Core Decomposition in Multilayer Networks: Theory, Algorithms, and Applications

EDOARDO GALIMBERTI, ISI Foundation and University of Turin
FRANCESCO BONCHI, ISI Foundation and Eurecat
FRANCESCO GULLO, UniCredit
TOMMASO LANCIANO, Sapienza University of Rome

Multilayer networks are a powerful paradigm to model complex systems, where multiple relations occur between the same entities. Despite the keen interest in a variety of tasks, algorithms, and analyses in this type of network, the problem of extracting dense subgraphs has remained largely unexplored so far.

As a first step in this direction, in this work, we study the problem of *core decomposition of a multilayer network*. Unlike the single-layer counterpart in which cores are all nested into one another and can be computed in linear time, the multilayer context is much more challenging as no total order exists among multilayer cores; rather, they form a lattice whose size is exponential in the number of layers. In this setting, we devise three algorithms, which differ in the way they visit the core lattice and in their pruning techniques. We assess time and space efficiency of the three algorithms on a large variety of real-world multilayer networks.

We then move a step forward and study the problem of extracting the *inner-most* (also known as *maximal*) cores, i.e., the cores that are not dominated by any other core in terms of their core index in all the layers. inner-most cores are typically orders of magnitude less than all the cores. Motivated by this, we devise an algorithm that effectively exploits the maximality property and extracts inner-most cores directly, without first computing a complete decomposition. This allows for a consistent speed up over a naïve method that simply filters out non-inner-most ones from all the cores.

Finally, we showcase the multilayer core-decomposition tool in a variety of scenarios and problems. We start by considering the problem of *densest-subgraph extraction in multilayer networks*. We introduce a definition of multilayer densest subgraph that tradesoff between high density and number of layers in which the high density holds, and exploit multilayer core decomposition to approximate this problem with quality guarantees. As further applications, we show how to utilize multilayer core decomposition to speed-up the extraction of *frequent cross-graph quasi-cliques* and to generalize the *community-search* problem to the multilayer setting.

CCS Concepts: • **Mathematics of computing** → **Graph algorithms**; *Graph theory*; *Approximation algorithms*; • **Information systems** → Data mining;

Additional Key Words and Phrases: Graph mining, multilayer networks, core decomposition, dense-subgraph extraction, cliques and quasi-cliques, community search

Authors' addresses: E. Galimberti and F. Bonchi are at ISI Foundation, Via Chisola 5, 10126, Torino, Italy; emails: {edoardo.galimberti, francesco.bonchi}@isi.it; F. Gullo is at Unicredit R&D Dept., Via Molfetta, 101, 00171 Rome, Italy; email: gullof@acm.org; T. Lanciano is at Sapienza University of Rome, Via Ariosto, 25, 00185 Roma, Italy; email: lanciano@diag.uniroma1.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1556-4681/2020/01-ART11 $15.00
https://doi.org/10.1145/3369872

ACM Reference format:
Edoardo Galimberti, Francesco Bonchi, Francesco Gullo, and Tommaso Lanciano. 2020. Core Decomposition in Multilayer Networks: Theory, Algorithms, and Applications. *ACM Trans. Knowl. Discov. Data* 14, 1, Article 11 (January 2020), 40 pages.
https://doi.org/10.1145/3369872

1 INTRODUCTION

In several real-world contexts, such as social media, biological networks, financial networks, and transportation systems, it is common to encounter multiple relations among the objects of the underlying domain. Data in these scenarios are therefore modeled as a graph[1] composed of a superimposition of different layers, i.e., where multiple edges of different types exist between a pair of vertices [23, 28, 56]. In the literature, different terminologies have been used for graphs of this kind: *multilayer networks*, *multiplex networks*, *multidimensional networks*, *multirelational networks*, *multislice networks*, and more. No uniformity exists in this regard, and the various terms may also refer to slightly different concepts. In this work, we deal with networks composed of multiple layers, *with no inter-layer links*, and hereinafter use the term "*multilayer networks*" to refer to them.[2]

Extracting dense structures from large graphs has emerged as a key graph-mining primitive in a variety of scenarios [57], ranging from web mining [43], to biology [36, 55], and finance [29]. Although the literature on multilayer graphs has grown fast in the last years, the problem of extracting dense subgraphs in this type of graph has remained, surprisingly, largely unexplored.

In single-layer graphs, among the many definitions of a dense structure, *core decomposition* plays a central role [21]. The *k-core* of a graph is defined as a maximal subgraph in which every vertex has at least k neighbors within that subgraph. The set of all k-cores of a graph G forms the *core decomposition* of G [69]. The appeal of core decomposition lies in the fact that it can be computed very fast, in linear time [15, 59], while, at the same time, being easily used to speed-up/approximate dense-subgraph extraction according to various other definitions. For instance, core decomposition may speed up maximal-clique finding [31], as a k-clique is guaranteed to be contained into a $(k-1)$-core, which can be significantly smaller than the original graph. Moreover, core decomposition can be used to approximate betweenness centrality [45], and to design linear-time approximation algorithms for the densest-subgraph [54] and the densest at-least-k-subgraph [4] problems.

In this work, we study the problem of *core decomposition in multilayer networks*. A major challenge with respect to the single-layer setting is that the number of multilayer cores is exponential in the number of layers. We achieve this challenge by devising a number of pruning techniques to avoid from-scratch computation of all the cores and exploit such techniques to devise efficient algorithms.

Nevertheless, computing multilayer core decomposition efficiently is not enough. Due the potentially high number of multilayer cores, a further major desideratum is to provide a data analyst with additional tools to browse through the output, and ultimately select the patterns of interest. The situation resembles that of the classic *association-rule* and *frequent-itemset* mining: a potentially exponential output, efficient algorithms to extract all the patterns, the need to have concise summaries of the extracted knowledge, and the opportunity of using the extracted patterns as building blocks for more sophisticated analyses. Following this direction, we present a number of problems and applications built on top of the multilayer core-decomposition tool. First, we focus on

[1] Throughout the article we use the terms "network" and "graph" interchangeably.
[2] Another popular terminology for networks with multiple layers and no inter-layer links is "*multiplex networks*."

the problem of extracting only the *maximal* or, as we call them in this work, the *inner-most cores*, i.e., cores that are not "dominated" by any other core. As experimentally observed, inner-most cores are orders of magnitude less than all the cores. Therefore, it is desirable to design algorithms that effectively exploit the maximality property and extract inner-most cores directly, without first computing a complete decomposition. Then, we show how multilayer core decomposition finds application to the problem of *densest-subgraph extraction in multilayer networks* [25, 49]. As a further application, we exploit multilayer core decomposition to speed-up the extraction of *frequent cross-graph quasi-cliques* [50]. Finally, we exploit multilayer core decomposition to generalize the *community-search* problem [72] to the multilayer setting.

1.1 Background and Related Work

Core decomposition. Given a single-layer graph $G = (V, E)$ and a vertex $u \in V$, let $deg(u)$ and $deg_S(u)$ denote the degree of u in G and in a subgraph S of G, respectively. Also, given a subset $C \subseteq V$ of vertices, let $E[C]$ denote the subset of edges induced by C.

Definition 1 (Core Decomposition). The *k-core* (or core of order k) of a single-layer graph $G = (V, E)$ is a *maximal* subgraph $G[C_k] = (C_k, E[C_k])$ such that $\forall u \in C_k : deg_{C_k}(u) \geq k$. The set $G = C_0 \supseteq C_1 \supseteq \cdots \supseteq C_{k^*}$ of all k-cores ($k^* = \arg\max_k C_k \neq \emptyset$) is the *core decomposition* of G.

Core decomposition can be computed in linear time by iteratively removing the smallest-degree vertex and setting its core number as its degree at the time of removal [15]. Core decomposition has established itself as an important tool for network analysis and visualization [3, 14] in several domains, e.g., bioinformatics [10, 82], software engineering [86], and social networks [41, 52]. It has been studied under various settings, such as distributed [2, 51, 60, 64], streaming [58, 68, 87], and external-memory [26, 79], and for various types of graph, such as uncertain [22], directed [42], weighted [40], and attributed [85] graphs. Core decomposition has been studied also for temporal networks: [80] defines the (k, h)-core, where h accounts for the number of multiple temporal edges between two vertices of degree at least k, while [38] introduces the concept of (maximal) span-core, i.e., a core structure assigned with clear temporal collocation. See [21] for a comprehensive survey.

In this article, we adopt the definition of a multilayer core by Azimi-Tafreshi et al. [9], that is a core identified by an $|L|$-dimensional integer vector $\mathbf{k}$ (with $|L|$ being the number of layers of the given multilayer network), where every component of $\mathbf{k}$ refers to the minimum-degree constraint in the corresponding layer, i.e., $\mathbf{k}[i]$ states the minimum degree required for that core in the ith layer, for all $i \in [1..|L|]$. Apart from introducing the multilayer-core definition, Azimi-Tafreshi et al. study the core-percolation problem from a physics standpoint, with no algorithmic contribution: they characterize cores of two-layer Erdős–Rényi and scale-free networks, and observe how this characterization fits real-world air-transportation networks.

To the best of our knowledge, *no prior work has studied how to efficiently compute the complete core decomposition of multilayer networks.*

Densest subgraph. Several notions of *density* exist in the literature, each of which leading to a different version of the dense-subgraph-discovery problem. While most variants are **NP**-hard and/or inapproximable, extracting dense subgraphs according to the *average-degree density* is solvable in polynomial time [44]. As a result, such a density has attracted most of the research in the field, so that the subgraph maximizing the average degree is commonly referred to as the *densest subgraph.* Goldberg [44] provides an exact algorithm for finding the densest subgraph, which is based on iteratively solving ad-hoc-defined minimum-cut problem instances. Although principled, the Goldberg's algorithm cannot scale to large graphs. Asahiro et al. [8] devises a linear-time greedy algorithm that has been shown to achieve $\frac{1}{2}$-approximation guarantee by Charikar [24]. Such a

greedy algorithm iteratively removes the smallest-degree vertex, and, among all the subgraphs yielded during this vertex-removal process, the densest one is ultimately output. Note that this algorithm resembles the one used for core decomposition. In fact, it can be proved that the innermost core of a graph is itself a $\frac{1}{2}$-approximation of the densest subgraph.

Variants of the densest-subgraph problem with size constraints turn out to be **NP**-hard. For these variants, approximation algorithms and other theoretical results have been presented [4, 6, 7, 35]. A number of works focus on extracting a subgraph maximizing densities other than the average degree. For instance, Tsourakakis et al. [77] resort to the notion of quasi-clique, while Tsourakakis [76] and Wang et al. [78] focus on notions of density based on k-cliques and/or triangles. The densest-subgraph problem has also been studied in different settings, such as streaming/dynamic context [11, 17, 30] and top-k fashion [12, 37, 62].

Dense structures in multilayer networks. Several recent works have dealt with the problem of extracting dense subgraphs from a set of multiple graphs sharing the same vertex set, which is a setting equivalent to the multilayer one we study in this work. Jethava and Beerenwinkel [49] define the *DCS* problem, i.e., find a subgraph maximizing the minimum average degree over *all* input graphs, and devise a linear-programming formulation and a greedy heuristic for it. Reinthal et al. [65] provide a Lagrangian relaxation of the Jethava and Beerenwinkel's linear program, which can be solved more efficiently. Semertzidis et al. [70] introduce three more variants of the problem, whose goal is to maximize the average average degree, the minimum minimum degree, and the average minimum degree, respectively. They show that the average-average variant reduces to the traditional densest-subgraph problem, and the minimum-minimum variant is polynomial-time solvable by a simple adaptation of the algorithm for core decomposition. They also devise heuristics for the remaining two variants. Charikar et al. [25] further focus on the minimum-average and average-minimum formulations, by providing several theoretical findings, including **NP**-hardness, hardness of the approximation (for both minimum-average and average-minimum), an integrality gap for the linear-programming relaxation introduced in [49, 65] (for minimum-average), and a characterization in terms of parameterized complexity (for average-minimum).

Other contributions in this area, less directly related to our work, deal with specific cases of two-layer networks [71, 81] and with the *community-detection* problem [16, 23, 61, 63, 73, 74, 84]. Boden et al. [19] study *subspace clustering* for multilayer graphs, i.e., find clusters of vertices that are densely connected by edges with similar labels for all possible label sets. Yan et al. [83] introduce the problem of mining *closed relational graphs*, i.e., frequent subgraphs of a multilayer graph exhibiting large minimum cut. Jiang et al. [50] focus on extracting *frequent cross-graph quasi-cliques*, i.e., subgraphs that are quasi-cliques in at least a fraction of layers equal to a certain minimum support and have size larger than a given threshold. Interdonato et al. [48] are the first to study the problem of *local community detection in multilayer networks*, i.e., when a seed vertex is given and the goal is to reconstruct its community by having only a limited local view of the network. Finally, Zhu et al. [88] address the problem of finding the k most diversified d-coherent cores, i.e., the k subgraphs having minimum degree at least d that maximize the coverage of the vertices.

In this work, in Section 5, we introduce a formulation of the densest-subgraph problem in multilayer networks that tradesoff between high density and number of layers where the high density holds. We apply multilayer core decomposition to provide provable approximation guarantees for this problem. We also show that our formulation generalizes the minimum-average densest-common-subgraph problem studied in [25, 49, 65, 70], and our method achieves approximation guarantees for that problem too. Furthermore, in Section 6, we show how to exploit multilayer core decomposition to speed-up the problem of finding frequent cross-graph quasi-cliques [50].

Community search. Given a (single-layer) graph and a set of query vertices, the *community search* problem aims at finding a cohesive subgraph containing the query vertices. Community search has received a great deal of attention in the data-mining community in the last few years (see e.g., a recent tutorial [47]). Sozio and Gionis [72] are the first to introduce the community-search problem, by employing the minimum degree as a cohesiveness measure. Their formulation can be solved by a simple (linear-time) greedy algorithm, which is very similar to the one proposed in [24] for the densest-subgraph problem. More recently, Cui et al. [27] devise a local-search approach to improve the efficiency of the method defined in [72], but only for the special case of a single query vertex. The minimum-degree-based problem has been further studied in [13], by exploiting core decomposition as a preprocessing step to allow more efficient and effective solutions.

Several formulations of the community search have also been studied under different names and in slightly different settings. Andersen and Lang [5] and Kloumann and Kleinberg [53] study *seed set expansion* in social graphs, in order to find communities with small conductance or that are well-resemblant of the characteristics of the query vertices, respectively. Other works define *connectivity subgraphs* based on electricity analogues [32], random walks [75], the minimum-description-length principle [1], the Wiener index [67], and network efficiency [66]. Community search has been formalized for attributed [33, 46] and spatial graphs [34] as well.

In this work, in Section 7, we formulate the community-search problem for multilayer graphs, by adapting the early definition by Sozio and Gionis [72], and show how our algorithms for multilayer core decomposition can be exploited to obtain optimal solutions to this problem.

1.2 Challenges, Contributions, and Roadmap

Let $G = (V, E, L)$ be a multilayer graph, where V is a set of vertices, L is a set of layers, and $E \subseteq V \times V \times L$ is a set of edges. Given an $|L|$-dimensional integer vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, the *multilayer* $\mathbf{k}$*-core* of G is a maximal subgraph whose vertices have at least degree k_ℓ in that subgraph, for all layers $\ell \in L$ [9]. Vector $\mathbf{k}$ is dubbed *coreness vector* of that core. The set of all *non-empty* and *distinct* multilayer cores constitutes the *multilayer core decomposition* of G. A major challenge of computing the core decomposition of a multilayer network is that *the number of multilayer cores are exponential in the number of layers*. This makes the problem inherently hard, as the exponential size of the output clearly precludes the existence of polynomial-time algorithms in the general case. In fact, unlike the single-layer case where cores are all nested into each other, no total order exists among multilayer cores. Rather, they form a *core lattice* defining partial containment. As a result, algorithms in the multilayer setting must be crafted carefully to handle this exponential blowup, and avoid, as much as possible, the computation of unnecessary (i.e., empty or non-distinct) cores.

A naïve way of computing a multilayer core decomposition consists in generating all coreness vectors, run for each vector $\mathbf{k}$, an algorithm that iteratively removes vertices whose degree in a layer ℓ is less than the ℓth component of $\mathbf{k}$, and filter out empty and duplicated cores. This method has evident efficiency issues, as every core is computed from the whole input graph, and it does not avoid generation of empty or non-distinct cores at all. As our first contribution, we devise three more efficient algorithms that exploit effective pruning rules during the visit of the lattice. The first two methods are based on a Breadth First Search (BFS) and a Depth First Search (DFS) strategy, respectively: the BFS method exploits the fact that a core is contained into the intersection of all its fathers in the lattice, while the DFS method iteratively performs a single-layer core decomposition to compute, one-shot, all cores along a path from a non-leaf lattice core to a leaf. The third method adopts a HYBRID strategy embracing the main pros of BFS and DFS, and equipped with a *look-ahead* mechanism to skip non-distinct cores.

We then shift the attention to the problem of computing *all and only the inner-most cores*, i.e., the cores that are not dominated by any other core in terms of their index on all the layers. A

straightforward way of approaching this problem would be to first compute the complete core decomposition, and then filter out the non-inner-most cores. However, as the inner-most cores are usually much less than the overall cores, it would be desirable to have a method that effectively exploits the maximality property and extracts the inner-most ones directly, without computing a complete decomposition. The design of an algorithm of this kind is an interesting challenge, as it contrasts the intrinsic conceptual properties of core decomposition, based on which a core of order k (in one layer) can be efficiently computed from the core of order $k-1$, of which it is a subset, thus naturally suggesting a bottom-up discovery. For this reason, at first glance, the computation of the core of the highest order would seem as hard as computing the overall core decomposition. In this work, we show that, by means of a clever core-lattice visiting strategy, we can prune huge portions of the search space, thus achieving higher efficiency than computing the whole decomposition.

