The Core Decomposition of Networks: Theory, Algorithms and Applications

Fragkiskos Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, Michalis Vazirgiannis

The Core Decomposition of Networks: Theory, Algorithms and Applications

Fragkiskos D. Malliaros*, Christos Giatsidis†, Apostolos N. Papadopoulos‡, and Michalis Vazirgiannis§

Abstract

The *core decomposition* of networks has attracted significant attention due to its numerous applications in real-life problems. Simply stated, the core decomposition of a network (graph) assigns to each graph node v, an integer number $c(v)$ (the core number), capturing how well v is connected with respect to its neighbors. This concept is strongly related to the concept of *graph degeneracy*, which has a long history in Graph Theory. Although the core decomposition concept is extremely simple, there is an enormous interest in the topic from diverse application domains, mainly because it can be used to analyze a network in a simple and concise manner by quantifying the significance of graph nodes. Therefore, there exists a respectable number of research works that either propose efficient algorithmic techniques under different settings and graph types or apply the concept to another problem or scientific area. Based on this large interest in the topic, in this survey, we perform an in-depth discussion of core decomposition, focusing mainly on: *i*) the basic theory and fundamental concepts, *ii*) the algorithmic techniques proposed for computing it efficiently under different settings, and *iii*) the applications that can benefit significantly from it.

Keywords: Core decomposition, graph mining, graph degeneracy, graph theory, algorithms

*Center for Visual Computing, CentraleSupélec, University of Paris-Saclay and Inria Saclay, France. Email: fragkiskos.malliaros@centralesupelec.fr

†Computer Science Laboratory, École Polytechnique, France. Email: xristosakamad@gmail.com

‡School of Informatics, Aristotle University of Thessaloniki, Greece. Email: papadopo@csd.auth.gr

§Computer Science Laboratory, École Polytechnique, France. Email: mvazirg@lix.polytechnique.fr

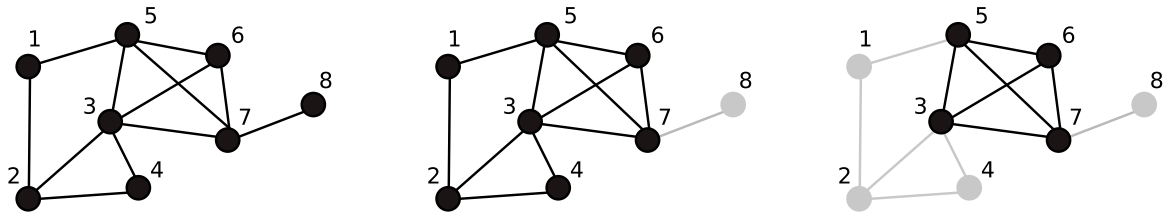
Contents

1 Introduction

Graph management and *graph mining* are two important research areas with a plethora of significant practical applications [3, 34]. The main reason for this is the fact that graphs are ubiquitous and, therefore, their efficient management and mining is necessary to guarantee fast and meaningful knowledge discovery. Research on graph processing and mining was boosted by the need to explore and analyze massive graphs in big data analytics tasks, aiming at computational efficiency and high scalability.

A *network* or *graph* (we will use the terms interchangeably) is denoted by $G(V, E)$, where V is the set of nodes or vertices and E is the set of edges or links. We will follow the trend in the literature and use the symbol n for the number of nodes ($n = |V|$) and the symbol m for the number of edges ($m = |E|$). The number of neighbors of a node $u \in V$ plays a central role in general, and it will be denoted by $deg(u)$.

Figure 1(a) presents a simple graph $G(V, E)$ with $n = 8$ nodes and $m = 12$ edges. Based on the degree definition, $deg(v_1) = 2$, $deg(v_4) = 2$ whereas node v_3 has the highest degree $deg(v_3) = 5$. Therefore, node v_8 has the smallest degree and node v_3 the highest.



(a) 1-core (b) 2-core (c) 3-core

Figure 1: A small graph G with $n = 8$ nodes and $m = 12$ edges and the corresponding 1-core, 2-core and 3-core of G.

In many modern applications graphs are first class citizens. For example:

- The Web [75] can be modeled as a graph, where nodes represent web pages and edges represent hyperlinks among them.
- In a social network [2], nodes may represent users and edges may provide information regarding friendship relationships.
- In protein analysis, a protein-protein interaction network (PPI for short) [118] may be represented as a graph, where nodes represent different proteins and edges capture the interaction between proteins during a specific experiment.
- Graphs may also represent interactions among different types of nodes. For example, purchase information may be captured by using a graph, where a set of nodes (e.g., customers) interact with another set of nodes (e.g., products). In this case, a link between a customer and a product captures the fact that this product was purchased by this specific customer.

These are only a few examples of applications that model elements and the interactions among them by using a graph structure. In these cases, it is important to rely on the structural properties of the interactions among the elements in order to discover useful and meaningful patterns.

Exploring and analyzing massive complex networks involves the execution of (usually) computationally intensive tasks, aiming at uncovering the network structure and detecting the presence of useful patterns that could be proven significant. Some important graph mining tasks involve: reachability queries, graph

partitioning, graph clustering, classification of graph nodes, predicting network evolution, discovering dense subgraphs, detecting influential spreaders.

In many cases we are searching for graph nodes that are considered "central" with respect to a specific problem at hand. Therefore, the concept of *node importance* is crucial in network analysis, since it is expected that among the nodes of a massive network, only a small fraction is of *high significance*. Evidently, one should first determine a method to quantify this significance (importance), since this concept is highly related to the application under consideration. For example, if we assume that we are interested in nodes with a large degree (simply because the number of links is important), then evidently, importance is directly measured by counting the number of neighbors of every node (a concept known as *degree centrality*). As another example, we may define importance in relation to the number of triangles each node participates in; the higher the number of triangles, the higher the node importance. Another popular measure of importance is the total number of shortest paths passing through a specific node (also known as *betweenness centrality* [44]). Also, one can quantify node importance using the concept of *random walks* and applying techniques similar to PageRank [25]. In such a case, the importance of a node is represented by the probability that this node will be visited by a random walker.

Based on the previous discussion, we realize that there is a plethora of different ways to define importance. However, it turns out that the concept of *core decomposition* can be used efficiently and effectively to quantify node importance in many different domains, thus, avoiding the use of more complex and computationally intensive algorithmic techniques. To be precise, the core decomposition of a simple graph G can be computed in linear time with respect to the number of edges of G, if the computation is done in main memory. Simply put, the k-core of a graph G is the maximal induced subgraph G_k, where the number of neighbors of every node u in G_k is at least k. The *core number* of a node u ($c(u)$) is defined as the maximum value of k such that u is contained in G_k. Figure 1 illustrates the 1-core, the 2-core and the 3-core of a toy graph. Based on the definition of the core number, it holds that $c(v_8) = 1$, $c(v_1) = 2$ and $c(v_6) = 3$.

Roadmap. Based on the fact that the core decomposition concept has numerous applications in diverse domains, in this survey we cover the topic as thorough as possible, presenting the *concepts*, the *algorithmic techniques* used and also the fundamental *applications* that base their main results on the core number (or variations) of graph nodes. This survey was inspired by four tutorials presented by the authors [94] in the following conferences: *i*) 7th IEEE/ACM International Conference on Social Network Analysis and Mining (ASONAM) 2015, *ii*) 19th EDBT/ICDT Joint Conference 2016, *iii*) IEEE International Conference on Data Mining (ICDM) 2016 and *iv*) The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD) 2017. To the best of our knowledge, this is the first comprehensive survey of the area. Existing works like [28, 22] cover only specific parts of the problem and they do not provide an in-depth discussion of the algorithms and the associated application domains.

Table 1 presents the timeline of the main research works related to core decomposition that are cited in this survey. We classify the related research contributions into two basic categories: *i*) concepts and algorithmic techniques and *ii*) representative application domains. The rest of this article is organized as follows:

- In Section 2 we present some fundamental concepts in addition to the basic definition of the core decomposition and some of its most widely used extensions for specific graph types. In particular, we will focus on simple, directed, weighted, signed, probabilistic, temporal, multilayer and hidden graphs which are frequently used in modern network-based applications.

- Section 3 offers a thorough overview of some of the the algorithmic techniques required for the core decomposition computation in different settings. In particular, we cover main memory computation, disk-based computation, local core number computation, parallel/distributed core decomposition and core decomposition in probabilistic graphs. Since it is not possible to cover every single algorithm in

Table 1: Core decomposition timeline.

Concepts and Algorithmic Techniques	Year	Representative Application Domains
coloring number [42]	1968	
degeneracy [88]	1970	
width [45]	1982	
	1983	dense subgraphs [128]
linkage [73]	1996	
	2000	web analysis [75]
generalized cores [16]	2002	
main-memory computation [17]	2003	
	2005	network analysis [8], visualization [6]
	2006	complex networks [38], fingerprinting and visualization [7]
	2007	internet topology [27]
neuroscience [62], truss decomposition [32]	2008	internet evolution [158], network analysis [64, 82, 9]
	2009	cell biology [92]
	2010	influential spreaders [74], mobile social networks [85]
D-cores [57], disk-based computation [30], distributed computation [105], anchored k-core [19]	2011	communities [58], influence [26], neuroscience [143]
triangle cores (truss) [146, 159], weighted networks [48]	2012	embeddings [120], dense subgraphs [142]
dynamic cores [124], distributed computation [106]	2013	influential spreaders [160], engagement [96], network analysis [1], neuroscience [130]
MapReduce [113], local estimation [109], uncertain graphs [23], S-cores [54]	2014	clustering [56], neuroscience [122]
temporal graphs [151], disk-based and in-memory [72], parallel computation [129], density-friendly decomposition [136]	2015	keyword extraction [121], community search [86], earthquake networks [63], software engineering [103], cell biology [68, 41], neuroscience [150, 20]
incremental computation [123], disk-based computation [60], distributed computation [12], uncertain truss [67]	2016	cell biology [39, 116], graph mining [131], network analysis [144], neuroscience [80]
hidden cores [134], multilayer graphs [47], parallel computation [148, 69], density-friendly decomposition [36], k-peak decomposition [61], (k, r)-core [157], core unravelling [155]	2017	distributed clustering [31, 98], truss community search [4], engagement [156], ecology [52, 51], criminal networks [99]
distributed computation [66], probabilistic cores [117], radius-bounded cores [147], bipartite peeling [125]	2018	biology [50], ecology [107], metabolic networks [43], patterns and anomaly detection [132], graph similarity [108]

its full extend, we have selected a representative set of algorithmic techniques which we describe in detail and for the rest we offer the corresponding links to related research.

- Next, Section 4 elaborates on diverse application domains that benefit significantly from the core decomposition concept. More specifically, we present the application of the core decomposition concept in domains such as: network analysis, temporal evolution, influence maximization, dense subgraph discovery, community detection, text mining, biological network analysis and neuroscience. Evidently, the list is not exhaustive but we believe that these domains cover the vast majority of research performed in the area.
- Finally, Section 5 concludes this survey and discusses briefly open problems and future research directions.

2 Fundamental Concepts

In this section, we cover the most important concepts related to core decomposition. In particular, we formally present the basic definitions and the most important properties which can be used by applications for performing more complex network analysis tasks. Table 2 illustrates some frequently used symbols and the corresponding interpretations.

Table 2: Frequently used symbols.

Symbol	**Interpretation**
G	a graph
V	set of vertices (or nodes) of G
E	set of edges of G
n	number of nodes ($n = \|V\|$)
m	number of edges ($m = \|E\|$)
u, v	some vertices of G
$N(u)$	set of direct neighbors of vertex u
$N_d(u)$	set of neighbors of u at a distance at most d
$deg(u)$	degree of node u (number of incident edges)
$c(u)$	core number of node u
$\delta^*(G)$	the degeneracy of graph G

2.1 Simple Graphs

Let $G(V, E)$ denote an undirected and unweighted graph, where V is the set of nodes and E is the set of edges. The k-core decomposition of G is a threshold-based hierarchical decomposition of G into nested subgraphs. The basic idea is that a threshold k is set on the degree of each node; nodes that do not satisfy the threshold, are excluded from the process. The following definitions provide some basic knowledge regarding the concepts around core decomposition.

Definition 1 (k-shell subgraph) *The k-shell is the subgraph of G defined by the nodes that belong to the k-core but not to the $(k+1)$-core.*

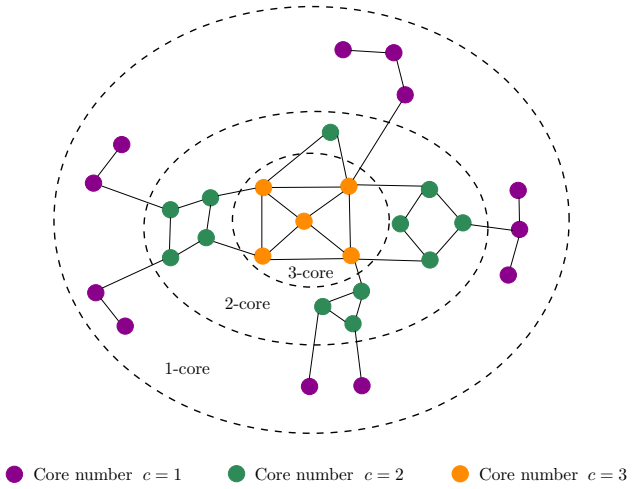


Figure 2: Example of the k-core decomposition.

Definition 2 (k-core subgraph) *Let H be a subgraph of G, i.e., $H \subseteq G$. H is defined to be the k-core of G, denoted by G_k, if it is a maximal subgraph of G in which all nodes have degree at least k.*

Definition 3 (graph degeneracy $\delta^*(G)$) *The degeneracy [88] $\delta^*(G)$ of a graph G is defined as the maximum k for which graph G contains a non-empty k-core subgraph.*

Definition 4 (core number) *A node v has core number $c(v) = k$, if it belongs to a k-core but not to the $(k+1)$-core.*

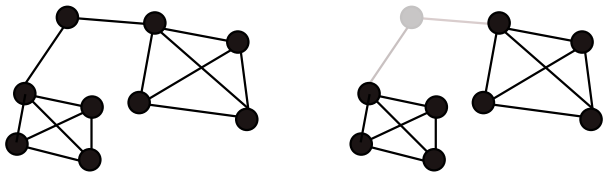
Based on the above definitions, it is evident that if all the nodes of the graph have degree at least one, i.e., $deg(v) \geq 1, \forall v \in V$, then the 1-core subgraph corresponds to the whole graph, i.e., $G_1 \equiv G$. Furthermore, assuming that $G_i,\ i = 0, 1, 2, \ldots, \delta^*(G)$ is the i-core of G, then the k-core subgraphs are nested. Formally:

$$G_0 \supseteq G_1 \supseteq G_2 \supseteq \ldots \supseteq G_{\delta^*(G)} \tag{1}$$

Typically, the subgraph $G_{\delta^*(G)}$ is called *the maximal k-core subgraph* of G. Figure 2 depicts an example of a graph and the corresponding k-core decomposition. As we observe, the degeneracy of this graph is $\delta^*(G) = 3$; therefore, the decomposition creates three nested k-core subgraphs, with the 3-core being the maximal one. An attempt to create a higher order core subgraph (i.e., the 4-core of the graph) would result in an empty subgraph, since the removal of one of the nodes belonging to the 3-core will force the removal of the remaining nodes. The nested structure of the k-core subgraphs is indicated by the dashed lines shown in Figure 2. Furthermore, the color of the nodes indicate the core number $c(u)$ of each node u.

It is important to note that the k-core subgraphs are not necessarily connected. As an example, consider the graph shown in Figure 3(a). The graph is composed of two cliques (complete subgraphs) of size four that are connected by a node x with a degree of 2. Evidently, the graph is a 2-core, since the degree os each node is at least 2. The transition from the 2-core to the 3-core of G will eliminate node x, since $deg(x) = 2$. The remaining nodes constitute the 3-core of the graph, which evidently is disconnected as shown in Figure 3(b).

