Density-friendly Graph Decomposition

Nikolaj Tatti
Aalto University
nikolaj.tatti@aalto.fi

Aristides Gionis
Aalto University
aristides.gionis@aalto.fi

ABSTRACT

Decomposing a graph into a hierarchical structure via k-core analysis is a standard operation in any modern graph-mining toolkit. k-core decomposition is a simple and efficient method that allows to analyze a graph beyond its mere degree distribution. More specifically, it is used to identify areas in the graph of increasing centrality and connectedness, and it allows to reveal the structural organization of the graph.

Despite the fact that k-core analysis relies on vertex degrees, k-cores do not satisfy a certain, rather natural, density property. Simply put, the most central k-core is not necessarily the densest subgraph. This inconsistency between k-cores and graph density provides the basis of our study.

We start by defining what it means for a subgraph to be *locally-dense*, and we show that our definition entails a nested chain decomposition of the graph, similar to the one given by k-cores, but in this case the components are arranged in order of increasing density. We show that such a *locally-dense decomposition* for a graph $G = (V, E)$ can be computed in polynomial time. The running time of the exact decomposition algorithm is $\mathcal{O}(|V|^2|E|)$ but is significantly faster in practice. In addition, we develop a linear-time algorithm that provides a factor-2 approximation to the optimal locally-dense decomposition. Furthermore, we show that the k-core decomposition is also a factor-2 approximation, however, as demonstrated by our experimental evaluation, in practice k-cores have different structure than locally-dense subgraphs, and as predicted by the theory, k-cores are not always well-aligned with graph density.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

Keywords

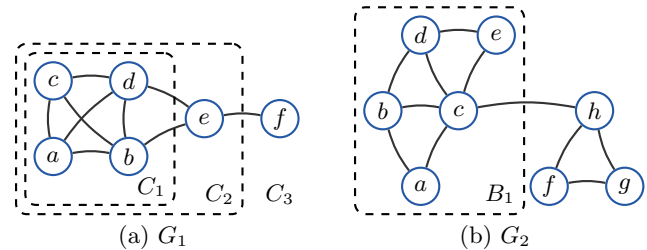community detection; k-core analysis; dense subgraphs

Figure 1: Toy graphs used in Example 1.

1. INTRODUCTION

Finding dense subgraphs and communities is one of the most well-studied problems in graph mining. Techniques for identifying dense subgraphs are used in a large number of application domains, from biology, to web mining, to analysis of social and information networks. Among the many concepts that have been proposed for discovering dense subgraphs, k-cores are particularly attractive for the simplicity of their definition and the fact that they can be identified in linear time.

The k-core of a graph is defined as a maximal subgraph in which every vertex is connected to at least k other vertices within that subgraph. A *k-core decomposition* of a graph consists of finding the set of all k-cores. A nice property is that the set of all k-cores forms a nested sequence of subgraphs, one included in the next. This makes the k-core decomposition of a graph a useful tool in analyzing a graph by identifying areas of increasing centrality and connectedness, and revealing the structural organization of the graph. As a result, k-core decomposition has been applied to a number of different applications, such as modeling of random graphs [8], analysis of the internet topology [12], social-network analysis [23], bioinformatics [6], analysis of connection matrices of the human brain [16], graph visualization [3], as well as influence analysis [19, 28] and team formation [9].

The fact that the k-core decomposition of a graph gives a chain of subgraphs where vertex degrees are higher in the inner cores, suggests that we should expect that the inner cores are, in certain sense, more dense or more connected than the outer cores. As we will show shortly, this statement is not true. Furthermore, in this paper we show how to obtain a graph decomposition for which the statement is true, namely, the inner subgraphs of the decomposition are denser than the outer ones. To quantify density, we adopt

a classic notion used in the densest-subgraph problem [13, 15], where density is defined as the ratio between the edges and the vertices of a subgraph. This density definition can be also viewed as the average degree divided by 2.

Our motivating observation is that k-cores are not ordered according to this density definition. The next example demonstrates that the most inner core is *not* necessarily the densest subgraph, and in fact, we can increase the density by either adding or removing vertices.

EXAMPLE 1. Consider the graph G_1 shown in Figure 1, consisting of 6 vertices and 9 edges. The density of the whole graph is $9/6 = 1.5$. The graph has three k-cores: a 3-core marked as C_1, a 2-core marked as C_2, and a 1-core, corresponding the the whole graph and marked as C_3. The core C_1 has density $6/4 = 1.5$ (it contains 6 edges and 4 vertices), while the core C_2 has density $8/5 = 1.6$ (it contains 8 edges and 5 vertices). In other words, C_1 has lower density than C_2, despite being an inner core.

Let us now consider G_2 shown in Figure 1. This graph has a single core, namely a 2-core, containing the whole graph. The density of this core is equal to $11/8 = 1.375$. However, a subgraph B_1 contains 7 edges and 5 vertices, giving us density $7/5 = 1.4$, which is higher than the density of the only core.

This example motivates us to define an alternative, more density-friendly, graph decomposition, which we call *locally-dense decomposition*. We are interested in a decomposition such that (*i*) the density of the inner subgraphs is higher than the density of the outer subgraphs, (*ii*) the most inner subgraph corresponds to the densest subgraph, and (*iii*) we can compute or approximate the decomposition efficiently.

We achieve our goals by first defining a *locally-dense* subgraph, essentially a subgraph whose density cannot be improved by adding and deleting vertices. We show that these subgraphs are arranged into a hierarchy such that the density decreases as we go towards outer subgraphs and that the most inner subgraph is in fact the densest subgraph.

We provide two efficient algorithms to discover this hierarchy. The first algorithm extends the exact algorithm for discovering the densest subgraph given by Goldberg [15]. This algorithm is based on solving a minimum cut problem on a certain graph that depends on a parameter α. Goldberg showed that for a certain value α (which can be found by binary search), the minimum cut recovers the densest subgraph. One of our contributions is to shed more light into Goldberg's algorithm and show that the same construction allows to discover *all* locally-dense subgraphs by varying α.

Our second algorithm extends the linear-time algorithm by Charikar for approximating dense subgraphs [13]. This algorithm first orders vertices by deleting iteratively a vertex with the smallest degree, and then selects the densest subgraph respecting the order. We extend this idea by using the same order, and finding first the densest subgraph respecting the order, and then iteratively finding the second densest subgraph containing the first subgraph, and so on. We show that this algorithm can be executed in linear time and it achieves a factor-2 approximation guarantee.

Charikar's algorithm and the algorithm for discovering a k-core decomposition are very similar: they both order vertices by deleting vertices with the smallest degree. We show that this connection is profoundly deep and we demonstrate that a k-core decomposition provides a factor-2 approxima-

tion for locally-dense decomposition. On the other hand, our experimental evaluation shows that in practice k-cores have different structure than locally-dense subgraphs, and as predicted by the theory, k-cores are not always well-aligned with graph density.

The remainder of paper is organized as follows. We give preliminary notation in Section 2. We introduce the locally-dense subgraphs in Section 3, present algorithms for discovering the subgraphs in Section 4, and describe the connection to k-core decomposition in Section 5. We present the related work in Section 6 and present the experiments in Section 7. Finally, we conclude the paper with discussion in Section 8.

2. PRELIMINARIES

Graph density. Let $G = (V, E)$ be a graph with $|V| = n$ vertices and $|E| = m$ edges. Given a subset of vertices $X \subseteq V$, it is common to define $E(X) = \{(x, y) \in E \mid x, y \in X\}$, i.e., the edges of G that have both end-points in X. The *density* of the vertex set X is then defined to be

$$d(X) = \frac{|E(X)|}{|X|},$$

that is, half of the *average degree* of the subgraph induced by X. The set of vertices $X \subseteq V$ that maximizes the density measure $d(X)$ is the *densest subgraph* of G.[1]

The problem of finding the densest subgraph can be solved in polynomial time. A very elegant solution that involves a mapping to a series of minimum-cut problems was given by Goldberg [15]. As the fastest algorithm to solve the minimum-cut problem runs in $\mathcal{O}(mn)$ time, this approach is not scalable to very large graphs. On the other hand, there exists a linear-time algorithm that provides a factor-2 approximation to the densest-subgraph problem [4, 13]. This is a greedy algorithm, which starts with the input graph, and iteratively removes the vertex with the lowest degree, until left with an empty graph. Among all subgraphs considered during this vertex-removal process, the algorithm returns the densest.

