Density-friendly Graph Decomposition

NIKOLAJ TATTI, HIIT, University of Helsinki, Aalto University

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 $O(|V|^2|E|)$ 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.

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 [26], bioinformatics [6], analysis of connection matrices

of the human brain [19], graph visualization [3], as well as influence analysis [22, 33] 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, 17], 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 1.1. 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.

Fig. 1. Toy graphs used in Example 1.1.

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 [17]. 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 [13] for approximating dense subgraphs. 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 approximation 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.

It is possible that the decomposition results a significant amount of subgraphs. In such a case it may be useful to constraint the number of the subgraphs. We approach this problem by defining an optimization criterion for a segmentation of k nested subgraphs. The objective function will be based on a statistical model. We will show that to optimize this particular objective, we need to (*i*) find locally-dense subgraphs, and (*ii*) reduce the number with a dynamic program. We also show that if we replace the first step with the greedy algorithm, then the resulting algorithm yields a factor-2 approximation guarantee.

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 introduce the constrained version of the problem in Section 6. We present the related work in Section 7 and present the experiments in Section 8. Finally, we conclude the paper with discussion in Section 9.

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 $E(X) = \{(x, y) \in E \mid x, y \in X\}$, that is, 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 [17]. As the fastest algorithm to solve the minimum-cut problem runs in $O(mn)$ 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.

---

[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.

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

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

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,

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

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

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

That is, these are the extra edges, on average, that we bring to Y if we expand it by appending X.

Now that we have defined a special case when X and Y are disjoint, we can now consider a more general case, that is, when X and Y are overlapping. Here 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

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

k-cores. We briefly review the basic background regarding k-cores. The concept was introduced by Seidman [26].

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

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

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 [23]. 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 3.1. A set of vertices W is *locally dense* if there are no $X \subseteq W$ and Y satisfying $Y \cap W = \emptyset$ such that

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

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 3.2. *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

$$d(X, U \setminus X) = d(X, U \cap W) \leq d(Y, U \cap W) \leq d(Y, U),$$

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. □

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 3.3. *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 3.3, defines the *locally-dense decomposition* of G.

Example 3.4. The locally-dense composition of G_1 given in Figure 1 is $\emptyset \subsetneq C_2 \subsetneq C_3 = V$, This is the k-core decomposition without C_1. The locally-dense composition of G_2 given in Figure 1 is $\emptyset \subsetneq B_1 \subsetneq V$. Note that both C_2 and B_1 are the densest subgraphs in their respective graphs.

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 3.5. *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},*

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

To prove the proposition we will use the following three lemmas.

LEMMA 3.6. *Let $X \subseteq Y$ be two sets of vertices with $Y \neq \emptyset$. Assume a third non-empty set Z with $Z \cap Y = \emptyset$. Then one of the following three cases follows:*

- *$d(Z, Y) > d(Y \cup Z, X) > d(Y, X)$, or*
- *$d(Z, Y) < d(Y \cup Z, X) < d(Y, X)$, or*
- *$d(Z, Y) = d(Y \cup Z, X) = d(Y, X)$.*

PROOF. Write $\alpha = \frac{|Y|}{|Y|+|Z|}$. We can rewrite $d(Y \cup Z, X)$ as

$$d(Y \cup Z, X) = \frac{|E_\Delta(Y \cup Z, X)|}{|Y| + |Z|} = \frac{|E_\Delta(Y, X)| + |E_\Delta(Z, Y)|}{|Y| + |Z|} = \alpha d(Y, X) + (1 - \alpha)d(Z, Y) \quad .$$

This shows that either $d(Z, Y) \geq d(Y \cup Z, X) \geq d(Y, X)$ or $d(Z, Y) \leq d(Y \cup Z, X) \leq d(Y, X)$. Since $0 < \alpha < 1$ it follows that $d(Z, Y) = d(Y \cup Z, X)$ if and only if $d(Y \cup Z, X) = d(Y, X)$. The three cases follows.   □

Let C_i be the sequence defined as $C_i = \arg\max_{W \supsetneq C_{i-1}} d(W)$, in case of a tie, select a larger graph, and $C_0 = \emptyset$.