As a major application of multilayer core decomposition, we then focus on the problem of *extracting the densest subgraph from a multilayer network*. As already discussed in Section 1.1, a number of works aim at extracting a subgraph that maximizes the minimum average degree over *all* layers [25, 49, 65, 70]. A major limitation of that formulation is that, considering all layers, even the noisy/insignificant ones would contribute to selecting the output subgraph, which might prevent us from finding a subgraph being dense in a still large subset of layers. Another simplistic approach at the other end of the spectrum corresponds to flattening the input multilayer graph and resorting to single-layer densest-subgraph extraction. However, this would mean disregarding the different semantics of the layers, incurring in a severe information loss. Within this view, in this work we generalize the problem studied in [25, 49, 65, 70] by introducing a formulation that accounts for a tradeoff between high density and number of layers exhibiting the high density. Specifically, given a multilayer graph $G = (V, E, L)$, the average-degree density of a subset of vertices S in a layer ℓ is defined as the number of edges induced by S in ℓ divided by the size of S, i.e., $\frac{|E_\ell[S]|}{|S|}$. We define the *multilayer densest subgraph* as the subset of vertices S^* such that the function

$$\max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S^*]|}{|S^*|} |\hat{L}|^\beta$$

is maximized. Parameter $\beta \in \mathbb{R}^+$ controls the importance of the two problem ingredients, i.e., high density and number of high-density layers. This problem statement naturally achieves the aforementioned desired tradeoff: the larger the subset $\hat{L}$ of selected layers, the smaller the minimum density $\min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|}$ in those layers. Similarly to the single-layer case where core decomposition provides a $\frac{1}{2}$-approximation of the densest subgraph, in this work we show that computing the multilayer core decomposition of the input graph and selecting the core maximizing the proposed multilayer density function achieves a $\frac{1}{2|L|^\beta}$-approximation for the general multilayer-densest-subgraph problem formulation, and a $\frac{1}{2}$-approximation for the all-layer variant in [25, 49, 65, 70].

As a further application of multilayer core decomposition, we show how it can speed up *frequent cross-graph quasi-clique* extraction [50]. We prove that searching for frequent cross-graph quasi-cliques in restricted areas of the graph—corresponding to multilayer cores complying with the quasi-clique condition—is still sound and complete, while also being much more efficient.

Finally, we also provide a generalization of the *community-search* problem [72] to the multilayer setting, and show how to exploit multilayer core decomposition to optimally solve this problem.

Summarizing, this work has the following contributions:

(1) We define the problem of *core decomposition in multilayer networks*, and characterize it in terms of relation to other problems, and complexity. We devise three algorithms that solve multilayer core decomposition efficiently (Section 3).
(2) We devise further algorithms to compute the *inner-most cores* only (Section 4).

(3) We study the problem of *densest-subgraph in multilayer networks*. We introduce a formulation that tradesoff between high density and number of layers exhibiting high density, and exploit multilayer core decomposition to solve it with approximation guarantees (Section 5).
(4) We show how the multilayer core-decomposition tool can be exploited to speed up the extraction of *frequent cross-graph quasi-cliques* (Section 6).
(5) We formulate the *multilayer community-search* problem and show that multilayer core decomposition provides an optimal solution to this problem (Section 7).

We also provide extensive experiments, on numerous real datasets, to assess the performance of our proposals. For each aforementioned context, experiments are provided within the corresponding section. A preliminary version of this work, covering Sections 3 and 5 only, was presented in [39].

Reproducibility. For the sake of reproducibility, all our code and some of the datasets used in this article are available at https://github.com/egalimberti/multilayer_core_decomposition.

2 PRELIMINARIES AND PROBLEM STATEMENTS

In this section, we introduce the needed preliminaries and notation, we provide some fundamental properties of multilayer cores, and then formally define all the problems studied in this work.

2.1 Multilayer Core Decomposition

We are given an undirected multilayer graph $G = (V, E, L)$, where V is a set of vertices, L is a set of layers, and $E \subseteq V \times V \times L$ is a set of edges. Let E_ℓ denote the subset of edges in layer $\ell \in L$. For a vertex $u \in V$ we denote by $deg(u, \ell)$ and $deg(u)$ its degree in layer ℓ and over all layers, respectively, i.e., $deg(u, \ell) = |\{e = (u, v, \ell) : e \in E_\ell\}|$, $deg(u) = |\{e = (u, v, \ell) : e \in E\}| = \sum_{\ell \in L} deg(u, \ell)$.

For a subset of vertices $S \subseteq V$ we denote by $G[S]$ the subgraph of G induced by S, i.e., $G[S] = (S, E[S], L)$, where $E[S] = \{e = (u, v, \ell) \mid e \in E, u \in S, v \in S\}$. For a vertex $u \in V$ we denote by $deg_S(u, \ell)$ and $deg_S(u)$ its degree in subgraph S considering layer ℓ only and all layers, respectively, i.e., $deg_S(u, \ell) = |\{e = (u, v, \ell) : e \in E_\ell[S]\}|$, $deg_S(u) = |\{e = (u, v, \ell) : e \in E[S]\}| = \sum_{\ell \in L} deg_S(u, \ell)$. Finally, let $\mu(\ell)$ and $\mu(\hat{L})$ denote the minimum degree of a vertex in layer ℓ and in a subset $\hat{L} \subseteq L$ of layers, respectively. Let also $\mu(S, \ell)$ and $\mu(S, \hat{L})$ denote the corresponding counterparts of $\mu(\ell)$ and $\mu(\hat{L})$ for a subgraph (induced by a vertex set) S.

A core of a multilayer graph is characterized by an $|L|$-dimensional integer vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, termed *coreness vector*, whose components k_ℓ denote the minimum degree allowed in layer ℓ. This corresponds to the notion of $\mathbf{k}$-core introduced by Azimi-Tafreshi et al. [9]. As discussed in Section 1.1, Azimi-Tafreshi et al. do not study (or devise any algorithm for) the problem of computing the entire multilayer core decomposition. They study core percolation by analyzing *a single* core of interest, computed with the simple iterative-peeling algorithm (Algorithm 1). Formally

Definition 2 (Multilayer Core and Coreness Vector [9]). Given a multilayer graph $G = (V, E, L)$ and an $|L|$-dimensional integer vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, the multilayer $\mathbf{k}$-*core* of G is a *maximal* subgraph $G[C] = (C \subseteq V, E[C], L)$ such that $\forall \ell \in L : \mu(C, \ell) \geq k_\ell$. The vector $\mathbf{k}$ is referred to as the *coreness vector* of $G[C]$.

Given a coreness vector $\mathbf{k}$, we denote by $C_\mathbf{k}$ the corresponding core. Also, as a $\mathbf{k}$-core is fully identified by the vertices belonging to it, we hereinafter refer to it by its vertex set $C_\mathbf{k}$ and the induced subgraph $G[C_\mathbf{k}]$ interchangeably. It is important noticing that a set of vertices $C \subseteq V$ may correspond to multiple cores. For instance, in the graph in Figure 1 the set {A,B,D,E} corresponds to both $(3, 0)$-core and $(3, 1)$-core. In other words, a multilayer core can be described by more than

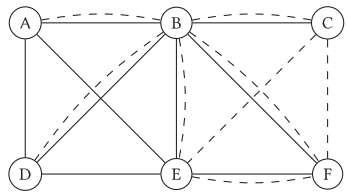

Fig. 1. Example two-layer graph (solid edges refer to the first layer, while dashed edges to the second layer) with the following $\mathbf{k}$-cores: $(0,0) = (1,0) = (0,1) = (1,1) = \{A,B,C,D,E,F\}$, $(2,0) = (2,1) = \{A,B,D,E,F\}$, $(3,0) = (3,1) = \{A,B,D,E\}$, $(0,2) = (1,2) = (0,3) = (1,3) = \{B,C,E,F\}$, $(2,2) = \{B,E,F\}$.

one coreness vector. However, as formally shown next, among such multiple coreness vectors there exists one and only one that is not dominated by any other. We call this vector the *maximal coreness vector* of C. In the example in Figure 1 the maximal coreness vector of $\{A,B,D,E\}$ is $(3,1)$.

Definition 3 (Maximal Coreness Vector). Let $G = (V, E, L)$ be a multilayer graph, $C \subseteq V$ be a core of G, and $\mathbf{k} = [k_\ell]_{\ell \in L}$ be a coreness vector of C. $\mathbf{k}$ is said *maximal* if there does not exist any coreness vector $\mathbf{k}' = [k'_\ell]_{\ell \in L}$ of C such that $\forall \ell \in L : k'_\ell \geq k_\ell$ and $\exists \hat{\ell} \in L : k'_{\hat{\ell}} > k_{\hat{\ell}}$.

Theorem 1. *Multilayer cores have a unique maximal coreness vector.*

Proof. We prove the theorem by contradiction. Assume two maximal coreness vectors $\mathbf{k} = [k_\ell]_{\ell \in L} \neq \mathbf{k}' = [k'_\ell]_{\ell \in L}$ exist for a multilayer core C. As $\mathbf{k} \neq \mathbf{k}'$ and they are both maximal, there exist two layers $\hat{\ell}$ and $\bar{\ell}$ such that $k_{\hat{\ell}} > k'_{\hat{\ell}}$ and $k'_{\bar{\ell}} > k_{\bar{\ell}}$. By definition of multilayer core (Definition 2), it holds that $\forall \ell \in L : \mu(C, \ell) \geq k_\ell, \mu(C, \ell) \geq k'_\ell$. This means that the vector $\mathbf{k}^* = [k^*_\ell]_{\ell \in L}$, with $k^*_\ell = \max\{k_\ell, k'_\ell\}, \forall \ell \in L$, is a further coreness vector of C. For this vector, it holds that $\forall \ell \neq \hat{\ell}, \ell \neq \bar{\ell} : k^*_\ell \geq k'_\ell, k^*_{\hat{\ell}} > k'_{\hat{\ell}}$, and $k^*_{\bar{\ell}} > k_{\bar{\ell}}$. Thus, $\mathbf{k}^*$ dominates both $\mathbf{k}$ and $\mathbf{k}'$, which contradicts the hypothesis of maximality of $\mathbf{k}$ and $\mathbf{k}'$. The theorem follows. □

The first (and main) problem we tackle in this work is the computation of the complete multilayer core decomposition, i.e., the set of all non-empty multilayer cores.

Problem 1 (Multilayer Core Decomposition). *Given a multilayer graph $G = (V, E, L)$, find the set of all* non-empty *and* distinct *cores of G, along with their corresponding maximal coreness vectors. Such a set forms what we hereinafter refer to as the* multilayer core decomposition *of G.*

2.2 Inner-Most Multilayer Cores

Cores of a single-layer graph are all nested one into another. This makes it possible to define the notions of (*i*) *inner-most core*, as the core of highest order, and (*ii*) *core index* (or *core number*) of a vertex u, which is the highest order of a core containing u. In the multilayer setting, the picture is more complex, as multilayer cores are not all nested into each other. As a result, the core index of a vertex is not unambiguously defined, while there can exist multiple inner-most cores:

Definition 4 (Inner-most Multilayer Cores). The *inner-most cores* of a multilayer graph are all those cores with maximal coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ such that there does not exist any other core with coreness vector $\mathbf{k}' = [k'_\ell]_{\ell \in L}$ where $\forall \ell \in L : k'_\ell \geq k_\ell$ and $\exists \hat{\ell} \in L : k'_{\hat{\ell}} > k_{\hat{\ell}}$.

To this purpose, look at the example in Figure 1. It can be observed that: (*i*) cores are not nested into each other, (*ii*) $(3,1)$-core, $(1,3)$-core and $(2,2)$-core are the inner-most cores, and (*iii*) vertices B and E belong to (inner-most) cores $(3,1)$, $(1,3)$, and $(2,2)$, thus making their core index not unambiguously defined.

The second problem we tackle in this work is the development of smart algorithms to compute all the inner-most cores, without the need of computing the complete multilayer core decomposition.

Problem 2 (Inner-most Cores Computation). *Given a multilayer graph $G = (V, E, L)$, find all* non-empty *and inner-most cores of G, along with their corresponding maximal coreness vectors.*

2.3 Multilayer Densest Subgraph

As anticipated in Section 1.2, the densest subgraph of a multilayer graph should provide a good tradeoff between large density and the number of layers where such a large density is exhibited. We achieve this intuition by means of the following optimization problem:

Problem 3 (Multilayer Densest Subgraph). *Given a multilayer graph $G = (V, E, L)$, a positive real number β, and a real-valued function $\delta : 2^V \to \mathbb{R}^+$ defined as*

$$\delta(S) = \max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|} |\hat{L}|^\beta, \tag{1}$$

find a subset $S^* \subseteq V$ of vertices that maximizes function δ, i.e.,

$$S^* = \arg\max_{S \subseteq V} \delta(S).$$

Parameter β controls the importance of the two ingredients of the objective function δ, i.e., density and number of layers exhibiting such a density: the smaller the β, the better the focus on the former aspect (density), and vice versa. Also, as a nice side effect, solving Multilayer Densest Subgraph allows for automatically finding a set of layers of interest for the densest subgraph S^*. In Section 5, we will show how to exploit it to devise an algorithm with approximation guarantees for Multilayer Densest Subgraph, thus extending to the multilayer case the intuition at the basis of the well-known $\frac{1}{2}$-approximation algorithm [8, 24] for single-layer densest subgraph.

2.4 Frequent Cross-Graph Quasi-Cliques

Another interesting insight into the notion of multilayer cores is about their relationship with (quasi-)cliques. In single-layer graphs, it is well-known that cores can speed-up clique finding, as a clique of size k is contained in the $(k-1)$-core. Interestingly, a similar relationship holds in the multilayer context too. Given a multilayer graph $G = (V, E, L)$, a layer $\ell \in L$, and a real number $\gamma \in (0, 1]$, a subgraph $G[S] = (S \subseteq V, E[S], L)$ of G is said to be a *γ-quasi-clique* in layer ℓ if all its vertices have at least $\gamma(|S| - 1)$ neighbors in layer ℓ within S, i.e., $\forall u \in S : deg_S(u, \ell) \geq \gamma(|S| - 1)$. Jiang et al. [50] study the problem of extracting *frequent cross-graph quasi-cliques*, defined next.

Problem 4 (Frequent Cross-graph Quasi-cliques Mining [50]). *Given a multilayer graph $G = (V, E, L)$, a function $\Gamma : L \to (0, 1]$ assigning a real value to every layer in L, a real number* min_sup $\in (0, 1]$, *and an integer* min_size ≥ 1, *find all maximal subgraphs $G[S]$ of G of size larger than* min_size *such that there exist at least* min_sup $\times |L|$ *layers ℓ for which $G[S]$ is a $\Gamma(\ell)$-quasi-clique.*

In Section 6, we will prove that a frequent cross-graph quasi-clique of size K is necessarily contained into a $\mathbf{k}$-core described by a maximal coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ such that there exists a fraction of at least *min_sup* layers ℓ where $k_\ell = \lfloor \Gamma(\ell)(K-1) \rfloor$. Based on this, exploiting multilayer core decomposition as a preprocessing step allows for speeding up any algorithm for Problem 4.

2.5 Multilayer Community Search

The last application we study is the problem of *community search*. Given a graph $G = (V, E)$ and a set $V_Q \subseteq V$ of query vertices, a very wide family of problem requires to find a connected subgraph

H of G, which contains all query vertices V_Q and exhibits an adequate degree of cohesiveness, compactness, or density. This type of problem has been termed in the literature in different ways, e.g., *community search* [13, 27, 72], *seed set expansion* [5, 53], *connectivity subgraphs* [1, 32, 66, 67, 75], just to mention a few: see [47] for a recent survey. In this work we adopt the early definition by Sozio and Gionis [72], which measures the cohesiveness of the resulting subgraph by means of the minimum degree inside the subgraph, and we adapt it to the multilayer setting as follows.

Problem 5 (Multilayer Community Search). *Given a multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, and a set $\hat{L} \subseteq L$ of layers, the minimum degree in the subgraph induced by S and $\hat{L}$ is*

$$\varphi(S, \hat{L}) = \min_{\ell \in \hat{L}} \min_{u \in S} deg_S(u, \ell).$$

Given a positive real number β, we define a real-valued density function $\vartheta : 2^V \to \mathbb{R}^+$ as

$$\vartheta(S) = \max_{\hat{L} \subseteq L} \varphi(S, \hat{L}) |\hat{L}|^{\beta}.$$

Given a set $V_Q \subseteq V$ of query vertices, find a subgraph containing all the query vertices and maximizing the density function, i.e.,

$$S^* = \arg\max_{V_Q \subseteq S \subseteq V} \vartheta(S). \tag{2}$$

In Section 7, we will show how to adapt multilayer core decomposition to solve Problem 5.

3 ALGORITHMS FOR MULTILAYER CORE DECOMPOSITION

A major challenge of Multilayer Core Decomposition is that the number of multilayer cores may be exponential in the number of layers. Specifically, denoting by K_ℓ the maximum order of a core for layer ℓ, the number of multilayer cores is $O(\prod_{\ell \in L} K_\ell)$. This makes Multilayer Core Decomposition intrinsically hard: *in the general case, no polynomial-time algorithm can exist.* The challenge hence lies in handling this exponential blowup by early recognizing and skipping unnecessary portions of the search space, such as non-distinct and/or empty cores.

Given a multilayer graph $G = (V, E, L)$ and a coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, finding the corresponding core can easily be solved in $O(|E| + |V| \times |L|)$ time by iteratively removing a vertex u having $deg_{G'}(u, \ell) < k_\ell$ in some layer ℓ, where G' denotes the current graph resulting from all previous vertex removals (Algorithm 1, where the set S of vertices to be considered is set to $S = V$). Hence, a naïve way of computing a multilayer core decomposition consists of generating all possible coreness vectors, run the multilayer core-detection algorithm just described for each vector, and retain only non-empty and distinct cores. This naïve method requires all vectors $[k_\ell]_{\ell \in L}$, where each k_ℓ component is varied within the interval $[0..K_\ell]$.[3] This corresponds to a $\Theta(\prod_{\ell \in L} K_\ell)$ number of vectors. As a result, the overall time complexity of the method is $O((|E| + |V| \times |L|) \times \prod_{\ell \in L} K_\ell)$.

This approach has the following two major weaknesses: (*i*) each core is computed starting from the whole input graph, and (*ii*) by enumerating all possible coreness vectors beforehand a lot of non-distinct and/or empty (thus, unnecessary) cores may be computed. In the following, we present three methods that solve Multilayer Core Decomposition much more efficiently.

3.1 Search Space

Although multilayer cores are not all nested into each other, a notion of partial containment can still be defined. Indeed, it can easily be observed that a $\mathbf{k}$-core with coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ is

[3] K_ℓ values can be derived beforehand by computing a single-layer core decomposition in each layer ℓ. This process overall takes $O(|E|)$ time.