Next we will provide graph-density definitions that relate pairs of vertex sets. Given two non-overlapping sets of vertices X and Y we first define the *cross edges* between X and Y as

$$E_\times(X, Y) = \{(x, y) \in E \mid x \in X, y \in Y\}.$$

We then define the *marginal edges* from X with respect to Y. Those are the edges that have one end-point in X and the other end-point in either X or Y, that is,

$$E_\Delta(X, Y) = E(X) \cup E_\times(X, Y).$$

The set $E_\Delta(X, Y)$ represents the additional edges that will be included in the induced subgraph of Y if we expand Y by adding X.

Assume that X and Y are non-overlapping. Then, we define the *outer density* of X with respect to Y as

$$d(X, Y) = \frac{|E_\Delta(X, Y)|}{|X|}.$$

Again, this is the "extra density" that we bring to Y if we expand it by appending X to it.

---

[1] We should point out that density is also often defined as $|E(X)|/\binom{|X|}{2}$. This is not the case for this paper.

We will be often dealing with the case where X and Y are overlapping and we would be interested in the outer density of vertices in X that are not already included in Y. Hence, we will expand the definition of outer density to a more general case by defining

$$d(X,Y) = d(X \setminus Y, Y).$$

k-cores. We briefly review the basic background regarding k-cores. The concept was introduced by Seidman [23].

Given a graph $G = (V,E)$, a set of vertices $X \subseteq V$ is a k-core if every vertex in the subgraph induced by X has degree at least k, and X is maximal with respect to this property. A k-core of G can be obtained by recursively removing all the vertices of degree less than k, until all vertices in the remaining graph have degree at least k.

It is not hard to see that if $\{C_i\}$ is the set of all distinct k-cores of G then $\{C_i\}$ forms a nested chain

$$\emptyset = C_0 \subsetneq C_1 \subsetneq \cdots \subsetneq C_\ell = V.$$

Furthermore, the set of vertices S_k that belong in a k-core but not in a $(k-1)$-core is called k-*shell*.

The k-*core decomposition* of G is the process of identifying all k-cores (and all k-shells). Therefore, the k-core decomposition of a graph identifies progressively the internal cores and decomposes the graph shell by shell. A linear-time algorithm to obtain the k-core decomposition was given by Matula and Beck [20]. The algorithm starts by provisionally assigning each vertex v to a core of index $\deg(v)$, an upper bound to the correct core of a vertex. It then repeatedly removes the vertex with the smallest degree, and updates the core index of the neighbors of the removed vertex. Note the similarity of this algorithm, with the 2-approximation algorithm for the densest-subgraph problem [13].

3. LOCALLY-DENSE GRAPH DECOMPOSITION

In this section we present the main concept introduced in this paper, the *locally-dense decomposition* of a graph. We also discuss the properties of this decomposition. We start by defining the concept of a *locally-dense subgraph*.

DEFINITION 2. *A set of vertices W is* locally dense *if there are no $X \subseteq W$ and Y satisfying $Y \cap W = \emptyset$ such that*

$$d(X, W \setminus X) \leq d(Y, W).$$

In other words, for W to be locally dense there should not be an X "inside" W and a Y "outside" W so that the density that Y brings to W is larger than the density that X brings.

Due to the notational simplicity, we will often refer to these sets of vertices as subgraphs.

Interestingly, the property of being locally dense induces a nested chain of subgraphs in G.

PROPOSITION 3. *Let U and W be locally-dense subgraphs. Then either $U \subseteq W$ or $W \subseteq U$.*

PROOF. Assume otherwise. Define $X = U \setminus W$ and $Y = W \setminus U$. Both X and Y should be non-empty sets. Then either $d(X, U \cap W) \leq d(Y, U \cap W)$ or $d(X, U \cap W) > d(Y, U \cap W)$. Assume the former. This implies

$$
\begin{aligned}
d(X, U \setminus X) &= d(X, U \cap W) \\
&\leq d(Y, U \cap W) \\
&\leq d(Y, U),
\end{aligned}
$$

which contradicts the fact that U is locally dense. For the first equality we used the fact that $U \setminus X = U \cap W$, while for the last inequality we used the fact that $E_\times(Y, U \cap W) \leq E_\times(Y, U)$.

The case $d(X, U \cap W) > d(Y, U \cap W)$ is similar. $\square$

The proposition implies that the set of locally-dense subgraphs of a graph forms a nested chain, in the same way that the set of k-cores does.

COROLLARY 4. *A set of locally-dense subgraphs can be arranged into a sequence $B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_k$, where $k \leq |V|$. Moreover, $d(B_i, B_{i-1}) > d(B_{i+1}, B_i)$ for $1 \leq i < k$.*

The chain of locally-dense subgraphs of a graph G, as specified by Corollary 4, defines the *locally-dense decomposition* of G.

We proceed to characterize the locally-dense subgraphs of the decomposition with respect to their *global* density in the whole graph G. We want to characterize the global density of subgraph B_i of the decomposition. B_i cannot be denser than the previous subgraph B_{i-1} in the decomposition, however, we want to measure the density that the additional vertices $S_i = B_i \setminus B_{i-1}$ bring. This density involves edges among vertices of S_i and edges from S_i to the previous subgraph B_{i-1}. This is captured precisely by the concept of *outer density* $d(B_i, B_{i-1})$ defined in the previous section. As the following proposition shows the outer density of B_i with respect to B_{i-1} is maximized over all subgraphs that contain B_{i-1}. In other words, B_i is the densest subgraph we can choose after B_{i-1}, given the containment constraint.

PROPOSITION 5. *Let $\{B_i\}$ be the chain of locally-dense subgraphs. Then $B_0 = \emptyset$, $B_k = V$, and B_i is the densest subgraph properly containing B_{i-1},*

$$B_i = \arg \max_{W \supsetneq B_{i-1}} d(W, B_{i-1}).$$

To prove the proposition we will use the following two following lemmas which we state without a proof.

LEMMA 6. *Let $X \subseteq Y$ be two sets of vertices. Assume $Z \cap Y = \emptyset$. If $d(Z, Y) \geq d(Y, X)$, then $d(Y \cup Z, X) \geq d(Y, X)$.*

LEMMA 7. *Let $X \subseteq Y$ be two sets of vertices. Assume $Z \subseteq Y \setminus X$. If $d(Z, Y \setminus Z) < d(Y, X)$, then $d(Y \setminus Z, X) > d(Y, X)$.*

PROOF OF PROPOSITION 5. Assume inductively that the proposition holds for all $j < i$.

Let $U = \arg \max_{W \supsetneq B_{i-1}} d(W, B_{i-1})$. We will first show that U is locally dense. We argue that there are no sets X and Y with $X \subseteq U$ and $Y \cap U = \emptyset$ that can serve as certificates for U being non locally-dense.

Fix any $X \subseteq U$. Define $X_j = X \cap (B_j \setminus B_{j-1})$ for $j < i$, and $X_i = X \cap (U \setminus B_{i-1})$. Define also $U_j = (U \setminus X) \cup B_{j-1}$ for $j \leq i$. Note that $B_j \subseteq U_j \cup X_j$.

If $X_i \neq \emptyset$, we have $d(X_i, U \setminus X_i) \geq d(U, B_{i-1})$, otherwise Lemma 7 implies that we can delete X_i from U and obtain a more dense subgraph.