LEMMA 3.7. $d(C_j, C_{j-1}) > d(C_i, C_{i-1})$ for $j < i$.

PROOF. We only need to show that the lemma holds $j = i - 1$. Assume otherwise: $d(C_i, C_{i-1}) \geq d(C_{i-1}, C_{i-2})$.

Write $Z = C_i \setminus C_{i-1}$, $Y = C_{i-1}$, and $X = C_{i-2}$. Since $d(Z, Y) = d(C_i, C_{i-1})$, Lemma 3.6 implies that

$$d(C_i, C_{i-2}) = d(Y \cup Z, X) \geq d(Y, X) = d(C_{i-1}, C_{i-2}),$$

violating the optimality of C_{i-1}.   □

LEMMA 3.8. If $Z \subseteq C_j \setminus C_{j-1}$ and $Z \neq \emptyset$, then $d(Z, C_j \setminus Z) \geq d(C_j, C_{j-1})$.

PROOF. Assume otherwise: $d(Z, C_j \setminus Z) < d(C_j, C_{j-1})$. Write $X = C_{j-1}$, $Y = C_j \setminus Z$. Lemma 3.6 implies that

$$d(C_j, C_{j-1}) = d(Z \cup Y, X) < d(Y, X) = d(C_j \setminus Z, C_{j-1}),$$

violating the optimality of C_j.   □

PROOF OF PROPOSITION 3.5. We need to show that $C_i = B_i$. Fix i and assume inductively that $C_j = B_j$ for all $j < i$.

We will first show that C_i is locally dense: we argue that there are no sets X and Y with $X \subseteq C_i$ and $Y \cap C_i = \emptyset$ that can serve as certificates for C_i being non locally-dense.

Fix any $X \subseteq C_i$. Define $X_j = X \cap (C_j \setminus C_{j-1})$ and $U_j = (C_i \setminus X) \cup C_{j-1}$ for $j \leq i$.

We claim that $C_j \subseteq U_j \cup X_j$. Let $x \in C_j$. If $x \in C_{j-1}$, then $x \in U_j$. Assume that $x \in C_j \setminus C_{j-1}$. If $x \in X$, then $x \in X_j$. If $x \notin X$, then $x \in C_j \setminus X \subseteq C_i \setminus X \subseteq U_j$. Thus, $C_j \subseteq U_j \cup X_j$, which in turns implies that $C_j \setminus X_j \subseteq U_j \setminus X_j$.

This leads to

$$\begin{aligned}
d(X_j, U_j \setminus X_j) &\geq d(X_j, C_j \setminus X_j) & (C_j \setminus X_j \subseteq U_j \setminus X_j) \\
&\geq d(C_j, C_{j-1}) & \text{(Lemma 3.8)} \\
&> d(C_i, C_{i-1}) \quad . & \text{(Lemma 3.7)}
\end{aligned}$$

This inequality leads to

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

Consider also any set Y with $Y \cap C_i = \emptyset$. Due to the optimality of C_i and Lemma 3.6 we must have $d(Y, C_i) < d(C_i, C_{i-1})$.

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

Now, we can safely assume $C_i = B_j$ for some j. We need to show that $j = i$. By induction we know that $C_{i-1} = B_{i-1}$. This guarantees that $j \geq i$. Assume $j > i$. Since C_i is maximal, we have $d(B_j \setminus B_i, B_i) < d(B_i, B_{i-1})$.

Since B_i is locally-dense, we have $d(B_j \setminus B_i, B_i) < d(B_i, B_{i-1})$. Lemma 3.6 now implies that

$$d(B_j, B_{i-1}) < d(B_i, B_{i-1})$$

which contradicts the optimality of $C_i = B_j$. Thus $i = j$. □

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

COROLLARY 3.9. *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*

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

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 $O(n^2 m)$, 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 [17], 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 [17] 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 f decreases as α increases, and as α exceeds a certain value, f 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 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 $O(nm)$ time, in

---

[2]This observation is an instance of *fractional programming* [16].