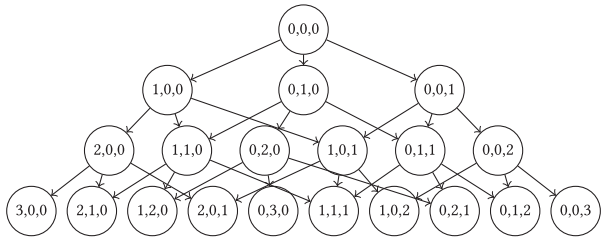

Fig. 2. Core lattice of a three-layer graph.

ALGORITHM 1: **k**-core

Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, an $|L|$-dimensional integer vector $\mathbf{k} = [k_\ell]_{\ell \in L}$.

Output: The **k**-core $C_\mathbf{k}$ of G.

1: **while** $\exists u \in S, \exists \ell \in L : \deg_S(u, \ell) < k_\ell$ **do**
2: $\quad S \leftarrow S \setminus \{u\}$
3: **end while**
4: $C_\mathbf{k} = S$

contained into any $\mathbf{k}'$-core described by a coreness vector $\mathbf{k}' = [k'_\ell]_{\ell \in L}$, whose components k'_ℓ are all no more than components k_ℓ, i.e., $k'_\ell \leq k_\ell$, $\forall \ell \in L$. This result is formalized next:

Fact 1. *Given a multilayer graph $G = (V, E, L)$ and two cores $C_\mathbf{k}$ and $C_{\mathbf{k}'}$ of G with coreness vectors $\mathbf{k} = [k_\ell]_{\ell \in L}$ and $\mathbf{k}' = [k'_\ell]_{\ell \in L}$, respectively, it holds that if $\forall \ell \in L : k'_\ell \leq k_\ell$, then $C_\mathbf{k} \subseteq C_{\mathbf{k}'}$.*

Proof. Combining the definition of multilayer core (Definition 2) and the hypothesis on vectors $\mathbf{k}$ and $\mathbf{k}'$, it holds that $\forall \ell \in L : \mu(C_\mathbf{k}, \ell) \geq k_\ell \geq k'_\ell$. This means that $C_\mathbf{k}$ satisfies the definition of $\mathbf{k}'$-core, thus implying that all vertices in $C_\mathbf{k}$ are part of $C_{\mathbf{k}'}$ too. The fact follows. □

Based on Fact 1, the search space of our problem can be represented as a lattice defining a partial order among all cores (Figure 2). Such a lattice, which we call the *core lattice*, corresponds to a Directed Acyclic Graph (DAG) where nodes represent cores,[4] and links represent relationships of containment between cores (a "father" node contains all its "child" nodes). We assume the core lattice keeping track of non-empty and not necessarily distinct cores: a core is present in the lattice as many times as the number of its coreness vectors. Each level i of the lattice represents the children of cores at lattice level $i - 1$. In particular, level i contains all those cores whose coreness vector results from increasing one and only one component of its fathers' coreness vector by one. Formally, a lattice level i contains all **k**-cores with coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ such that there exists a core at lattice level $i - 1$ with coreness vector $\mathbf{k}' = [k'_\ell]_{\ell \in L}$, where $\exists \ell \in L : k_\ell = k'_\ell + 1$, and $\forall \hat{\ell} \neq \ell : k_{\hat{\ell}} = k'_{\hat{\ell}}$. As a result, level 0 contains the root only, which corresponds to the whole input graph (i.e., the $[0]_{|L|}$-core), the leaves correspond to inner-most cores, and any non-leaf node has at least one and at most $|L|$ children. Moreover, every level i contains all cores whose coreness-vector components sum to i.

Solving the Multilayer Core Decomposition problem is hence equivalent to building the core lattice of the input graph. The efficient methods we present next are all based on smart core-lattice building strategies that extract cores from smaller subgraphs, while also attempting to minimize the visit/computation of unnecessary (i.e., empty/non-distinct) cores.

[4]Throughout the article we use the term "node" to refer to elements of the core lattice, and "vertex" for the elements of the multilayer graph.

ALGORITHM 2: BFS-ML-cores

Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of G.

```
C ← ∅, Q ← {[0]_|L|}, F([0]_|L|) ← ∅                    # F keeps track of father nodes
while Q ≠ ∅ do
    dequeue k = [k_ℓ]_ℓ∈L from Q
    if |{k_ℓ : k_ℓ > 0}| = |F(k)| then                   # Corollary 2
        F_∩ ← ⋂_{F∈F(k)} F                               # Corollary 1
        C_k ← k-core(G, F_∩, k)                          # Algorithm 1
        if C_k ≠ ∅ then
            C ← C ∪ {C_k}
            for all ℓ ∈ L do                             # enqueue child nodes
                k' ← [k_1, ..., k_ℓ + 1, ..., k_|L|]
                enqueue k' into Q
                F(k') ← F(k') ∪ {C_k}
            end for
        end if
    end if
end while
```

3.2 Breadth-First Algorithm

Two interesting corollaries can be derived from Fact 1. First, any non-empty $\mathbf{k}$-core is necessarily contained in the intersection of all its father nodes of the core lattice. Second, any non-empty $\mathbf{k}$-core has *exactly* as many fathers as the number of non-zero components of its coreness vector $\mathbf{k}$:

Corollary 1. *Given a multilayer graph G, let C be a core of G and $\mathcal{F}(C)$ be the set of fathers of C in the core lattice of G. It holds that $C \subseteq \bigcap_{\hat{C} \in \mathcal{F}(C)} \hat{C}$.*

Proof. By definition of core lattice, the coreness vector of all father cores $\mathcal{F}(C)$ of C is dominated by the coreness vector of C. Fact 1 ensures that $C \subseteq C'$, $\forall C' \in \mathcal{F}(C)$. Assume $u \in C$, $u \notin \bigcap_{\hat{C} \in \mathcal{F}(C)} \hat{C}$ exists. This implies existence of a father core $C' \in \mathcal{F}(C)$ s.t. $C \nsubseteq C'$, which is a contradiction. □

Corollary 2. *Given a multilayer graph G, let C be a core of G with coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, and $\mathcal{F}(C)$ be the set of fathers of C in the core lattice of G. It holds that $|\mathcal{F}(C)| = |\{k_\ell : \ell \in L, k_\ell > 0\}|$.*

Proof. By definition of core lattice, a core C at level i has a coreness vector whose components sum to i, while the fathers $\mathcal{F}(C)$ of C have coreness vector whose components sum to $i - 1$. The coreness vector of a father of C can be obtained by decreasing a non-zero component of the coreness vector of C by one. This means that the number of fathers of C is upper-bounded by the non-zero components of its coreness vector. More precisely, the number of fathers of C is exactly equal to this number, as, according to Corollary 1, no father of C can be empty, otherwise C would be empty too and would not be part of the core lattice. □

The above corollaries pave the way to a breadth-first search building strategy of the core lattice, where cores are generated level-by-level by properly exploiting the rules in the two corollaries (Algorithm 2). Although the worst-case time complexity of this BFS-ML-cores method remains unchanged with respect to the naïve algorithm, the BFS method is expected to be much more efficient in practice, due to the following main features: (i) cores are not computed from the initial

ALGORITHM 3: DFS-ML-cores

Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of G.

```
C ← {V},  R ← L,  Q ← {[0]_|L|},  Q' ← ∅
while R ≠ ∅ do
    remove a layer from R
    for all k ∈ Q do
        ∀ℓ∈R s.t. k_ℓ = 0 : Q' ← Q' ∪ {k' | C_k' ∈ k-coresPath(G, C_k, k, ℓ)}
        ∀ℓ∈L \ R s.t. k_ℓ = 0 : C ← C ∪ k-coresPath(G, C_k, k, ℓ)
    end for
    C ← C ∪ {C_k | k ∈ Q'},  Q ← Q',  Q' ← ∅
end while
```

graph every time, but from a much smaller subgraph given by the intersection of all their fathers; (*ii*) in many cases, i.e., when the rule in Corollary 2 (which can be checked in constant time) arises, no overhead due to the intersection among father cores is required; (*iii*) the number of empty cores computed is limited, as no empty core may be generated from a core that has already been recognized as empty.

3.3 Depth-First Algorithm

Although being much smarter than the naïve method, BFS-ML-cores still has some limitations. First, it visits every core as many times as the number of its fathers in the core lattice. Also, as a second limitation, consider a path $\mathcal{P}$ of the lattice connecting a non-leaf node to a leaf by varying the same ℓ-th component of the corresponding coreness vectors. It is easy to see that the computation of all cores within $\mathcal{P}$ with BFS-ML-cores takes $O(|\mathcal{P}| \times (|E| + |V| \times |L|))$ time, as the core-decomposition process is re-started at every level of the lattice. This process can in principle be performed more efficiently, i.e., so as to take $O(|\mathcal{P}| + |E| + |V| \times |L|)$ time, as it actually corresponds to (a simple variant of) a single-layer core decomposition.

To address the above two cons, we propose a method performing a depth-first search on the core lattice. The method, dubbed DFS-ML-cores (Algorithm 3), iteratively picks a non-leaf core $\mathbf{k} = [k_1, \ldots, k_\ell, \ldots, k_{|L|}]$ and a layer ℓ such that $k_\ell = 0$, and computes all cores $[k_1, \ldots, k_\ell + 1, \ldots, k_{|L|}], \ldots, [k_1, \ldots, K_\ell, \ldots, k_{|L|}]$ with a run of the $\mathbf{k}$-coresPath$(G, C_\mathbf{k}, \mathbf{k}, \ell)$ subroutine. Specifically, such a subroutine returns the cores corresponding to all coreness vectors obtained by varying the ℓth component of $\mathbf{k}$ within $[0, \ldots, K_\ell]$. Also, it discards vertices violating the coreness condition specified by vector $\mathbf{k}$, i.e., vertices whose degree in some layer $\hat{\ell} \neq \ell$ is less than the $\hat{\ell}$th component of $\mathbf{k}$. The pseudocode of $\mathbf{k}$-coresPath is reported as Algorithm 4: it closely resembles the traditional core-decomposition algorithm for single-layer graphs, except for the addition of the cycle starting at Line 13, which identifies the aforementioned vertices to be discarded.

A side effect of this strategy is that the same core may be computed multiple times. As an example, in Figure 2 the $(1, 2, 0)$-core is computed by core decompositions initiated at both cores $(1, 0, 0)$ and $(0, 2, 0)$. To reduce (but not eliminate) these multiple core computations, the DFS-ML-cores method exploits the following result.

THEOREM 2. *Given a multilayer graph $G = (V, E, L)$, let $[\ell_1, \ldots, \ell_{|L|}]$ be an order defined over set L. Let $\mathbf{Q}_0 = \{[0]_{|L|}\}$, and, $\forall i \in [1..|L|]$, let $\mathbf{Q}_i = \{\mathbf{k}' \mid C_{\mathbf{k}'} \in \mathbf{k}\text{-}coresPath(G, C_\mathbf{k}, \mathbf{k}, \ell), \mathbf{k} \in \mathbf{Q}_{i-1}, \ell \in (\ell_i..\ell_{|L|}], k_\ell = 0\}$ and $\mathbf{C}_i = \{\mathbf{k}' \mid C_{\mathbf{k}'} \in \mathbf{k}\text{-}coresPath(G, C_\mathbf{k}, \mathbf{k}, \ell), \mathbf{k} \in \mathbf{Q}_{i-1}, \ell \in [\ell_1..\ell_i], k_\ell = 0\}$. The set $\mathbf{C} = \{C_\mathbf{k} \mid \mathbf{k} \in \bigcup_{i=0}^{|L|} \mathbf{Q}_i \cup \bigcup_{i=1}^{|L|} \mathbf{C}_i\}$ is the multilayer core decomposition of G.*

ALGORITHM 4: **k**-coresPath

Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, an $|L|$-dimensional integer vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, and a layer $\ell \in L$.

Output: The set $\mathbf{C}_{\mathbf{k},\ell}$ of the multilayer cores of G varying the ℓ-th component of $\mathbf{k}$.

```
C_{k,ℓ} ← ∅, 𝒟 ← ∅
for all u ∈ S do
    𝒟(deg_S(u, ℓ)) ← 𝒟(deg_S(u, ℓ)) ∪ {u}
end for
for all k ∈ [0, ..., K_ℓ] do
    while D(k) ≠ ∅ do
        remove a vertex u from 𝒟(k)
        S ← S \ {u}
        for all v ∈ S : (u, v, ℓ) ∈ E ∧ deg_S(v, ℓ) ≥ k do
            𝒟(deg_S(v, ℓ) + 1) ← 𝒟(deg_S(v, ℓ) + 1) \ {v}
            𝒟(deg_S(v, ℓ)) ← 𝒟(deg_S(v, ℓ)) ∪ {v}
        end for
        for all ℓ̂ ∈ L \ {ℓ} do
            for all v ∈ S : (u, v, ℓ̂) ∈ E ∧ deg_S(v, ℓ̂) < k_ℓ̂ do
                𝒟(deg_S(v, ℓ)) ← 𝒟(deg_S(v, ℓ)) \ {v}
                𝒟(k) ← 𝒟(k) \ {v}
            end for
        end for
    end while
    C_{k,ℓ} ← C_{k,ℓ} ∪ {S}
end for
```

Proof. The multilayer core decomposition of G is formed by the union of all non-empty and distinct cores of all paths $\mathcal{P}$ of the lattice connecting a non-leaf node to a leaf by varying the same ℓ-th component of the corresponding coreness vectors.

For any $i \in [1..|L|]$, let $\mathcal{P}_i \in \mathcal{P}$ denote the subset of paths whose coreness vectors $\mathbf{k}' = [k'_\ell]_{\ell \in L}$ have a number of non-zero components equal to i. By definition of $\mathbf{Q}_i$ and $\mathbf{C}_i$ it holds that all coreness vectors $\mathbf{k}'$ of the cores along the paths in $\mathcal{P}_i$ are in $\mathbf{Q}_i \cup \mathbf{C}_i = \{\mathbf{k}' : |\{k'_\ell : \ell \in L, k'_\ell > 0\}| = i\}$. Also, since some of the paths may overlap, all cores along the paths $\mathcal{P}_i$ are computed by executing single-layer core decompositions initiated at a subset of cores along the paths $\mathcal{P}_{i-1}$. Such a subset of cores is represented by the subset of coreness vectors within $\mathbf{Q}_{i-1} = \{\mathbf{k} : |\{k_\ell : \ell \in [\ell_2..\ell_{|L|}], k_\ell > 0\}| = i - 1\}$. Moreover, note that single-layer core decompositions for the layers where $k_\ell \neq 0$ are discarded, as it boils down to visit cores in $\mathcal{P}_{i-1}$. As a result, the set $\{C_\mathbf{k} \mid \mathbf{k} \in \bigcup_{i=0}^{|L|} \mathbf{Q}_i \cup \bigcup_{i=1}^{|L|} \mathbf{C}_i\}$ correctly contains all possible coreness vectors of the core lattice. □

Referring to Algorithm 3, the result in Theorem 2 is implemented by keeping track of a subset of layers $R \subseteq L$. At the beginning $R = L$, and, at each iteration of the main cycle, a layer ℓ is removed from it. The algorithm is guaranteed to be sound and complete regardless of the layer ordering, which may instead affect running time. In our experiments we test several orders, such as random, or non-decreasing/non-increasing average-degree density. All those orders resulted in comparable running times, we therefore decided to stick to the simplest one, i.e., random. Set $\mathbf{Q}$ keeps track of (the coreness vector of) all lattice nodes where the current single-layer core-decomposition processes need to be run from. $\mathbf{Q}'$ stores the (coreness vector of) cores computed from each node in $\mathbf{Q}$ and for each layer within R, while also forming the basis of $\mathbf{Q}$ for the next iteration.

ALGORITHM 5: HYBRID-ML-cores

Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{C}$ of all non-empty multilayer cores of G.

```
Q ← {[0]_|L|},  F([0]_|L|) ← ∅                                  # F keeps track of father nodes
Q' ← ⋃_{ℓ∈L} {k | C_k ∈ k-coresPath(G,V,[0]_|L|,ℓ)}              # looked-ahead cores
C ← {C_k | k ∈ Q'}
while Q ≠ ∅ do
    dequeue k = [k_ℓ]_{ℓ∈L} from Q
    if |{k_ℓ : k_ℓ > 0}| = |F(k)| ∧ k ∉ Q' then                  # Corollary 2
        F_∩ ← ⋂_{F∈F(k)} F                                      # Corollary 1
        C_k ← k-core(G, F_∩, k)                                 # Algorithm 1
        if C_k ≠ ∅ then
            C ← C ∪ {C_k}
            d_μ(C_k) ← [μ(C_k, ℓ)]_{ℓ∈L}                        # look-ahead mechanism (Corollary 3)
            Q' ← Q' ∪ {k' | k ≤ k' ≤ d_μ(C_k)}
        end if
    end if
    if k ∈ Q' then
        for all ℓ ∈ L do                                        # enqueue child nodes
            k' ← [k_1, ..., k_ℓ + 1, ..., k_|L|]
            enqueue k' into Q
            F(k') ← F(k') ∪ {C_k}
        end for
    end if
end while
```

In summary, compared to BFS-ML-cores, the DFS method reduces both the time complexity of computing all cores in a path $\mathcal{P}$ from a non-leaf node to a leaf of the core lattice (from $O(|\mathcal{P}| \times (|E| + |V| \times |L|))$ to $O(|\mathcal{P}| + |E| + |V| \times |L|)$), and the number of times a core is *visited*, which may now be smaller than the number of its fathers. On the other hand, DFS-ML-cores comes with the aforementioned issue that some cores may be *computed* multiple times (while in BFS-ML-cores every core is computed only once). Furthermore, cores are computed starting from larger subgraphs, as intersection among multiple fathers cannot exploited.

3.4 Hybrid Algorithm

The output of both BFS-ML-cores and DFS-ML-cores correctly corresponds to all distinct cores of the input graph and the corresponding maximal coreness vectors.[5] Nevertheless, none of these methods is able to skip the computation of non-distinct cores. Indeed, both methods need to compute every core C as many times as the number of its coreness vectors to guarantee completeness. To address this limitation, we devise a further method where the main peculiarities of BFS-ML-cores and DFS-ML-cores are joined into a "hybrid" lattice-visiting strategy. This HYBRID-ML-cores method exploits the following corollary of Theorem 1, stating that the maximal coreness vector of a core C is given by the vector containing the minimum degree of a vertex in C for each layer:

[5]Pseudocodes in Algorithms 2 and 3 guarantee this as cores are added to a set $\mathbf{C}$ that does not allow duplicates. Any real implementation can easily take care of this by checking whether a core is already in $\mathbf{C}$, and update it in case the corresponding coreness vector contains the previously stored one.