Similarly, for $j < i$ with $X_j \neq \emptyset$, we have

$$
\begin{aligned}
d(X_j, U_j \setminus X_j) &\geq d(X_j, B_j \setminus X_j) \\
&\geq d(B_j, B_{j-1}) \\
&> d(U, B_{i-1}),
\end{aligned}
$$

where the first inequality follows from $B_j \setminus X_j \subseteq U_j \setminus X_j$, the second inequality is implied by Lemma 7 and the induction assumption on j, and the last inequality is implied by Lemma 6 and the induction assumption on $j+1, \ldots, i-1$.

These inequalities imply

$$
\begin{aligned}
d(X, U \setminus X) &= \sum_{j=1, X_j \neq \emptyset}^{i} \frac{|X_j|}{|X|} d(X_j, U_j \setminus X_j) \\
&\geq \sum_{j=1}^{i} \frac{|X_j|}{|X|} d(U, B_{i-1}) \\
&= d(U, B_{i-1}).
\end{aligned}
$$

Consider also any set Y with $Y \cap U = \emptyset$. Due to the optimality of U and Lemma 6 we must have $d(Y, U) < d(U, B_{i-1})$.

We conclude that for any X and Y with $X \subseteq U$ and $Y \cap U = \emptyset$ it is $d(X, U \setminus X) > d(Y, U)$, which shows that U is locally dense.

Assume now $U = B_j$ for $j \geq i$. We need to show that $j = i$. Assume otherwise. Since B_i is locally dense, we have $d(B_j \setminus B_i, B_i) < d(B_i, B_{i-1})$. With a simple calculation we can show that $d(B_j \setminus B_i, B_i) < d(B_j, B_{i-1})$. Lemma 7 implies that removing $B_j \setminus B_i$ produces a more dense subgraph which contradicts the optimality of U. $\square$

As a consequence of the previous proposition we can characterize the first subgraph in the decomposition.

COROLLARY 8. *Let $\{B_i\}$ be a locally-dense decomposition of a graph G. Then B_1 is the densest subgraph of G.*

The above discussion motivates the problem of locally-dense graph decomposition, which is the focus of this paper.

PROBLEM 1. *Given a graph $G = (V, E)$ find a maximal sequence of locally-dense subgraphs*

$$
\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_k = V.
$$

4. DECOMPOSITION ALGORITHMS

In this section we propose two algorithms for the problem of locally-dense graph decomposition (Problem 1). The first algorithm gives an *exact solution*, and runs in worst-case time $\mathcal{O}(|V|^2|E|)$, but it is significantly faster in practice. The second algorithm is a linear-time algorithm that provides a factor-2 approximation guarantee.

Both algorithms are inspired by corresponding algorithms for the densest-subgraph problem. The first algorithm by the exact algorithm of Goldberg [15], and the second algorithm by the greedy linear-time algorithm of Charikar [13].

4.1 Exact algorithm

We start our discussion on the exact algorithm for locally-dense graph decomposition by reviewing Goldberg's algorithm [15] for the densest-subgraph problem.

Recall that the densest-subgraph problem asks to find the subset of vertices W that maximizes $d(W) = |E(W)|/|W|$. Given a graph $G = (V, E)$ and a positive number $\alpha \geq 0$ define a function

$$
f(\alpha) = \max_{W \subseteq V} \{|E(W)| - \alpha|W|\},
$$

and the maximizer

$$
F(\alpha) = \arg \max_{W \subseteq V} \{|E(W)| - \alpha|W|\},
$$

where ties are resolved by picking the largest W. Note that the value of the function f decreases as α increases, and as α exceeds a certain value the function becomes 0 by taking $W = \emptyset$. Goldberg observed that the densest-subgraph problem is equivalent to the problem of finding the largest value of α^* for which $f(\alpha^*) \geq 0$ and the maximizer set $F(\alpha^*) = W^*$ is non empty.[2] The densest subgraph is precisely this maximizer set W^*. Furthermore, Goldberg showed how to find the vertex set $W = F(\alpha)$, for a given value of α. This is done by mapping the problem to an instance of the *min-cut problem*, which can be solved in time $\mathcal{O}(|V||E|)$, in a recent breakthrough by Orlin [22]. We will present an extension of this transformation in the next section, where we discuss how to speed-up the algorithm.

Thus, Goldberg's algorithm uses binary search over α and finds the largest value of α^* for which $f(\alpha^*) \geq 0$ and the maximizer set W^* is non empty. Each iteration of the binary search involves a call to a min-cut instance for the current value of α.

Our algorithm for finding the locally-dense decomposition of a graph builds on Goldberg's algorithm [15]. We show that Goldberg's construction has the following, rather remarkable, property: there is a sequence of values $\alpha^* = \alpha_1 \geq \ldots \geq \alpha_k$, for $k \leq n$, which gives all the distinct values of the function f. Furthermore, the corresponding set of subgraphs $\{F(\alpha_1), \ldots, F(\alpha_k)\}$ is exactly the set of all locally-dense subgraphs of G, and thus the solution to our decomposition problem.

Therefore, our algorithm is a simple extension of Goldberg's algorithm: instead of searching only for the optimal value $\alpha_1 = \alpha^*$, we find the whole sequence of α_i's and the corresponding subgraphs.

Next we prove the claimed properties and discuss the algorithm in more detail.

We first show that the distinct maximizers of the function F correspond to the set of locally-dense subgraphs.

PROPOSITION 9. *Let $\{B_i\}$ be the set of locally-dense subgraphs. Then*

$$
B_i = F(\alpha), \text{ for } d(B_{i+1}, B_i) < \alpha \leq d(B_i, B_{i-1}).
$$

PROOF. We first show that $U = F(\beta)$ is a locally-dense subgraph, for any β. Note that for any $X \subseteq U$, we must have $|E_\Delta(X, U \setminus X)| - \beta|X| \geq 0$, otherwise we can delete X from U and obtain a better solution which violates the optimality of $U = F(\beta)$. This implies that $d(X, U \setminus X) = E_\Delta(X, U \setminus X)/|X| \geq \beta$. Similarly, for any Y such that $Y \cap U = \emptyset$, we have $|E_\Delta(Y, U)| - \beta|Y| < 0$ or, equivalently, $d(Y, U) < \beta$. Thus, U is locally dense.

Fix i and select α s.t. $d(B_{i+1}, B_i) < \alpha \leq d(B_i, B_{i-1})$. Let $B_j = F(\alpha)$. If $j > i$, then, due to Corollary 4, $d(B_j, B_{j-1}) < \alpha$ which we can rephrase as

$$
c = |E_\Delta(B_j \setminus B_{j-1}, B_{j-1})| - \alpha|B_j \setminus B_{j-1}| < 0.
$$

If we delete $B_j \setminus B_{j-1}$ from U, then we increase the cost exactly by $-c$, that is, we obtain a better solution which violates the optimality of U. If $j < i$, then Corollary 4

---

[2] This observation is an instance of *fractional programming* [14].

Algorithm 1: EXACTLD(G, X, Y)

input : Graph $G = (V, E)$
         locally-dense subgraphs X and Y with $X \subsetneq Y$
1 $\alpha \leftarrow d(Y, X) + |V|^{-2}$;
2 $Z \leftarrow F(\alpha)$;
3 **if** $Z \neq X$ **then**
4   **output** Z;
5   EXACTLD(G, X, Z);
6   EXACTLD(G, Z, Y);

---

implies that $d(B_{j+1}, B_j) \geq \alpha$, so we can add $B_{j+1} \setminus B_j$ to obtain a better solution. It follows that $B_i = F(\alpha)$. □

Next we need to show that it is possible to search efficiently for the sequence of α's that give the set of locally-dense subgraphs. To that end we will show that if we have obtained two subgraphs $B_x \subsetneq B_y$ of the decomposition (corresponding to values $\alpha_x \geq \alpha_y$), it is possible to pick a new value α so that computing $F(\alpha)$ allows us to make progress in the search process: we either find a new locally-dense subgraph $B_x \subsetneq B_z \subsetneq B_y$ or we establish that no such subgraph exists between B_x and B_y, in other words, B_x and B_y are consecutive subgraphs in our decomposition.