The concept of degeneracy in graphs, as defined above, is also known as *width* [45] and *linkage* [73]. It is also related to the *coloring number* α of a graph [42], which is defined as the least k for which there is an ordering $\prec$ of the graph nodes, such that for every $v \in V$, the number of adjacent nodes $w \prec v$ is less than α.



(a) 2-core (b) 3-core

Figure 3: Example of a disconnected k-core subgraph. The 2-core is shown in (a). The removal of the node with degree 2 leads to (b) which depicts the 3-core (and also the maximum core of the graph). The 3-core is disconnected.

2.2 Extensions to Other Types of Graphs

The k-core decomposition described above considers that graphs are unweighted and undirected. However, many real-world networks carry rich semantics, as expressed by more complex graph types. To that end, there exist research efforts towards meaningful extensions of the k-core decomposition to other types of graphs. In most of the cases, these extensions pose additional challenges to the efficient computation of the decomposition as well.

Directed Graphs. Directed graphs or digraphs [13] are characterized by rich semantics in comparison to simple graphs, simply because edge direction is important. In a directed graph the degree of a node u may refer to the number of incoming links ($deg_{in}(u)$) or to the number of outgoing links ($deg_{out}(u)$). These are also known as the in-degree and the out-degree respectively.

Giatsidis et al. [59, 57] introduced D-cores, an extension of the k-core structure to *directed graphs*. In this case, the notion of (k, ℓ)-core is used to represent subgraphs in which all nodes have in-degree at least k and out-degree at least ℓ respectively.

Weighted Graphs. A weighted graph is characterized by the existence of weights on the graph edges. Each edge e is associated with a weight $w(e)$ that may represent the cost of the edge, or the strength of the link between the participating nodes, or any other type of quantification, depending on the application. Computing the core decomposition in a weighted graph is significantly harder than the computation in a simple graph, mainly because there is no easily derived bound on the core number of a node. In [58, 48, 40], the authors propose efficient algorithms for computing the core decomposition in weighted graphs.

Signed Graphs. In [54], an extension of the k-core decomposition for *signed networks* was proposed. Signed networks [76, 77, 83] are used to capture the notion of positive and negative interactions among the nodes of a graph (e.g., trust/distrust, friend/foe relationships). Examples of such networks include the trust networks that can be produced by product review websites like Epinions[1] and the voting election network between the administrators of Wikipedia[2].

Dynamic Graphs. A *dynamic graph* is characterized by changes performed on the set of nodes and/or the set of edges. These changes may correspond to insertion or deletions of edges. Changes in the graph may have an impact on the core numbers of nodes. In the worst case, an insertion of a single edge may change all core numbers of nodes. On the other extreme, the insertion may cause no changes at all. The typical case

[1] www.epinions.com
[2] www.wikipedia.org

is that an edge insertion (or deletion) will have an impact on some core numbers. Therefore, the challenge is to be able to monitor the core numbers of all nodes by applying only a few computations, avoiding the re-computation of the core decomposition.

In Section 3 we will discuss in detail an algorithm for monitoring the core decomposition in a dynamic graph reported in [124], where insertions and deletions of edges may be applied at any time. The main characteristic of the algorithm is that it detects the minimal set of nodes that must be checked for changes in the core numbers, thus reducing the overall processing cost significantly.

Temporal Graphs. A *temporal graph* is a special case of a dynamic graph. Two nodes u and v may me connected by an edge at multiple time instances or intervals. We may assume that each edge is annotated with a timestamp, denoting the time instance of the specific interaction. Also, two nodes may be linked for a specific time interval $[t_i, t_j]$ defined by two time instances t_i, t_j, where $t_i \leq t_j$.

In [151], the definition of the core decomposition is adapted to the case of temporal graphs. The concept of (k, h)-core is defined, where as usually k represents the degree of a node and h represents the number of multiple temporal edges between two vertices. Given a temporal graph G, the (k, h)-core of G is the largest subgraph $H_{(k,h)}$ of G such that every vertex u in $H_{(k,h)}$ has at least k direct neighbors, and there are at least h temporal edges between u and each one of its neighbors in $H_{(k,h)}$.

Probabilistic Graphs. A special category of graphs, includes graphs that introduce *uncertainty* with respect to the existence of nodes and edges [70]. For example, an edge e between nodes u and v may exist or not. The existence of an edge depends on several factors, mainly on the particular application under consideration. For example, in a social network where an edge corresponds to a message exchange between two users, the message will be sent with some probability (i.e., it is not sure that user u will send a message to user v). As another example, consider a protein-protein interaction network, where each node corresponds to a protein and each edge denotes interactions among proteins. In this case, we may realize that proteins u and v interact in 70% of the cases, which means that the edge $e_{u,v}$ will be present in the graph with a probability of 0.7. Also, uncertainty may be introduced on purpose for privacy reasons.

Computing the core decomposition of an uncertain graph is not trivial. One approach could be to transform the uncertain graph into a weighted graph, where the weight $w(e)$ of an edge e is inversely proportional to the existential probability $p(e)$ of the edge. However, this simplistic approach has severe drawbacks, since the meaning of the probability is lost during this transformation and it does not give any insight regarding the importance of the computed cores. To attack the problem, Bonchi et al. [23] proposed a core decomposition methodology for uncertain graphs. The problem was also studied later in [117]. The algorithm of [23] will be covered in detail in the following section that covers the major algorithmic techniques.

Multilayer Graphs. Usually, we assume that graph nodes are of the same type and also graph edges represent the same relationship among nodes. However, in many cases this simple view of the network may not represent reality. People interact in many different ways. Fore example, two persons may be friends in real life, but also may be friends in a social network, may collaborate in a research project or may work in the same company. These are different relationship types that may be present.

In its simplest form, a *multilayer graph* (a.k.a. *multidimensional graph*) $G(V, \mathcal{E})$ is composed of a set of nodes V and a set of edge subsets $\mathcal{E} = E_1 \cup E_2 \cup ... \cup E_l$, where l is the total number of layers and E_j contains the set of edges present in the j-th layer of G. The first algorithm for computing the core decomposition of a multilayer graph is reported in [47]. The authors not only provide a novel definition for the core numbers on a multilayer graph but also show that this definition has some nice properties regrading the density of the k-core subgraphs which are in sync with the core decomposition concept in simple graphs.

Hidden Graphs. Conventional graphs are characterized by the fact that both the set of vertices V and the set of edges E are known in advance, and are organized in such a way to enable efficient execution of basic tasks. Usually, the adjacency lists representation is being used, which is a good compromise between space requirements and computational efficiency. However, a concept that recently has started to gain significant interest is that of *hidden graphs* [11]. In contrast to conventional graphs, a hidden graph is defined as $G(V, f())$, where V is the set of vertices and $f()$ is a function $V \times V \rightarrow \{0, 1\}$ which takes as an input two vertex identifiers and returns true or false if the edge exists or not respectively. Therefore, in a hidden graph the edge set E is not given explicitly and it is inferred by using the function $f()$.

Hidden graphs constitute an interesting tool and an promising alternative to conventional graphs, since there is no need to represent the edges explicitly. This enables the analysis of different graph types that are implicitly produced by changing the function $f()$. Note that the total number of possible graphs that can be produced for the same set of vertices equals $2^{\binom{n}{2}}$, where $n = |V|$ is the number of vertices. It is evident, that the materialization of all possible graphs is not an option, especially when n is large. Therefore, hidden graphs is a tempting alternative to model relationships among a set of entities. On the other hand, there are significant challenges posed, since the existence of an edge must be verified by evaluating the function $f()$, which is costly in general.

Motivated by recent developments in the area [135, 153] for detecting the top-k nodes with the highest degrees in bipartite graphs, in [134], an algorithm is proposed to discover if a hidden graph contains a k-core subgraph or not, by applying as few edge probing queries as possible.

2.3 Generalized Core Decomposition

The k-core decomposition was initially introduced for the degree property of the nodes in a graph. A natural inquiry would be "why do we focus on node degrees?". An obvious answer to this question would be that the degree of a node is relatively easily computed and it is a very simple concept. However, the degree is not the only node property that could be applied in this framework. Batagelj and Zaveršnik proposed the notion of *generalized cores* [16], which extends cores from degree to other node properties. In fact, any node property can be used potentially to define a different kind of core decomposition, where the concept of the core is associated with the node property under consideration.

Definition 5 (Generalized Cores or p-cores) *Let $G = (V, E)$ be a graph and let $w : E \rightarrow \mathbb{R}$ be a function assigning values (or weights) to the edges of the graph. A node property function $p()$ that assigns real values on graph G, is defined as $p(v, C)$, where $v \in V$ and $C \subseteq V$. Then, a subgraph $H = (V_H, E_H)$ induced by the set $V_H \subseteq V$ is called a p-core at level $t \in \mathbb{R}$ if and only if (i) $\forall v \in V_H : t \leq p(v, V_H)$ and (ii) V_H is a maximal set.*

Recall that, a function $p()$ is called *monotone* if and only if the following property holds:

$$C_1 \subseteq C_2 \Rightarrow p(v, C_1) \leq p(v, C_2), \forall v \in V. \tag{2}$$

In [16] it was shown that for a monotone function p, the p-core at level t of the decomposition can be found be successively removing nodes with value of p less than t – as has been already described for the k-core decomposition. Furthermore, the subgraphs corresponding to the cores are nested, i.e., $t_1 < t_2 \Rightarrow H_{t_2} \subseteq H_{t_2}$. In fact, if we consider that function p corresponds to the degree of a node, i.e., $p(v, C) = d_v^C$, where d_v^C is the degree of node v in subgraph C, this function is monotone. Also, many other functions on the nodes v of the graph including the in-degree and out-degree, the weighted degree (i.e., sum of weights of the adjacent edges) and the number of cycles of length l that pass through node v, have been proven to be monotone; thus, the same procedure can be used to extract the corresponding p-cores.

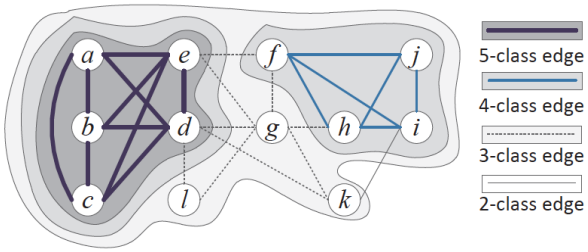


Figure 4: Example of a graph and its K-classes, $2 \leq K \leq 5$. The figure is courtesy of Wang and Cheng [146]. ©2012 VLDB Endowment.

2.4 Extensions of the Core Decomposition

In this section, we describe various extensions of the core decomposition, covering among others the notion of truss decomposition – a particular type of generalized cores based on the property of triangles.

2.4.1 Truss Decomposition

The *K-truss decomposition* extends the notion of k-core using triangles, i.e., cycle subgraphs of length three [32, 146, 159]. Let $G(V, E)$ be an undirected graph. We define as a *triangle* $\triangle_{uvw}$ a cycle subgraph of nodes $u, v, w \in V$. Additionally, the set of triangles of G is denoted by $\triangle_G$. The *support* of an edge $e = (u, v) \in E$ is defined as $sup(e, G) = |\{\triangle_{uvw} : \triangle_{uvw} \in \triangle_G\}|$ and expresses the number of triangles that contain edge e.

Given an undirected graph G, the K-truss, $K \geq 2$, denoted by $T_K = (V_{T_K}, E_{T_K})$, is defined as the largest subgraph of G, where every edge is contained in at least $K - 2$ triangles within the subgraph, i.e., $\forall e \in E_{T_K}, sup(e, T_K) \geq K - 2$. Based on that, the *truss number* of an edge $e \in G$ is defined as $t_{edge}(e) = \max\{K : e \in E_{T_K}\}$. Thus, if $t_{edge}(e) = K$, then $e \in E_{T_K}$ but $e \notin E_{T_{K+1}}$. We use $K_{\max}$ to denote the maximum truss number of any edge $e \in E$. The *K-class* of a graph $G = (V, E)$ is defined as $\Phi_K = \{e : e \in E, t_{edge}(e) = K\}$. Figure 4 shows an example graph and its K-classes.

Based on the above definitions, we can now introduce the concept of K-truss decomposition.

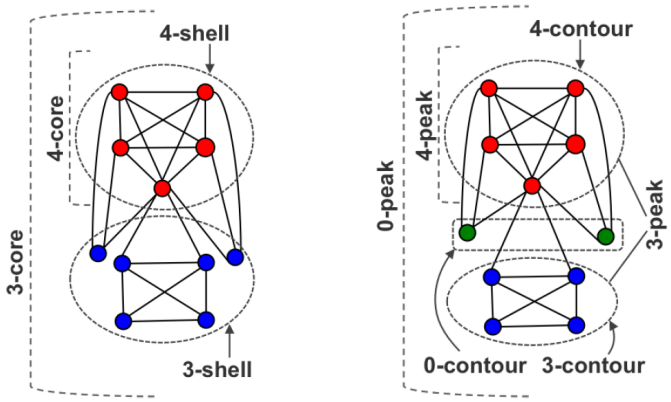
Definition 6 (K-truss decomposition) *Given a graph $G = (V, E)$, the K-truss decomposition is defined as the task of finding the K-truss subgraphs of G, for all $2 \leq K \leq K_{\max}$. That is, the K-truss can be obtained by the union of all edges that have truss number at least K, i.e., $E_{T_K} = \bigcup_{j \geq K} \Phi_j$.*

Since the K-truss decomposition is defined based on the number of triangles – a more "strict" criterion compared to the one of degree – it can intuitively be considered as the "core" of the k-core subgraph.

Lastly, we mention here that the concept of K-truss decomposition has recently been extended to the case of probabilistic (or uncertain) graphs [67, 162].

2.4.2 Density-friendly Core Decomposition

One of the drawbacks of the k-core decomposition is that the nested k-core subgraphs do not satisfy a natural density property – simply defined as the ratio between the number of edges and nodes of the subgraph. In other words, the maximal k-core subgraph is not necessarily the densest subgraph of the graph. Based on this observation, Tatti and Gionis [136] introduced the concept of *density-friendly* graph decomposition, where *i*) the density of the inner core subgraphs given by the decomposition is higher than the density of the outer ones, and *ii*) the most inner subgraph will correspond to the densest subgraph. Furthermore, the authors of [136] have shown that the locally-dense decomposition can be computed in polynomial time. Note



(a) k-cores and k-shells (b) k-contours and k-peaks

Figure 5: Example of the k-core and k-peak decomposition. The peak number of a node is at most its core number. The figure is courtesy of Govindan et al. [61]. ©2017 International World Wide Web Conference Committee.

that, more recently, Danisch et al. [36] proposed a scalable algorithm for computing such a decomposition, based on convex programming.

2.4.3 Peak Decomposition

Another drawback of the k-core decomposition defined earlier has to do with the fact that it is computed globally; if the graph contains distinct regions of different densities, the sparser among these regions might be neglected by the decomposition. To deal with this issue, the authors of [61] have proposed the notion of k-peak decomposition, which aims at finding the centers of distinct regions in the graph – viewing the global structure of the graph as a set of regions, each one resembling a mountain with a central peak.

More precisely, given a graph G, the k-contour can be defined as follows.

Definition 7 (k-contour) *Given a graph $G(V, E)$, a k-contour is the induced subgraph over the maximal set of nodes, such that i) the k-contour does not contain nodes from a higher contour (i.e., values higher than k), and ii) each node in the k-contour has at least k links to other nodes in the k-contour.*

Based on that, we can define the *peak number* of a node as the value k such that the node belongs to a k-contour. Then, a *k-peak decomposition* of a graph G is defined as the assignment of each node to exactly one contour. Figure 5 shows an example graph and the corresponding k-core and k-peak decomposition.

Definition 8 (k-peak) *Given a graph G, a k-peak is the induced subgraph of the union of j-contours, $\forall j \leq k$.*

Lastly, as shown in the paper [61], similar to the case of k-core decomposition, the k-peak decomposition is also unique, i.e., each node has a single unique peak number.

2.4.4 Nucleus Decomposition

Another extension of the core decomposition is the *nucleus decomposition* [126]. The basic motivation here comes from the problem of dense subgraph detection, where the goal is to identify structures of dense subgraphs within a large graph and to understand how those structures are related to each other. In particular, we are interested in extracting a global, hierarchical representation of many dense subgraphs.