Corollary 3. *Given a multilayer graph $G = (V, E, L)$, the maximal coreness vector of a multilayer core C of G corresponds to the $|L|$-dimensional integer vector $\mathbf{d}_\mu(C) = [\mu(C, \ell)]_{\ell \in L}$.*

Proof. By Definition 2, vector $\mathbf{d}_\mu(C)$ is a coreness vector of C. Assume that $\mathbf{d}_\mu(C)$ is not maximal, meaning that another coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ dominating $\mathbf{d}_\mu(C)$ exists. This implies that $k_\ell \geq \mu(C, \ell)$, and $\exists \hat{\ell} \in L : k_{\hat{\ell}} > \mu(C, \hat{\ell})$. By definition of multilayer core, all vertices in C have degree larger than the minimum degree $\mu(C, \hat{\ell})$ in layer $\hat{\ell}$, which is a clear contradiction. □

Corollary 3 gives a rule to skip the computation of non-distinct cores: given a core C with coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, all cores with coreness vector $\mathbf{k}' = [k'_\ell]_{\ell \in L}$ such that $\forall \ell \in L : k_\ell \leq k'_\ell \leq \mu(C, \ell)$ are guaranteed to be equal to C and do not need to be explicitly computed. For instance, in Figure 2, assume that the min-degree vector of the $(0, 0, 1)$-core is $(0, 1, 2)$. Then, cores $(0, 0, 2)$, $(0, 1, 1)$, and $(0, 1, 2)$ can immediately be set equal to the $(0, 0, 1)$-core. The hybrid-ml-cores algorithm we present here (Algorithm 5) exploits this rule by performing a breadth-first search equipped with a "look-ahead" mechanism resembling a depth-first search. Moreover, hybrid-ml-cores starts with a single-layer core decomposition for each layer so as to have more fathers early-on for intersections. Cores interested by the look-ahead rule are still *visited* and stored in $\mathbf{Q}'$, as they may be needed for future core computations. However, no further computational overhead is required for them. The time complexity of hybrid-ml-cores is the same as bfs-ml-cores, plus an additional $O(|E|)$ time for every visited multilayer core, which is needed in the look-ahead rule to compute the min-degree vector of that core.

3.5 Discussion

To have a concrete comparison of the characteristics of the proposed algorithms for multilayer core decomposition, we report in Figure 3 a running example over a core lattice of a simple two-layer graph. All the algorithms start by visiting the root of the core lattice, which corresponds to the whole input multilayer graph (this preliminary step is left out from Figure 3 since it is shared by all the methods). bfs-ml-cores visits the core lattice level by level, and exploits every containment relationship. The execution pattern of dfs-ml-cores is instead much different: it starts by finding those multilayer cores having a single component of the coreness vector other than zero and, in a later step, visits the rest of the core lattice. In both steps (a) and (b) dfs-ml-cores visits cores following straight paths in the search space, i.e., from a core to a leaf. As a result, not all the containment relationships are exploited. For instance, the computation of the $(2, 1)$-core exploits the containment from the $(2, 0)$-core, but not from the $(1, 1)$-core. hybrid-ml-cores is, as expected, a mix of the two other methods. The first step is identical to dfs-ml-cores. At step (b), hybrid-ml-cores starts to visit the remaining cores by a breadth-first-search strategy, while also exploiting the look-ahead mechanism. In particular, the minimum degree vector of the $(1, 1)$-core is found to be equal to $(1, 2)$; therefore, the $(1, 2)$-core is not computed directly, but set equal to the $(1, 1)$-core. In the final step, hybrid-ml-cores visits the remaining core by going on with the breadth-first search.

We already discussed (in the respective paragraphs) the strengths and weaknesses of bfs-ml-cores and dfs-ml-cores: the best among the two is determined by the peculiarities of the specific input graph. On the other hand, hybrid-ml-cores profitably exploits the main nice features of both bfs-ml-cores and dfs-ml-cores, thus is expected to outperform both methods in most cases. However, in those graphs where the number of non-distinct cores is limited, the overhead due to the look-ahead mechanism can make the performance of hybrid-ml-cores degrade.

In terms of space requirements, bfs-ml-cores needs to keep in memory all those cores having at least a child in the queue, i.e., at most two levels of the lattice (the space taken by a multilayer core is $O(|V|)$). The same applies to hybrid-ml-cores with the addition of the cores computed through single-layer core decomposition and look-ahead, until all their children have been processed.

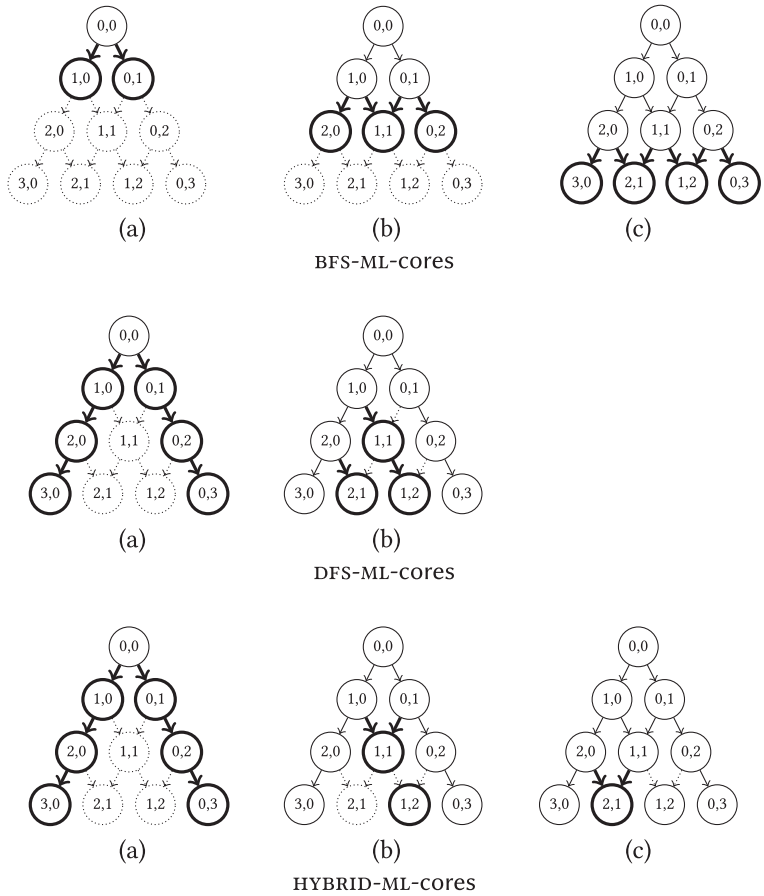

Fig. 3. Running example of our algorithms for multilayer core decomposition over a core lattice of a two-layer graph. Nodes and links depicted by solid lines have been visited in previous steps of the algorithm, those in thick lines are visited during the current step, while the remaining in dotted lines have not been visited yet.

DFS-ML-cores instead requires to store all cores where the single-layer core-decomposition process should be started from, both in the current iteration and the next one. Thus, we expect DFS-ML-cores to take more space than BFS-ML-cores and HYBRID-ML-cores, as in practice the number of cores to be stored should be more than the cores belonging to two lattice levels.

3.6 Experimental Results

In this subsection, we present experiments to (*i*) compare the proposed algorithms in terms of runtime, memory consumption, and search-space exploration; (*ii*) characterize the output core decompositions, also by comparing total number of cores and number of inner-most cores.

Datasets. We select publicly available real-world multilayer networks, whose main characteristics are summarized in Table 1. Homo[6] and SacchCere[6] are networks describing different types of genetic interactions between genes in *Homo sapiens* and *Saccharomyces cerevisiae*, respectively.

[6]http://deim.urv.cat/~manlio.dedomenico/data.php.

Table 1. Characteristics of the Real-World Datasets: Number of Vertices ($|V|$), Number of Overall Edges ($|E|$), Number of Layers ($|L|$), Minimum, Average, and Maximum Number of Edges in a Layer (min $|E_\ell|$, avg $|E_\ell|$, max $|E_\ell|$), and Application Domain

Dataset	$\|V\|$	$\|E\|$	$\|L\|$	min $\|E_\ell\|$	avg $\|E_\ell\|$	max $\|E_\ell\|$	Domain
Homo	18k	153k	7	256	21k	83k	genetic
SacchCere	6.5k	247k	7	1.3k	35k	91k	genetic
DBLP	513k	1.0M	10	96k	101k	113k	co-authorship
ObamaInIsrael	2.2M	3.8M	3	557k	1.2M	1.8M	social
Amazon	410k	8.1M	4	899k	2.0M	2.4M	co-purchasing
FriendfeedTwitter	155k	13M	2	5.2M	6.8M	8.3M	social
Higgs	456k	13M	4	28k	3.4M	12M	social
Friendfeed	510k	18M	3	226k	6.2M	18M	social

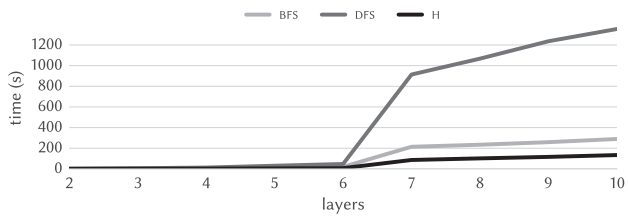

Fig. 4. Runtime of the proposed methods with varying the number of layers (DBLP dataset).

ObamaInIsrael[6] represents different types of social interaction (e.g., *re-tweeting*, *mentioning*, and *replying*) among Twitter users, focusing on Barack Obama's visit to Israel in 2013. Similarly, Higgs[6] is built by tracking the spread of news about the discovery of the Higgs boson on Twitter, with the additional layer for the *following* relation. Friendfeed[7] contains public interactions among users of Friendfeed collected over 2 months (e.g., *commenting*, *liking*, and *following*). FriendfeedTwitter[7] is a multi-platform social network, where layers represent interactions within Friendfeed and Twitter between users registered to both platforms [28]. Amazon[8] is a co-purchasing *temporal network*, containing four snapshots between March and June 2003. Finally, DBLP[9] is derived following the methodology in [20]. For each co-authorship relation (edge), the bag of words resulting from the titles of all papers co-authored by the two authors is collected. Then, Latent Dirichlet Allocation (LDA) topic modeling [18] is applied to automatically identify a hundred topics. Among these, 10 topics that are recognized as the most relevant to the data-mining area have been hand-picked. Every selected topic corresponds to a layer. An edge between two co-authors in a certain layer exists if the relation between those co-authors is labeled with the topic corresponding to that layer.

Implementation. All methods are implemented in Python (v. 2.7.12) and compiled by Cython: all our code are available at github.com/egalimberti/multilayer_core_decomposition. All experiments are run on a machine equipped with Intel Xeon CPU at 2.1 GHz and 128 GB RAM except for Figure 4, whose results are obtained on Intel Xeon CPU at 2.7 GHz with 128 GB RAM.

Comparative evaluation. We compare the naïve baseline (for short N) and the three proposed methods BFS-ML-cores (for short BFS), DFS-ML-cores (DFS), and HYBRID-ML-cores (H) in terms of

[7] http://multilayer.it.uu.se/datasets.html.
[8] https://snap.stanford.edu/data/.
[9] http://dblp.uni-trier.de/xml/.

Table 2. Comparative Evaluation: Proposed Methods and Baseline

Dataset	#output cores	Method	Runtime (s)	Memory (MB)	#computed cores
Homo	1,845	N	1,145	27	12,112
		BFS	13	26	3,043
		DFS	27	27	6,937
		H	**12**	**25**	**2,364**
SacchCere	74,426	N	24,469	55	278,402
		BFS	**1,134**	**34**	89,883
		DFS	2,627	57	223,643
		H	1,146	35	**83,978**
DBLP	3,346	N	103,231	608	34,572
		BFS	68	612	6,184
		DFS	282	627	38,887
		H	**29**	**521**	**5,037**
Obama InIsrael	2,573	N	37,554	1,286	3,882
		BFS	226	1,299	3,313
		DFS	**150**	1,384	3,596
		H	177	**1,147**	**2,716**
Amazon	1,164	N	11,990	**425**	1,823
		BFS	3,981	534	1,354
		DFS	5,278	619	2,459
		H	**3,913**	536	**1,334**
Friendfeed Twitter	76,194	N	409,489	220	80,954
		BFS	61,113	**215**	80,664
		DFS	**1,973**	267	80,745
		H	59,520	268	**76,419**
Higgs	8,077	N	163,398	474	22,478
		BFS	2,480	**465**	12,773
		DFS	**640**	490	14,119
		H	2,169	493	**9,389**
Friendfeed	365,666	BFS	58,278	**465**	546,631
		DFS	**13,356**	591	568,107
		H	47,179	490	**389,323**

Runtime differs from [39] since a different server was employed. For each dataset, best performances are bolded.

running time, memory usage, and number of computed cores (as a measure of the explored search-space portion). The results of this comparison are shown in Table 2. As expected, N is the least efficient method: it is outperformed by our algorithms by 1–4 orders of magnitude. Due to its excessive requirements, we could not run it in reasonable time (i.e., 30 days) on the Friendfeed dataset.

Among the proposed methods, H is recognized as the best method (in absolute or with performance comparable to the best one) in the first five (out of a total of eight) datasets. In the remaining three datasets the best method is DFS. This is mainly motivated by the fact that those three datasets have a relatively small number of layers, an aspect that DFS takes particular advantage from (as also better testified by the experiment with varying the number of layers discussed below). In some cases, H is also comparable to BFS, thus confirming the fact that in datasets where the number of non-distinct cores is not so large, the performance of the two methods gets closer. A similar

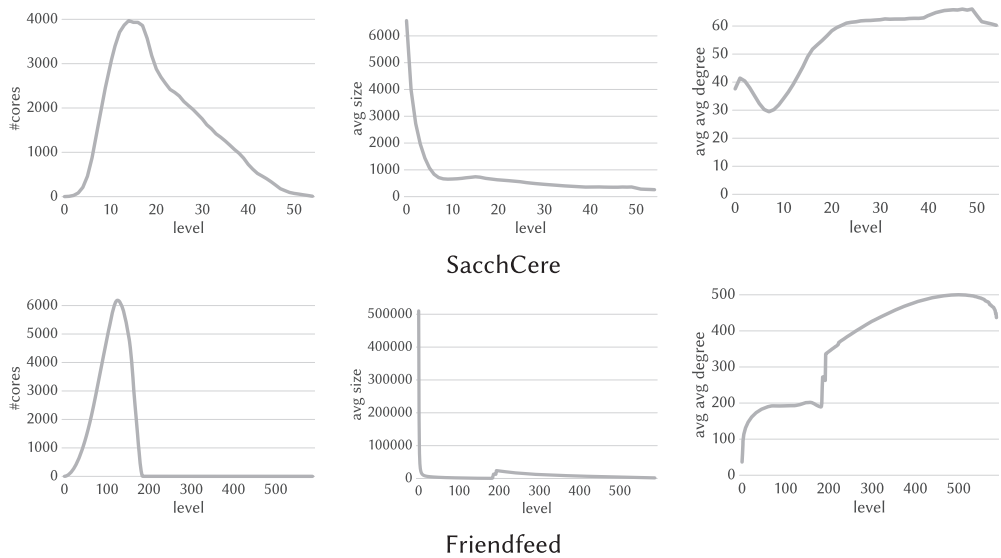

Fig. 5. Distribution of number of cores (left), average core size (center), and average average-degree density of a core (right) to the core-lattice level, for datasets SacchCere (top) and Friendfeed (bottom).

reasoning holds between BFS and DFS (at least with a small/moderate number of the layers, see next): BFS is faster in most cases, but, due to the respective pros and cons discussed in Section 3, it is not surprising that the two methods achieve comparable performance in a number of other cases.

To test the behavior with varying the number of layers, Figure 4 shows the running times of the proposed methods on different versions of the DBLP dataset, obtained by selecting a variable number of layers, from 2 to 10. While the performance of the three methods is comparable up to six layers, beyond this threshold the execution time of DFS grows much faster than BFS and H. This attests that the pruning rules of BFS and H are more effective as the layers increase. To summarize, DFS is expected to have runtime comparable to (or better than) BFS and H when the number of layers is small, while H is faster than BFS when the number of non-distinct cores is large.

The number of computed cores is always larger than the output cores as all methods might compute empty cores or, in the case of DFS, the same core multiple times. Table 2 shows that DFS computes more cores than BFS and H, which conforms to its design principles.

Finally, all methods turn out to be memory-efficient, taking no more than 1.5 GB of memory.

Core-decomposition characterization. Figure 5 reports the distribution of number of cores, core size, and average-degree density (i.e., number of edges divided by number of vertices) of the subgraph corresponding to a core. Distributions are shown by level of the lattice[10] for the SacchCere and Friendfeed datasets. Although the two datasets have very different scales, the distributions exhibit similar trends. Being limited by the number of layers, the number of cores in the first levels of the lattice is very small, but then it exponentially grows until reaching its maximum within the first 2–30% visited levels. The average size of the cores is close to the number of vertices in the first lattice level, when cores' degree conditions are not very strict. Then it decreases as the number of cores gets larger, with a maximum reached when very small cores stop "propagating" in the lower lattice levels. Finally, the average (average-degree) density tends to increase for higher

[10]Recall that the lattice level has been defined in Section 3.1: level i contains all cores whose coreness-vector components sum to i.

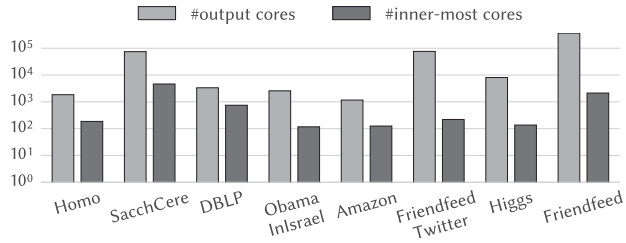

Fig. 6. Number of output cores (total and inner-most).

lattice level. However, there are a couple of exceptions: it decreases (*i*) in the first few levels of SacchCere's lattice, and (*ii*) in the last levels of both SacchCere and Friendfeed, where the core size starts getting smaller, thus implying small average-degree values.

In Figure 6, we show the comparison between the number of all cores and inner-most cores for all the datasets. The number of cores differs quite a lot from dataset to dataset, depending on dataset size, number of layers, and density. The fraction of inner-most cores exhibits a non-decreasing trend as the layers increase, ranging from 0.3% of the total number of output cores (FriendfeedTwitter) to 22% (DBLP).