PROPOSITION 10. *Let $\{B_i\}$ be the set of locally-dense subgraphs. Let $B_x \subsetneq B_y$ be two subgraphs. Set $\alpha = d(B_y, B_x) + |V|^{-2}$ and let $B_z = F(\alpha)$. If $x + 1 < y$, then $x < z < y$. If $x + 1 = y$, then $z = x$.*

PROOF. A simple calculation shows that $\alpha > d(B_y, B_x) \geq d(B_y, B_{y-1})$. Proposition 9 now implies that $z < y$.

Assume that $x + 1 < y$. This implies that $d(B_y, B_{y-1}) < d(B_{x+1}, B_x)$ which implies that $d(B_y, B_x) < d(B_{x+1}, B_x)$. Let us write

$$a = E_\Delta(B_y \setminus B_x, B_x), \qquad b = |B_y| - |B_x|,$$
$$c = E_\Delta(B_{x+1} \setminus B_x, B_x), \text{ and} \qquad d = |B_{x+1}| - |B_x|.$$

Let us now bound the difference between the densities as

$$
\begin{aligned}
d(B_{x+1}, B_x) - d(B_y, B_x) &= \frac{a}{b} - \frac{c}{d} \\
&= \frac{ad - bc}{bd} \\
&\geq \frac{1}{bd} \\
&> \frac{1}{|V|^2}.
\end{aligned}
$$

This implies that $\alpha < d(B_{x+1}, B_x)$. Proposition 9 now implies that $z \geq x + 1 > x$.

Assume that $x + 1 = y$. Since $d(B_y, B_{y-1}) < d(B_x, B_{x-1})$, the same argument as above shows that $z \geq x$, which guarantees that $x = z$. □

The exact decomposition algorithm uses Proposition 10 to guide the search process. Starting by the two extreme subgraphs of the decomposition, $\emptyset$ and V, the algorithm maintains a sequence of locally-dense subgraphs. Recursively, for any two currently-adjacent subgraphs in the sequence, we use Proposition 10 to check whether the two subgraphs are consecutive or not in the decomposition. If they are

consecutive, the recurrence at that branch of the search is terminated. If they are not, a new subgraph between the two is discovered and it is added in the decomposition. The algorithm is named EXACTLD and it is illustrated as Algorithm 1.

With the next propositions we prove the correctness of the algorithm and we bound its running time.

PROPOSITION 11. *The algorithm* EXACTLD *initiated with input $(G, \emptyset, V)$ visits all locally-dense subgraphs of G.*

PROOF. Let $\{B_i\}$ be the set of locally-dense subgraphs. We will prove the proposition by showing that for $i < j$, the algorithm EXACTLD(G, B_i, B_j) visits all monotonic subgraphs that are between B_i and B_j. We will prove this by induction over $j - i$. The first step $j = i + 1$ is trivial. Assume that $j > i + 1$. Then Proposition 10 implies that $B_k = F(\alpha)$, where $i < k < j$. The inductive assumption now guarantees that EXACTLD(G, B_i, B_k) and EXACTLD(G, B_k, B_j) will visit all monotonic subgraphs between B_i and B_j. □

PROPOSITION 12. *The worst-case running time of algorithm* EXACTLD *is $\mathcal{O}(|V|^2|E|)$.*

PROOF. We will show that the algorithm EXACTLD, initiated with input $(G, \emptyset, V)$ makes $2k - 3$ calls to the function F, where k is the number of locally-dense subgraphs.

Let k_i be the number of calls of F when the input parameter $Y = B_i$. Out of these k_i calls one call will result in $F(\alpha) = X$. There are $k - 1$ such calls, since $Y = \emptyset$ is never tested. Each of the remaining calls will discover a new locally-dense subgraph. Since there are $k - 2$ new subgraphs to discover, it follows that $2k - 3$ calls to F are needed.

Since a call to F corresponds to a min-cut computation, which has running time $\mathcal{O}(|V||E|)$ [22], and since $k \in \mathcal{O}(|V|)$, the claimed running-time bound follows. □

4.2 Speeding up the exact algorithm

Our next step is to speed-up EXACTLD. This speed-up does not improve to the theoretical bound for the computational time but, in practice, it improves the performance of the algorithm dramatically.

The speed-up is based on the following observation. We know from Proposition 10 that EXACTLD(G, X, Y) visits only subgraphs Z with the property $X \subseteq Z \subseteq Y$. This gives us immediately the first speed-up: we can safely ignore any vertex outside Y, that is, EXACTLD$(G(Y), X, Y)$ will yield the same output.

Our second observation is that any subgraph Z visited by EXACTLD(G, X, Y) must contain vertices X. However, we cannot simply delete them because we need to take into account the edges between X and Z. To address this let us consider the following maximizer

$$F(\alpha; X) = \arg \max_{X \subseteq W \subseteq V} \{|E(W)| - \alpha|W|\}.$$

We can replace the original $F(\alpha)$ in Algorithm 1 with $F(\alpha; X)$. To compute $F(\alpha; X)$ we will use a straightforward extension of the Goldberg's algorithm [15] and transform this problem into a problem of finding a minimum cut.

In order to do this, given a graph $G = (V, E)$, let us define a weighted graph H that consists of vertices $V \setminus X$ and edges $E(V \setminus X)$ with weights of 1. Add two auxiliary vertices s and t into H and connect these vertices to every vertex in

$V \setminus X$. Given a vertex $y \in V \setminus X$, assign a weight of 2α to the edge (y, t) and a weight of

$$w(y) = \deg(y; V \setminus X) + 2\deg(y; X)$$

to the edge (s, y), where $\deg(y; U)$ stands for the number of neighbors of y in U. We claim that solving a minimum cut such that s and t are in different cuts will solve $F(\alpha; X)$. This cut can be obtained by constructing a maximum flow from s to t.

To prove this claim let $C \subsetneq V(H)$ be a subset of vertices containing s and not containing t. Let $Z = C \setminus \{s\}$ and also let $W = V \setminus (Z \cup X)$. There are three types of cross-edges from C to $V(H) \setminus C$: *(i)* edges from $x \in Z$ to t, *(ii)* edges from s to $x \in W$, and *(iii)* edges from $x \in Z$ to $y \in W$. The total cost of C is then

$$2|Z|\alpha + \sum_{y \in W} w(y) + |E_\times(Z, W)|.$$

We claim that the last two terms of the cost are equal to $2|E| - 2|E(X \cup Z)|$. To see this, consider an edge $e = (x, y)$ in $E \setminus E(X \cup Z)$. This implies that one of the end points, assume it is y, has to be in W. There are three different cases for x: *(i)* if $x \in W$, then e contributes 2 to the cost: 1 to $w(x)$ and 1 to $w(y)$, *(ii)* if $x \in X$, then e contributes 2 to $w(y)$, and *(iii)* if $x \in Z$, then e contributes 1 to $w(y)$ and 1 to the third term. Thus, we can write the cut as

$$2|Z|\alpha + 2|E| - 2|E(X \cup Z)|$$
$$= 2|E| - 2|X|\alpha - 2(|E(X \cup Z)| - \alpha|Z \cup X|).$$

The first two terms in the right-hand side are constant which implies that that finding the minimum cut is equivalent of maximizing $|E(X \cup Z)| - \alpha|Z \cup X|$. Consequently, if Z^* is the min-cut solution, then $F(\alpha) = X \cup Z^*$.