To this direction, the authors of [126] defined the notion of *nuclei* in a graph: an (r, s)-nucleus, for fixed and small positive integers $r < s$, is defined as a maximal subgraph where every r-clique (i.e., complete graph of r nodes) is part of many s-cliques. Furthermore, nuclei subgraphs that do not contain one another, cannot share an r-clique. Based on that, for various values of r and s ($r < s$), it can be shown that the (r, s)-nuclei form an hierarchical decomposition of the graph – where the density of the nuclei is increasing as we move towards the leaves of the decomposition. In practice, the authors of [126] have observed that the $(3, 4)$-nuclei provide the most interesting decomposition of real-world graphs. Figure 6 depicts an example of the hierarchical structure of $(3, 4)$-nuclei decomposition in a snapshot of the Facebook graph composed by 88K nodes. Each node of the structure corresponds to a $(3, 4)$-nucleus, and the tree edges indicate containment. More generally, an ancestor nucleus contains all descendant nuclei. The figure also shows the scale and densities of the various nuclei subgraphs.

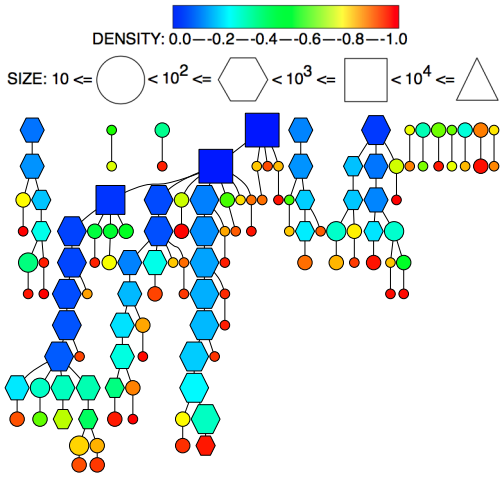


Figure 6: $(3, 4)$-nuclei subgraphs for a snapshot of the Facebook graph [126]. Branching depicts the different regions in the graph. The figure is courtesy of Sarıyüce et al. [126]. ©2015 International World Wide Web Conference Committee (IW3C2).

2.4.5 Peeling Bipartite Networks

A graph $G(V_h, V_a, E_b)$ is called *bipartite* if the node set V can be partitioned into two disjoint sets V_h and V_a, where $V = V_h \cup V_a$, such that every edge $e \in E_b$ connects a node of V_h to a node of V_a, i.e., $e = (i, j) \in E \Rightarrow i \in V_h$ and $j \in V_a$. In other words, there are no edges between nodes of the same partition. A common approach to analyze bipartite networks is to project them into weighted or unweighted unipartite ones. Nevertheless, this simplistic approach has several drawbacks – with the major one being the fact that, a node with degree d in the original bipartite network will result in a d-clique in the projected one. Thus, hub nodes will dominate the maximal k-core subgraph produced by the core decomposition on the projected unipartite network.

Motivated by the task of dense subgraph detection in bipartite networks, Sarıyüce and Pinar [125] introduced the concept of bipartite *graph peeling* to detect dense subgraphs and the relationships among them. The main idea is to rely on *higher-order structural motifs* [18] that are able to capture cohesiveness in bipartite graphs. More precisely, the authors of [125] have used a particular motif, called *butterfly* subgraph, which corresponds to a $(2, 2)$-biclique (bipartite clique with 2 nodes at each partition) – the overall goal is to discover bipartite subgraphs including many butterfly structures and to construct relations among them. Then, two types of decomposition can be defined: the *tip decomposition* and the *wing decomposition*.

Definition 9 (k-tip decomposition) *A bipartite subgraph $H(U, V, E) \subset G$ induced on U, is a k-tip if and only if: i) each node $u \in U$ takes part in at least k butterflies; ii) each node pair $(u, v) \in U$ is connected by a series of butterflies; iii) H is maximal, i.e., there is no other k-tip that subsumes H.*

Note that, two nodes $u, w \in U$ are connected by a series of butterfly subgraphs, if there exists a sequence of nodes $u = v_1, v_2, \ldots, v_k = w$ such that some butterfly contains v_i and v_{i+1}, for each i.

Definition 10 (Tip number $\theta(u)$) *The tip number $\theta(u)$ of a node $u \in U$ is the largest value t such that there exists a t-tip that contains u. Then, the tip decomposition of a bipartite graph $G(U, V, E)$ is to find the tip numbers of nodes in U.*

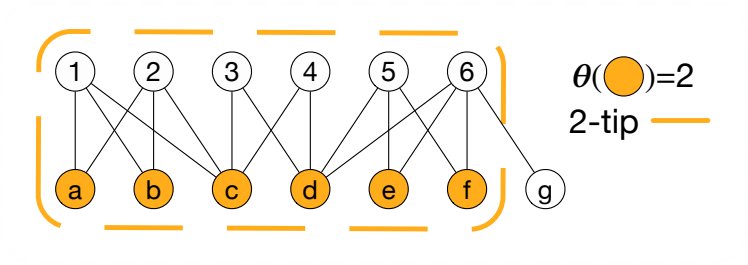


Figure 7: Example of k-tip decomposition. Nodes a, b, e, and f participate at two butterfly subgraphs, while nodes c and d at three. Notice that, nodes c and d cannot have a tip number of 3 since their induced subgraph has just one butterfly. Therefore, nodes a to f form a 2-tip, and their tip number will be $\theta(a - f) = 2$. The figure is courtesy of Sarıyüce and Pinar [125]. ©2018 Association for Computing Machinery.

Figure 7 gives an example graph and its k-tip decomposition. Notice that, the k-tip decomposition does not allow node overlaps, which might be the case in many real-world bipartite networks (e.g., author-paper collaboration networks). To allow node overlaps, the authors of [125] have introduced the concept of *k-wing* decomposition, where the focus is on the edges of the network instead of the nodes.

Definition 11 (k-wing decomposition) *A bipartite subgraph $H(U, V, E) \subset G$ induced on U, is a k-tip if and only if: i) each edge $(u, v) \in E$ participates in at least k butterflies; ii) each edge pair $(u_1, v_1), (u_2, v_2) \in E$ is connected by a series of butterflies; iii) H is maximal, i.e., there is no other k-wing that subsumes H.*

Definition 12 (Wing number $\psi(e)$) *The wing number $\psi(e)$ of an edge $e \in E$ is the largest value s such that there exists a s-wing that contains e. Then, the wing decomposition of a bipartite graph $G(U, V, E)$ is to find the wing numbers of edges in a graph G.*

3 Algorithmic Techniques

In this section, we focus on algorithmic techniques for the computation of the k-core decomposition in different settings. The design of an algorithm for core decomposition depends on many diverse factors such as the type of the graph (simple, directed, signed, weighted, probabilistic), the hardware infrastructure (memory-based, disk-based, parallel, distributed), the type of the output (exact, approximate), just to name a few. Since there is a plethora of different algorithms we are going to discuss the most representative ones.

3.1 In-Memory Computation

The computation of the k-core decomposition of a graph can be performed through a simple process that is based on the following rationale: to extract the k-core subgraph, all nodes with degree less than k and their adjacent edges should be recursively deleted [128]. That way, beginning with $k = 1$, the algorithm removes all the nodes (and the incident edges) with degree equal to or less than k, until no such node remains in the graph. Also notice that, removing edges that are incident to a node may cause reductions to the degree of neighboring nodes; the degree of some nodes may become at most k, and thus, they should also be removed at this step of the algorithm. When all remaining nodes have degree $d_v > k$, k is increased by one and the process is repeated until no more remaining nodes are left in the graph.

A straight-forward implementation of this algorithm requires a priority queue to store the nodes, prioritized by their degree. The removal of a node requires in the worst case the deletion of $n-1$ edges which translates to the execution of $n-1$ decrease-key operations in the priority queue. Since each edge is examined exactly once, the worst case complexity of the naive algorithms becomes $\mathcal{O}(m \cdot \log n)$.

However, the problem can be solved in linear time, by using bin-sorting as it was demonstrated in [17] by Batagelj and Zaveršnik. However, the same idea was applied by Matula and Beck [100]. All nodes are maintained sorted with respect to their degrees by using a comparison-free sorting algorithm which maintains separate bins for each degree value. Clearly, for a graph with n nodes, the minimum degree of a node is 1 (assuming no isolated nodes exist) and the maximum is $n-1$. Thus, by keeping an in-memory array of all possible degree values and keeping track of bin boundaries, each edge deletion can be handled in $\mathcal{O}(1)$ time, resulting in a total complexity of $\mathcal{O}(m+n)$. The corresponding pseudocode is given in Algorithm 1. Note that, maintaining the k-core decomposition of a graph is equivalent of keeping the core number $c_i, \forall i \in V$.

Algorithm 1: CoreDecomposition (G)

Input: the graph G
Result: the core numbers (array C)

```
1  V ← set of vertices of G
2  array D ← vertex degrees
3  sort array D in non-decreasing order
4  for each v ∈ V in the order do
5  |  C[v] ← D[v]
6  |  for each u ∈ N(v) do
7  |  |  if D[u] > D[v] then
8  |  |  |  D[u] ← D[u] − 1
9  |  |  |  reorder array D accordingly
10 return C
```

3.2 Disk-Resident Graphs

A natural extension to the previous algorithm is based on the fact that most of the interesting real-world networks are too large to fit in main memory. Therefore, we need efficient algorithmic techniques for providing the core decomposition in cases where the graph is stored in secondary storage (i.e., disk).

The first algorithm (EMCore) to attack the problem in secondary storage was reported in [30]. EMCore assumes that the graph resides on disk and it performs the following steps: *i*), graph partitioning, *ii*) core number estimation and *iii*) recursive top-down processing.

Graph partitioning. The purpose of this step is to decompose the graph into small subgraphs, so that core number computation can be performed in each subgraph separately. Evidently, each of these subgraphs must fit in main memory.

Although many graph partitioning techniques are available, we are interested in a partitioning disk-resident graphs. The algorithms scans the input graph only once and partitions the vertex set V into a set of p mutually disjoint vertex subsets $U = \{U_1, ..., Up\}$, where $V = \bigcup_i U_i$ and for any i, j, $U_i \cap U_j = \emptyset$.

The graph partitioning algorithm starts reading nodes from the disk-resident graph and maintains in memory as many nodes as possible. Let U_{mem} denote the current memory-resident part of the graph. For any examined node u, if u is not connected to any of the vertex subsets currently in U_{mem}, a new partition is created and u becomes the only member of it. Otherwise, u is assigned to the vertex subset to with u has the most connections. In case we reach the block limits, a new block is flushed to the disk. Also, if the memory capacity is reached, the largest partition of the memory-resident part U_{mem} is flushed to the disk. This process continuous until all nodes have been scanned.

Core number estimation. For each subgraph determined by the partitioning process, the upper bound of the core number of each node is determined. This bound becomes tighter during the course of the algorithm. Initially, the upper bound $\psi(u)$ of the core number of each node u, is set to the degree of u, i.e., $\psi(u) = deg(u)$.

The refinement of the upper bound is performed by using the following observation: for a vertex u let Z denote the subset of the neighbors of u such that their upper bound is strictly less than $\psi(u)$. Then, for any nonempty subset $Z' \subseteq Z$, if $deg(u) - |Z'| < \psi(u)$, then the upper bound $\psi(u)$ can be tighten as $\psi(u) = \max\{deg(u) - |Z'|, \psi_{max}(Z')\}$, where $\psi_{max}(Z')$ denotes the maximum upper bound found in Z'.

Algorithm 2: ESTIMATE (G)

Input: the graph G
Result: core number estimation

```
1 for each vertex u ∈ V do
2     initialize ψ(u) ← deg(u)
3 for each U ∈ V do
4     set Z ← {v ∈ N(u), ψ(v) < ψ(u)
5     if deg(u) − |Z| < ψ(u) then
6         let f(X) ← max{deg(u) − |X|, ψ_max(X)}
7         set ψ(u) ← min{f(Z') : Z' ⊆ Z, Z' ≠ ∅}
8 repeat lines 3-7
```

The outline of the estimation process is shown in Algorithm 2. This algorithm is executed every time a block needs to be flushed to disk during the partitioning process described previously. Note that in order to determine the subset Z', there is no need to generate all possible subsets of Z which would lead to checking $2^{|Z|}$ subsets. Instead, nodes in Z are sorted in decreasing order of $\psi(u)$ and keep the first nodes in the order that minimize the value of $f(Z')$. This results in a much more efficient computation, since we only need to check at most $|Z|$ subsets.

Recursive top-down processing. Based on the upper bound of the core number, the k-classes are recursively computed for a convenient value of k. The value of k is determined so that the relevant subgraph can fit in main memory. More specifically, a range of values $[k_l, k_u]$ is determined, where $k_l \leq k_u$. The target is to determine the value of k_l, given the value of k_u and b, which is the maximum number of blocks that can

be accommodated in main memory. The value of b is simply $\lfloor M/B \rfloor$, where M is the total memory capacity and B is the size of the disk block.

Let $\Psi_{k_l}^{k_u}$ denote the subset of nodes with core upper bound estimation in the range $[k_l, k_u]$, i.e., $\Psi_{k_l}^{k_u} = \{u : u \in V, k_l \leq \psi(u) \leq k_u\}$. At each recursive step, the algorithm constructs a subgraph that is relevant for the computation of the exact core numbers of the nodes $v \in \Psi_{k_l}^{k_u}$.

We will discuss briefly the way k_l is determined. Let K be the set of values of k such that the nodes in the set $\Psi_k^{k_u}$ are distributed in at most b different vertex sets of U. Formally, the set K is computed as follows:

$$K = \{k : 1 < k \leq k_u, |U_x : U_x \in U, U_x \cap \Psi_k^{k_u} \neq \emptyset| \leq b\}$$

Based on this, the value of k_l is set to $\min\{k : k \in K\}$ if $K \neq \emptyset$ or it is set to k_u otherwise. This way, at each recursive step, the algorithm loads as many parts of the graph as possible into main memory. Since both k_l and k_u are determined, the algorithm proceeds with the computation of the core numbers of all nodes in $\Psi_{k_l}^{k_u}$ by loading in main memory the corresponding subgraphs. Then, the core number of each node u currently in main memory is refined. Before the execution of the next recursive step, the nodes that have their core number refined are removed from main memory (together with the corresponding edges) after *depositing* some important bookkeeping information that may be needed by the next recursive step. A new value for the upper bound of the core number is set as $k_u \leftarrow (k_l - 1)$, the new value of k_l is computed and the next recursive step is executed. The process terminates when no more k values are left to examine (essentially this translates to $k_l \leq 1$).

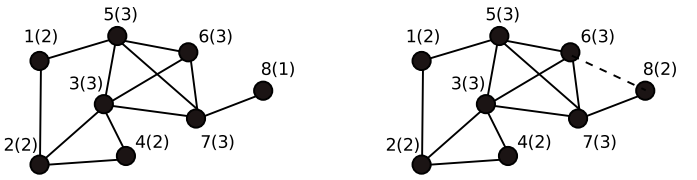
In conclusion, EMCore manages to compute the core number of all nodes without the requirement that the graph fits in main memory. This fact enables the computation of core decomposition of large disk-resident graphs. The algorithm performs $\mathcal{O}(k_{max})$ iterations over a graph G, where k_{max} is the maximum core number of G. A limitation of the EMCore algorithm is that it may require a significant number of I/O operations to detect the appropriate partitions. In [60] a space-efficient algorithm is proposed (NimbleCore), that provides accurate estimates of the core numbers by using $\mathcal{O}(n)$ space for graphs with power-law degree distribution and $\mathcal{O}(n \log d_{max})$ space for arbitrary graphs, where d_{max} is the maximum node degree. Another implementation of EMCore has been reported in [72], which is based on GraphChi [78]. In the same work, the classic code decomposition algorithm [17] is implemented using the Webgraph [21] framework. Webgraph is a graph compression framework that provides efficient access to a compressed graph.