Given that the inner-most cores are *per-se* interesting and typically one or more orders of magnitude fewer in number than the total cores, it would be desirable to have a method that effectively exploits the maximality property and extracts the inner-most ones directly, without computing a complete decomposition. This is presented in the next section.

4 ALGORITHMS FOR INNER-MOST MULTILAYER CORES

In this section, we focus on the problem of finding the non-empty inner-most multilayer cores of a multilayer graph (Problem 2). Specifically, the main goal here is to devise a method that is more efficient than a naïve one that computes the whole multilayer core decomposition and then a-posteriori filters non-inner-most cores out. To this end, we devise a recursive algorithm, which is termed IM-ML-cores and whose outline is shown as Algorithm 6 (and Algorithm 7). We provide the details of the algorithm next. In the remainder of this section we assume the layer set L of the input multilayer graph $G = (V, E, L)$ to be an ordered list $[\ell_1, \ldots, \ell_{|L|}]$. The specific ordering we adopt in this work is by non-decreasing average-degree density, as, among the various orderings tested, this is the one that provides the best experimental results.

The proposed IM-ML-cores algorithm is based on the notion of *ℓ_r-right-inner-most multilayer cores* of a core $C_{\mathbf{k}}$, i.e., all those cores having coreness vector $\mathbf{k}'$ equal to $\mathbf{k}$ up to layer ℓ_{r-1}, and for which the inner-most condition holds for layers from ℓ_r to $\ell_{|L|}$.

Definition 5 (ℓ_r-right-inner-most Multilayer Cores). Given a multilayer graph $G = (V, E, L)$ and a layer $\ell_r \in L$, the *ℓ_r-right-inner-most multilayer cores* of a core $C_{\mathbf{k}}$ of G, where $\mathbf{k} = [k_\ell]_{\ell \in L}$, correspond to all the cores of G with coreness vector $\mathbf{k}' = [k'_\ell]_{\ell \in L}$ such that $\forall \ell \in [\ell_1, \ell_r) : k'_\ell = k_\ell$, and there does not exist any other core with coreness vector $\mathbf{k}'' = [k''_\ell]_{\ell \in L}$ such that $\forall \ell \in [\ell_1, \ell_r) : k''_\ell = k_\ell$, $\forall \ell \in [\ell_r, \ell_{|L|}] : k''_\ell \geq k'_\ell$, and $\exists \hat{\ell} \in [\ell_r, \ell_{|L|}] : k''_{\hat{\ell}} > k'_{\hat{\ell}}$.

Let $C_{[0]_{|L|}}$ be the root of the core lattice. $C_{[0]_{|L|}}$ has a coreness vector composed of zero components. Therefore, according to the above definition, it is easy to observe that the ℓ_1-right-inner-most multilayer cores of $C_{[0]_{|L|}}$ correspond to the desired ultimate output, i.e., to all inner-most multilayer cores of the input multilayer graph.

Fact 2. *Given a multilayer graph $G = (V, E, L)$, let $\mathbf{I}_{\ell_1}$ be the set of all ℓ_1-right-inner-most multilayer cores of core $C_{[0]_{|L|}}$. $\mathbf{I}_{\ell_1}$ corresponds to all inner-most multilayer cores of G.*

ALGORITHM 6: IM-ML-cores

Input: A multilayer graph $G = (V, E, L)$.
Output: The set $\mathbf{I}$ of all inner-most multilayer cores of G.

1: sort L by non-decreasing average-degree density
2: $\mathcal{M} \leftarrow \emptyset$
3: $\mathbf{I} \leftarrow$ RIM-ML-cores$(G, V, [0]_{|L|}, \ell_1, \mathcal{M})$

ALGORITHM 7: RIM-ML-cores

Input: A multilayer graph $G = (V, E, L)$, a set $S \subseteq V$ of vertices, a coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, a layer $\ell_r \in L$, and a data structure $\mathcal{M}$.
Output: The set $\mathbf{I}_r$ of all right-inner-most multilayer cores of $C_\mathbf{k}$ given ℓ_r.

1: $\mathbf{I}_r \leftarrow \emptyset$
2: **if** $\ell_r \neq \ell_{|L|}$ **then**
3: $\quad \mathbf{Q} \leftarrow \{\mathbf{k}' \mid C_{\mathbf{k}'} \in \mathbf{k}\text{-coresPath}(G, S, \mathbf{k}, \ell_r)\} \cup \{\mathbf{k}\}$
4: $\quad \mathbf{C} \leftarrow \mathbf{k}\text{-coresPath}(G, S, \mathbf{k}, \ell_r) \cup \{S\}$
5: $\quad$ **for all** $\mathbf{k}' \in \mathbf{Q}$ in decreasing order of k'_{ℓ_r} **do**
6: $\quad\quad \mathcal{M}(\mathbf{k}', \ell_r) \leftarrow \emptyset$
7: $\quad\quad \mathbf{I}_r \leftarrow \mathbf{I}_r \cup$ RIM-ML-cores$(G, C_{\mathbf{k}'}, \mathbf{k}', \ell_{r+1}, \mathcal{M})$
8: $\quad$ **end for**
9: **else**
10: $\quad k_\mathcal{M} \leftarrow 0$
11: $\quad$ **for all** $\ell \in [\ell_1, \ell_{|L|})$ **do**
12: $\quad\quad \mathbf{k}^\ell = [k_{\ell_1}, \ldots, k_\ell + 1, \ldots, k_{\ell_{|L|}}]$
13: $\quad\quad k_\mathcal{M} \leftarrow \max\{k_\mathcal{M}, \mathcal{M}(\mathbf{k}^\ell, \ell_{|L|-1})\}$
14: $\quad$ **end for**
15: $\quad \mathbf{k}' \leftarrow [k_{\ell_1}, \ldots, k_{\ell_{|L|-1}}, k_\mathcal{M}]$
16: $\quad \mathbf{k}^I \leftarrow$ Inner-mostCore$(G, S, \mathbf{k}', \ell_{|L|})$
17: $\quad$ **if** $\mathbf{k}^I \neq$ NULL **then**
18: $\quad\quad \mathbf{I}_r \leftarrow \mathbf{I}_r \cup \mathbf{k}^I$
19: $\quad\quad \mathcal{M}(\mathbf{k}^I, \ell_{|L|-1}) \leftarrow k^I_{\ell_{|L|}} + 1$
20: $\quad$ **else**
21: $\quad\quad \mathcal{M}(\mathbf{k}', \ell_{|L|-1}) \leftarrow k'_{\ell_{|L|}}$
22: $\quad$ **end if**
23: **end if**

The proposed IM-ML-cores algorithm recursively computes ℓ_r-right-inner-most multilayer cores, starting from the root of the core lattice (Algorithm 6). The goal is to exploit Fact 2 and ultimately have the ℓ_1-right-inner-most multilayer cores of core $C_{[0]_{|L|}}$ computed. The algorithm makes use of a data structure $\mathcal{M}$, which consists of a sequence of nested maps, one for each layer but the last one (i.e., $\ell_{|L|}$). For every layer ℓ_r that has been so far processed by the recursive procedure, $\mathcal{M}$ keeps track of the minimum-degree that a core should have in layer ℓ_r to be recognized as an ineer-most one. Specifically, given a coreness vector $\mathbf{k}$ and a layer ℓ_r, the instruction $\mathcal{M}(\mathbf{k}, \ell_r)$ iteratively accesses the nested maps using the elements of $\mathbf{k}$ up to layer ℓ_r as keys. As an example, consider a coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, with $|L| = 3$. $\mathcal{M}(\mathbf{k}, \ell_{|L|-1})$ first queries the outer-most map with key k_{ℓ_1}, and obtains a further map. Then, this second map is queried with key k_{ℓ_2}, to finally get the ultimate desired numerical value. Note that, if $\ell_r < \ell_{|L|-1}$, then $\mathcal{M}(\mathbf{k}, \ell_r)$ returns a further

map. Conversely, if $\ell_r = \ell_{|L|-1}$, then $\mathcal{M}(\mathbf{k}, \ell_r)$ returns a numerical value. If $\mathbf{k}$ does not correspond to a sequence of valid keys for $\mathcal{M}$, we assume that 0 is returned as a default value. $\mathcal{M}$ is initialized as empty, and populated during the various recursive iterations.

Algorithm 7 may be logically split into the following two main blocks: the first one (Lines 3–8) taking care of the recursion, and the second one (Lines 10–22) computing the ℓ_r-right-inner-most cores. The first block of the algorithm is executed when the current ℓ_r layer is not the last one. In that block the $\mathbf{k}$-coresPath subroutine (already used in Algorithm 3 and described in Section 3.3) is run on set S of vertices, layer ℓ_r, and taking into account the constraints in vector $\mathbf{k}$ (Lines 3 and 4). Then, for each coreness vector $\mathbf{k}'$ that has been found, a recursive call is made, where the layer of interest becomes the next layer ℓ_{r+1}, and the data structure $\mathcal{M}$ is augmented by adding a further (empty) nested map (this new map will be populated within the upcoming recursive executions). The coreness vectors are processed in decreasing order of k'_{ℓ_r}. This processing order ensures the correctness of the following: once a multilayer core has been identified as ℓ_r-right-inner-most, it permanently becomes part of the ultimate output cores (no further recursive call will remove it from the output). Note also that, for each $\mathbf{k}'$, RIM-ML-cores can be run on $C_{\mathbf{k}'}$ only, i.e., the core of coreness vector $\mathbf{k}'$. This guarantees better efficiency, without affecting correctness.

The second block of the algorithm (Lines 10–22) works as follows. When the last layer has been reached, i.e., $\ell_r = \ell_{|L|}$, the current recursion ends, and an ℓ_r-right-inner-most multilayer core is returned (if any). First of all, the algorithm computes a coreness vector $\mathbf{k}'$, which is potentially ℓ_r-right-inner-most (Lines 10–15). In this regard, note that the $k_{\mathcal{M}}$ value is derived from the information that has been stored in $\mathcal{M}$ in the earlier recursive iterations. Finally, the algorithm computes the inner-most core in $\ell_{|L|}$ constrained by $\mathbf{k}'$, by means of the Inner-mostCore subroutine. Such a subroutine, similarly to the $\mathbf{k}$-coresPath one, takes as input a multilayer graph G, a subset S of vertices, a coreness vector $\mathbf{k}$, and a layer ℓ. It returns the multilayer core having coreness vector of highest ℓth component of the vertices in S, considering the constraints specified in $\mathbf{k}$. If the Inner-mostCore procedure actually returns a multilayer core, then it is guaranteed that such a core is ℓ_r-right-inner-most, and is therefore added to the solution (and $\mathcal{M}$ is updated accordingly).

In Figure 7, we show an example of the execution of the proposed IM-ML-cores algorithm for a simple three-layer graph, while Figure 8 reports the content of the $\mathcal{M}$ data structure for this example. Every box corresponds to a call of Algorithm 7, for which we specify (*i*) the input parameters (G and $\mathcal{M}$ are omitted for the sake of brevity), (*ii*) the calls to the $\mathbf{k}$-coresPath or Inner-mostCore subroutines, and (*iii*) the content of $\mathbf{Q}$ (when it is instantiated). For instance, the coreness vector given as input to Inner-mostCore at box 1.3.4 has the last element equal to the maximum between what is stored in $\mathcal{M}$ at the end of the paths $1 \rightarrow 0$ and $0 \rightarrow 1$, i.e., 8 and 5, that have been set at boxes 1.2.3 and 1.3.3, respectively.

4.1 Experimental Results

Running times. We assess the efficiency of IM-ML-cores (for short IM) by comparing it to the aforementioned naïve approach for computing inner-most multilayer cores, which consists in firstly computing all multilayer cores (by means of one of the three algorithms presented in Section 3) and filtering out the non-inner-most ones. The results of this experiment are reported in Table 3. First of all, it can be observed that the a-posteriori filtering of the inner-most multilayer cores does not consistently affect the runtime of the algorithms for multilayer core decomposition: this means that most of the time is spent for computing the overall core decomposition. The main outcome of this experiment is that the running time of the proposed IM method is smaller than the time required by BFS, DFS, or H summed up to the time spent in the a-posteriori filtering, with considerable speed-up from 1.3 to an order of magnitude on the larger datasets, e.g., FriendfeedTwitter

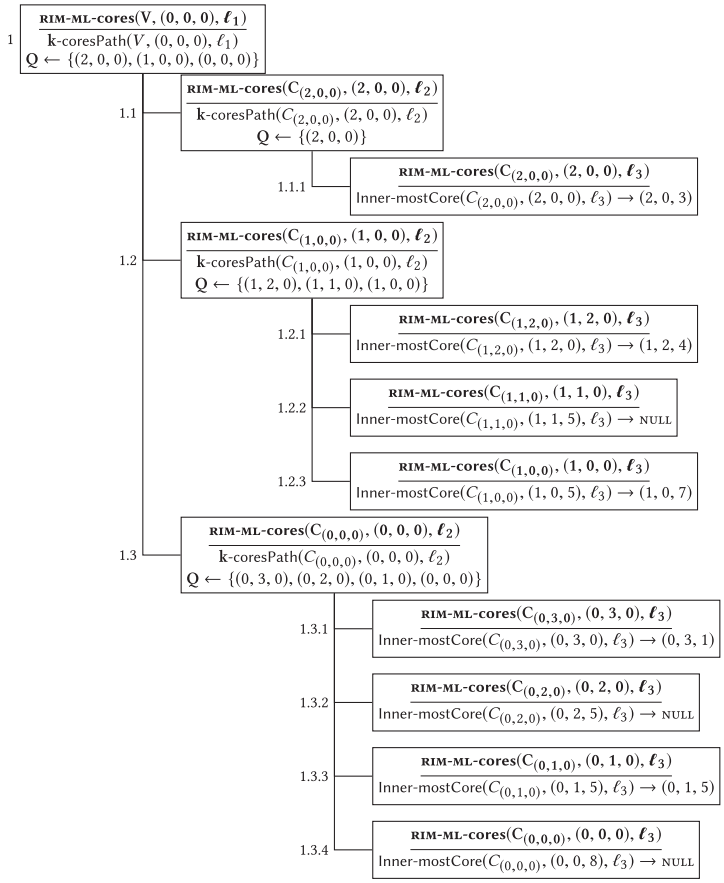

Fig. 7. Execution of the IM-ML-cores algorithm (Algorithm 6) on a toy three-layer graph.

and Friendfeed. The only exception is on the DBLP dataset where BFS and H run slightly faster, probably due to fact that its edges are (almost) equally distributed among the layers, which makes the effectiveness of the ordering vanish.

Characterization. We also show the characteristics of the inner-most multilayer cores. Figure 9 reports the distribution of number, size, and average-degree density of all cores and inner-most cores only. Distributions are shown in a way similar to what previously done in Figure 5, i.e., by level of the core lattice, and for the SacchCere and Amazon datasets.

For both datasets, there are no inner-most cores in the first levels of the lattice. As expected, the number of inner-most cores considerably increases when the number of all cores decreases. This is due to the fact that some cores stop propagating throughout the lattice, hence they are recognized as inner-most. In general, inner-most cores are on average smaller than all multilayer cores. Nonetheless, for the levels 12 and 13 of the Amazon dataset, inner-most cores have greater size than all cores. This behavior is consistent with our definitions: inner-most cores are cores without descendants, thus they are expected to be the smallest-sized ones, but they do not necessarily have to. Finally, the distribution of the average-degree density exhibits a similar trend to the distribution of the size: this is expected as the two measures depend on each other.

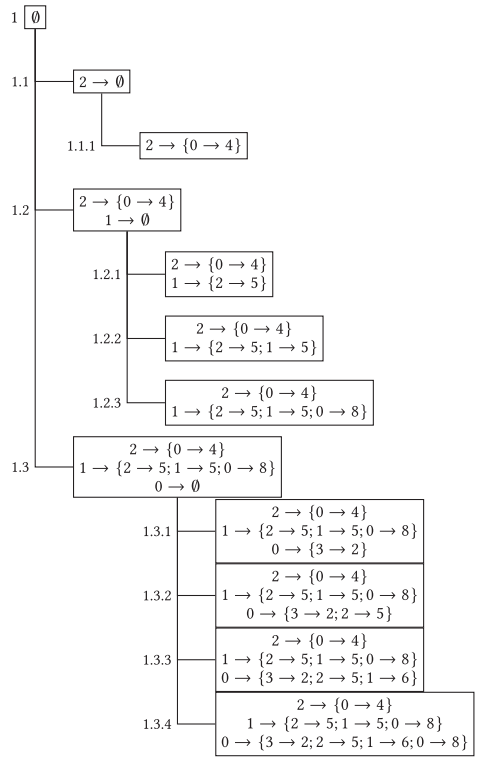

Fig. 8. Content of the $\mathcal{M}$ data structure during the execution of the IM-ML-cores algorithm as per the example shown in Figure 7.