Algorithm 1: ExactLD(G, X, Y)

---

   input : Graph $G = (V, E)$
               locally-dense subgraphs X and Y with $X \subsetneq Y$

1   $\alpha \leftarrow d(Y, X) + n^{-2}$;
2   $Z \leftarrow F(\alpha)$;
3   **if** $Z \neq X$ **then**
4   **output** Z;
5   ExactLD(G, X, Z);
6   ExactLD(G, Z, Y);

---

a recent breakthrough by Orlin [25]. 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 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 [17]. We show that Goldberg's construction has the following, rather remarkable, property: there is a sequence of values $\alpha^* = \alpha_1 > \cdots > \alpha_k$, 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 4.1. *Let $\{B_i\}$ be the set of locally-dense subgraphs. Then*

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

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 such that $d(B_{i+1}, B_i) < \alpha \leq d(B_i, B_{i-1})$. Let $B_j = F(\alpha)$. If $j > i$, then, due to Corollary 3.3, $d(B_j, B_{j-1}) \leq d(B_{i+1}, B_i) < \alpha$ which we can rephrase as

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

If we delete $B_j \setminus B_{j-1}$ from U, then we improve the quality exactly by $-c$, that is, we obtain a better solution which violates the optimality of U. If $j < i$, then Corollary 3.3 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 $B_x \subseteq B_y$ of the decomposition (corresponding to values $\alpha_x > \alpha_y$), 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 4.2. *Let* $\{B_i\}$ *be the set of locally-dense subgraphs. Let* $B_x \subsetneq B_y$ *be two subgraphs. Set* $\alpha = d(B_y, B_x) + n^{-2}$ *and let* $B_z = F(\alpha)$. *If* $x + 1 < y$, *then* $x < z < y$. *If* $x + 1 = y$, *then* $z = x$.

LEMMA 4.3. $d(B_k, B_i) \geq d(B_\ell, B_j)$, *for* $i \leq j < k \leq \ell$. *The equality holds if and only if* $i = k$ *and* $j = \ell$.

PROOF. Corollary 3.3 states that $d(B_o, B_{o-1})$ is monotonically strictly decreasing as a function of o. Lemma 3.6, applied recusively, states that

$$d(B_k, B_i) \geq d(B_k, B_j) \geq d(B_\ell, B_j) \quad .$$

The inequality is strict if and only if $i \neq k$ or $j \neq \ell$. □

PROOF OF PROPOSITION 4.2. Lemma 4.3 states that $d(B_y, B_{y-1}) \leq d(B_y, B_x) < \alpha$. Proposition 4.1 now implies that $z < y$.

Assume that $x + 1 < y$. Lemma 4.3 implies that $d(B_y, B_x) < d(B_{x+1}, B_x)$. Write

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

Let us now bound the difference between the densities as

$$d(B_{x+1}, B_x) - d(B_y, B_x) = \frac{a}{b} - \frac{c}{d} = \frac{ad - bc}{bd} \geq \frac{1}{bd} > \frac{1}{n^2} = \alpha - d(B_y, B_x) \quad .$$

This implies that $\alpha \leq d(B_{x+1}, B_x)$. Proposition 4.1 now implies that $z \geq x + 1 > x$.

Assume that $x + 1 = y$. Lemma 4.3 implies that $d(B_y, B_{y-1}) < d(B_x, B_{x-1})$, and the same argument as above shows that $\alpha \leq d(B, B_{x-1})$ and, consequently, $z \geq x$. This guarantees that $x = z$. □

The exact decomposition algorithm uses Proposition 4.2 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 4.2 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 4.4. *The algorithm* EXACTLD *initiated with input* $(G, \emptyset, V)$ *visits all non-trivial 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 4.2 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 4.5. *The worst-case running time of algorithm* EXACTLD *is* $O(n^2 m)$.