3.3 Core Decomposition in Dynamic Graphs

The previous techniques discussed so far assume that the graph is available in the very beginning of the execution. However, many modern applications are characterized by frequent updates, meaning that the structure of the graph may change over time by the insertion/deletion of nodes or edges. Evidently, if the graph changes then we can reevaluate the core numbers by running the algorithm again. Although this approach will provide the correct results, it is expected that the performance will degrade, especially of updates are frequent.

As an example, consider the graph shown in Figure 8(a). The core numbers are shown in parentheses near each node identifier. Assume that a new edge between nodes 6 and 8 is inserted. The updated core numbers are shown in Figure 8(b). It is evident that only the core numbers of node 8 will change from 1 to 2. No other changes are required, since the core number of the rest of the nodes does not change.

A possible alternative is to maintain the previous core numbers and perform only incremental changes to the core numbers, based on the parts of the graph that have change. For example, the insertion of a new edge in the graph may impact the core numbers of specific nodes. The idea is to detect the set of affected nodes and recompute their core numbers without recomputing the core numbers for all the nodes. A similar approach can be followed when an edge is deleted from the graph. Performing incremental changes



(a) initial graph (b) after inserting edge (6,8)

Figure 8: Changes performed to the core numbers after inserting a new edge linking nodes 6 and 8. Only the core number of node 8 needs to be updated.

is expected to be much more efficient than re-executing the core decomposition algorithm from scratch. In this section, we discuss the methodology proposed in [124], which provides an incremental way to update the core numbers. The incremental algorithm is based on the following foundations:

1. If an edge is inserted to or removed from G, the core number of any node u can change by at most one.
2. If an edge (u, v) is inserted to or removed from G, where $c(u) < c(v)$, then $c(v)$ cannot change.
3. If an edge (u, v) is inserted into G, then all of the vertices whose core numbers have changed should form a connected subgraph. Similarly, if an edge (u, v) is removed from G, then all the vertices whose core numbers have changed should form a connected subgraph.
4. If an edge (u, v) is inserted (removed) and $c(u) \leq c(v)$, then only the vertices w that have $c(w) = c(u)$ and are reachable from u via a path that consists of vertices with core numbers equal to $c(u)$, may have their core numbers incremented (decremented).

In [124], three different algorithms are studied: *i*) SubCore, which is based on the aforementioned foundations, *ii*) PureCore, which applies some additional optimizations and *iii*) Traversal, which manages to reduce the number of examined nodes even further and shows the best overall performance. Based on the experimental results in [124], even the SubCore algorithm, which is the simplest among the three proposed, manages to update the core numbers up to 14,000 faster than the standard baseline algorithm which recomputes the core decomposition from scratch. Interestingly, a very similar set of foundations is also defined in [87] for the same topic. The two pieces of work are seemingly developed in parallel and there is not direct comparison of their proposed algorithms but the theorems appear to be equivalent.

The later work [151] address [124] and [87] as being impractical for large graphs (in the aspect of number of nodes). Dynamic graphs are referred as temporal in [151] and the number of edges between two vertices may be more than one as a mean to describe an edge being in multiple instances in time (i.e. if an edge exists in $t1$ and $t2$ then it is counted as two edges). This transforms the problem of k-core decomposition to a (k, h)-core decomposition where the h is the number of edges a node shares with all of its neighbors and it is an additional degree/dimension in the core decomposition model. Effectively, by thresholding h (similar to k) one calculates cores as different temporal instances.

Two distributed algorithms are proposed for the calculation of the temporal core decomposition. The first is based on the Pregel model [93] and is similar to the classic k-core algorithm on Pregel but it is considered inefficient as it has high memory, communication and computation costs. A more efficient distributed algorithm is then presented based a block-centric model for graph computations [152] (Blogel) which is elaborated in the **Distributed Computation** section.

In [46] cores in a temporal graph are considered to exist in temporal internals Δ and are named span-cores. The authors then make the the note that a span-core at k, Δ is contained in a span core k, Δ' if

$k' \leq k$ & $\Delta' \subseteq \Delta$. Based on this, they define a maximal span-core as one where it is not contained in another span core (the previous condition cannot be satisfied by another core).

In order to figure out whether graph of $\Delta = [t_s, t_e]$ contains a maximal span-core, the authors prove that one only needs the core numbers from $\Delta' = [t_s - 1, t_e]$ and $\Delta'' = [t_s, t_e + 1]$. This eventually motivates the authors to start from larger temporal spans instead of calculating the core decomposition at each time-instance t of the graph. In this manner, the authors do not consider every core decomposition at each t as a maximal one.

3.4 Local Computation of Core Numbers

The main characteristic of the aforementioned techniques is that the core number of a node is determined by taking into account the whole graph. Moreover, the output in all cases is the core number of every node $u \in V$. In interesting alternative is to try to estimate the core number of a node u by considering only the neighborhood of u (e.g., the 1-hop, 2-hop 3-hop neighborhood, etc). The main advantage of such an approach is that we do not need to consider the whole graph, which potentially leads to more efficient computation. Also, in many cases we need to compute the core number of a small subset of nodes. However, we expect that the core number determined by such a *local computation* may not much the core number determined if the whole graph is taken into account.

The first work towards local core number estimation was reported in [109]. For a node u let $N_d(u)$ denote the set of nodes at a *distance* at most d from u. In its simplest form, the distance can be the shortest path distance, which, in the case of unweighted graphs, translates to the minimum number of hops between two nodes. Let G_d^u denote the graph induced by the d-neighborhood of u. One possible approach is to compute the core number of node u in the induced subgraph G_d^u. Let $c_d(u)$ denote the core number of node u computed in the induced subgraph G_d^u. By increasing d it is expected that a more accurate estimate of the core number may be achieved, since a larger induced subgraph is being used. Evidently, if d is large enough so that $G_d^u = G$, then $c_d(u) = c(u)$, meaning that the core number will be accurately computed. It is not hard to prove that always $c_d(u) \leq c(u)$, i.e., the core number computed in the induced subgraph G_d^u is a lower bound on the exact core number $c(u)$ computed in the whole graph. The outline of the lower bound computation is given in Algorithm 3.

Algorithm 3: LowerBound (G,d,u)

Input: graph G, number of hops d, node u,
Result: core number estimation in induced subgraph G_d^u

```
if d == 0 then
    return (deg(u))
else
    V_d(u) ← the d-neighborhood of u
    E_d(u) ← edges among V_d(u)
    create induced graph G_d^u using V_d(u) and E_d(u)
    execute CoreDecomposition on G_d^u
    return c_d(u) (the core number of u in G_d^u)
```

In the sequel, we present a more sophisticated estimator which, in contrast to the previous one, uses some additional foundations to provide tighter bounds for the core number. Let u the node we are interested in. In case $d = 0$, the only available bound that we may use is the degree of u, since $c(u) \leq deg(u)$. By setting $d = 1$, we may use additional information related to the 1-hop neighborhood of u. For example, if

the core numbers of the direct neighbors of u are known, then the value of $c(u)$ can be computed accurately, by utilizing the following rationale:

- A node u belongs to the k-core of G, if and only if u has at least k direct neighbors in the k-core.
- Let u_1, u_2, ... be the direct neighbors of u with known core numbers. Then, it holds that:

$$c(u) = \max_{1 \leq i \leq deg(u)} (\min(c(u_i), deg(u) - i + 1)) \tag{3}$$

Evidently, the assumption that the core numbers of all neighbors of u are known, is quite restrictive. However, the rationale of the previous idea is very useful in deriving an upper bound of the core number of a node, based on upper bounds of the core numbers if its neighbors. In fact, Equation 3 is valid for any $f(u) \geq c(u), \forall u \in V$. Thus, we can derive the following recurrence, where u_1, u_2, ... are the direct neighbors of u ordered in increasing order based on $\hat{c}_{d-1}()$, i.e., $\hat{c}_{d-1}(u_i) \leq \hat{c}_{d-1}(u_{i+1})$:

$$\hat{c}_d(u) = \begin{cases} \max\limits_{1 \leq i \leq deg(u)} (\min(\hat{c}_{d-1}(u_i), deg(u) - i + 1)), & d > 0 \\ 0, & d = 0 \end{cases}$$

Evidently, $\hat{c}_d(u)$ is an upper bound for $c(u)$. The outline of the computation of $\hat{c}_d(u)$ for any node u and any value of d is given in Algorithm 4.

Algorithm 4: UPPERBOUND (G,d,u)

Input: graph G, number of hops d, node u,
Result: core number estimation $\hat{c}_d(u)$

```
if d == 0 then
    return (deg(u))
else
    for v ∈ N_1(u) do
        compute ĉ_{d-1}(v)
    order nodes in N_1(u) based on ĉ_{d-1}
    ĉ_d(u) ← deg(u)
    for 1 ≤ i ≤ |N_1(u)| do
        j ← min{ĉ_{d-1}(u), deg(u) − i + 1}
        if j > ĉ_d(u) then
            ĉ_d(u) ← j
    return ĉ_d(u)
```

Based on the previous discussion, the exact core number $c(u)$ of a node u is bounded by the two values $c_d(u)$ and $\hat{c}_d(u)$:

$$c_d(u) \leq c(u) \leq \hat{c}_d(u), \forall d \geq 0$$

Based on the experimental results given in [109], both estimators provide satisfactory results for $d = 2$ for different input graphs. The accuracy of the estimators is quantified by considering the percentage of the total number of nodes for which the estimators give the exact core number. In most of the cases, the propagating estimator $\hat{c}_d(u)$ is more efficient than the induced estimator $c_d(u)$ in terms of accuracy as defined previously. In fact, the propagating estimator manages to achieve an accuracy between 80% and 90%, in the majority of the experiments performed, for $d = 2$.

As a concluding remark, in cases were the core number of specific nodes needs to be computed, the methodology described is an effective alternative to the complete core decomposition process since it is more efficient with respect to runtime and provides satisfactory accuracy results.

3.5 Parallel and Distributed Techniques

The main feature of the algorithmic techniques discussed so far is that they work in a centralized environment. However, mining massive graphs in a centralized manner does not provide scalable solutions. A very promising alternative is the exploitation of *multiple resources* to attack the problem. Towards this goal, there are two different research directions based on architectural assumptions: *i*) solving the problem in a shared-memory *multi-core machine* (the *parallel* case) and *ii*) solving the problem in a *cluster of machines* (the *distributed* case).

Parallel Computation. Parallel computation of core decomposition in multi-core processors was first investigated in [129], where the ParK algorithm was proposed. ParK was designed to work efficiently in multi-core processors where locality of reference is very important. In contrast to other techniques, ParK carefully reduces the number of random memory accesses performed. Note that random memory accesses may invalidate the caches in a multi-core system leading to performance degradation.

ParK uses three data structures: $Core$, $Curr$ and $Next$. $Core$ is an array if size n initialized to the degrees of the nodes, i.e., initially $Core(u) = deg(u)$ for any node u. It is updated continuously during the course of the algorithm and its final values correspond to the core numbers of the nodes. $Curr$ contains the set of nodes to be processed during the current iteration, whereas $Next$ contains the nodes to be processed in the next iteration.

To process a node u involves accessing the direct neighbors of u and decreasing their degree if they have not already been processed. Computation is performed in different levels. While processing level l, all nodes that belong to the l-shell are processed. This is performed in two steps, Scan and Loop:

- During the Scan phase, the array $Core$ is scanned and all relevant nodes belonging to the l-shell are collected in the set $Curr$. Formally: $Curr = \{u : deg(u) = l\}$.
- The Loop phase is executed in sublevels. In each sublevel, all nodes in the set $Curr$ are processed. While processing node u, if a neighbor v is moved to $Curr$, which means that $deg(v) = \mathrm{l}$, then node v is added to the set $Next$. At the end of the sublevel, the contents of $Next$ are transferred to the set $Curr$ to be processed in the next sublevel.

The outline of ParK is given in Algorithm 5. It is supported by two procedures: Scan and SubLevel.

In the previous discussion, the description of ParK is based on a centralized environment. In the sequel we provide the changes need to be applied in order to use ParK in a parallel (shared-memory) setting. First, we assume that processing will be performed by a set of threads T. Let t denote the total number of threads in T. Next, procedures Scan and SubLevel must be enhanced to allow multi-thread execution. To achieve this, nodes are split among the t threads, meaning that each thread handles roughly n/t nodes. However, since the data structures used may need to be written by multiple threads concurrently, race conditions may appear and they must be handled carefully to avoid inconsistencies.

The outline of the parallel version of Park is given in Algorithm 6. By inspecting the pseudocode we observe that there are some differences in comparison to the centralized variation. For example, changes performed to the $Curr$ set must be atomic, which means that the write operations must be protected due to race conditions. For example, adding a new node id to $Curr$ requires an `atomicIncrement()` operation (Line 23) in Procedure Scan. Also, similar atomic operations are required in Procedure SubLevel (Lines

Algorithm 5: ParK (G) */* centralized */*

Input: the graph G
Result: core numbers

```
1  Curr ← ∅
2  Next ← ∅
3  remaining = n
4  level = 1
5  while remaining > 0 do
6      invoke Scan(Core, level, Curr)
7      while |Curr| > 0 do
8          remaining ← remaining − |Curr|
9          invoke SubLevel(Curr, Core, level, Next)
10         Curr ← Next
11         Next ← ∅
12     level ← level + 1
13 return Core
14 Procedure Scan (Core, level, Curr)
15 for u ∈ V do
16     if Core[u] = level then
17         Curr ← Curr ∪ {u}
18 Procedure SubLevel (Curr, Core, level, Next)
19 for u ∈ Curr do
20     for v ∈ N(u) do
21         if Core[v] > level then
22             Core[v] ← Core[v] − 1
23             if Core[v] == level then
24                 Next ← Next ∪ {v}
```

30, 32, 34). In addition, in the main algorithm we need to invoke `fork()` (Line 3) to create multiple instances that will execute in parallel and also `join()` (Line 17) to wait for the threads to finish before we report the core numbers back to the caller of ParK. Moreover, there is need for include synchronization calls (Lines 8, 12, 15). The invocation of `synchronize()` sets a barrier which must be reached by all running threads before code execution can continue. In Algorithm 6 the additions required to guarantee consistency during execution are depicted framed.

To get an idea of the performance improvement that ParK achieves, for the Friendster graph [84] which contains 65 million nodes and 1.8 billion edges, ParK needs roughly 160 seconds to compute the core numbers whereas the centralized algorithm requires almost 1000 seconds on a machine with 8 physical cores. Similar behavior is observed in the majority of the datasets used for experimental evaluation.

Distributed Computation. Although the computation of the core decomposition using parallelism is very attractive compared to the centralized alternative, still there are significant limitations. The use of shared memory may become a bottleneck by increasing the number of parallel resources in the system. Anyway, the level of parallelism may increase up to a point using a shared-memory architecture. In addition, if the size of the graph grows significantly, we may face storage problems since the graph may not be accommodated

Algorithm 6: ParK (G) */* parallel */*

Input: the graph G
Result: core numbers

```
1  Curr ← ∅
2  Next ← ∅
3  invoke fork()
4  remaining = n
5  level = 1
6  while remaining > 0 do
7      invoke Scan(Core, level, Curr)
8      invoke synchronize()
9      while |Curr| > 0 do
10         remaining ← remaining − |Curr|
11         invoke SubLevel(Curr, Core, level, Next)
12         invoke synchronize()
13         Curr ← Next
14         Next ← ∅
15         invoke synchronize()
16     level ← level + 1
17 invoke join()
18 return Core
19 Procedure Scan (Core, level, Curr)
20 for i = 0 to n − 1 in parallel do
21     idx ← 0
22     if Core[u] = level then
23         a ← atomicIncrement(idx, 1)
24         Curr[a] ← i
25 Procedure SubLevel (Curr, Core, level, Next)
26 idx ← 0
27 for u ∈ Curr in parallel do
28     for v ∈ N(u) do
29         if Core[v] > level then
30             a ← atomicDecrement(Core[v], 1)
31             if a ≤ level then
32                 atomicIncrement(Core[v], 1)
33             if a + 1 == level then
34                 b ← atomicIncrement(idx, 1)
35                 Next[b] ← v
```

in main memory. One may argue that in such a case, disk-based techniques, like the EMCore algorithm which was covered in a previous section, could be applied. The problem with the algorithms that utilize secondary storage is that they perform a large number of passes over the data. As the graph grows larger the number of passes is expected to increase leading to inefficient computation.