Table 3. Runtime (in seconds) of the Methods for Multilayer Core Decomposition, the a-posteriori Filtering of the Inner-Most Multilayer Cores, and the Proposed IM-ML-cores Method for Directly Computing Inner-Most Multilayer Cores

Dataset	BFS	DFS	H	Filtering	IM
Homo	13	27	12	0.5	5
SacchCere	1,134	2,627	1,146	24	336
DBLP	68	282	29	1	148
ObamaInIsrael	226	150	177	7	120
Amazon	3,981	5,278	3,913	129	2,530
FriendfeedTwitter	61,113	1,973	59,520	276	1,583
Higgs	2,480	640	2,169	33	356
Friendfeed	58,278	13,356	47,179	394	2,640

5 MULTILAYER DENSEST SUBGRAPH

In this section, we showcase the usefulness of multilayer core-decomposition in the context of multilayer densest-subgraph discovery. Particularly, we show how to exploit the multilayer core-decomposition to devise an algorithm with approximation guarantees for the MULTILAYER DENSEST SUBGRAPH problem introduced in Section 2 (Problem 3), thus extending to the multilayer

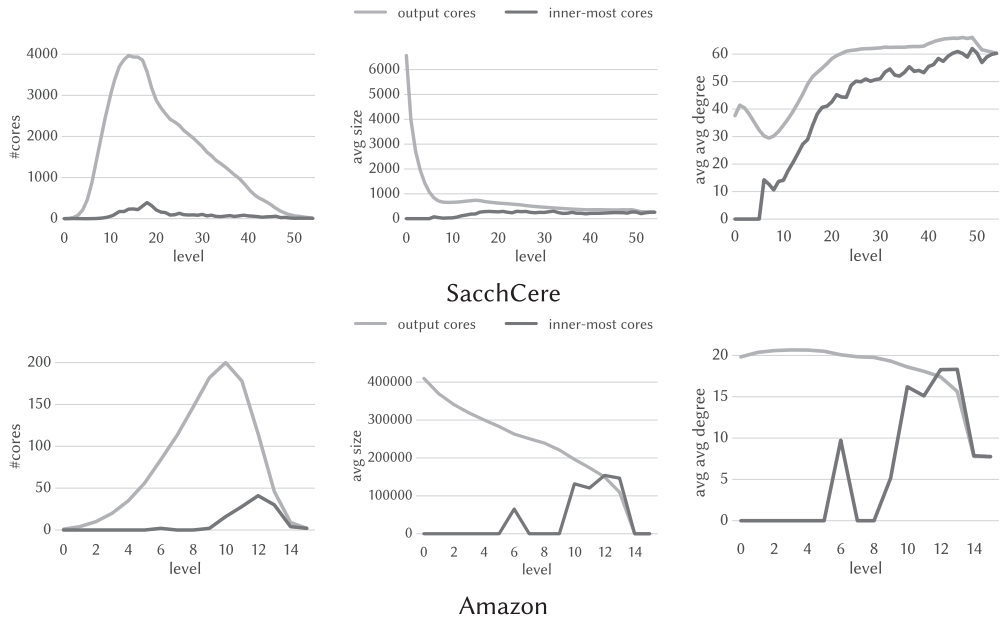

Fig. 9. Comparison of the distributions, to the core-lattice level, of number (left), average size (center), and average average-degree density (right) of multilayer cores and inner-most multilayer cores, for datasets SacchCere (top) and Amazon (bottom).

setting the intuition at the basis of the well-known $\frac{1}{2}$-approximation algorithm [8, 24] for single-layer densest-subgraph extraction.

5.1 Hardness

We start by formally showing that the Multilayer Densest Subgraph problem (Problem 3) is **NP**-hard.

Theorem 3. *Problem 3 is* **NP**-*hard.*

To prove the theorem, we introduce two variants of Problem 3's objective function, i.e., $\delta_{\text{ALL}}(\cdot)$, which considers all layers in L, and $\delta_{\neg\text{ALL}}(\cdot)$, which considers all subsets of layers but the whole layer set L. Specifically, for any given multilayer graph $G = (V, E, L)$ and vertex subset $S \subseteq V$, the two functions are defined as

$$\delta_{\text{ALL}}(S) = \min_{\ell \in L} \frac{|E_\ell[S]|}{|S|} |L|^\beta, \tag{3}$$

$$\delta_{\neg\text{ALL}}(S) = \max_{\hat{L} \in 2^L \setminus \{L\}} \min_{\ell \in \hat{L}} \frac{|E_\ell[S]|}{|S|} |\hat{L}|^\beta. \tag{4}$$

We also define deg_{max} as the maximum degree of a vertex in a layer:

$$deg_{max} = \max_{\ell \in L} \max_{u \in V} deg(u, \ell), \tag{5}$$

and introduce the following three auxiliary lemmas.

Lemma 1. $\delta_{\text{ALL}}(S) \geq \frac{1}{|V|}|L|^\beta$, *for all* $S \subseteq V$ *such that* $\forall \ell \in L : |E_\ell[S]| > 0$.

Proof. For a vertex set S spanning at least one edge in every layer, it holds that $\min_{\ell \in L} \frac{|E_\ell[S]|}{|S|} \geq \frac{1}{|V|}$, and, therefore, $\delta_{\text{ALL}}(S) = \min_{\ell \in L} \frac{|E_\ell[S]|}{|S|} |L|^\beta \geq \frac{1}{|V|}|L|^\beta$. □

Lemma 2. *$\delta_{\neg\text{ALL}}(S) \leq \frac{deg_{max}}{2}(|L|-1)^{\beta}$, for all $S \subseteq V$.*

Proof. The maximum density of a vertex set S in a layer can be at most equal to the density of the maximum clique, i.e., at most $\frac{(deg_{max}+1)\ deg_{max}}{2\ (deg_{max}+1)} = \frac{deg_{max}}{2}$. At the same time, the size of a layer set $\hat{L}$ in the function $\delta_{\neg\text{ALL}}(\cdot)$ can be at most $|L|-1$ (as the whole layer set L is not considered in $\delta_{\neg\text{ALL}}(\cdot)$). This means that $\delta_{\neg\text{ALL}}(S) = \max_{\hat{L}\in 2^L\setminus\{L\}} \min_{\ell\in\hat{L}} \frac{|E_\ell[S]|}{|S|}|\hat{L}|^{\beta} \leq \frac{deg_{max}}{2}(|L|-1)^{\beta}$. □

Lemma 3.

$$\beta > \frac{\log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1)}{1-\log_{|L|}(|L|-1)} \quad \Leftrightarrow \quad \frac{1}{|V|}|L|^{\beta} > \frac{deg_{max}}{2}(|L|-1)^{\beta}.$$

Proof.

$$\begin{aligned}
\beta &> \frac{\log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1)}{1-\log_{|L|}(|L|-1)} \\
&\Leftrightarrow \left(1-\log_{|L|}(|L|-1)\right)\beta \ > \ \log_{|L|-1}\left(\tfrac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1) \\
&\Leftrightarrow \beta \ > \ \log_{|L|-1}\left(\tfrac{|V|}{2}deg_{max}\right) \times \log_{|L|}(|L|-1) + \beta\log_{|L|}(|L|-1) \\
&\Leftrightarrow \frac{\beta}{\log_{|L|}(|L|-1)} \ > \ \log_{|L|-1}\left(\tfrac{|V|}{2}deg_{max}\right) + \beta \\
&\Leftrightarrow \frac{\log_{|L|}|L|^{\beta}}{\log_{|L|}(|L|-1)} \ > \ \log_{|L|-1}\left(\tfrac{|V|}{2}deg_{max}\right) + \log_{|L|-1}(|L|-1)^{\beta} \\
&\Leftrightarrow \log_{|L|-1}|L|^{\beta} \ > \ \log_{|L|-1}\left(\tfrac{|V|}{2}deg_{max}(|L|-1)^{\beta}\right) \\
&\Leftrightarrow |L|^{\beta} \ > \ \tfrac{|V|}{2}deg_{max}(|L|-1)^{\beta} \\
&\Leftrightarrow \frac{1}{|V|}|L|^{\beta} \ > \ \frac{deg_{max}}{2}(|L|-1)^{\beta}.
\end{aligned}$$

□

With Lemmas 1–3 in place, we are now ready to provide the ultimate proof of Theorem 3.

Proof. We reduce from the Min-Avg Densest Common Subgraph (DCS-MA) problem [49], which aims at finding a subset of vertices $S \subseteq V$ from a multilayer graph $G = (V, L, S)$ maximizing $\min_{\ell\in L}\frac{E_\ell[S]}{|S|}$, and has been recently shown to be **NP**-hard in [25]. We distinguish two cases. The first (trivial) one is when G has a layer with no edges. In this case any vertex subset would be an optimal solution for DCS-MA (with overall objective function equal to zero), including the optimal solution to our Multilayer Densest Subgraph problem run on the same G (no matter which β is used). In the second case, G has at least one edge in every layer. In this case solving our Multilayer Densest Subgraph problem on G, with β set to any value $> \frac{\log_{|L|-1}\left(\frac{|V|}{2}deg_{max}\right)\times\log_{|L|}(|L|-1)}{1-\log_{|L|}(|L|-1)}$, gives a solution that is optimal for DCS-MA as well. Indeed, it can be observed that, for all $S \subseteq V$ such that $\forall\ell \in L : |E_\ell[S]| > 0$:

$$\begin{aligned}
\delta_{\text{ALL}}(S) \quad &\geq \quad \frac{1}{|V|}|L|^{\beta} && \{\text{Lemma 1}\} \\
&> \quad \frac{deg_{max}}{2}(|L|-1)^{\beta} && \{\text{Lemma 3}\} \\
&\geq \quad \delta_{\neg\text{ALL}}(S). && \{\text{Lemma 2}\}
\end{aligned}$$

ALGORITHM 8: ML-densest

Input: A multilayer graph $G = (V, E, L)$ and a real number $\beta \in \mathbb{R}^+$.
Output: $C^* \subseteq V$.

```
C ← MultiLayerCoreDecomposition(G)        # Any of Algorithms 2, 3, 5 can be used
C* ← arg max_{C∈C} δ(C)                   # Equation (1)
```

This means that, for that particular value of β, the optimal solution of Multilayer Densest Subgraph on input G is given by maximizing the $\delta_{\text{ALL}}(\cdot)$ function, which considers all layers and is, as such, equivalent to the objective function underlying the DCS-MA problem. This completes the proof. □

5.2 Algorithms

The approximation algorithm we devise for the Multilayer Densest Subgraph problem is very simple: it computes the multilayer core decomposition of the input graph, and, among all cores, takes the one maximizing the objective function δ as the output densest subgraph (Algorithm 8). Despite its simplicity, the algorithm achieves provable approximation guarantees proportional to the number of layers of the input graph, precisely equal to $\frac{1}{2|L|^\beta}$. We next formally prove this result.

Let $\mathbf{C}$ be the core decomposition of the input multilayer graph $G = (V, E, L)$ and C^* denote the core in $\mathbf{C}$ maximizing the density function δ, i.e., $C^* = \arg\max_{C \in \mathbf{C}} \delta(C)$. Then, C^* corresponds to the subgraph output by the proposed ML-densest algorithm. Let also $C^{(\mu)}$ denote the subgraph maximizing the minimum degree in a single layer, i.e., $C^{(\mu)} = \arg\max_{S \subseteq V} f(S)$, where $f(S) = \max_{\ell \in L} \mu(S, \ell)$, while $\ell^{(\mu)} = \arg\max_{\ell \in L} \mu(C^{(\mu)}, \ell)$. It is easy to see that $C^{(\mu)} \in \mathbf{C}$. Finally, let S^*_{SL} be the densest subgraph among all single-layer densest subgraphs, i.e., $S^*_{\text{SL}} = \arg\max_{S \subseteq V} g(S)$, where $g(S) = \max_{\ell \in L} \frac{|E_\ell[S]|}{|S|}$, and ℓ^* be the layer where S^*_{SL} exhibits its largest density, i.e., $\ell^* = \arg\max_{\ell \in L} \frac{|E_\ell[S^*_{\text{SL}}]|}{|S^*_{\text{SL}}|}$. We start by introducing the following two lemmas that can straightforwardly be derived from the definitions of C^*, $C^{(\mu)}$, S^*_{SL}, $\ell^{(\mu)}$, and ℓ^*:

Lemma 4. $\delta(C^*) \geq \delta(C^{(\mu)})$.

Proof. By definition, $C^{(\mu)}$ is a multilayer core described by (among others) the coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ with $k_{\ell^{(\mu)}} = \max_{\ell \in L} \mu(C^{(\mu)}, \ell)$, and $k_\ell = 0$, $\forall \ell \neq \ell^{(\mu)}$. Then $C^{(\mu)} \in \mathcal{C}$. As $C^* = \arg\max_{C \in \mathcal{C}} \delta(C)$, it holds that $\delta(C^*) \geq \delta(C^{(\mu)})$. □

Lemma 5. $\delta(S^*) \leq \frac{|E_{\ell^*}[S^*_{SL}]|}{|S^*_{SL}|} |L|^\beta$.

Proof.

$$\delta(S^*) = \max_{\hat{L} \subseteq L} \min_{\ell \in \hat{L}} \frac{|E_\ell[S^*]|}{|S^*|} |\hat{L}|^\beta \leq \max_{\ell \in L} \frac{|E_\ell[S^*]|}{|S^*|} |L|^\beta \leq \frac{|E_{\ell^*}[S^*_{\text{SL}}]|}{|S^*_{\text{SL}}|} |L|^\beta. \qquad \square$$

The following further lemma shows a lower bound on the minimum degree of a vertex in S^*_{SL}:

Lemma 6. $\mu(S^*_{SL}, \ell^*) \geq \frac{|E_{\ell^*}[S^*_{SL}]|}{|S^*_{SL}|}$.

Proof. As S^*_{SL} is the subgraph maximizing the density in layer ℓ^*, removing the minimum-degree node from S^*_{SL} cannot increase that density. Thus, it holds that

$$\frac{|E_{\ell^*}[S^*_{\text{SL}}]|}{|S^*|} \geq \frac{|E_{\ell^*}[S^*_{\text{SL}}]| - \mu(S^*_{\text{SL}}, \ell^*)}{|S^*_{\text{SL}}| - 1}$$
$$\Leftrightarrow \mu(S^*_{\text{SL}}, \ell^*) \geq |E_{\ell^*}[S^*_{\text{SL}}]| \frac{|S^*_{\text{SL}}| - 1}{|S^*_{\text{SL}}|} - |E_{\ell^*}[S^*_{\text{SL}}]|$$
$$\Leftrightarrow \mu(S^*_{\text{SL}}, \ell^*) \geq \frac{|E_{\ell^*}[S^*_{\text{SL}}]|}{|S^*_{\text{SL}}|}.$$

□

The approximation factor of the proposed ML-densest algorithm is ultimately stated in the next theorem:

Theorem 4. $\delta(C^*) \geq \frac{1}{2|L|^\beta}\delta(S^*)$.

Proof.

$$\begin{aligned}
\delta(C^*) &\geq \delta(C^{(\mu)}) && \{\text{Lemma 4}\}\\
&\geq \max_{\ell \in L} \frac{|E_\ell[C^{(\mu)}]|}{|C^{(\mu)}|} 1^\beta = \max_{\ell \in L} \frac{|E_\ell[C^{(\mu)}]|}{|C^{(\mu)}|} && \{\text{Equation (1)}\}\\
&\geq \frac{1}{2} \max_{\ell \in L} \mu(C^{(\mu)}, \ell) && \{\text{as avg degree} \geq \text{min degree}\}\\
&= \frac{1}{2}\mu(C^{(\mu)}, \ell^{(\mu)}) && \{\text{by definition of } C^{(\mu)}\}\\
&\geq \frac{1}{2}\mu(S^*_{\text{SL}}, \ell^*) && \{\text{optimality of } C^{(\mu)} \text{ w.r.t. min degree}\}\\
&\geq \frac{1}{2}\frac{|E_{\ell^*}[S^*_{\text{SL}}]|}{|S^*_{\text{SL}}|} && \{\text{Lemma 6}\}\\
&\geq \frac{1}{2|L|^\beta}\delta(S^*). && \{\text{Lemma 5}\}
\end{aligned}$$

□

The following corollary shows that the theoretical approximation guarantee stated in Theorem 4 remains the same even if only the inner-most cores are considered (although, clearly, considering the whole core decomposition may lead to better accuracy in practice).

Corollary 4. *Given a multilayer graph $G = (V, E, L)$, let $\mathbf{C}_{\text{IM}}$ be the set of all inner-most multilayer cores of G, and let $C^*_{\text{IM}} = \arg\max_{C \in \mathbf{C}_{\text{IM}}} \delta(C)$. It holds that $\delta(C^*_{\text{IM}}) \geq \frac{1}{2|L|^\beta}\delta(S^*)$.*

Proof. Let $C^{(\mu)}_{\text{IM}} \in \mathbf{C}_{\text{IM}}$ be an inner-most core of G whose coreness vector has a component equal to $\ell^{(\mu)}$. It is easy to see that the result in Lemma 4 holds for C^*_{IM} and $C^{(\mu)}_{\text{IM}}$ too, i.e., becoming $\delta(C^*_{\text{IM}}) \geq \delta(C^{(\mu)}_{\text{IM}})$, while the proof of Theorem 4 holds as is, by simply replacing C^* with C^*_{IM} and $C^{(\mu)}$ with $C^{(\mu)}_{\text{IM}}$. □

Finally, we observe that the result in Theorem 4 carries over to the DCS-MA problem studied in [25, 49, 65, 70] as well, as that problem can be reduced to our Multilayer Densest Subgraph problem (as shown in Theorem 3).

5.3 Experimental Results

We experimentally evaluate our ML-densest algorithm (Algorithm 8) on the datasets in Table 1. Figure 10 reports the results—minimum average-degree density in a layer, number of selected

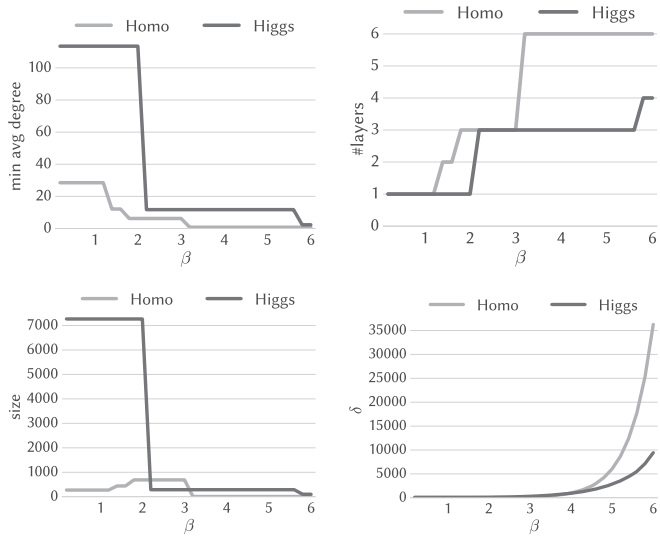

Fig. 10. Multilayer densest-subgraph extraction (Homo and Higgs datasets): minimum average-degree density in a layer, number of selected layers, size, and objective-function value δ of the output densest subgraphs with varying β.

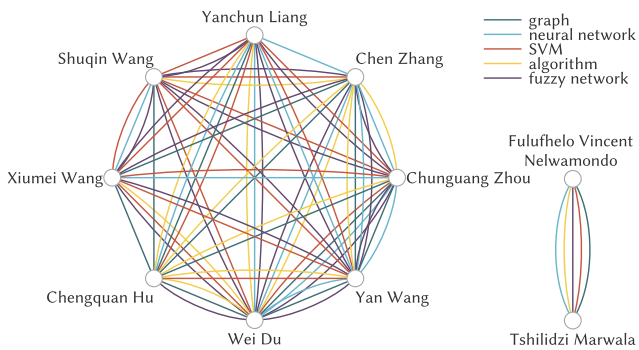

Fig. 11. Multilayer densest subgraph extracted by Algorithm 8 from the DBLP dataset ($\beta = 2.2$).

layers, size, objective-function value δ—on the Homo and Higgs datasets, with varying β. The remaining datasets, which we omit due to space constraints, exhibit similar trends on all measures.

