural as the observations with large dataset were performed on Linux machine and moreover, we did not follow the same data structure as it was difficult to implement. We could improve the run time by performing speed up techniques. According to current results we can conclude that the algorithm is performed in polynomial time but could take more computation with large graphs, especially, when the number of edges is too large (as shown in the graph), as the probability is associated with the edge and the algorithm needs to re-compute η-degree in step 12.

1.4 EXPERIMENT

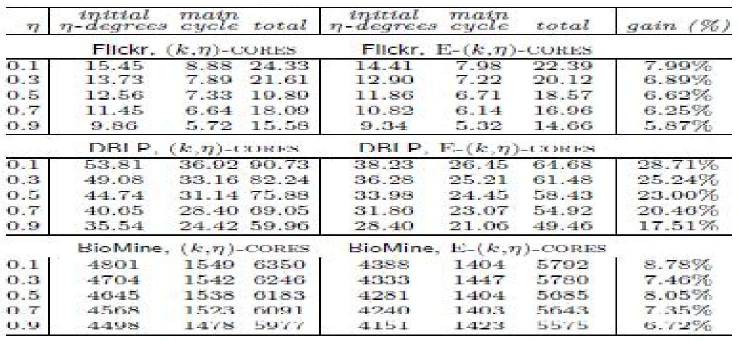
The authors presented quantitive experiments on efficiency and numerical stability of the (k,n)-cores and E(k,n)-core decomposition based on a real-world uncertain graph.

The dataset used are:

Flickr (www.flickr.com, |V | = 24 125, |E| = 300 836)

DBLP (www.informatik.uni-trier.de/~ley/db/, |V | =684 911, |E| = 2 284 991).

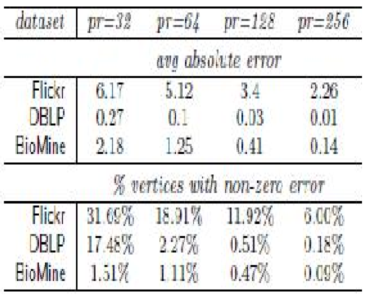
BioMine (biomine.org, |V | = 1 008 200, |E| = 6 742 939).



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	40.05	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%

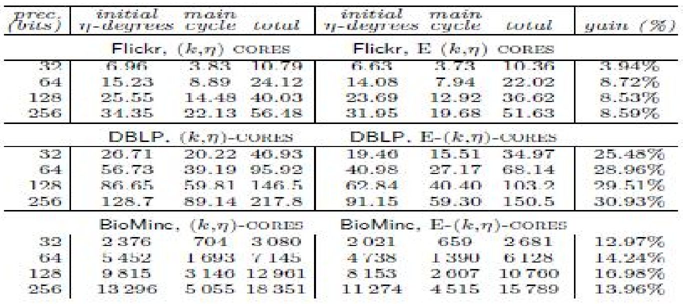
Figure 1 Times (in seconds) of the proposed methods for computing (k, η)-core decomposition (precision 64 bits).

- The column gain (%) depicts the gain of the E-(k,η)-cores algorithm over the (k, η)-cores algorithm.
- Both algorithms took an average of 20 and 60 seconds respectively on Flickr and DBLP.
- On BioMine the average increases to up an hour which is reasonable for networks of such size.
- The E-(k, η) algorithm runs faster than the basic (k,η)-cores algorithm with a more evident gain on a large dataset.



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%

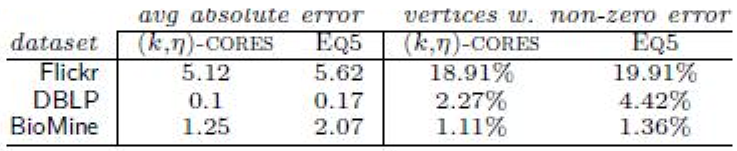
Figure 2 Accuracy of (k, η)-core index for η=0 wrt for different values of precision (bits)



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	31.05	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	650	2 681	12.97%
64	5 452	1 693	7 145	4 738	1 390	6 128	14.24%
128	9 815	3 140	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%

Figure 3 Times (secs) of the two proposed methods of computing (k, η)-core decomposition

- The representation of the probabilities may lead to numerical instability.
- Enlarging the precision via increasing the number of bits of the numerical representation (of the probabilities) could slow down the overall computation due to slowing arithmetic computation.
- For η=0 the (k, η) core decomposition of an uncertain graph G corresponds to the k-core decomposition of the deterministic graph.(by ignoring probabilities)
- By increasing the precision level of representing the probabilities, the average absolute error and the efficiency decrease.
- The results show a linear trend: by doubling the precision, the time doubles while errors get halved.



	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%

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

NB. (k, η)-core corresponds to (k, η)-core with the dynamic programming method

EQ5 is the combination of (k, η)-core and the equation 5

- (k, η)-core outperforms EQ5 in terms of both average absolute error and percentage of vertices with non-zero errors.
- The average absolute error of EQ5 is reduced by 9% in Flickr, 41% in DBLP and 40% in BioMine.

1.5 SUMMARY

The paper [1] helped us in understanding the notion of k-cores in uncertain graph and the various applications associated with. The most challenging part in the computation of k-core in uncertain graph was to understand and implement the η-degree function; we tried our best to follow it completely as presented in the paper [1]. Although, our implementation could be improved by adopting the speed up techniques as mentioned in the paper [1], but due to time constraint and the complexity of adapting these techniques, we could not perform them. However, we mentioned in our implementation results about the differences of our resultant graphs and author's work; we observed that the use of the JGraphT library and usage of loops might take the execution bit longer. In order to stabilize the implementation, we tried to adopt many data types but ended up with using float (with one decimal) as it appeared to be more suitable in this perspective.

REFERENCES

[1] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1316–1325. ACM, 2014.

[2] W. Khaouid, M. Barsky, S. Venkatesh, and A. Thomo. K-core decomposition of large networks on a single PC. PVLDB, 9(1):13–23, 2015

[3] D. Eas1ey and J. Kleinberg, Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press, 2010.

[4] http://jgrapht.org/

[5] https://docs.oracle.com/javase/8/docs/api/

[6] http://delab.csd.auth.gr/~apostol/tutorial-edbt-2016-summary.pdf

[7] https://snap.stanford.edu/data/com-DBLP.html