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 $O(nm)$ [25], and since $k \in O(n)$, 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 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 4.2 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

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

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 [17] 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

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

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 at least 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

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

Algorithm 2: GREEDYLD(G)

---

input : Graph $G = (V, E)$
output: Collection C of approximate locally-dense subgraphs

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

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, 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 C from the ordered vertices takes $O(n^2)$ time, if done naively. However, it is possible to implement this step in $O(n)$ 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,

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

Consequently, we can see that recovering C is an instance of the following problem,

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

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

Luckily, Calders et al. [11] demonstrated that we can use the classic PAVA algorithm by Ayer et al. [5] to solve this problem for *every* value of j in total $O(n)$ 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 the smallest approximate locally-dense subgraph of size at least i is compared with the density of the smallest optimal locally-dense subgraph of size at least i. This is defined below via the concept of *profile*.

Definition 4.6. 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

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

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

PROPOSITION 4.7. *Let* $\mathcal{B} = \{B_i\}$ *be the set of locally-dense subgraphs. Let* $C = \{C_i\}$ *be the subgraphs obtained by* GREEDYLD. *Then*

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

First, we need the following lemma.

LEMMA 4.8. $d(v, B_i \setminus \{v\}) \ge d(B_i, B_{i-1})$, *for* $v \in B_i \setminus B_{i-1}$,

PROOF. Assume otherwise. Lemma 3.6 now states that $d(B_i \setminus \{v\}, B_{i-1}) < d(B_i, B_{i-1})$, which violates the optimality of B_i as indicated by Proposition 3.5. □

PROOF OF PROPOSITION 4.7. 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 $\text{in}(v_j) \ge d(v_j, B_i \setminus \{v_j\})$, and, due to Lemma 4.8, $d(v_j, B_i \setminus \{v_j\}) \ge d(B_i, B_{i-1})$. In summary, we have

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

Algorithm 3: CORE(G)

---

input : Graph $G = (V, E)$
output: Collection C of k-cores
1 $C \leftarrow \{V\}$;
2 $k \leftarrow \min_w \deg(w)$;
3 **for** $i = n, \ldots, 1$ **do**
4 $\quad$ $w_i \leftarrow$ the vertex with the smallest degree;
5 $\quad$ **if** $\deg(w) > k$ **then**
6 $\quad\quad$ add V to C;
7 $\quad\quad$ $k \leftarrow \deg(w)$;
8 $\quad$ delete w_i from V;
9 **return** C;

Let C_x be the smallest subgraph such that $|C_x| \geq k$. Let v_z be the vertex with the smallest index that is still in $C_x \setminus C_{x-1}$ and define $A = \{v_z, \ldots, v_j\}$. Let $g(v)$ be the degree of $v \in A$ right before v_j is removed during GREEDYLD. Note that, by definition, $\text{in}(v_j) \leq g(v)$, and that

$$\sum_{v \in A} g(v) = 2|E(A)| + |E_\times(A, C_{x-1})| \leq 2|E(A)| + 2|E_\times(A, C_{x-1})| = 2 \sum_{v \in A} \text{in}(v) \quad .$$

This leads to

$$p(k; C) = d(C_x, C_{x-1}) \geq d(A, C_{x-1}) = \frac{1}{|A|} \sum_{v \in A} \text{in}(v) \geq \frac{1}{2|A|} \sum_{g(v) \in A} g(v) \geq \frac{\text{in}(v_j)}{2},$$

where the optimality of C_x implies the first inequality. $\qquad\qquad\qquad\qquad\qquad\qquad$ □

We should point out that $p(1, \mathcal{B})$ is equal to the density of the densest subgraph, while $p(1, C)$ is equal to the density of the subgraph discovered by the Charikar's algorithm. Consequently, Proposition 4.7 provides automatically the 2-approximation guarantee of the Charikar's algorithm.

We should also point out that $p(i, C)$ can be larger than $p(i, \mathcal{B})$. However, if j is the *first* index, for which $p(j, C) \neq p(j, \mathcal{B})$, then Proposition 3.5 guarantees that $p(j, C) < p(j, \mathcal{B})$.

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 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 4.7, 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 5.1. *Let $\{C_i\}$ be the set of all k-cores of a graph $G = (V, E)$. Then $\{C_i\}$ forms a nested chain,*

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

Similar to Proposition 4.7, k-cores provide a factor-2 approximation with respect to the locally-dense subgraphs. The proof is in fact quite similar to that of Proposition 4.7.