The trends observed in the figure conform to what expected: the smaller β, the more the objective function privileges solutions with large average-degree density in a few layers (or even just one layer, for β close to zero). The situation is overturned with larger values of β, where the minimum average-degree density drops significantly, while the number of selected layers stands at 6 for Homo and 4 for Higgs. In-between β values lead to a balancing of the two terms of the objective function, thus giving more interesting solutions. Also, by definition, δ as a function of β draws exponential curves.

Finally, as anecdotal evidence of the output of Algorithm 8, in Figure 11 we report the densest subgraph extracted from DBLP. The subgraph contains 10 vertices and 5 layers automatically selected by the objective function δ. The minimum average-degree density is encountered on the layers corresponding to topics "*graph*" and "*algorithm*" (green and yellow layers in the figure), and is equal to 1.2. The objective-function value is $\delta = 41.39$. Note that the subgraph is composed of

two connected components. In fact, like the single-layer case, multilayer cores are not necessarily connected.

6 MULTILAYER QUASI-CLIQUES

Another interesting insight into the notion of multilayer cores is about their relationship with (quasi-)cliques. In single-layer graphs it is well-known that cores can be exploited to speed-up the problem of finding cliques, as a clique of size k is guaranteed to be contained into the $(k-1)$-core. Interestingly, a similar relationship holds in the multilayer context too. Given a multilayer graph $G = (V, E, L)$, a layer $\ell \in L$, and a real number $\gamma \in (0, 1]$, a subgraph $G[S] = (S \subseteq V, E[S], L)$ of G is said to be a *γ-quasi-clique* in layer ℓ if all its vertices have at least $\gamma(|S| - 1)$ neighbors in layer ℓ within S, i.e., $\forall u \in S : deg_S(u, \ell) \geq \gamma(|S| - 1)$. Jiang et al. [50] study the problem of extracting *frequent cross-graph quasi-cliques*[11]: given a multilayer graph $G = (V, E, L)$, a function $\Gamma : L \rightarrow (0, 1]$ assigning a real value to every layer in L, a real number $min_sup \in (0, 1]$, and an integer $min_size > 1$, find all maximal subgraphs $G[S]$ of G of size larger than min_size such that there exist at least $min_sup \times |L|$ layers ℓ for which $G[S]$ is a $\Gamma(\ell)$-quasi-clique.

The following theorem shows that a frequent cross-graph quasi-clique of size $\geq min_size$ is necessarily contained into a $\mathbf{k}$-core described by a coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ such that there exists a fraction of min_sup layers ℓ where $k_\ell = \lceil \Gamma(\ell)(min_size - 1) \rceil$.

Theorem 5. *Given a multilayer graph $G = (V, E, L)$, a real-valued function $\Gamma : L \rightarrow (0, 1]$, a real number* $\text{min_sup} \in (0, 1]$, *and an integer* $\text{min_size} > 1$, *a frequent cross-graph quasi-clique of G complying with parameters* Γ, min_sup, *and* min_size *is contained into a* $\mathbf{k}$*-core with coreness vector* $\mathbf{k} = [k_\ell]_{\ell \in L}$ *such that* $|\{\ell \in L : k_\ell = \lceil \Gamma(\ell)(\text{min_size} - 1) \rceil\}| = \lceil \text{min_sup} \times |L| \rceil$.

Proof. Assume that a cross-graph quasi-clique S of G complying with parameters Γ, min_sup, and min_size is not contained into any $\mathbf{k}$-core with coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$ such that $|\{\ell \in L : k_\ell = \lceil \Gamma(\ell)(min_size - 1) \rceil\}| = \lceil min_sup \times |L| \rceil$. This means that S contains a vertex u such that $|\{\ell \in L : deg_S(u, \ell) \geq \Gamma(\ell)(min_size - 1)\}| < min_sup \times |L|$, which means that $|\{\ell \in L : deg_S(u, \ell) \geq \Gamma(\ell)(|S| - 1)\}| < min_sup \times |L|$ as well, since $|S| \geq min_size$. This violates the definition of frequent cross-graph quasi-clique. □

As a simple corollary, the computation of frequent cross-graph quasi-cliques can therefore be circumstantiated to the subgraph given by the union of all multilayer cores complying with the condition stated in Theorem 5.

Corollary 5. *Given a multilayer graph $G = (V, E, L)$, a real-valued function $\Gamma : L \rightarrow (0, 1]$, a real number* $\text{min_sup} \in (0, 1]$, *and an integer* $\text{min_size} > 1$, *let $G' = (V', E', L)$ the subgraph of G given by the union of all multilayer cores of G complying with Theorem 5. It holds that all cross-graph quasi-cliques of G complying with parameters* Γ, min_sup, *and* min_size *are contained into G'.*

The finding in Corollary 5 can profitably be exploited to have a more efficient extraction of frequent cross-graph quasi-cliques. Specifically, the idea is to (*i*) compute *all* multilayer cores of the input graph G (including the non-distinct ones, as the condition stated in Theorem 5 refers to not necessarily maximal coreness vectors); (*ii*) process all multilayer cores of G one by one, retain only the ones complying with Theorem 5, and compute the subgraph G' induced by the union of all such cores; (*iii*) run any algorithm for frequent cross-graph quasi-cliques on G'. Based on the above theoretical results, such a procedure is guaranteed to be sound and complete, and it is

[11]The input in [50] has the form of a set of graphs sharing the same vertex set, which is clearly fully equivalent to the notion of multilayer graph considered in this work.

Table 4. Comparison of the Runtime of the Efficient Extraction of Frequent Cross-Graph Quasi-Cliques by Corollary 5 and of the Original Algorithm [50], for the SacchCere Dataset

Γ							*min_sup*	*min_size*	# solution Quasi-Cliques	$\lvert V'\rvert$	Runtime (s) Corollary 5	Runtime (s) [50]
1	1	1	1	.2	.2	1	0.5	6	2	371	**3**	169
.9	.9	.9	.9	.2	.2	.9			2	371	**25**	17,561
.8	.8	.8	.8	.2	.2	.8			6	1,196	**734**	22,932
.7	.7	.7	.7	.2	.2	.7			6	1,196	**728**	23,376
.6	.6	.6	.6	.2	.2	.6			59	2,300	**5 200**	28,948
.5	.5	.5	.5	.2	.2	.5			59	2,300	**5 123**	29,677

Γ							*min_sup*	*min_size*	# solution Quasi-Cliques	$\lvert V'\rvert$	Runtime (s) Corollary 5	Runtime (s) [50]
.5	.5	.5	.5	.2	.2	.5	1	3	2	152	**2**	281
							0.9		2	152	**2**	282
							0.8		28	940	**23**	292
							0.7		323	3,271	**205**	411
							0.6		323	3,271	**203**	414
							0.5		1,630	4,581	**2,569**	3,075

Γ							*min_sup*	*min_size*	# solution Quasi-Cliques	$\lvert V'\rvert$	Runtime (s) Corollary 5	Runtime (s) [50]
.5	.5	.5	.5	.2	.2	.5	0.5	7	27	2,254	**5,606**	34,904
								6	59	2,300	**5,123**	29,677
								5	357	3,363	**4,493**	21,206
								4	378	3,363	**3,704**	15,465
								3	1,630	4,581	**2,569**	3,075

The evaluation is proposed varying one of the parameters, i.e., Γ, *min_sup*, and *min_size*, at a time. The number of solution quasi-cliques and the number of vertices $|V'|$ of the subgraph G' are also reported. For each run of the experiment, the smallest runtime is bolded.

expected to provide a significant speed-up, as G' is expected to be much smaller than the original graph G.

6.1 Experimental Results

We show in Tables 4 and 5 the experimental results about the comparison of the algorithm proposed by Jiang et al. [50] and the more efficient extraction of frequent cross-graph quasi-cliques by Corollary 5. Table 4 refers to the SacchCere dataset, while Table 5 to the DBLP dataset. To evaluate the effect of the parameters, i.e., the function Γ, *min_sup*, and *min_size*, on the performance of the two approaches, we vary a parameter at a time keeping the other two fixed. With regards to the values selected for Γ, we fix $\Gamma(\ell_5) = \Gamma(\ell_6) = 0.2$ in all the experiments involving the SacchCere dataset, due to the imbalance of the distribution of the edges in favor of the other five layers (i.e., layers $\ell_1, \ldots, \ell_4, \ell_7$). Instead, given the uniformity of the edge density across the layers of the DBLP dataset, Γ is modified coherently for all the layer in this latter case. In addition to the execution times, for each configuration of the parameters, we also report the number of solution frequent cross-graph quasi-cliques and the number of vertices $|V'|$ of the subgraph G' identified by Corollary 5.

Table 5. Comparison of the Runtime of the Efficient Extraction of Frequent Cross-Graph Quasi-Cliques by Corollary 5 and of the Original Algorithm [50], for the DBLP Dataset

Γ										*min_sup*	*min_size*	# solution Quasi-Cliques	$\lvert V'\rvert$	Runtime (s) Corollary 5	Runtime (s) [50]
1	1	1	1	1	1	1	1	1	1	0.2	8	2	18	**0.2**	26,496
.9	.9	.9	.9	.9	.9	.9	.9	.9	.9			2	18	**0.2**	26,112
.8	.8	.8	.8	.8	.8	.8	.8	.8	.8			13	75	**0.3**	26,867
.7	.7	.7	.7	.7	.7	.7	.7	.7	.7			18	196	**1**	27,387
.6	.6	.6	.6	.6	.6	.6	.6	.6	.6			18	196	**1**	27,084
.5	.5	.5	.5	.5	.5	.5	.5	.5	.5			121	801	**18**	31,508

Γ										*min_sup*	*min_size*	# solution Quasi-Cliques	$\lvert V'\rvert$	Runtime (s) Corollary 5	Runtime (s) [50]
.5	.5	.5	.5	.5	.5	.5	.5	.5	.5	0.5	3	8	182	**0.2**	26,969
										0.4		195	2,375	**1**	26,964
										0.3		3,394	22,659	**210**	32,981

Γ										*min_sup*	*min_size*	# solution Quasi-Cliques	$\lvert V'\rvert$	Runtime (s) Corollary 5	Runtime (s) [50]
.5	.5	.5	.5	.5	.5	.5	.5	.5	.5	0.2	13	1	75	**0.2**	26,644
											12	1	75	**0.2**	27,136
											11	8	196	**0.7**	26,966
											10	10	196	**0.7**	27,116
											9	116	801	**18**	32,372
											8	121	801	**18**	31,508
											7	1,292	3,468	**181**	113,558
											6	1,370	3,468	**198**	113,520
											5	7,599	15,316	**3,790**	++
											4	8,578	15,316	**3,502**	++

The evaluation is proposed varying one of the parameters, i.e., Γ, *min_sup*, and *min_size*, at a time. The number of solution quasi-cliques and the number of vertices $|V'|$ of the subgraph G' are also reported. ++ indicates runtime longer than 259, 200 seconds (i.e., 3 days). For each run of the experiment, the smallest runtime is bolded.

The first thing to notice is that, in both datasets and for every configuration, our approach is faster than the algorithm by Jiang et al. [50]. The actual speed-up varies with the size of $|V'|$ (with respect to $|V|$), which, in turn, is affected by the mining parameters. For the SacchCere dataset, we obtain the most extreme cases when varying *min_sup* (middle table): our approach is able to prune from 30% ($min_sup = 0.5$) up to 98% ($min_sup = 1$) of the input multilayer graph. For the DBLP dataset, the results are even stronger: in the worst case (i.e., $\Gamma(\ell) = 0.5\ \forall \ell \in L$, $min_sup = 0.3$, and $min_size = 3$) we prune the 95% of the original vertex set. The runtime of both our approach and Jiang et al.'s [50] algorithm varies consistently according to parameters and to $|V'|$. The speed-up that our method reaches ranges from 1.2 to 2 orders of magnitude for the SacchCere dataset, and from one order up to six orders of magnitude for the DBLP dataset.

7 COMMUNITY SEARCH IN MULTILAYER NETWORKS

The idea here is very similar to that of the multilayer densest subgraph.

Problem 5 (Multilayer Community Search). *Given a multilayer graph $G = (V, E, L)$, a set of vertices $S \subseteq V$, and a set of layers $\hat{L} \subseteq L$, we define the minimum degree of a vertex in S, within the subgraph induced by S and $\hat{L}$ as*

$$\varphi(S, \hat{L}) = \min_{\ell \in \hat{L}} \min_{u \in S} deg_S(u, \ell).$$

Given a positive real number β, we define a real-valued density function $\vartheta : 2^V \to \mathbb{R}^+$ as

$$\vartheta(S) = \max_{\hat{L} \subseteq L} \varphi(S, \hat{L}) |\hat{L}|^\beta.$$

Given a set $V_Q \subseteq V$ of query vertices, find a subgraph containing all the query vertices and maximizing the density function, i.e.,

$$S^* = \arg\max_{V_Q \subseteq S \subseteq V} \vartheta(S). \tag{6}$$

Let $\mathbf{C}$ be the set of all non-empty multilayer cores of G. For a core $C \in \mathbf{C}$ with coreness vector $\mathbf{k} = [k_\ell]_{\ell \in L}$, we define the score

$$\sigma(C) = \max_{\hat{L} \subseteq L} (\min_{\ell \in \hat{L}} k_\ell) |\hat{L}|^\beta,$$

and denote by C^* a core that contains all query vertices in V_Q and maximizes the score σ, i.e.,

$$C^* = \arg\max_{C \in \mathbf{C}, V_Q \subseteq C} \sigma(C). \tag{7}$$

As shown in the following theorem, C^* is a (not necessarily unique) *exact* solution to Problem 5.

Theorem 6. *Given a multilayer graph $G = (V, E, L)$, and a set $V_Q \subseteq V$ of query vertices, let S^* and C^* be the vertex sets defined as in Equation (6) and Equation (7), respectively. It holds that $\vartheta(C^*) = \vartheta(S^*)$.*

Proof. We prove the statement by contradiction, assuming that $\vartheta(C^*) < \vartheta(S^*)$. Let $\mu_\ell = \min_{u \in S^*} deg_{S^*}(u, \ell)$, and $\mu = [\mu_\ell]_{\ell \in L}$. By definition of multilayer core, there exists a core $C \in \mathbf{C}$ of G with coreness vector μ such that $S^* \subseteq C$. This means that

$$\sigma(C) = \max_{\hat{L} \subseteq L} (\min_{\ell \in \hat{L}} \mu_\ell) |\hat{L}|^\beta = \max_{\hat{L} \subseteq L} (\min_{\ell \in \hat{L}} \min_{u \in S^*} deg_{S^*}(u, \ell)) |\hat{L}|^\beta = \vartheta(S^*).$$

Thus, there exists a core $C \in \mathbf{C}$ whose $\vartheta(\cdot)$ score is equal to $\vartheta(S^*)$, which contradicts the original assumption $\vartheta(C^*) < \vartheta(S^*)$. □

Algorithms. The core C^* can be straightforwardly found by running any of the proposed algorithms for multilayer core decomposition—BFS-ML-cores (Algorithm 2), DFS-ML-cores (Algorithm 3), or HYBRID-ML-cores (Algorithm 5)—and taking from the overall output core set the core maximizing the $\sigma(\cdot)$ score. However, thanks to the constraint about containment of query vertices V_Q, the various algorithms can be speeded up by preventively skipping the computation of cores that do not contain V_Q. Specifically, this corresponds to the following simple modifications:

- BFS-ML-cores (Algorithm 2): replace the condition at Line 7 with "**if** $V_Q \subseteq C_{\mathbf{k}}$ **then**."
- DFS-ML-cores (Algorithm 3): stop the **k**-coresPath subroutine used at Lines 5 and 6 as soon as a core not containing V_Q is encountered and make the subroutine return only the cores containing V_Q.
- HYBRID-ML-cores (Algorithm 5): replace the condition at Line 9 with "**if** $V_Q \subseteq C_{\mathbf{k}}$ **then**."

Table 6. Comparison of the Average Runtime (in seconds) between the Original Algorithms for Multilayer Core Decomposition and Modified Methods for Community Search, with Varying the Number $|V_Q|$ of Query Vertices

			$\lvert V_Q \rvert$									
Dataset	Method	Original	1	2	3	4	5	6	7	8	9	10
Homo	BFS	13	2	1	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.6
	DFS	27	3	2	1	1	1	0.9	0.9	0.9	0.9	0.9
	H	12	**0.9**	**0.3**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**
SacchCere	BFS	1,134	**162**	**25**	6	3	1	1	0.7	0.7	0.5	0.5
	DFS	2,627	390	58	13	6	2	2	1	1	0.7	0.6
	H	1,146	166	**25**	**5**	**2**	**0.5**	**0.8**	**0.2**	**0.2**	**0.1**	**0.1**
DBLP	BFS	68	35	35	34	34	34	34	35	34	35	36
	DFS	282	55	42	39	39	38	38	38	38	39	39
	H	29	**5**	**5**	**5**	**5**	**5**	**6**	**6**	**6**	**6**	**6**
Obama InIsrael	BFS	226	42	36	34	33	31	32	32	32	32	33
	DFS	150	51	38	34	33	31	31	31	30	31	31
	H	177	**15**	**10**	**10**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
Amazon	BFS	3,981	2,125	1,364	608	582	441	234	231	192	175	167
	DFS	5,278	3,103	2,105	1,198	1,072	851	523	515	434	406	371
	H	3,913	**2,109**	**1,342**	**570**	**546**	**405**	**190**	**190**	**150**	**134**	**127**
Friendfeed Twitter	BFS	61,113	2,464	1,004	597	333	243	185	117	108	85	59
	DFS	1,973	**129**	**73**	**48**	**33**	**30**	**27**	**22**	**21**	**19**	**17**
	H	59,520	2,340	916	523	278	193	136	78	69	49	28
Higgs	BFS	2,480	351	149	91	65	62	56	50	45	40	41
	DFS	640	**125**	**77**	60	52	51	46	46	42	42	39
	H	2,169	239	80	**43**	**23**	**21**	**16**	**14**	**9**	**8**	**8**
Friendfeed	BFS	58,278	150	51	27	25	25	24	23	23	23	23
	DFS	13,356	803	220	82	68	68	66	58	58	59	57
	H	47,179	**10**	**4**	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**2**

In each dataset and for each $|V_Q|$, the smallest runtime is bolded.