Note that the graph H does not have vertices included in X. By combining both speed-ups we are able to reduce the running time of EXACTLD(X, Y) by considering only the vertices that are in $Y \setminus X$.

4.3 Linear approximation algorithm

As we saw in the last section, the exact algorithm can be significantly accelerated, and indeed, our experimental evaluation shows that it is possible to run the exact algorithm for a graph of millions of vertices and edges within 2 minutes. Nevertheless, the worst-case complexity of the algorithm is cubic, and thus, it is not truly scalable for massive graphs.

Here we present a more lightweight algorithm for performing a locally-dense decomposition of a graph. The algorithm runs in linear time and offers a factor-2 approximation guarantee. As the exact algorithm builds on Goldberg's algorithm for the densest-subgraph problem, the linear-time algorithm builds on Charikar's approximation algorithm for the same problem [13]. As already explained in Section 2, Charikar's approximation algorithm iteratively removes the vertex with the lowest degree, until left with an empty graph, and returns the densest graph among all subgraphs considered during this process.

Our extension to this algorithm, called GREEDYLD, is illustrated in Algorithm 2, and it operates in two phases. The first phase is identical to the one in Charikar's algorithm: all vertices of the graph are iteratively removed, in increasing order of their degree in the current graph. In the second phase, the algorithm proceeds to discover approximate locally-dense subgraphs, in an iterative manner,

---

Algorithm 2: GREEDYLD(G)

> **input** : Graph $G = (V, E)$
> **output**: Collection $\mathcal{C}$ of approximate locally-dense subgraphs

1 **for** $i = |V|, \ldots, 1$ **do**
2 $\quad$ $w_i \leftarrow$ the vertex with the smallest degree;
3 $\quad$ delete w_i from V;
4 $\mathcal{C} \leftarrow \{\emptyset\}$;
5 $j \leftarrow 0$;
6 **while** $j < |V|$ **do**
7 $\quad$ $i \leftarrow \arg\max_{i > j} d(\{w_1, \ldots, w_i\}, \{w_1, \ldots, w_j\})$;
8 $\quad$ add $\{w_1, \ldots, w_j\}$ to $\mathcal{C}$;
9 $\quad$ $j \leftarrow i$;
10 **return** $\mathcal{C}$;

---

from B_1 to B_k. The first subgraph B_1 is the approximate densest subgraph, the same one returned by Charikar's algorithm. In the j-th step of the iteration, having discover subgraphs $B_1, \ldots, B_{j-1}$ the algorithm selects the subgraph B_j that maximizes the density $d(B_j, B_{j-1})$. To select B_j the algorithm considers subsets of vertices only in the degree-based order that was produced in the first phase.

Discovering $\mathcal{C}$ from the ordered vertices takes $\mathcal{O}(n^2)$ time, if done naively. However, it is possible to implement this step in $\mathcal{O}(n)$ time. In order to do this, sort vertices in the reverse visit order, and define in(v) to be the number of edges of v from the earlier neighbors. Then, we can we express the density as an average,

$$d(\{w_1, \ldots, w_i\}, \{w_1, \ldots, w_j\}) = \frac{1}{i - j} \sum_{k=j+1}^{i} \text{in}(v_k).$$

Consequently, we can see that recovering $\mathcal{C}$ is an instance of the following problem,

PROBLEM 2. *Given a sequence $y_1, \ldots, y_n$, compute the maximal interval*

$$m(j) = \arg\max_{j \le i \le n} \frac{1}{i - j + 1} \sum_{k=j}^{i} y_k,$$

for every $1 \le j \le n$.

Luckily, Calders et al. [11] demonstrated that we can use the classic PAVA algorithm [5] to solve this problem for *every* value of j in total $\mathcal{O}(n)$ time.

To quantify the approximation guarantee of GREEDYLD, note that the sequence of approximate locally-dense subgraphs produced by the algorithm are not necessarily aligned with the locally-dense subgraphs of the optimal decomposition. In other words, to assess the quality of the density of an approximate locally-dense subgraph B_j produced by GREEDYLD, there is no direct counterpart in the optimal decomposition to compare. To overcome this difficulty we develop a scheme of "vertex-wise" comparison, where for any $1 \le i \le n$, the density of a the smallest approximate locally-dense subgraph of size at least i is compared with the density of a the smallest optimal locally-dense subgraph of size at least i. This is defined below via the concept of *profile*.

DEFINITION 13. *Let $\mathcal{B} = (\emptyset = B_0 \subsetneq B_1 \subsetneq \cdots \subsetneq B_k = V)$ be a nested chain of subgraphs, the first subgraph being the*

empty graph and the last subgraph being the full graph. For an integer i, $1 \le i \le n$ define

$$j = \min \{x \mid |B_x| \ge i\}$$

to be the index of the smallest subgraph in $\mathcal{B}$ whose size is at least i. We define a profile function $p : \{1, \ldots, n\} \to \mathbb{R}$ *to be*

$$p(i; \mathcal{B}) = d(B_j, B_{j-1}).$$

Our approximation guarantee is now expressed as a guarantee of the profile function of the approximate decomposition with respect to the optimal decomposition.

PROPOSITION 14. *Let $\mathcal{B} = \{B_i\}$ be the set of optimal locally-dense subgraphs. Let $\mathcal{C} = \{C_i\}$ be the subgraphs obtained by* GREEDYLD. *Then*

$$p(i; \mathcal{C}) \ge p(i; \mathcal{B}) / 2.$$

PROOF. Sort the set of vertices V according to the reverse visiting order of GREEDYLD and let $\text{in}(v)$ be the number of edges of v from earlier neighbors.

Fix k to be an integer, $1 \le k \le n$ and let B_i be the smallest subgraph such that $|B_i| \ge k$. Let v_j be the last vertex occurring in B_i. We must have $\text{in}(v_j) \ge d(v_j, B_i)$, and $d(v_j, B_i) \ge d(B_i, B_{i-1})$ as otherwise we can remove v_j from B_i and improve the density due to Lemma 7. We have

$$p(k; \mathcal{B}) = d(B_i, B_{i-1}) \le \text{in}(v_j).$$

Let C_x be the smallest subgraph such that $|C_x| \ge k$. Let v_z be the vertex with the smallest index that is still in $C_x \setminus C_{x-1}$. Let g_y be the degree of v_y right before v_j is removed during GREEDYLD. Note that $\text{in}(v_j) \le g_y$, and we can easily show that $\sum_{y=z}^{j} g_y \le 2 \sum_{y=z}^{j} \text{in}(v_y)$. We now have

$$
\begin{aligned}
p(k; \mathcal{C}) &= d(C_x, C_{x-1}) \\
&\ge \frac{1}{j - z + 1} \sum_{y=z}^{j} \text{in}(v_y) \\
&\ge \frac{1}{2(j - z + 1)} \sum_{y=z}^{j} g_y \\
&\ge \frac{\text{in}(v_j)}{2},
\end{aligned}
$$

where the optimality of C_x implies the first inequality. $\square$

We should point out that $p(1, \mathcal{B})$ is equal to the density of the densest subgraph, while $p(1, \mathcal{C})$ is equal to the density of the subgraph discovered by the Charikar's algorithm. Consequently, Proposition 14 provides automatically the 2-approximation guarantee of the Charikar's algorithm.

We should also point out that $p(i, \mathcal{C})$ can be larger than $p(i, \mathcal{B})$ but for the first index, say j, for which $p(j, \mathcal{C}) \ne p(j, \mathcal{B})$, Proposition 5 guarantees that $p(j, \mathcal{C}) < p(j, \mathcal{B})$.