During the last fifteen years we have witnessed a tremendous progress in data-driven distributed computing. In addition to the numerous ad-hoc solutions, several unified distributed platforms like MapReduce [37],

Hadoop [149] and Spark [154] appeared and paved the way for nowadays cluster computing. These platforms are based on a cluster of shared-nothing machines (usually of commodity hardware) and they are able to execute complex data mining and machine learning algorithms over massive datasets efficiently. For the remaining of this section we will focus on a distributed core decomposition algorithm that was initially designed for a cluster of machines without any specific organization. The only requirement is that the processors may communicate my exchanging messages through the interconnect (usually a high-speed LAN). However, the algorithm was later adapted in order to be applicable in the Spark distributed engine.

The first distributed core decomposition algorithm was reported in [105, 106]. In the general case, each processing unit is responsible for multiple graph nodes. To simplify the presentation we will adopt the *one-to-one* scenario, meaning that we assume that each graph node corresponds to a single processing unit (a processor or a core). This model, resembles the Pregel's [93] "*think as a vertex*" point of view. In other words, to design an algorithm we should take the point of view of a graph node u and try to provide the answer based on the information collected from the neighbors of u in an iterative manner. Each graph node u maintains the following information:

- $core(u)$: This is the currently most accurate estimate for the core number of u, which is initialized to the degree of u.
- $est[u_1, ..., u_l]$: This is an array (or hashmap) storing the current estimates for the core numbers of u's neighbors. More specifically, $est[u_j]$ is the most up-to-date estimate of the core number of u_j known by u. Initially, $est[u_j] \leftarrow \infty$.
- $changed$(u): This is a boolean flag that is set to `true` whenever the value of $core(u)$ changes. This attribute is initialized to false.

The computation for a node u begins by sending a message to all it's neighbors. Since initially there is no accurate estimate for $c(u)$ (the core number of u), we may use the node degree as an upper bound. Node u prepares a message $msg[u, deg(u)]$ containing the node identifier of u and the current best estimate of the core number. This message is transmitted to all neighbors of u.

Assume now that u receives a message $msg[u_i, k]$ from node u_i (i.e., the i-th neighbor of u). From this message, we know that k is the current best estimate for $core(u_i)$. Therefore, if $k < est[u_i]$, this means that the previous estimate that node u knows about u_i is larger than the last received, and it should be updated by setting $est[u_i] \leftarrow k$. Once node u receives such a message, a new estimate for $core(u)$ may be produced. Let h denote the newly estimated value of the core number based on the information contained in $est[u_1, ..., u_l]$. If $h < core(u)$, then $core(u) \leftarrow h$ and also $changed(u) \leftarrow true$, which means that a new estimate for $core(u)$ is available.

The outline of this technique is given in Algorithm 7. The main part of the algorithm is composed of three phases, depending on the event being handled. Initially, and before the start of execution an initialization step is applied in Lines 1-6. In this step, the only valid estimate for the core number of the degree of u. Therefore, the message send by node u to all its neighbors simply contains the degree of u. Every time a new message is received by u from one of its neighbors u_i, an update is performed in case the new value received has an impact on the estimation the $core(u)$ (Lines 7-13). The function UpdateCore is used in this case (Lines 18-29). Node u sends periodically (every Δt time instances) the value of $core(u)$ to its neighbors, in case $core(u)$ has changed (Lines 14-17).

The algorithm terminates when no change is performed on any core number. If this happens, then the core number estimate for each node equals its actual core number. To be able to achieve this convergence several techniques may be applied, e.g., centralized, decentralized, barrier synchronization. Another more efficient technique is to execute the algorithm for a fixed number of rounds. The main motivation for this alternative is that after the the first few rounds the core number estimates are quite close to the actual

Algorithm 7: PERNODE (u)

Input: node u
Result: new estimate h for $core(u)$

On Initialize / initialization step */*
  $changed(u) \leftarrow false$
  $core(u) \leftarrow deg(u)$
  for *each* $u_i \in N(u)$ **do**
    $est[u_i] \leftarrow \infty$
  send$(msg[u, u_i])$, $\forall i$ */* inform neighbors */*
On Receive$([u_i, k])$ */* msg from a neighbor */*
  if $k < est[u_i]$ **then**
    $est[u_i] \leftarrow k$
    $h \leftarrow$ UPDATECORE$(u, est[], core(u))$
  if $h < core(u)$ **then**
    $core(u) \leftarrow h$ */* update core estimate */*
    $changed(u) \leftarrow true$
Repeat (every Δt time instances)
if $changed(u)$ **then**
  send$(msg[u, core(u)])$ */* inform neighbors */*
  $changed(u) \leftarrow false$

Function UPDATECORE $(u$,$est[]$,$k)$
for $i = 1$ *to* k **do**
  $count[i] \leftarrow 0$
for *each* $v \in N(u)$ **do**
  $j \leftarrow \min\{k, est[v]\}$
  $count[j] \leftarrow count[j] + 1$
for $i = k$ *downto* 2 **do**
  $count[i-1] \leftarrow count[i-1] + count[i]$
$i \leftarrow k$
while $i > 1$ *and* $count[i] < i$ **do**
  $i \leftarrow i - 1$
return i

core numbers. Therefore, there is an efficiency vs. accuracy trade-off since on one hand the number of rounds is fixed but on the other hand the core numbers reported may not be 100% accurate. We have performed some experiments to test the accuracy of the algorithm by using a fixed number of iterations. Table 3 presents some representative results. More specifically, the table shows the percentage of nodes that have the correct core number after 20 rounds (iterations). We observe, that the accuracy is adequate for most realistic scenarios taking into account that an exact computation would require a significant number of rounds (shown in column **max num of iters**). All datasets are available at [84].

In [151], the Pregel model is deemed inefficient for large temporal graphs and an alternative is proposed based on the Bogel model [152]. This approach partitions the graph into blocks V_b that are accompanied with the information of which vertices are also connected to V_b (V_b^+). The core numbers in each V_b are computed and then the core numbers of the V_b^+ vertices are used to update their respective block (V_b). The

Table 3: Percentage of nodes with correct core numbers after 20 iterations (Accuracy@20).

Graph	Max Num of Iters	Accuracy@20
Orkut	191	88.76%
LiveJournal	99	98.49%
Web-Stanford	538	90.2%
Enron Email	28	99.7%

degree of a node in V_b takes into account also V_b^+. The main intuition of the update is that the vertices in V_b^+ with a core number lower than the minimum degree of V_b will not contribute to the core number of vertices in the future. Those vertices are removed from V_b^+ and core numbers are recomputed in V_b recursively.

3.6 Probabilistic Core Decomposition

The aforementioned algorithmic techniques operate on *certain* graphs, meaning that graph nodes and edges are present with certainty (they always exist). However, as stated in Section 2, many applications require some kind of *uncertainty* associated with nodes or edges. In such a case, the graph becomes *uncertain* or *probabilistic*, meaning that the edge (or node) will be present in the network with some probability. For the following discussion, we will assume that graph nodes are certain (i.e., exist at all times) whereas uncertainty is associated only with edges.

In the sequel we describe the algorithm reported in [23]. Let $\mathcal{G} = (V, E, p)$ be an uncertain (a.k.a probabilistic) graph, where $p : E \rightarrow (0, 1]$ is a function that assigns probabilities to the edges of the graph. A widely used approach to analyze uncertain graphs is the one of *possible worlds*, where each possible world constitutes a deterministic realization of $\mathcal{G}$. According to this model, an uncertain graph $\mathcal{G}$ is interpreted as a set $\{G = (V, E_G)\}_{E_G \subseteq E}$ of $2^{|E|}$ possible deterministic graphs [119, 110, 111]. Let $G \sqsubseteq \mathcal{G}$ indicate that G is a possible world of $\mathcal{G}$. Then, the probability that $G = (V, E_G)$ is observed as a possible world of $\mathcal{G}$ (assuming independence of edge existence) is given by the following formula:

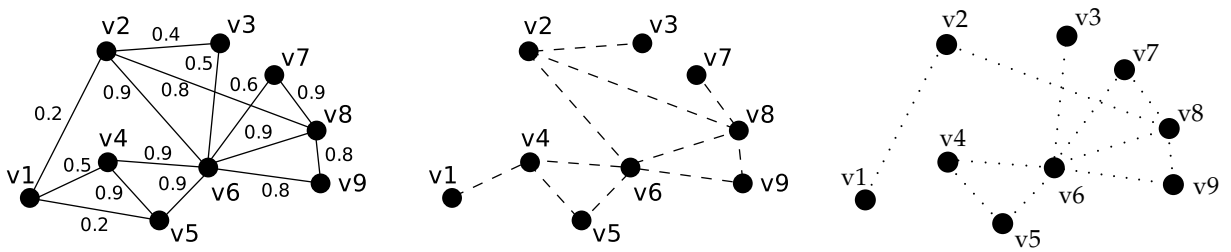
$$\Pr[G|\mathcal{G}] = \prod_{e \in E_G} p(e) \prod_{e \in E \setminus E_G} (1 - p(e)) \tag{4}$$

As an example, consider the probabilistic graph $\mathcal{G}$ shown in Figure 9(a). Two possible instances of $\mathcal{G}$ are given in Figure 9(b) and 9(c). High-probability edges are expected to show up more frequently in instances of $\mathcal{G}$ in comparison to low-probability edges. In this example, G_1 and G_2 are two of the *possible worlds* that can be produced by using $\mathcal{G}$ as a template.

One of the novelties of the approach proposed in [23] is that the degree of a node is expressed by using probabilistic arguments. First, we need an expression of the probability that the degree of a node u is more than k. Note that this is a natural concept taking into account that the degree of a node is in general different in different instances of the probabilistic graph $\mathcal{G}$. This probability is expressed as follows:

$$\Pr[deg(u) \geq k] = \sum_{G \in \mathcal{G}_u^k} \Pr[G] \tag{5}$$

where $\mathcal{G}_u^k$ is the set of instances of $\mathcal{G}$ where u has a degree at least k. Next, we introduce the η-degree of a



(a) $\mathcal{G}$ (b) G_1 (c) G_2

Figure 9: A probabilistic graph $\mathcal{G}$ and two possible instances G_1 and G_2. The numbers near the edges denote existential probabilities.

node u, denoted as $\eta deg(u)$ and defined as follows:

$$\eta deg(u) = \max\{k \leq |N(u)| | \Pr[deg(u) \geq k] \geq \eta\} \tag{6}$$

More specifically, the $\eta deg(u)$ is the maximum k for which the probability that the degree of u is more than k, is more than η. Recall that $|N(u)|$ is the number of direct neighbors of u in the probabilistic graph $\mathcal{G}$. Based on the variables k and η, the concept of (k, η)-core is defined: the (k, η)-core of a probabilistic graph $\mathcal{G}$ is a maximal subgraph $\mathcal{H}(V_{\mathcal{H}}, E_{\mathcal{H}}, p)$ such that the probability that each vertex $u \in V_{\mathcal{H}}$ has degree no less than k in $\mathcal{H}$ is greater than or equal to η, i.e., $\forall u \in V : \Pr[deg_{\mathcal{H}}(u) \geq k] \geq \eta$.

Definition 13 (Probabilistic core decomposition) *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 $\mathcal{G}$.*

The outline of the probabilistic core decomposition technique is shown in Algorithm 8. We observe that in contrast to the standard core decomposition approach, degree computation is substituted by η-degree computation in the probabilistic case. More specifically, all η-degrees are computed at the beginning of the processing (Lines 1,2), and η-degrees of neighboring nodes are updated accordingly (Lines 11-15). However, while the degree computation is straightforward in the deterministic case, computing η-degrees are far from trivial. In [23], interesting techniques are developed for computing η-degrees efficiently, based on *dynamic programming*.

The probabilistic core decomposition was also studied recently in [117], where (k, θ)-cores were proposed. In contrast to (k, η)-cores, (k, θ)-cores capture the likelihood of a node to be a k-core member in different instances of an uncertain graph $\mathcal{G}$. Given k and the probability threshold θ, the technique of [117] detects nodes such that with probability at least θ are included in the k-core in possible worlds.

4 Representative Applications

The core decomposition concept, despite its simplicity, has been applied successfully in many different scientific disciplines. It turns out that the core number of network nodes plays different roles, depending on the context being used and the type of the network applied. In this section, we present the most representative use cases and the associated results obtained.

Algorithm 8: ProbCores ($\mathcal{G}$, η)

Input: the probabilistic graph $\mathcal{G}$, probability threshold η
Result: the η-core number for each node

```
1  forall u ∈ V do
2      compute ηdeg(u)
3  c ← ∅, d ← ∅, D ← [∅, ..., ∅]
4  forall u ∈ V do
5      d[u] ← ηdeg(u)
6      D[ηdeg(u)] ← D[ηdeg(u)] ∪ {u}
7  forall k = 0, 1, ..., n do
8      while D[k] ≠ ∅ do
9          D[k] ← D[k] − {v}, random v ∈ D[k]
10         c[v] ← k
11         forall u : (u, v) ∈ E, d[u] > k do
12             recompute η-deg(u)
13             D[d[u]] ← D[d[u]] − {u}
14             D[η-deg(u)] ← D[η-deg(u)] ∪ {u}
15             d[u] ← η-deg(u)
16         V ← V − {v}
17 return c /* n-dimensional vector containing the η-core number of each vertex in G */
```

4.1 Network Modeling and Analysis

The k-core decomposition and its extensions have been extensively used in several applications. Seidman [128] was the first that proposed to use the tool of k-core decomposition in social network analysis, as an easy to compute and effective way to extract dense subgraphs. Later, other studies in large scale real networks followed [64, 58], including the analysis of Microsoft Instant Messenger (MSN) [82] and Facebook [141] social graphs. In a similar way, the decomposition has been applied to study [9] and model [27] the Internet graph.

Furthermore, several theoretical studies about the structure of real networks have been presented from the *statistical physics* community [38]. In [65], it was shown that the k-core plays a central role for the modeling of real world graphs and their percolation properties. Based on that, a graph generation model was introduced, and the properties of the generated graphs has been compared against a variety of real networks.

4.2 Temporal Evolution

An interesting application of temporal cores is found in the aforementioned work of [46] (Dynamic Graphs) for the detection of anomalous contacts in social networks. Besides the extensions of the k-core structure into temporal graphs, the evolution of degeneracy has also been studied as a property through time (e.g. like density, diameter etc.). In [158] the authors use k-core to study the evolution of the Internet through time and discover that the majority of new nodes are added to the periphery of the graph while the size of the maximal k-core is quite stable through time.

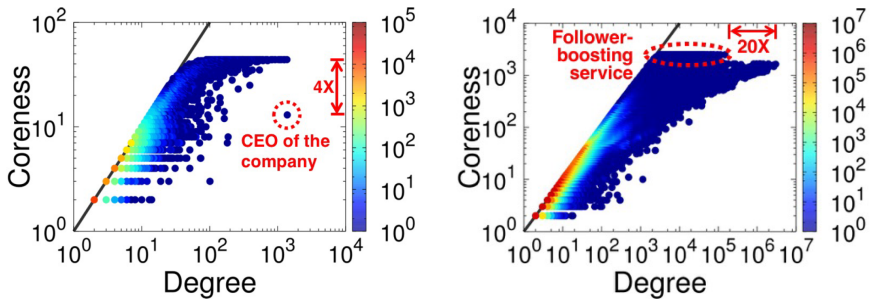


Figure 10: The *mirror pattern* of an Email communication graph (left) and a snapshot of the Twitter network (right). Each plot depicts the correlation between the degree and core number of the nodes of the graph. The figure is courtesy of Shin et al. [131]. ©2016 IEEE.