7.1 Experimental Results

We experimentally prove the efficiency of the modifications adopted by our algorithms for multilayer community search by reporting a comparison against the original algorithms with no such modifications. Therefore, we consider as baselines the algorithms introduced in Section 3 for computing the entire multilayer core decomposition, i.e., the BFS-ML-cores, DFS-ML-cores, and HYBRID-ML-cores algorithms. We vary the size $|V_Q|$ of the query-vertex set from 1 to 10. For every query-set size, we select—uniformly at random—a number of 100 different query-vertex sets from the whole vertex set. We also vary β from 0.1 and 100. The runtime with varying $|V_Q|$ is shown in Table 6. All results are averaged over the various query-vertex sets sampled.

In all datasets and for all algorithms, the modifications yield considerable improvement. For $|V_Q| = 1$, which is the most demanding scenario in terms of runtime, we achieve from one to three orders of magnitude of speedup in all the cases (with the exception of Amazon). As the number of query vertices increases, the modifications become even more effective: for $|V_Q| > 2$, we obtain at least one order of magnitude of speedup, up to a maximum of four orders of magnitude on the Friendfeed dataset.

As a further insight, for a number of query vertices $|V_Q| \leq 2$, the runtime of the methods for multilayer community search is strongly dependent on the underlying algorithm for multilayer core decomposition. For example, on the SacchCere and Higgs datasets, H is outperformed by BFS and DFS, respectively. The picture is instead different for $|V_Q| > 2$: H turns out to be the fastest algorithm in all the datasets, with the exception of FriendfeedTwitter, for which DFS achieves better performance up to 10 query vertices. Therefore, in general, the core-lattice visit performed by H results to be more effective in identifying the solution multilayer core quickly. In the case of FriendfeedTwitter instead, the gap between the original runtime of DFS and H is so marked that, even if H yields better speedup, it is not able to outperform DFS. This behavior is mainly motivated by the small number of layers of FriendfeedTwitter (only 2), which, as already observed beforehand, favors DFS in terms of runtime.

8 CONCLUSIONS

Core decomposition has been proven to be a fundamental graph-analysis tool with plenty of applications. In this work, we study core decomposition in multilayer networks, characterizing its usefulness, its relation to other problems, and its complexity. We then devise three efficient algorithms for computing the whole core decomposition of a multilayer network and we show a series of non-trivial applications of the core decomposition to solve related problems. In particular:

- Given the large number of multilayer cores, we devise an efficient algorithm for efficiently computing the inner-most cores only.
- We study densest-subgraph extraction in multilayer graphs as a proper optimization problem tradingoff between high density and layers exhibiting high density, and show how core decomposition can be used to approximate this problem with quality guarantees.
- We show how the multilayer core-decomposition tool can be theoretically exploited as a data-reduction, preliminary step to speed up the extraction of frequent cross-graph quasi-cliques, and experimentally prove the effectiveness of our approach with respect to the original algorithm that searches for frequent cross-graph quasi-cliques in the whole input graph.
- We generalize the multilayer community-search problem to the multilayer case and show how to exploit multilayer core decomposition to obtain optimal solutions to this problem.

In our ongoing and future investigation we plan to employ multilayer core decomposition for the analysis of multilayer brain networks in which each layer represents a patient, vertices are brain regions, and edges are co-activation interactions measured by functional magnetic resonance imaging scans. In this scenario, the multilayer core-decomposition tool may be particularly powerful in the task of identifying common patterns to patients affected by diseases or under the assumption of drugs and, also, to select features in order to discriminate diseased patients from healthy individuals.

REFERENCES

[1] Leman Akoglu, Duen Horng Chau, Christos Faloutsos, Nikolaj Tatti, Hanghang Tong, and Jilles Vreeken. 2013. Mining connection pathways for marked nodes in large graphs. In *Proceedings of the SIAM International Conference on Data Mining (SDM'13)*. 37–45.

[2] Hidayet Aksu, Mustafa Canim, Yuan-Chi Chang, Ibrahim Korpeoglu, and Özgür Ulusoy. 2014. Distributed k-core view materialization and maintenance for large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2439–2452.

[3] J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2005. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Proceedings of the Conference on Advances in Neural Information Processing Systems (NIPS'05)*. 41–50.

[4] Reid Andersen and Kumar Chellapilla. 2009. Finding dense subgraphs with size bounds. In *Proceedings of the International Work on Algorithms and Models for the Web-Graph (WAW'09)*. 25–37.

[5] Reid Andersen and Kevin J. Lang. 2006. Communities from seed sets. In *Proceedings of the World Wide Web Conference (WWW'06)*. 223–232.

[6] Sanjeev Arora, David R. Karger, and Marek Karpinski. 1995. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'95)*. 284–293.

[7] Yuichi Asahiro, Refael Hassin, and Kazuo Iwama. 2002. Complexity of finding dense subgraphs. *Discrete Applied Mathematics* 121, 1–3 (2002), 15–26.

[8] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily finding a dense subgraph. *Journal of Algorithms* 34, 2 (2000), 203–221.

[9] N. Azimi-Tafreshi, J. Gómez-Gardeñes, and S. N. Dorogovtsev. 2014. k-core percolation on multiplex networks. *Physical Review E* 90, 3 (2014), 032816.

[10] Gary D. Bader and Christopher W. V. Hogue. 2003. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics* 4, 3 (2003), 2.

[11] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Densest subgraph in streaming and MapReduce. *Proceedings of the VLDB Endowment* 5, 5 (2012), 454–465.

[12] Oana Denisa Balalau, Francesco Bonchi, T. H. Chan, Francesco Gullo, and Mauro Sozio. 2015. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the International Conference on Web Search and Data Mining (WSDM'15)*. 379–388.

[13] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data Mining and Knowledge Discovery* 29, 5 (2015), 1406–1433.

[14] Vladimir Batagelj, Andrej Mrvar, and Matjaz Zaversnik. 1999. Partitioning approach to visualization of large graphs. In *Proceedings of the International Symposium on Graph Drawing (GD'99)*. 90–97.

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

[16] Michele Berlingerio, Michele Coscia, and Fosca Giannotti. 2011. Finding redundant and complementary communities in multidimensional networks. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'11)*. 2181–2184.

[17] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. 2015. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'15)*. 173–182.

[18] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022. https://dl.acm.org/citation.cfm?id=944937.

[19] Brigitte Boden, Stephan Günnemann, Holger Hoffmann, and Thomas Seidl. 2012. Mining coherent subgraphs in multilayer graphs with edge labels. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1258–1266.

[20] Francesco Bonchi, Aristides Gionis, Francesco Gullo, Charalampos E. Tsourakakis, and Antti Ukkonen. 2015. Chromatic correlation clustering. *ACM Transactions on Knowledge Discovery from Data* 9, 4 (2015), 34.

[21] Francesco Bonchi, Francesco Gullo, and Andreas Kaltenbrunner. 2018. Core decomposition of massive, information-rich graphs. In *Encyclopedia of Social Network Analysis and Mining* (2nd ed.). Springer, 419–428.

[22] Francesco Bonchi, Francesco Gullo, Andreas Kaltenbrunner, and Yana Volkovich. 2014. Core decomposition of uncertain graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1316–1325.

[23] Deng Cai, Zheng Shao, Xiaofei He, Xifeng Yan, and Jiawei Han. 2005. Community mining from multi-relational networks. In *Proceedings of the European Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'05)*. 445–452.

[24] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'00)*. 84–95.

[25] Moses Charikar, Yonatan Naamad, and Jimmy Wu. 2018. On finding dense common subgraphs. arXiv:1802.06361.

[26] James Cheng, Yiping Ke, Shumo Chu, and M Tamer Özsu. 2011. Efficient core decomposition in massive networks. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'11)*. 51–62.

[27] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'14)*. 991–1002.

[28] Mark E. Dickison, Matteo Magnani, and Luca Rossi. 2016. *Multilayer Social Networks*. Cambridge University Press.

[29] Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E. Lee, and John H. Thornton, Jr. 2009. Migration motif: A spatial-temporal pattern mining approach for financial markets. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1135–1144.

[30] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. 2015. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the World Wide Web Conference (WWW'15)*. 300–310.

[31] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing all maximal cliques in sparse graphs in near-optimal time. In *Proceedings of the International Symposium on Algorithms and Computation (ISAAC'10)*. 403–414.

[32] Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. 2004. Fast discovery of connection subgraphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 118–127.

[33] Yixiang Fang, Reynold Cheng, Yankai Chen, Siqiang Luo, and Jiafeng Hu. 2017. Effective and efficient attributed community search. *The VLDB Journal* 26, 6 (2017), 803–828.

[34] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment* 10, 6 (2017), 709–720.

[35] Uriel Feige, Guy Kortsarz, and David Peleg. 2001. The dense *k*-subgraph problem. *Algorithmica* 29, 3 (2001), 410–421.

[36] Eugene Fratkin, Brian T. Naughton, Douglas L. Brutlag, and Serafim Batzoglou. 2006. MotifCut: Regulatory motifs finding with maximum density subgraphs. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB'06)*. 156–157.

[37] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. 2016. Top-k overlapping densest subgraphs. *Data Mining and Knowledge Discovery* 30, 5 (2016), 1134–1165.

[38] Edoardo Galimberti, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2018. Mining (maximal) span-cores from temporal networks. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'18)*. 107–116.

[39] Edoardo Galimberti, Francesco Bonchi, and Francesco Gullo. 2017. Core decomposition and densest subgraph in multilayer networks. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'17)*. 1807–1816.

[40] Antonios Garas, Frank Schweitzer, and Shlomo Havlin. 2012. A *k*-shell decomposition method for weighted networks. *New Journal of Physics* 14, 8 (2012), 083030.

[41] David García, Pavlin Mavrodiev, and Frank Schweitzer. 2013. Social resilience in online communities: The autopsy of Friendster. In *Proceedings of the ACM Conference on Online Social Networks (COSN'13)*. 39–50.

[42] Christos Giatsidis, Dimitrios M. Thilikos, and Michalis Vazirgiannis. 2013. D-cores: Measuring collaboration of directed graphs based on degeneracy. *Knowledge and Information Systems* 35, 2 (2013), 311–343.

[43] David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering large dense subgraphs in massive graphs. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'05)*. 721–732.

[44] A. V. Goldberg. 1984. *Finding a Maximum Density Subgraph.* Technical Report. University of California at Berkeley.

[45] John Healy, Jeannette Janssen, Evangelos Milios, and William Aiello. 2006. Characterization of graphs using degree cores. In *Proceedings of the International Workshop on Algorithms and Models for the Web-Graph (WAW'06)*. 137–148.

[46] Xin Huang and Laks V. S. Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[47] Xin Huang, Laks V. S. Lakshmanan, and Jianliang Xu. 2017. Community search over big graphs: Models, algorithms, and opportunities. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'17)*. 1451–1454.

[48] Roberto Interdonato, Andrea Tagarelli, Dino Ienco, Arnaud Sallaberry, and Pascal Poncelet. 2017. Local community detection in multilayer networks. *Data Mining and Knowledge Discovery* 31, 5 (2017), 1444–1479.

[49] Vinay Jethava and Niko Beerenwinkel. 2015. Finding dense subgraphs in relational graphs. In *Proceedings of the European Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'15)*. 641–654.

[50] Daxin Jiang and Jian Pei. 2009. Mining frequent cross-graph quasi-cliques. *ACM Transactions on Knowledge Discovery from Data* 2, 4 (2009), 16.

[51] Wissam Khaouid, Marina Barsky, S. Venkatesh, and Alex Thomo. 2015. K-core decomposition of large networks on a single PC. *Proceedings of the VLDB Endowment* 9, 1 (2015), 13–23.

[52] Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley, and Hernán A. Makse. 2010. Identification of influential spreaders in complex networks. *Nature Physics* 6, 11 (2010), 888–893.

[53] Isabel M. Kloumann and Jon M. Kleinberg. 2014. Community membership identification from small seed sets. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1366–1375.

[54] Guy Kortsarz and David Peleg. 1994. Generating sparse 2-spanners. *Journal of Algorithms* 17, 2 (1994), 222–236.

[55] Michael A. Langston, Lan Lin, Xinxia Peng, Nicole E. Baldwin, Christopher T. Symons, Bing Zhang, and Jay R. Snoddy. 2005. A combinatorial approach to the analysis of differential gene expression data: The use of graph algorithms for disease prediction and screening. In *Methods of Microarray Data Analysis IV*. Springer, 223–238.

[56] Kyu-Min Lee, Byungjoon Min, and Kwang-Il Goh. 2015. Towards real-world complexity: an introduction to multiplex networks. *The European Physical Journal B* 88, 2 (2015), 48.

[57] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu C. Aggarwal. 2010. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*. Springer.

[58] Rong-Hua Li, Jeffrey Xu Yu, and Rui Mao. 2014. Efficient core maintenance in large dynamic graphs. *IEEE Transactions on Knowledge and Data Engineering* 26, 10 (2014), 2453–2465.

[59] David W. Matula and Leland L. Beck. 1983. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM* 30, 3 (1983), 417–427.

[60] Alberto Montresor, Francesco De Pellegrini, and Daniele Miorandi. 2013. Distributed k-core decomposition. *IEEE Transactions on Parallel and Distributed Systems* 24, 2 (2013), 288–300.

[61] Peter J. Mucha, Thomas Richardson, Kevin Macon, Mason A. Porter, and Jukka-Pekka Onnela. 2010. Community structure in time-dependent, multiscale, and multiplex networks. *Science* 328, 5980 (2010), 876–878.

[62] Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. 2017. Fully dynamic algorithm for top-k densest subgraphs. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'17)*. 1817–1826.

[63] Evangelos E. Papalexakis, Leman Akoglu, and Dino Ience. 2013. Do more views of a graph help? Community detection and clustering in multi-graphs. In *Proceedings of the International Conference on Information Fusion (FUSION'13)*. 899–905.

[64] Katerina Pechlivanidou, Dimitrios Katsaros, and Leandros Tassiulas. 2014. MapReduce-based distributed k-shell decomposition for online social networks. In *Proceedings of the IEEE World Congress on Services (SERVICES'14)*. IEEE, 30–37.

[65] Alexander Reinthal, Anton Törnqvist, Arvid Andersson, Erik Norlander, Philip Stålhammar, and Sebastian Norlin. 2016. Finding the densest common subgraph with linear programming. B.Sc. Thesis. Chalmers University of Technology, University of Gothenburg, Department of Computer Science and Engineering.

[66] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2017. To be connected, or not to be connected: That is the minimum inefficiency subgraph problem. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'17)*. 879–888.

[67] Natali Ruchansky, Francesco Bonchi, David García-Soriano, Francesco Gullo, and Nicolas Kourtellis. 2015. The minimum Wiener connector problem. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'15)*. 1587–1602.

[68] Ahmet Erdem Saríyüce, Buğra Gedik, Gabriela Jacques-Silva, Kun-Lung Wu, and Ümit V. Çatalyürek. 2013. Streaming algorithms for k-core decomposition. *Proceedings of the VLDB Endowment* 6, 6 (2013), 433–444.

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

[70] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. 2019. Finding lasting dense subgraphs. *Data Mining and Knowledge Discovery* 33, 5 (2019), 1417–1445.

[71] Chih-Ya Shen, Hong-Han Shuai, De-Nian Yang, Yi-Feng Lan, Wang-Chien Lee, Philip S. Yu, and Ming-Syan Chen. 2015. Forming online support groups for internet and behavior related addictions. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM'15)*. 163–172.

[72] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 939–948.

[73] Andrea Tagarelli, Alessia Amelio, and Francesco Gullo. 2017. Ensemble-based community detection in multilayer networks. *Data Mining and Knowledge Discovery* 31, 5 (2017), 1506–1543.

[74] Lei Tang, Xufei Wang, and Huan Liu. 2010. *Community Detection in Multi-dimensional Networks*. Technical Report. DTIC Document.

[75] Hanghang Tong and Christos Faloutsos. 2006. Center-piece subgraphs: Problem definition and fast solutions. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 404–413.

[76] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *Proceedings of World Wide Web Conference (WWW'15)*. 1122–1132.

[77] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 104–112.

[78] Nan Wang, Jingbo Zhang, Kian-Lee Tan, and Anthony K. H. Tung. 2010. On triangulation-based dense neighborhood graph discovery. *Proceedings of the VLDB Endowment* 4, 2 (2010), 58–68.

[79] Dong Wen, Lu Qin, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2016. I/O efficient core graph decomposition at Web scale. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'16)*. 133–144.

[80] Huanhuan Wu, James Cheng, Yi Lu, Yiping Ke, Yuzhen Huang, Da Yan, and Hejun Wu. 2015. Core decomposition in large temporal graphs. In *Proceedings of the IEEE International Conference on Big Data*. 649–658.

[81] Yubao Wu, Ruoming Jin, Xiaofeng Zhu, and Xiang Zhang. 2015. Finding dense and connected subgraphs in dual networks. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'15)*. 915–926.

[82] Stefan Wuchty and Eivind Almaas. 2005. Peeling the yeast protein network. *Proteomics* 5, 2 (2005), 444–449.
[83] Xifeng Yan, X. Zhou, and Jiawei Han. 2005. Mining closed relational graphs with connectivity constraints. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 324–333.
[84] Zin Mar Yin and Soe Soe Khaing. 2013. Multi-layered Graph Clustering in Finding the Community Cores. *International Journal of Advanced Research in Computer Engineering & Technology* 2, 10 (2013), 2674–2677.
[85] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. When engagement meets similarity: Efficient (k, r)-core computation on social networks. *Proceedings of the VLDB Endowment* 10, 10 (2017), 998–1009.
[86] Haohua Zhang, Hai Zhao, Wei Cai, Jie Liu, and Wanlei Zhou. 2010. Using the k-core decomposition to analyze the static structure of large-scale software systems. *Journal of Supercomputing* 53, 2 (2010), 352–369 .
[87] Yikai Zhang, Jeffrey Xu Yu, Ying Zhang, and Lu Qin. 2017. A fast order-based approach for core maintenance. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'17)*. 337–348.
[88] Rong Zhu, Zhaonian Zou, and Jianzhong Li. 2018. Diversified coherent core search on multi-layer graphs. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE'18)*. 701–712.

Received December 2018; revised July 2019; accepted October 2019