5. LOCALLY-DENSE SUBGRAPHS AND CORE DECOMPOSITION

Here we study the connection of graph cores, obtained with the well-known k-core decomposition algorithms, with local-density, studied in this paper. We are able to show that from the theory point-of-view, graph cores are as good

---

Algorithm 3: CORE(G)

input : Graph $G = (V, E)$
output: Collection $\mathcal{C}$ of k-cores

1   $\mathcal{C} \leftarrow \{V\}$;
2   $k \leftarrow \min_w \deg(w)$;
3   **for** $i = |V|, \ldots, 1$ **do**
4     $w_i \leftarrow$ the vertex with the smallest degree;
5   **if** $\deg(w) > k$ **then**
6   add V to $\mathcal{C}$;
7       $k \leftarrow \deg(w)$;
8   delete w_i from V;
9   **return** $\mathcal{C}$;

---

approximation to the optimal locally-dense graph decomposition as the subgraphs obtained by the GREEDYLD algorithm. In particular we show a similar result to Proposition 14, namely, a factor-2 approximation on the profile function of the core decomposition.

However, as we will see in our empirical evaluation, the behavior of the two algorithms, GREEDYLD and k-core decomposition are different in practice, with GREEDYLD giving in general more dense subgraphs and closer to the ones given by exact locally-dense decomposition.

Before stating and proving the result regarding k-cores, recall that a set of vertices $X \subseteq V$ is a k-core if every vertex in the subgraph induced by X has degree at least k, and X is maximal with respect to this property. A linear-time algorithm for obtaining all k-cores is illustrated in Algorithm 3.

It is a well-known fact that the set of all k-cores of a graph forms a nested chain of subgraphs, in the same way that locally-dense subgraphs do.

PROPOSITION 15. *Let $\{C_i\}$ be the set of all k-cores of a graph $G = (V, E)$. Then $\{C_i\}$ forms a nested chain,*

$$\emptyset = C_0 \subsetneq C_1 \subsetneq \cdots \subsetneq C_l = V.$$

Similar to Proposition 14, k-cores provide a factor-2 approximation with respect to the optimal locally-dense subgraphs. The proof is in fact quite similar to that of Proposition 14.

PROPOSITION 16. *Let $\mathcal{B} = \{B_i\}$ be the set of optimal locally-dense subgraphs. Let $\mathcal{C} = \{C_i\}$ be the set of k-cores. Then*

$$p(i; \mathcal{C}) \ge p(i; \mathcal{B}) / 2.$$

PROOF. Sort V according to the reverse visiting order of CORE and let $\text{in}(v)$ be the number of edges of v from earlier neighbors.

Fix k to be an integer, $1 \le k \le n$ and let B_i be the smallest subgraph such that $|B_i| \ge k$. Let v_j be the last vertex occurring in B_i. We must have $\text{in}(v_j) \ge d(v_j, B_i)$, and $d(v_j, B_i) \ge d(B_i, B_{i-1})$ as otherwise we can remove v_j from B_i and improve the density due to Lemma 7. We have

$$p(k; \mathcal{B}) = d(B_i, B_{i-1}) \le \text{in}(v_j).$$

Let C_x be the smallest smallest core such that $|C_x| \ge k$. Let v_s be the vertex with the smallest index that is still in $C_x \setminus C_{x-1}$. Let v_l be the vertex with the largest index that is still in $C_x \setminus C_{x-1}$.

If $j > l$, then $\mathrm{in}(v_j) < \mathrm{in}(v_l)$ otherwise C_x is not a core. If $j < l$, then $\mathrm{in}(v_j) \leq \mathrm{in}(v_l)$, otherwise C_x would is not the smallest core as $j \geq k$. Hence, $\mathrm{in}(v_j) \leq \mathrm{in}(v_l)$.

Let g_y be the degree of v_y right before v_l is removed during CORE. We now have

$$
\begin{aligned}
p(k; \mathcal{C}) &= d(C_x, C_{x-1}) \\
&= \frac{1}{l - s + 1} \sum_{y=s}^{l} \mathrm{in}(v_y) \\
&\geq \frac{1}{2(l - s + 1)} \sum_{y=s}^{l} g_y \\
&\geq \frac{\mathrm{in}(v_l)}{2} \\
&\geq \frac{\mathrm{in}(v_j)}{2},
\end{aligned}
$$

which proves the proposition. $\square$

6. RELATED WORK

Our paper is related to previous work on discovering dense subgraphs, clique-like structures, and hierarchical communities. We review some representative work on these topics.

Clique relaxations. The densest possible subgraph is a clique. Unfortunately finding large cliques is computationally intractable [17]. Additionally, the notion of clique does not provide a robust definition for practical situations, as a few absent edges may completely destroy the clique. To address these issues, researchers have come up with relaxed clique definitions. A relaxation, k-plex was suggested by Seidman and Foster [24]. In a k-plex a vertex can have at most $k - 1$ absent edges. Unfortunately, discovering maximal k-plexes is also an **NP**-hard problem [7]. An alternative relaxation for a clique is the one of an n-clique, a maximal subgraph where each vertex is connected to every vertex with a path, possibly outside of the subgraph, of at most n-length [10]. So, according to this definition a clique is an 1-clique. As maximal n-cliques may produce sparse graphs, the concept of n-clans was also proposed by limiting the diameter of the subgraph to be at most n [21]. Since 1-clan corresponds to a maximal clique, discovering n-clans is a computationally intractable problem.

Quasi-cliques. For the definition of graph density we have chosen to work with $d(X)$, the average degree of the subgraph induced by X. While this is a popular density definition, there are other alternatives. One such alternative would be to divide the number of edges present in the subgraph with the total number of possible edges, that is, divide by $\binom{n}{2}$. This would give us a normalized density score that is between 0 and 1. Subgraphs that maximize this density definition are called *quasi-cliques*, and algorithms for enumerating all quasi-cliques, which can be exponentially many, have been proposed by Abello et al. [1] and Uno [29]. However, the definition of quasi-cliques is problematic. Note that a single edge already provides maximal density. Consequently additional objectives are needed. One natural objective is to maximize the size of a graph with density of 1, however, this makes the problem equivalent to finding a maximal clique which, as mentioned above, is a computationally-intractable problem [17].

Alternative definitions for density. Other definitions of graph density have been proposed. Recently, Tsourakakis

proposed to measure density by counting triangles, instead of counting edges [26]. Interestingly enough, it is possible to find an approximate densest subgraph under this definition. An interesting future direction for our work is to study if the decomposition proposed in this paper can be extended for the triangle-density definition. Density definitions of the form $g(|E|) - \alpha h(|V|)$, where g and h are some increasing functions were studied by Tsourakakis et al. [27], with specific focus on $h(x) = \binom{x}{2}$. It not known whether the densest-subgraph problem according to this definition is polynomially-time solvable or **NP**-hard. Finally, a variant for $d(X)$ adopted for directed graph, along with polynomial-time discovery algorithm, was suggested by Khuller and Saha [18]. Such a definition could serve for defining decompositions of directed graphs, which is also left for future work.

Hierarchical communities. Discovering hierarchy of k nested communities with as homogeneous shells as possible with the constraint that inner communities are denser was studied by the authors of this paper [25]. Here, $|E|/\binom{|V|}{2}$ was used as density definition, and a heuristic algorithm was proposed. Unfortunately, no exact polynomial-time algorithm is known for this problem. As a potential future work it would be interesting to see whether the ideas presented in this paper can be merged with the idea of discovering k-homogeneous communities.

7. EXPERIMENTAL EVALUATION

We will now present our experimental evaluation. We test the two proposed algorithms, EXACTLD and GREEDYLD, for decomposing a graph into locally-dense subgraphs, and we contrast the resulting decompositions against k-cores, obtained with the CORE algorithm. We compare the three algorithms in terms of running time, decomposition size (number of subgraphs they provide), and relative density of the subgraphs they return. We also use the Kendall-τ to measure how similar are the decompositions in terms of the order they induce on the graph vertices.