Algorithm 9: CORE-D

Input: graph stream: G, sampling probability : p, coefficients: w_0, w_1
Result: estimated degeneracy: $\hat{k}_{max}$

$G_{Sample} = \emptyset$
for *each edge* $e \in G$ **do**
  add e to G_{Sample} with probability p
$\#\Delta_{Sample} \leftarrow TriangleCounting(G_{Sample})$ [140]
$\#\hat{\Delta} \leftarrow \#\Delta_{Sample}(1/p)^3$
$\hat{k}_{max} \leftarrow exp(w_0 + w_1 log(\#\hat{\Delta}))$
return $\hat{k}_{max}$

4.3 Anomaly Detection

Shin et al. [131, 132] examined the properties of the k-core decomposition in a wide range of real-world networks. Their main observations include a set of empirical patterns that hold across several real-world graphs and can further be used to detect anomalies. The main observation, called the *mirror pattern*, indicates that the core number of the nodes of a graph has a strong positive correlation with the degree (which essentially represents an upper bound for the core number).

Figure 10 depicts the mirror pattern of two real-world graphs: an Email communication network and a snapshot of the Twitter graph. As one can observe, their is strong positive correlation between node degree and core number. For example, in the case of the Email network, Spearmans rank correlation coefficient has value $\rho = 0.99$, while in Twitter network the correlation coefficient is $\rho = 0.95$. Intuitively, the mirror pattern implies that nodes with high core number have also the tendency to have high degree and vice versa.

Nevertheless, one may observe that some nodes deviate from this "ideal" behavior; as the authors mention, these nodes correspond to two different types of anomalies: "lonerstars" (i.e., nodes mostly connected to "loners") and "lockstep behavior" (i.e., a group of similarly behaving nodes). In the Email network of Figure 10 (left), the marked node has the highest degree but relatively low core number; this node corresponds to a secondary email account of the former CEO of the company, which was used only to receive emails.

In the case of the Twitter network, the nodes with the highest core number in Figure 10 (right) have been marked. Those nodes, have relatively low degrees and slightly deviate from the mirror pattern. Taking a closer look on the corresponding Twitter accounts, the authors noticed that at least 78% of those nodes were directly involved in a "Follower-Boosting" service – thus, can be annotated as anomalies.

In order to conduct this analysis, they devise a novel approach for estimating the core number in huge graphs (CORE-D, Algorithm 9). Their algorithm is based on estimating the the number of triangles by using the work of [140] for sampling and estimating the triangle count in large networks. The degeneracy of the graph can be estimated then based on coefficients (w_0, w_1) which are calculated from real data using linear regression.

4.4 Detection of Influential Spreaders

Detecting influential spreaders is an important topic for understanding how information diffuses in social networks. An intuitive notion in this domain is that individuals with high connectivity would contribute more in the diffusion process. This would naturally lead to metrics like betweenness centrality to be utilized for the identification of good spreaders (e.g., [112]).

The work of Kitsak et al. [74] finds a contradiction to that – as being highly connected in a network is not sufficient. It is pointed out that the quality of the connections are also important (i.e., the neighbors must be also well connected) and that the k-core is better at finding highly influential nodes. The observation that the core number of a node is a good predictor of its spreading capabilities, formed a new line of research in the area of influence spreading. A similar study that was conducted in [115], including PageRank as well as additional influence spreading metrics, lead to the same conclusion in favour of k-core.

Naturally, several improvements and extensions have been made to this approach (e.g., [91, 114, 79]). One such improvement ranks nodes of a network by the sum of the core numbers of its neighbors [89]. In [26], the authors apply k-core (k-shell) on data from Twitter and notice that the skew in the degrees of the nodes creates a unnecessary number of cores (thousands) with most of the nodes existing in the lowered ranked ones (up to 4). To limit the number of cores, instead of mapping k connection to the k-th level, they assign $2^k - 1$ to k. In order to provide a more sophisticated decomposition, weights can be introduced in the graph. the authors of [5] provide a weighting scheme that represents the interaction among nodes and apply a weighted version of the k-core algorithm. This weight is specific to the nature of the graph (e.g. based on retweets in Twitter).

Other extensions of the k-core decomposition can be utilized as well. A prime example is the use of k-truss (triangle-based) [95]. This more restrictive version of graph degeneracy provides a smaller and more refined set of nodes in the maximal k-truss subgraph (which is a subset of the maximal k-core). The k-truss structure also captures the cohesiveness of the graph.

The work of [86] utilizes the k-core as a prepossessing step for an algorithm that extracts influential communities. A basic assumption in that work is that we can weight the graphs with some influential metric (e.g., Pagerank). Then the k-influential community is defined as an induced subgraph H^k of G where:

- H^k is connected.
- Each node in H^k has a degree of k.
- There is no other subgraph that satisfies the other two criteria, is not a subgraph of H^k and has a minimum weight (among its nodes) lower than H^k.

While there is a much more efficient algorithm presented in that work, we present here in Algorithm 10 the naive basic version of computing the top-r k-connected communities as it is more intuitive. The authors note that this is better than k-truss as it includes the influence of each community (the minimum node-weight) but there is no direct comparison.

The work of [132] uses k-core decomposition for detecting influential spreaders as well (besides anomaly detection). In their approach (CORE-S Algorithm 11) for this application, they apply "vanilla" k-core decomposition and rank the potential spreaders with their eigen-centrality within the core (instead of the

Algorithm 10: TopCom (G,W,r,k)

Input: $G(V, E)$, W, r, and k
Result: The top-r k-influential communities

$G_0 \leftarrow G, i \leftarrow 0$;
while G_i *contains a* $k-core$ **do**
  Compute the maximal $k-core$ $C^k(G_i)$;
  Let $H^k(i)$ *be the maximal connected component of* $C^k(G_i)$ *with the smallest influence value*;
  Let u *be the smallest* $-$ *weight node in* $H^k(i)$;
  Delete u;
  Let G_{i+1} *be a subgraph of* $C^k(G_i)$ *after deleting* u;
  $i \leftarrow i + 1$;
if $i \geq r$ **then**
  return $H^k(i1), \ldots, H^k(ir)$
else
  return $H^k(i1), \ldots, H^k(0)$

global graph) . They compare their approach against k-core, k-truss, and eigen-centrality presenting better results than them both in efficiency and accuracy.

Algorithm 11: Core-S for top-k spreaders

Input: Graph: G, Number of spreaders: $k(nmax)$
Result: k influential spreaders

Run the core decomposition of G
Extract the degeneracy-core $G'(V', E')$ from G
Compute the in-core centrality of the vertices in V' using power iteration in G'
Return top-k vertices with the highest in-core centralities

4.5 Network Visualization

The nested decomposition of k-core organizes vertices efficiently into groups for visual analysis as well. One of the earliest pieces of work for k-core based graph visualization focused on the presentation of the graph's adjacency matrix [15].The main idea is to reorder the vertices in rows and columns by their core number.

In general, the k-core has been used to display examples of results from the analysis of real world graphs by focusing in the most dense cores. In [58] the authors display dense cliques of collaboration found in the academia with fractional (weighted) cores. Triangle cores are also used on publication data in [159] as well as the Wikipedia graph and protein to protein interaction networks among others to display examples of discovered cliques. A similar concept to the triangle cores, the m-core is defined based on the number of triangles an edge (instead of a vertex) belongs to [161]. Based on the m-coreness of an edge, a vertex will belong to an m-core if at least one of its endpoint vertices belongs to it. The m-core is utilized in [33] on an internet graph as well as on the E. Coli metabolic network to support that real world networks are organized with mechanisms that are based on local instead of global properties.

A variety of system and software applications have been developed to provide visualizations of graphs

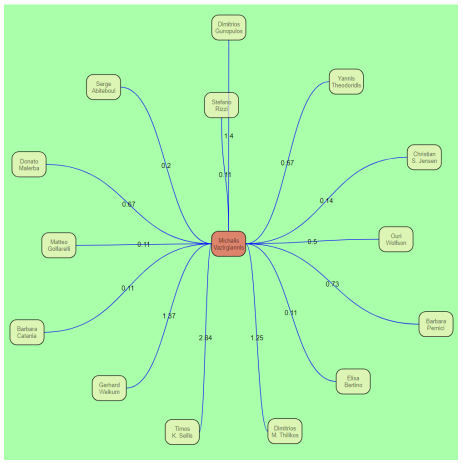


Figure 11: Example of k-core-based ego network.

either at their entirety or at specific sub-graphs. Focusing on those that utilize graph degeneracy, Gephi [14] is a popular graph visualization tool that includes -among many others- k-core as a vertex positioning algorithm to organize vertices in concentric nested circles that are equal to the number of cores while the position of the a vertex in each circle is random. In a work with a specific focus of evaluation of individuals, the authors of [53], present a tool for selecting and displaying the ego network of researchers in the graph of academic collaborations where only the colleagues of at least equal core number are included while the rest are hidden (example at Figure 11).

Finally, [7] offers a well developed tool for degeneracy based graph visualization. An mock-up of what it produces can be found in Fig 12. The nodes are organized in a nested manner that indicates their core number while the degree of the vertices and their proximity among them is also taken into account. Specifically the degree is represented by the size of the vertex while the proximity is displayed by positioning them in relevant proximity at the nested circles.

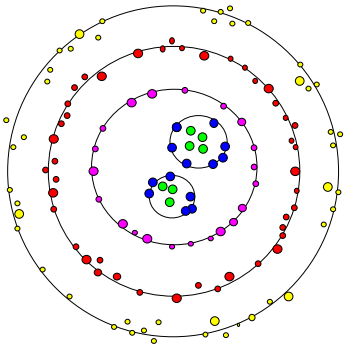


Figure 12: Example of the model for graph visualization with k-cores of [7].

4.6 Dense Subgraph Discovery

A common application of the k-core decomposition is the identification of dense subgraphs; Andersen and Chellapilla [10] were based on this to propose solutions with approximations guarantees for variants of

the densest subgraph problem. In a similar spirit, variants of the community detection problem has been addressed utilizing the properties of k-core decomposition, including local community detection techniques [35] and the influential community search problem [86, 4] where the notion of influence is defined as the minimum weight of the nodes in that community.

4.7 Community Detection

In a recent work on community detection [56],the authors built upon the properties of the decomposition to speed-up the execution time of computationally intensive graph clustering algorithms, such as *spectral clustering*. In particular, they have proposed CoreCluster, an efficient graph clustering framework that can be used along with any known graph clustering algorithm. The approach capitalizes on processing the graph in a hierarchical way provided by its k-core decomposition. That way, the nodes are clustered in an incremental manner that preserve the clustering structure of the graph, while making the execution of the chosen clustering algorithm much faster due to the smaller size of the graph's partitions onto which the algorithm operates.

4.8 Text Analytics

Recently, the concept of k-core decomposition has been also applied in information networks used to represent textual information. Models for graph construction from textual data can be found at [104]. In short, one may consider elements of text (n-grams, single terms etc.) as vertices and as edges the co-occurrence of those elements. The edges can be enriched with additional properties depending on the application. One common approach assigns weights to the edges based on the frequency of term co-occurrence withing a fixed window - where a sliding window is assumed over the text.

A variety of graph based techniques have been utilized on those graphs. For example, [104] utilizes Pagerank and [90] uses HITS for ranking words in text based on their ranking in the corresponding graph. This, in both cases, aims at of the task of keyword extraction. Naturally, the ranking provided from degeneracy has been utilized as well for keyword extraction [121]. As the authors note in their work, beyond better results than the the other aforementioned approaches, the maximal k-core automatically decides the number of keywords in contrast with other methodologies where a fixed number is selected. Later on, Tixier et al. [138] further refined the concept of core (and truss) decomposition for the task of keyword extraction. Overall, the additional benefit of flexibility on the properties of the graph motivated the utilization of advanced approaches and applications of the k-core in text mining.

One such example is found in the later work of detecting events in Twitter streams [101]. The graph of words is build there from tweets in time-windows of fixed size and the detection of the event is based on thresholding the vertex/term weighted degree in the maximal core (i.e. the degrees define the event appearance). Another example [139], utilizes the k-core and k-truss structure for text visualization and summarization as an online real-time application. Moreover, the degeneracy approach has found application on keyword extraction from multiparty conversations [102]. In all cases there is a benefit from the efficient calculation of the k-core which allows almost instant results.

4.9 The Anchored k-Core Problem and Engagement Dynamics in Social Graphs

A common behavior of users in social networks is that their decisions are influenced by that of their neighbors, exhibiting the so-called *positive network effect*: assuming some notion of utility or gain per individual, this is increasing with the number of friends that behave in a certain way. For example, it has been empirically observed that users are more likely to engage to the activities of a social network if their friends do so. Based on that, how can we design or modify a social network in order to maximize the engagement of its users?

Let us assume that all users in a community are initially engaged and each individual has two strategies: to remain engaged in the activities of the community or to drop out [96]. An individual will remain engaged if at least k of his/her friends are engaged (i.e., degree constraint). A user with less than k engaged friends will decide to drop out, and his/her decision might spread over the network forming a cascade of departures (i.e., other individuals might drop out too). When the collapse stops, the remaining engaged users correspond to the k-core subgraph. That way, the size of k-core can be used to measure the overall engagement of the social network. In the related literature, many empirical studies have used the core number of nodes or the size of the maximal k-core subgraph to characterize the engagement properties of individual nodes or even the engagement characteristics of the whole graph [96, 49, 97].

Based on the previously described model of user engagement in social networks, Bhawalkar et al. [19] introduced the *anchored k-core* problem, which examines how to prevent unravelling on the network: we aim at retaining (anchoring) some individuals, so as to maximize the number of users that will remain engaged when the unraveling stops (i.e., the size of the maximal k-core subgraph). Once a node v in G is *anchored*, it is always retained by the k-core decomposition regardless of its degree (i.e., it is never removed by the decomposition) [155, 156].

Definition 14 (Anchored k-core subgraph) *Given an undirected graph G and a vertex set $A \subset G$, the anchored k-core subgraph, denoted by $C_k(G_A)$, is the corresponding k-core of G with vertices in A anchored.*

Definition 15 (Anchored k-core problem) *Given an undirected graph G, a degree constraint k and a budget b, the anchored k-core problem aims at finding a set A of b nodes, such that the size of the resulting anchored k-core, $C_k(G_A)$, is maximized.*

If we have a set A of anchor nodes, then we can directly use the linear time algorithm presented in Section 3 to compute $C_k(G_A)$. However, finding the optimal A is a computational difficult problem; it has been shown that, when $k \geq 3$, the anchored k-core problem is NP-hard [19]. Zhang et al. [155] have proposed an efficient heuristic algorithm, called OLAK, to deal with the complexity constraints of the anchored k-core problem.

4.10 Graph Similarity

In a very recent work [108], the hierarchy of the core decomposition is utilized to provide a general framework for computing similarity metrics among graphs.

Graph similarity is an upcoming topic in the domains of computational biology, chemistry and natural language processing. Simply put, when computing similarities among graphs basic structures (e.g. trees, cycles) are compared between graphs in a local or global level with graph kernels. The aforementioned work [108], contributes by utilizing existing kernels at equivalent core levels between graphs in order to compare the structures at similar levels of connectivity.

Algorithm 12 is straightforward as it accumulates the similarities of a base-kernel along the corresponding cores between two graphs. Despite its simplicity in implementation, it outperforms the utilized baselines in comparison with the result if they (the baselines) were used without the framework. The evaluation is performed on classification tasks of real world graphs from a variety of domains.

4.11 Biology and Ecology

Many interactions in real organisms are modeled as networks. Biological networks have been studied significantly by many different perspectives. One of these, is related to the discovery of core/periphery structure of biological networks. In [92] this idea is studied for protein-protein interactions networks (PPI). It turns out

Algorithm 12: GRAPHSIMKERNEL(G,G')

Input: A pair of graphs G and G'
Result: Result of the kernel function, val

$val = 0$
$\delta_{min} = min(\delta(G), \delta(G'))$
Let C_i, C'_i *be the* $i-$*core of* G, G' *for* $i = 0, \ldots, \delta_{min}$
for $i = \delta_{min}$ *to* 0 **do**
    $val = val + kernel(C_i, C'_i)$

that in addition to the discovery of interesting correlations between connectivity and biological properties, the core/periphery structures help to reveal the existence of multiple levels of protein expression dynamics. Moreover, as reported in [68], residues belonging to inner cores are more conserved than those at the periphery of the network and also it seems that these groups are functionally and structurally critical. Another important result was reported in [41] which studied the relation between the core numbers of proteins and mutation rates. It turns out that the mutation rates for the interior cores is lower.