7.1 Experimental setup

We perform our evaluation on 11 graphs of different sizes and densities. A short description of the graphs is given below, and their basic characteristics can be found in Table 1.

- `dolphins`: an undirected social network of frequent associations between dolphins in a community living off Doubtful Sound in New Zealand.
- `karate`: the social network of friendships between members of a karate club at a US university in the 1970.
- `lesmis`: co-appearance of characters in Les Miserables novel by Victor Hugo.
- `astro`: a co-authorship network among arXiv Astro Physics publications.
- `enron`: an e-mail communication network by Enron employees.
- `fb1912`: an ego-network obtained from Facebook.
- `hepph`: a co-authorship network among arXiv High Energy Physics publications.
- `dblp`: a co-authorship network among computer science researchers.
- `gowalla`: a friendship network of `gowalla.com`.

Table 1: Basic characteristics of the datasets and the running times of the algorithms. E stands for ExactLD, G for GreedyLD, and C for Core.

Name	$|V|$	$|E|$	running time C	G	E
dolphins	62	159	1ms	1ms	2ms
karate	34	78	1ms	1ms	2ms
lesmis	77	254	2ms	2ms	3ms
astro	18 772	396 160	0.4s	0.4s	2s
enron	36 692	183 831	0.3s	0.3s	2s
fb1912	747	30 025	44ms	44ms	0.2s
hepph	12 008	237 010	0.2s	0.2s	0.9s
dblp	317 080	1 049 866	2s	2s	14s
gowalla	196 591	950 327	2s	2s	9s
roadnet	1 965 206	5 533 214	7s	8s	1m6s
skitter	1 696 415	11 095 298	21s	21s	1m46s

- `roadnet`: a road network of California, where vertices represent intersections and edges represent road segments.
- `skitter`: an internet topology graph, obtained from traceroutes run daily in 2005.

The first three datasets are obtained from UCIrvine Network Data Repository,[3] and the remaining datasets are obtained from Stanford SNAP Repository.[4]

We apply CORE, GREEDYLD, and EXACTLD to every dataset. We use a computer equipped with 3GHz Intel Core i7 and 8GB of RAM.[5]

7.2 Results

We begin by reporting the running times of the three algorithms for all of our datasets. They are shown in Table 1. As expected, the linear-time algorithms CORE and GREEDYLD are both very fast; the largest graph with 11 million edges and 1.7 million vertices is processed in 21 seconds. However, we are also able to run the exact decomposition for all the graphs in reasonable time, despite its running-time complexity of $\mathcal{O}(|V|^2|E|)$. It takes less than 2 minutes for EXACTLD to process the largest graph. There are three reasons that contribute to achieving this performance. First, we need to compute the minimum cut only $\mathcal{O}(k)$ times, where k is the number of locally-dense graphs. In practice, k is much smaller than the number of vertices. Second, computing minimum cut in practice is faster than the theoretical $\mathcal{O}(|V||E|)$ bound. Third, as described in Section 4, most of the minimum cuts are computed using subgraphs. While in theory these subgraphs can be as large as the original graph, in practice these subgraphs are significantly smaller.

Next, we compare how well CORE and GREEDYLD approximate the exact locally-dense decomposition. In order to do that we compute the ratio

$$r(\mathcal{C}, \mathcal{B}) = \min_i \frac{p(i; \mathcal{C})}{p(i; \mathcal{B})}, \qquad (1)$$

Table 2: Smallest ratio of the profile function, and the profile function of the exact solution as defined in Equation (1), and the ratio of the most inner discovered subgraph versus the actual densest subgraph.

Name	$r(\mathcal{C}, \mathcal{B})$ CORE	GREEDYLD	$d(C_1)/d(B_1)$ CORE	GREEDYLD
dolphins	0.94	0.83	0.98	0.98
karate	0.95	0.99	0.95	0.99
lesmis	0.86	0.87	0.96	1.00
astro	0.85	0.85	0.87	0.92
enron	0.83	0.82	0.94	1.00
fb1912	0.69	0.74	0.91	1.00
hepph	0.74	0.75	1.00	1.00
dblp	0.80	0.86	1.00	1.00
gowalla	0.89	0.92	0.87	1.00
roadnet	0.81	0.87	0.84	0.87
skitter	0.73	0.84	0.84	1.00

Table 3: Sizes of the discovered decompositions and Kendall-τ statistics between the decompositions. E stands for ExactLD, G for GreedyLD, and C for Core.

Name	C	G	E	C-vs-E	G-vs-E	C-vs-G
dolphins	4	6	7	0.76	0.77	0.99
karate	4	3	4	0.80	0.95	0.78
lesmis	8	8	9	0.94	0.99	0.95
astro	52	83	435	0.93	0.93	0.99
enron	43	162	357	0.92	0.92	0.99
fb1912	87	55	75	0.95	0.98	0.97
hepph	64	63	283	0.93	0.93	0.98
dblp	47	97	1087	0.88	0.89	0.97
gowalla	51	161	899	0.97	0.96	0.98
roadnet	3	43	2710	0.57	0.80	0.68
skitter	111	266	3501	0.98	0.97	0.99

where $\mathcal{B}$ is the locally-dense decomposition and $\mathcal{C}$ is obtained by either from GREEDYLD or CORE. These ratios are shown in Table 2. We also compare $p(1; \mathcal{C})/p(1; \mathcal{B})$, that is, the ratio of density for the inner most subgraph in $\mathcal{C}$ against the density of $\mathcal{B}_1$, the densest subgraph. Propositions 14 and 15 guarantee that there ratios are at least 1/2. In practice, the ratios are larger, typically over 0.8. In most cases, but not always, GREEDYLD obtains better ratios than CORE. When comparing the ratio for the inner most subgraph, GREEDYLD, by design, will always be better or equal than CORE. We see that only in three datasets CORE is able to find the same subgraph as GREEDYLD.

Let us now compare the different solutions found by the three algorithms. In Table 3 we report the sizes of discovered communities and their Kendall-τ statistics, which compares the ordering of the vertices induced by the decompositions. In particular, the Kendall-τ statistic is computed by assigning each vertex an index based on which subgraph the vertex belongs. To handle ties, we use the b-version of Kendall-τ, as given by Agresti [2]. If the statistic is 1, the decompositions are equal.
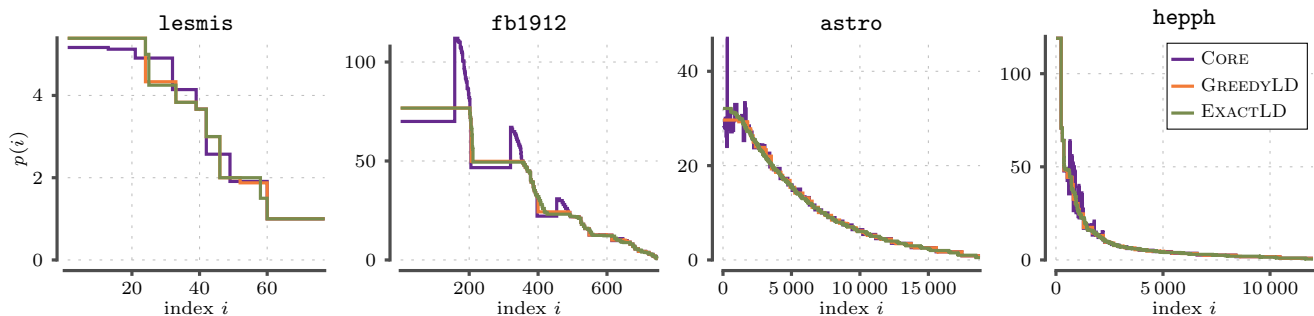
Figure 2: Profile functions for `lesmis`, `fb1912`, `astro`, and `hepph`.

Our first observation is that typically the locally-dense decomposition algorithms return more subgraphs than the k-core decomposition. As an extreme example, `roadnet` contains only 3 k-cores while GREEDYLD finds 43 subgraphs and EXACTLD finds 2710. This can be explained by the fact that the vertices in the graph have low degrees, which results in a very coarse k-core decomposition. On the other hand, EXACTLD and GREEDYLD exploit density to discover more fine-grained decompositions. This result is similar to what we presented in the Example 1 in the introduction.

The Kendall-τ statistics are typically close to 1, especially for large datasets suggesting that all 3 methods result in similar decompositions. The statistic between CORE and GREEDYLD is typically larger than to the exact solution. This is expected since CORE and GREEDYLD use the exact same order for vertices—the only difference between these two methods is how they partition the vertex order. In addition, decompositions produced by GREEDYLD are closer to the exact solution than the decompositions produced by CORE, which is also a natural result.

Let us now compare the solutions in terms of profile functions as defined in Definition 13. We illustrate several prototypical examples of such profile functions in Figure 2. From the figure we see that GREEDYLD produces similar profiles as the exact locally-dense decomposition. We also see that CORE does not respect the local density constraint. In `fb1912`, `astro`, and `hepph` there exist k-shells that are denser than their inner shells, that is, joining these shells would increase the density of the inner shell. GREEDYLD does not have this problem since by definition it will have a monotonically decreasing profile.

Finally, in Table 4 we present the decompositions obtained by the three algorithms for the `lesmis` graph. We see that GREEDYLD obtains very similar result to the exact solution, the only difference is the second subgraph and the third subgraph is merged and the 3rd last subgraph lends vertices to the 2nd last subgraph. While GREEDYLD has the same first subgraph as the exact solution, which is the densest subgraph, CORE breaks this subgraph into 3 subgraphs. Interestingly enough, the main character of the book, Jean Valjean, is not placed into the first shell by CORE.

8. CONCLUSIONS

Inspired by k-core analysis and density-based graph mining, we propose density-friendly graph decomposition, a new tool for analyzing graphs. Like k-core decomposition, our approach decomposes a given graph into a nested sequence of

Table 4: Decompositions of the `lesmis` dataset. The numbers represent the greedy peeling and the k-core decomposition, respectively. The names are ordered according to locally dense decomposition, the groups are marked with chanding color.

Valjean	1 2	Fameuil	2 3	Pontmercy	6 7
MmeThenardier	1 3	Blacheville	2 3	MotherInnocent	7 7
Thenardier	1 2	Favourite	2 3	Magnon	7 7
Javert	1 2	Dahlia	2 3	MmePontmercy	7 7
Eponine	1 2	Zephine	2 3	BaronessT	6 7
Gavroche	1 1	Fantine	2 3	Child1	7 7
Marius	1 1	Bamatabois	3 4	Child2	7 7
Mabeuf	1 1	Judge	3 4	Napoleon	8 8
Enjolras	1 1	Champmathieu	3 4	CountessDeLo	8 8
Combeferre	1 1	Brevet	3 4	Geborand	8 8
Prouvaire	1 1	Chenildieu	3 4	Champtercier	8 8
Feuilly	1 1	Cochepaille	3 4	Cravatte	8 8
Courfeyrac	1 1	Gillenormand	4 5	Count	8 8
Bahorel	1 1	MlleGillenormand	4 5	OldMan	8 8
Bossuet	1 1	LtGillenormand	4 5	Labarre	8 8
Joly	1 1	Simplice	5 6	MmeDeR	8 8
Grantaire	1 1	Anzelma	5 6	Isabeau	8 8
Gueulemer	1 2	Woman2	5 6	Gervais	8 8
Babet	1 2	Toussaint	5 6	Scaufflaire	8 8
Claquesous	1 2	Myriel	6 6	Boulatruelle	8 8
Montparnasse	1 2	MlleBaptistine	6 6	Gribier	8 8
Brujon	1 3	MmeMagloire	6 6	Jondrette	8 8
MmeHucheloup	1 3	Marguerite	7 7	MmeBurgon	8 8
Cosette	2 4	Fauchelevent	6 7	MlleVaubois	8 8
Tholomyes	2 3	Perpetue	7 7	MotherPlutarch	8 8
Listolier	2 3	Woman1	7 7		

subgraphs These subgraphs have the property that the inner subgraphs are always denser than the outer ones; additionally the most inner subgraph is the densest one—properties that the k-cores do not satisfy.

We provide two efficient algorithms to discover such a decomposition. The first algorithm is based on minimum cut and it extends the exact algorithm of Goldberg for the densest-subgraph problem. The second algorithm extends a linear-time algorithm by Charikar for approximating the same problem. The second algorithm runs in linear time, and thus, in addition to finding subgraphs that respect better the density structure of the graph, it is as efficient as the k-core decomposition algorithm.

In addition to offering a new alternative for decomposing a graph into dense subgraphs, we significantly extend the analysis, the understanding, and the applicability of previous well-known graph algorithms: Goldberg's exact algorithm and Charikar's approximation algorithm for finding the densest subgraph, as well as the k-core decomposition algorithm itself.

9. REFERENCES

[1] J. Abello, M. Resende, and S. Sudarsky. Massive quasi-clique detection. In *LATIN 2002: Theoretical Informatics*, pages 598–612, 2002.

[2] A. Agresti. *Analysis of Ordinal Categorical Data*. John Wiley & Sons, 2nd edition, 2010.

[3] J. I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. k-core decomposition: a tool for the visualization of large scale networks. *CoRR*, abs/cs/0504107, 2005.

[4] Y. Asahiro, K. Iwama, H. Tamaki, and T. Tokuyama. Greedily finding a dense subgraph. *SWAT*, pages 136–148, 1996.

[5] M. Ayer, H. Brunk, G. Ewing, and W. Reid. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, 26(4):641–647, 1955.

[6] G. Bader and C. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(1), 2003.

[7] B. Balasundaram, S. Butenko, and I. V. Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011.

[8] B. Bollobás. The evolution of random graphs. *Transactions of the American Mathematical Society*, 286(1):257–274, 1984.

[9] F. Bonchi, F. Gullo, A. Kaltenbrunner, and Y. Volkovich. Core decomposition of uncertain graphs. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1316–1325, 2014.

[10] C. Bron and J. Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[11] T. Calders, N. Dexters, J. J. M. Gillis, and B. Goethals. Mining frequent itemsets in a stream. *Information Systems*, 39:233–255, 2014.

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

[13] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. *APPROX*, 2000.

[14] W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.

[15] A. V. Goldberg. Finding a maximum density subgraph. *University of California Berkeley Technical report*, 1984.

[16] P. Hagmann, L. Cammoun, X. Gigandet, R. Meuli, C. J. Honey, V. J. Wedeen, and O. Sporns. Mapping the structural core of human cerebral cortex. *PLoS, Biology*, 6(7):888–893, 2008.

[17] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, pages 627–636, 1996.

[18] S. Khuller and B. Saha. On finding dense subgraphs. In *Automata, Languages and Programming*, volume 5555, pages 597–608, 2009.

[19] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *Nature physics*, 6(11):888–893, 2010.

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

[21] R. J. Mokken. Cliques, clubs and clans. *Quality and Quantity*, 13(2):161–173, 1979.

[22] J. Orlin. Max flows in $\mathcal{O}(nm)$ time, or better. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 765–774, 2013.

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

[24] S. B. Seidman and B. L. Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 2010.

[25] N. Tatti and A. Gionis. Discovering nested communities. In *Machine Learning and Knowledge Discovery in Databases—European Conference, ECML PKDD 2013*, pages 32–47, 2013.

[26] C. E. Tsourakakis. A novel approach to finding near-cliques: The triangle-densest subgraph problem. *CoRR*, 2014.

[27] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013*, pages 104–112, 2013.

[28] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences*, 109(16):5962–5966, 2012.

[29] T. Uno. An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica*, 56(1):3–16, 2010.