In addition to homogeneous networks that appear in biology, bipartite networks are also quite frequent: gene-protein, host-pathogen, predator-prey. In [50], two new visualization types are proposed that exploit the structural properties of these networks to improve readability. The basis of these methods is the core decomposition of the bipartite graph.

Other Biology-related works that use the concept of core decomposition include: [39], which applies the concept in weighted biological networks, [116] that performs protein complex prediction and [43] which uses core decomposition in plant metabolic networks.

The core decomposition has been effectively applied to other branches of Biology, such as Ecology. For example, in [51] an application is presented to plot bipartite ecological networks. Also, in [52], the authors study different techniques to identifying the species for which the networks are most vulnerable to cascade extinctions. It turns out that the core decomposition concept sheds light on the understanding of the robustness properties in mutualistic networks. Also, similar ideas can be found in [107], which studies the concept of structural collapse in mutualistic ecosystems. More specifically, based on the authors, it was shown that "when species located at the maximum k-core of the network go extinct as a consequence of sufficiently weak interaction strengths, the system will reach the *tipping point* of its collapse".

4.12 Neuroscience

The study and the understanding of the brain is an ongoing adventure. It turns out the core decomposition concept plays an important role in this study. One of the first works that applied the concept was published in [62]. In that work, the authors performed an in-depth analysis of the brain functional network which is composed of parts of the brain that are functionally interconnected in a dense manner. The main result of this study is that regions of the brain that belong to the structural core, share high degree, strength, and betweenness centrality, and they operate as hubs linking other major structural modules.

In a similar line, the work in [143] demonstrates that parts of the brain with high connectivity (i.e., brain hubs) form a so-called "rich club", which means that there is a tendency for high-degree nodes to be more densely connected among themselves than nodes of a lower degree. The result is that this "rich club" provides important information on the higher-level topology of the brain functional network.

Later, this idea was developed further in [130] in order to compare the brain organization of pigeons and mammals. The main result of this study was that the pigeon *telencephalon*[3] is organized in a similar manner

[3]The telencephalon is the most highly developed and anterior part of the forebrain, composed mainly of the cerebral

to that of a mammal.

Additional research efforts in the area that use the core decomposition concept as a first class citizen include: [122] which studies the influence of wiring cost on the large-scale architecture of human cortical connectivity, [20] which examines the way the brain functional network reorganizes during cognition, [150] which links the concept of *cell assemblies* to that of k-core and studies a specific type of cell assembly called k-assembly and [80] which studies the hierarchical cortical organization of the human brain.

5 Conclusions and Further Research

The core decomposition of a graph is a concept that has been studied for many years and is applied in many different problems in diverse scientific areas – also displaying its value in real world applications. The main reason for its ubiquity lies in the fact that it provides an efficient manner for organizing the graph into hierarchical structures of increasing cohesiveness.

In this survey article, we have provided a thorough review on existing approaches for applications and algorithmic techniques concerning degeneracy-based graph decompositions. In the application domain, most of the works have focused primarily on real-world scale-free networks. A possible explanation is that, random graphs do not exhibit interesting degeneracy-based properties. In other words, there is no actual discrepancy of the core numbers of nodes in random graphs. Concerning algorithmic techniques, we have examined k-core (and its variants) being applied to a multitude of computational models covering many scenarios. One perhaps issue, we notice that in most cases the approaches are tailored by utilizing specialized structures which makes difficult to utilize the same model in a more generic graph analysis scenario (we elaborate on this below).

We have organized this review by three major axes *i*) graph types, *ii*) algorithmic techniques, and *iii*) applications. We note that many publications could be mentioned in all of these axes as the graph type is usually defined in a specific scenario for "not simple" graphs which in turn requires re-definition (or extension of the definition) of degeneracy and consecutively the re-design of the algorithm. For this reason we place each piece work into one of these axes based on its major focus.

To the extend of our knowledge, this survey is the first in attempting to cover degeneracy to this extend. Two notable attempts in similar reviews exist in the general literature as book chapters found in "Encyclopedia of Social Network Analysis and Mining" [22] and in "Cohesive Subgraph Computation over Large Sparse Graphs" [29] both covering to a lesser extend a subset of the main topics presented in this manuscript.

Although core decomposition has been covered by many different perspectives, still there is room for more work in the area. Next, we discuss briefly possible topics of interest for future research.

Algorithmic Aspects. A variety of computational models have been studied for k-core decomposition (in memory, streaming, in parallel) but most of them focus on a single dimension of decomposition based on a function upon properties of the node i.e. the degree, the number of triangles etc. As graph models become more elaborate (direction, label, timestamp), the algorithms are redesigned for a specific computational model. While this is expected (one designs an algorithm specific to the needs of application and core extension), there is a lack for a uniform model regardless of the dimensions across which the core is computed. One of the original works for Generalized Cores [16] attempted at providing such a general model but this would not be applicable in e.g. temporal cores as the additional dimension can be used in a different manner than the degree (which lead to a variety of works specifically for temporal cores).

hemispheres (https://en.wikipedia.org/wiki/Cerebrum).

Through out this review, the reader may notice that different models for graph decomposition try to solve the same problem in principle for computing the decomposition -while they study different graph properties. Never the less, a variety of algorithms has been introduced that solve the same problem (with similar approaches). As such, it is expected in future research for "generalized cores" to re-appear for a greater extension of dimensions and computational models.

Furthermore, there is a lot of potential for graph computational models in general in distributed environments due to the advances in this domain both in academia but in the industry as well combined with need for scalable algorithms on fast evolving data. While we cover here cases for distributed, parallel and streaming computation for large graphs, these approaches are yet to be adopted in general in real world applications and for now are specific implementations. One major factor for this is due to the fact that they require specific data structures and designs that do not generalize in usability by other algorithms e.g. the model used in incremental k-core decomposition [123] is very efficient but it might not be very efficient in other graph analysis tasks like eigen-centrality computation or triangle counting. As such, it is hard to adopt a solution if it requires a great cost for transformation in order to use other graph mining algorithms.

Core Decomposition in Machine Learning. While we saw here the potential of k-core decomposition even in graphs that are extracted from not graph data (in graphs of words), it seems that it has not been explored in too many cases in the domain of Machine Learning or Data Science in general. While the topic is beyond the scope of this review, there are many cases where a graph structure is assumed as representation of the data in some form (e.g. manifold based decomposition [137], spectral clustering on none graph data through affinity kernels [145]).

Despite that graph structures are used quite frequently through data transformations and affinity kernels, the k-core structure of those graphs has not even been examined for its properties and potentials (e.g. feature extraction). This is somewhat, surprising as the k-core algorithm is quite efficient and -as it has been seen in this review- it offers the potential of finding out-liers or anomalies in real-world graphs. Of course, other types of data (e.g., points in the Euclidean space) will not have the same properties as the graphs that are usually studied with degeneracy. Nevertheless, at the minimum a study is lacking on the properties of "other" graphs.

Representation Learning on Graphs. A recent work (core2vec [127]) showed the potential of using k-core decomposition in feature engineering with deep learning techniques. In general this work lies in the domain of representing nodes with latent features – as an embedding in a multidimensional space – that are learned automatically. Usually, this embedding represents similar vertices as vectors with a high similarity in the vector space model. The aforementioned work describes an alternative to using random walk during one of the phases of an existing technique, by biasing the walk with core-based information (focusing the walk in the same core). Although this can be considered a marginal modification, it offers motivation on working in this sub-domain of machine learning. We have seen throughout this review a variety of core models with different interpretations – capturing different (structural) properties of the vertices and subgraphs in general. While these properties can be considered as features, one could utilize them in a similar way to the core2vec model (e.g., as prepossessing techniques), in order to optimize graph embedding techniques.

Influence Maximization. The topic of influence maximization and the identification of influential spreaders has received a lot of attention in the domain of online social networks and recommender systems due to its economical applications but also due to the general emergence of interest in the study of information diffusion.

As we have seen in Section 4, the k-core has played a major role in this domain and further improvements

did not involve only the extension of the k-core but also its combination with other methodologies. As only a few basic models have been studied, the exploration of the more elaborate ones remains still an open subject (e.g., in directed graphs). Moreover, any lateral property defined on this topic will always have the potential of being combined with the k-core structure in similar manners to the ones reviewed. Lastly, the k-core has mainly been used so far as a heuristic algorithm to identify nodes with good spreading properties. It remains open problem how to combine the properties of the core decomposition with the greedy algorithm by Kempe et al. [71], towards a scalable core-based influence maximization algorithm with theoretical guarantees.

References

[1] A. Adiga and A. K. S. Vullikanti. How robust is the core of a network? In H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 541–556, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[2] C. C. Aggarwal, editor. *Social Network Data Analytics*. Springer, 2011.

[3] C. C. Aggarwal and H. Wang. *Managing and Mining Graph Data*. Springer, 2010.

[4] E. Akbas and P. Zhao. Truss-based community search: A truss-equivalence based indexing approach. *Proceedings of the VLDB Endowment*, 10(11):1298–1309, Aug. 2017.

[5] M. A. Al-garadi, K. D. Varathan, and S. D. Ravana. Identification of influential spreaders in online social networks using interaction weighted k-core decomposition method. *Physica A: Statistical Mechanics and its Applications*, 468:278–288, 2017.

[6] J. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. K-core decomposition: A tool for the visualization of large scale networks. *Adv. Neural Inf. Process. Syst.*, 18, 04 2005.

[7] J. I. Alvarez-hamelin, A. Barrat, and A. Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *NIPS '06: Advances in Neural Information Processing Systems*, pages 41–50, 2006.

[8] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. k-core decomposition: a tool for the analysis of large scale internet graphs, 2005.

[9] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. k-core decomposition of internet graphs: Hierarchies, self-similarity and measurement biases. *NHM*, 3(2):371, 2008.

[10] R. Andersen and K. Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, pages 25–37, 2009.

[11] D. Angluin and J. Chen. Learning a hidden graph using o(logn) queries per edge. *J. Comput. Syst. Sci.*, 74(4):546–556, June 2008.

[12] S. Aridhi, M. Brugnara, A. Montresor, and Y. Velegrakis. Distributed k-core decomposition and maintenance in large dynamic graphs. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*, DEBS '16, pages 161–168, New York, NY, USA, 2016. ACM.

[13] J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[14] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8(2009):361–362, 2009.

[15] V. Batagelj, A. Mrvar, and M. Zaveršnik. Partitioning approach to visualization of large graphs. In *International Symposium on Graph Drawing*, pages 90–97. Springer, 1999.

[16] V. Batagelj and M. Zaversnik. Generalized cores. *CoRR*, cs.DS/0202039, 2002.

[17] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks, 2003. cite arxiv:cs/0310049.

[18] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. 353(6295):163–166, 2016.

[19] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. In *ICALP '11: Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming*, pages 440–451, 2011.

[20] M. Bola and B. Sabel. Dynamic reorganization of brain functional networks during cognition. *NeuroImage*, 114, 03 2015.

[21] P. Boldi and S. Vigna. The webgraph framework i: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 595–602, New York, NY, USA, 2004. ACM.

[22] F. Bonchi, F. Gullo, and A. Kaltenbrunner. *Core Decomposition of Massive, Information-Rich Graphs*, pages 1–11. Springer New York, New York, NY, 2017.

[23] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *KDD*, pages 1316–1325, 2014.

[24] U. Brandes. Social network analysis and visualization [applications corner]. *IEEE Signal Processing Magazine*, 25(6), 2008.

[25] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[26] P. Brown and J. Feng. Measuring user influence on twitter using modified k-shell decomposition. In *The Social Mobile Web*, volume WS-11-02 of *AAAI Workshops*. AAAI, 2011.

[27] S. Carmi, S. Havlin, S. Kirkpatrick, Y. Shavitt, and E. Shir. A model of internet topology using k-shell decomposition. *PNAS*, 104(27):11150–11154, 2007.

[28] L. Chang and L. Qin. *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer, 2018.

[29] Q.-L. Chang, Lijun. *Minimum Degree-Based Core Decomposition*, pages 21–39. Springer Series in the Data Sciences.

[30] J. Cheng, Y. Ke, S. Chu, and M. T. Ozsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[31] S.-T. Cheng, Y.-C. Chen, and M.-S. Tsai. Using k-core decomposition to find cluster centers for k-means algorithm in graphx on spark. In *he Eighth International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 93–98, 2017.

[32] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 2008.

[33] P. Colomer-de Simón, M. A. Serrano, M. G. Beiró, J. I. Alvarez-Hamelin, and M. Boguná. Deciphering the global organization of clustering in real complex networks. *Scientific reports*, 3:2517, 2013.

[34] D. J. Cook and L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.

[35] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[36] M. Danisch, T.-H. H. Chan, and M. Sozio. Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 233–242, 2017.

[37] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[38] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. k-core organization of complex networks. *Physical Review Letters*, 96:040601, 2006.

[39] M. Eidsaa. *Core Decomposition Analysis of Weighted Biological Networks*. PhD thesis, NTNU, 2016.

[40] M. Eidsaa and E. Almaas. s-core network decomposition: A generalization of k-core analysis to weighted networks. *Phys. Rev. E*, 88:062819, 2013.

[41] A. I. Emerson, S. Andrews, I. Ahmed, T. K. Azis, and J. A. Malek. K-core decomposition of a protein domain co-occurrence network reveals lower cancer mutation rates for interior cores. *Journal of Clinical Bioinformatics*, 5(1):1, Mar 2015.

[42] P. Erds and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungarica*, 17(1-2):61–99, 1966.

[43] H. A. Filho, J. Machicao, and O. M. Bruno. A hierarchical model of metabolic machinery based on the kcore decomposition of plant metabolic networks. *PLOS ONE*, 13(5):1–15, 05 2018.

[44] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[45] E. C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982.

[46] E. Galimberti, A. Barrat, F. Bonchi, C. Cattuto, and F. Gullo. Mining (maximal) span-cores from temporal networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 107–116. ACM, 2018.

[47] E. Galimberti, F. Bonchi, and F. Gullo. Core decomposition and densest subgraph in multilayer networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1807–1816, New York, NY, USA, 2017. ACM.

[48] A. Garas, F. Schweitzer, and S. Havlin. A k-shell decomposition method for weighted networks. *New Journal of Physics*, 14(8), 2012.

[49] D. Garcia, P. Mavrodiev, and F. Schweitzer. Social resilience in online communities: The autopsy of friendster. In *COSN '13: Proceedings of the First ACM Conference on Online Social Networks*, pages 39–50, 2013.

[50] J. Garcia-Algarra, J. Pastor, M. L. Mouronte, and J. Galeano. A structural approach to disentangle the visualization of bipartite biological networks. *Complexity*, 2018:1–11, 02 2018.

[51] J. Garcia-Algarra, J. M. M. Pastor, M. L. Mouronte, and J. Galeano. Bipartgraph: An interactive application to plot bipartite ecological networks. *bioRxiv*, 2017.

[52] J. Garca-Algarra, J. Pastor, J. Iriondo, and J. Galeano. Ranking of critical species to preserve the functionality of mutualistic networks using the k-core decomposition. *PeerJ*, 5(e3321), 2017.

[53] C. Giatsidis, K. Berberich, D. M. Thilikos, and M. Vazirgiannis. Visual exploration of collaboration networks based on graph degeneracy. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1512–1515. ACM, 2012.

[54] C. Giatsidis, B. Cautis, S. Maniu, D. M. Thilikos, and M. Vazirgiannis. Quantifying trust dynamics in signed graphs, the s-cores approach. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 668–676, 2014.

[55] C. Giatsidis, B. Cautis, S. Maniu, D. M. Thilikos, and M. Vazirgiannis. Quantifying trust dynamics in signed graphs, the s-cores approach. In *SDM*, pages 668–676, 2014.

[56] C. Giatsidis, F. D. Malliaros, D. M. Thilikos, and M. Vazirgiannis. Corecluster: A degeneracy based graph clustering framework. In *AAAI '14: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 44–50, 2014.

[57] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. D-cores: Measuring collaboration of directed graphs based on degeneracy. In *ICDM '11: Proceedings of the 11th IEEE International Conference on Data Mining*, pages 201–210, 2011.

[58] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. Evaluating cooperation in communities with the k-core structure. In *ASONAM '11: Proceedings of the International Conference on Advances in Social Networks Analysis and Mining*, pages 87–93, 2011.

[59] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowl. Inf. Syst.*, 35(2):311–343, 2013.

[60] P. Govindan, S. Soundarajan, T. Eliassi-Rad, and C. Faloutsos. Nimblecore: A space-efficient external memory algorithm for estimating core numbers. In *ASONAM*, pages 207–214. IEEE Computer Society, 2016.

[61] P. Govindan, C. Wang, C. Xu, H. Duan, and S. Soundarajan. The k-peak decomposition: Mapping the global structure of graphs. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1441–1450, 2017.

[62] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, and O. Sporns. Mapping the structural core of human cerebral cortex. *PLOS Biology*, 6(7):e159, 2008.

[63] X. He, H. Zhao, W. Cai, G.-G. Li, and F.-D. Pei. Analyzing the structure of earthquake network by k-core decomposition. *Physica A: Statistical Mechanics and its Applications*, 421:34–43, 2015.

[64] J. Healy, J. Janssen, E. Milios, and W. Aiello. Characterization of graphs using degree cores. In *WAW '08: Algorithms and Models for the Web-Graph*, pages 137–148, 2008.

[65] L. Hébert-Dufresne, A. Allard, J.-G. Young, and L. J. Dubé. Percolation on random networks with arbitrary k-core structure. volume 88, page 062820. APS, 2013.

[66] X. Hu, F. Liu, V. Srinivasan, and A. Thomo. k-core decomposition on giraph and graphchi. In L. Barolli, I. Woungang, and O. K. Hussain, editors, *Advances in Intelligent Networking and Collaborative Systems*, pages 274–284, Cham, 2018. Springer International Publishing.

[67] X. Huang, W. Lu, and L. V. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 77–90, 2016.

[68] A. E. Isaac and S. Sinha. Analysis of core–periphery organization in protein contact networks reveals groups of structurally and functionally critical residues. *Journal of Biosciences*, 40(4):683–699, Oct 2015.

[69] H. Kabir and K. Madduri. Parallel k-core decomposition on multicore platforms. In *IPDPS Workshops*, pages 1482–1491. IEEE Computer Society, 2017.

[70] V. Kassiano, A. Gounaris, A. N. Papadopoulos, and K. Tsichlas. Mining uncertain graphs: An overview. In T. Sellis and K. Oikonomou, editors, *Algorithmic Aspects of Cloud Computing*, pages 87–116, Cham, 2017. Springer International Publishing.

[71] D. Kempe, J. Kleinberg, and E. Tardos. 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 '03, pages 137–146, 2003.

[72] 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.

[73] L. M. Kirousis and D. M. Thilikos. The linkage of a graph. *SIAM J. Comput.*, 25(3):626–647, 1996.

[74] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljerosand, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature Physics*, 2010.

[75] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. The web as a graph. In *PODS*, 2000.

[76] J. Kunegis, A. Lommatzsch, and C. Bauckhage. The slashdot zoo: Mining a social network with negative edges. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 741–750, New York, NY, USA, 2009. ACM.

[77] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. D. Luca, and S. Albayrak. Spectral analysis of signed graphs for clustering, prediction and visualization. In *SDM*, pages 559–570. SIAM, 2010.

[78] A. Kyrola, G. Blelloch, and C. Guestrin. Graphchi: Large-scale graph computation on just a pc. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 31–46, Berkeley, CA, USA, 2012. USENIX Association.

[79] L. L˙ Vital nodes identification in complex networks. *Physics Reports*, 650:1 – 63, 2016. Vital nodes identification in complex networks.

[80] N. Lahav, B. Ksherim, E. Ben-Simon, A. Maron-Katz, R. Cohen, and S. Havlin. K -shell decomposition reveals hierarchical cortical organization of the human brain. *New Journal of Physics*, 18(8):083013, 2016.

[81] S. Lahiri, S. R. Choudhury, and C. Caragea. Keyword and keyphrase extraction using centrality measures on collocation networks. *CoRR*, 2014.

[82] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *WWW '08: Proceedings of the 17th International Conference on World Wide Web*, pages 915–924, 2008.

[83] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Signed networks in social media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1361–1370, New York, NY, USA, 2010. ACM.

[84] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[85] M. Li, W. Zhou, and L. Gao. S-kcore: A social-aware kcore decomposition algorithm in pocket switched networks. In *2010 IEEE/IFIP 8th International Conference on Embedded and Ubiquitous Computing (EUC 2010)(EUC)*, volume 00, pages 737–742, 12 2010.

[86] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *Proceedings of the VLDB Endowment*, 8(5):509–520, 2015.

[87] R.-H. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering*, 26(10):2453–2465, 2014.

[88] D. R. Lick and A. T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22:1082–1096, 1970.

[89] J.-H. Lin, Q. Guo, W.-Z. Dong, L.-Y. Tang, and J.-G. Liu. Identifying the node spreading influence with largest k-core values. *Physics Letters A*, 378(45):3279–3284, 2014.

[90] M. Litvak and M. Last. Graph-based keyword extraction for single-document summarization. In *Proceedings of the workshop on Multi-source Multilingual Information Extraction and Summarization*, pages 17–24. Association for Computational Linguistics, 2008.

[91] L. Lü, T. Zhou, Q.-M. Zhang, and H. E. Stanley. The h-index of a network node and its relation to degree and coreness. *Nature Communications*, 7:10168 EP –, 01 2016.

[92] F. Luo, B. Li, X.-F. Wan, and R. H. Scheuermann. Core and periphery structures in protein interaction networks. *BMC Bioinformatics*, 10(Suppl 4):s8, 2009.

[93] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[94] F. D. Malliaros, A. N. Papadopoulos, and M. Vazirgiannis. Core decomposition in graphs: Concepts, algorithms and applications. In *EDBT*, pages 720–721. OpenProceedings.org, 2016.

[95] F. D. Malliaros, M.-E. G. Rossi, and M. Vazirgiannis. Locating influential nodes in complex networks. *Scientific reports*, 6:19307, 2016.

[96] F. D. Malliaros and M. Vazirgiannis. To stay or not to stay: modeling engagement dynamics in social graphs. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13*, pages 469–478, 2013.

[97] F. D. Malliaros and M. Vazirgiannis. Vulnerability assessment in social networks under cascade-based node departures. *EPL (Europhysics Letters)*, 110(6):68006, 2015.

[98] A. Mandal and M. A. Hasan. A distributed k-core decomposition algorithm on spark. In *Proceedings of the 2017 IEEE International Conference on Big Data (Big Data)*, BIG DATA '17, pages 976–981. IEEE Computer Society, 2017.

[99] C. Z. Marshak. *Applications of Network Science to Criminal Networks, University Education, and Ecology*. PhD thesis, UCLA, 2017.

[100] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983.

[101] P. Meladianos, G. Nikolentzos, F. Rousseau, Y. Stavrakas, and M. Vazirgiannis. Degeneracy-based real-time sub-event detection in twitter stream. In *ICWSM*, pages 248–257, 2015.

[102] P. Meladianos, A. Tixier, I. Nikolentzos, and M. Vazirgiannis. Real-time keyword extraction from conversations. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, volume 2, pages 462–467, 2017.

[103] P. Meyer, H. Siy, and S. Bhowmick. Identifying important classes of large software systems through k-core decomposition. *Advances in Complex Systems*, 17:1550004, 04 2015.

[104] R. Mihalcea and P. Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, 2004.

[105] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. In *PODC*, pages 207–208, 2011.

[106] A. Montresor, F. De Pellegrini, and D. Miorandi. Distributed k-core decomposition. *IEEE Transactions on Parallel and Distributed Systems*, 24(2):288–300, 2013.

[107] F. Morone, G. Ferraro, and H. A. Makse. The k-core as a predictor of structural collapse in mutualistic ecosystems. *Nature Physics*, 10 2018.

[108] G. Nikolentzos, P. Meladianos, S. Limnios, and M. Vazirgiannis. A degeneracy framework for graph similarity. In *IJCAI*, pages 2595–2601, 2018.

[109] M. P. O'Brien and B. D. Sullivan. Locally estimating core numbers. In *ICDM*, pages 460–469, 2014.

[110] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. The pursuit of a good possible world: Extracting representative instances of uncertain graphs. In *SIGMOD*, pages 967–978, 2014.

[111] P. Parchas, F. Gullo, D. Papadias, and F. Bonchi. Uncertain graph processing through representative instances. *ACM Trans. Database Syst.*, 40(3):20:1–20:39, 2015.

[112] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001.

[113] K. Pechlivanidou, D. Katsaros, and L. Tassiulas. Mapreduce-based distributed k-shell decomposition for online social networks. *2014 IEEE World Congress on Services*, 0:30–37, 2014.

[114] S. Pei and H. A. Makse. Spreading dynamics in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2013(12):P12002, 2013.

[115] S. Pei, L. Muchnik, J. S. Andrade Jr, Z. Zheng, and H. A. Makse. Searching for superspreaders of information in real-world social media. *Scientific reports*, 4:5547, 2014.

[116] M. Pellegrini, M. Baglioni, and F. Geraci. Protein complex prediction for large protein protein interaction networks with the core & peel method. *BMC Bioinformatics*, 17(12):372, Nov 2016.

[117] Y. Peng, Y. Zhang, W. Zhang, X. Lin, and L. Qin. Efficient probabilistic k-core computation on uncertain graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1192–1203, 2018.

[118] E. M. Phizicky and S. Fields. Protein-protein interactions: methods for detection and analysis. *Microbiological reviews*, 59 1:94–123, 1995.

[119] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, pages 997–1008, 2010.

[120] S. Qing, J. Liao, J. Wang, X. Zhu, and Q. Qi. Hybrid virtual network embedding with k-core decomposition and time-oriented priority. In *ICC*, pages 2695–2699, 2012.

[121] F. Rousseau and M. Vazirgiannis. Main core retention on graph-of-words for single-document keyword extraction. In *ECIR '15: Proceedings of the 37th European Conference on Information Retrieval*, pages 382–393, 2015.

[122] D. Samu, A. K. Seth, and T. Nowotny. Influence of wiring cost on the large-scale architecture of human cortical connectivity. *PLOS Computational Biology*, 10(4):1–24, 04 2014.

[123] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek. Incremental k-core decomposition: algorithms and evaluation. *The VLDB Journal*, 25(3):425–447, Jun 2016.

[124] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek. Streaming algorithms for k-core decomposition. *Proceedings of the VLDB Endowment*, 6(6):433–444, Apr. 2013.

[125] A. E. Sariyüce and A. Pinar. Peeling bipartite networks for dense subgraph discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM*, pages 504–512, 2018.

[126] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 927–937, 2015.

[127] S. Sarkar, A. Bhagwat, and A. Mukherjee. Core2vec: A core-preserving feature learning framework for networks. In *IEEE/ACM 2018 International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2018*, pages 487–490, 2018.

[128] S. B. Seidman. Network Structure and Minimum Degree. *Social Networks*, 5:269–287, 1983.

[129] N. Shailaja Dasari, D. Ranjan, and M. Zubair. Park: An efficient algorithm for k-core decomposition on multicore processors. *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*, pages 9–16, 01 2015.

[130] M. Shanahan, V. Bingman, T. Shimizu, M. Wild, and O. Gntrkn. Large-scale network organization in the avian forebrain: a connectivity matrix and theoretical analysis. *Frontiers in Computational Neuroscience*, 7:89, 2013.

[131] K. Shin, T. Eliassi-Rad, and C. Faloutsos. Corescope: Graph mining using k-core analysis - patterns, anomalies and algorithms. In *ICDM*, pages 469–478. IEEE, 2016.

[132] K. Shin, T. Eliassi-Rad, and C. Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowl. Inf. Syst.*, 54(3):677–710, 2018.

[133] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[134] P. Strouthopoulos and A. N. Papadopoulos. Core discovery in hidden graphs. *CoRR (to appear in Data and Knowledge Engineering)*, abs/1712.02827, 2017.

[135] Y. Tao, C. Sheng, and J. Li. Finding maximum degrees in hidden bipartite graphs. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 891–902, New York, NY, USA, 2010. ACM.

[136] N. Tatti and A. Gionis. Density-friendly graph decomposition. In *WWW*, pages 1089–1099, 2015.

[137] J. B. Tenenbaum, V. De Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[138] A. Tixier, F. D. Malliaros, and M. Vazirgiannis. A graph degeneracy-based approach to keyword extraction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1860–1870. Association for Computational Linguistics, 2016.

[139] A. Tixier, K. Skianis, and M. Vazirgiannis. Gowvis: a web application for graph-of-words-based text visualization and summarization. *Proceedings of ACL-2016 System Demonstrations*, pages 151–156, 2016.

[140] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 837–846. ACM, 2009.

[141] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the facebook social graph, 2011. cite arxiv:1111.4503Comment: 17 pages, 9 figures, 1 table.

[142] E. Valari, M. Kontaki, and A. N. Papadopoulos. Discovery of top-k dense subgraphs in dynamic graph collections. In A. Ailamaki and S. Bowers, editors, *Scientific and Statistical Database Management*, pages 213–230, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[143] M. P. van den Heuvel and O. Sporns. Rich-club organization of the human connectome. *Journal of Neuroscience*, 31(44):15775–15786, 2011.

[144] T. Verma, F. Russmann, N. Arajo, J. Nagler, and H. Herrmann. Emergence of coreperipheries in networks. *Nature Communications*, 7, 2016.

[145] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[146] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment*, 5(9):812–823, May 2012.

[147] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 233–244, 2018.

[148] N. Wang, D. Yu, H. Jin, C. Qian, X. Xie, and Q. Hua. Parallel algorithm for core maintenance in dynamic graphs. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, volume 00, pages 2366–2371, June 2017.

[149] T. White. *Hadoop: The Definitive Guide*. O'Reilly, 4 edition, 2015.

[150] C. I. Wood and I. V. Hicks. The minimal k-core problem for modeling k-assemblies. *The Journal of Mathematical Neuroscience (JMN)*, 5(1):14, Jul 2015.

[151] H. Wu, J. Cheng, Y. Lu, Y. Ke, Y. Huang, D. Yan, and H. Wu. Core decomposition in large temporal graphs. In *BigData*, pages 649–658. IEEE, 2015.

[152] D. Yan, J. Cheng, Y. Lu, and W. Ng. Blogel: A block-centric framework for distributed computation on real-world graphs. *Proceedings of the VLDB Endowment*, 7(14):1981–1992, 2014.

[153] M. L. Yiu, E. Lo, and J. Wang. Identifying the most connected vertices in hidden bipartite graphs using group testing. *IEEE Transactions on Knowledge & Data Engineering*, 25:2245–2256, 10 2013.

[154] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, Oct. 2016.

[155] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. Olak: An efficient algorithm to prevent unraveling in social networks. *Proceedings of the VLDB Endowment*, 10(6):649–660, Feb. 2017.

[156] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 245–251, 2017.

[157] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: Efficient (k,r)-core computation on social networks. *Proceedings of the VLDB Endowment*, 10(10):998–1009, June 2017.

[158] G.-Q. Zhang, G.-Q. Zhang, Q.-F. Yang, S.-Q. Cheng, and T. Zhou. Evolution of the Internet and its cores. *New Journal of Physics*, 10(12):123027+, Dec. 2008.

[159] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE '12: Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, pages 1049–1060, 2012.

[160] R. Zhuo-Ming, L. Jian-Guo, S. Feng, H. Zhao-Long, and G. Qiang. Analysis of the spreading influence of the nodes with minimum k-shell value in complex networks. *Acta Physica Sinica*, 62(10):108902, 2013.

[161] V. Zlatić, D. Garlaschelli, and G. Caldarelli. Networks with arbitrary edge multiplicities. *EPL (Europhysics Letters)*, 97(2):28005, 2012.

[162] Z. Zou and R. Zhu. Truss decomposition of uncertain graphs. *Knowledge and Information Systems*, 50(1):197–230, Jan. 2017.