PROPOSITION 5.2. *Let $\mathcal{B} = \{B_i\}$ be the set of locally-dense subgraphs. Let $C = \{C_i\}$ be the set of k-cores. Then*

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

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

Fix k to be an integer, $1 \leq k \leq n$ and let B_i be the smallest subgraph such that $|B_i| \geq k$. Let v_j be the last vertex occurring in B_i. We must have $in(v_j) \geq d(v_j, B_i \setminus \{v_j\})$, and, due to Lemma 4.8, $d(v_j, B_i \setminus \{v_j\}) \geq d(B_i, B_{i-1})$. In summary, we have

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

Let C_x be the smallest core such that $|C_x| \geq k$, and write $A = C_x \setminus C_{x-1}$. Let v_s be the vertex with the smallest index that is still in A, and let v_l be the vertex with the largest index that is still in A, that is, $\{v_s, \ldots, v_l\} = A$.

If $j > l$, then $in(v_j) < in(v_l)$, otherwise C_x is not a core. If $j < l$, then $in(v_j) \leq in(v_l)$, otherwise $væ \notin C_x$, and since $j \geq k$, then C_x is not the smallest core with at least k vertices, which is a contradiction. Hence, $in(v_j) \leq in(v_l)$.

Let $g(v)$ be the degree of $v \in A$ right before v_l is removed during CORE. We now have

$$
\begin{aligned}
p(k; C) &= & d(C_x, C_{x-1}) \\
&= & \frac{1}{|A|} \sum_{v \in A} in(v) \\
&\geq & \frac{1}{2|A|} \sum_{v \in A} g(v) \\
&\geq & \frac{in(v_l)}{2} \\
&\geq & \frac{in(v_j)}{2},
\end{aligned}
$$

which proves the proposition.   □

6 SEGMENTATION PROBLEM: CONSTRAINING THE NUMBER OF SUBGRAPHS

It is possible that the decomposition yields a significant amount of subgraphs. In such a case it may be useful to constraint the number of the subgraphs. In order to do so we need to define an optimization criterion, which will be our first step. We then demonstrate how to solve the problem exactly, and how to estimate the solution efficiently.

6.1 Problem definition

Our goal is to discover k nested subgraphs that minimize a certain cost. We base the cost on the degree of a node, relative to the subgraph. A natural approach here is to model the degree, that is, our goal is to maximize the log-likelihood $\sum_v \log p(\deg(v; C_i); \lambda_i)$, where C_i is the smallest subgraph containing v and λ_i is a parameter of the distribution. Unfortunately, this is problematic due to the following reason: an edge (x, y), where $x, y \in C_i \setminus C_{i-1}$ increases the degrees of both x

and y, whereas an edge (x, y), with $x \in C_i$ and $y \in C_{i-1}$ increases the degrees only for x and *not* for y. The distribution we will consider favors small degrees, so this leads to a scenario where the cost function implicitly favors having a lot of cross-edges. To rectify this problem we introduce the notion of *adjusted degree*, where we count each cross-edge twice.

Definition 6.1. Assume a sequence of nested subgraphs $C = (\emptyset = C_0 \subsetneq C_1 \subsetneq \cdots \subsetneq C_k = V)$. Let v be a vertex and let C_i be the smallest set containing v. Define the *adjusted degree* as

$$\mathrm{adg}(v; C) = |\{(v, u) \mid u \in C_i \setminus C_{i-1}\}| + 2|\{(v, u) \mid u \in C_{i-1}\}| \quad .$$

To reduce the clutter, we typically omit C from the notation and write $\mathrm{adg}(v)$.
Next we give a formal definition of the problem.

Definition 6.2. Assume that we are given a distribution $p(\cdot; r)$ for the adjusted degree. This distribution has one parameter r; small values indicate the likelihood of high degrees. Given a graph G and an integer k, find a k-*segmentation*, a sequence of nested subgraphs $C = (\emptyset = C_0 \subsetneq C_1 \subsetneq \cdots \subsetneq C_k = V)$ and parameters $\lambda_1 \leq \cdots \leq \lambda_k$, minimizing the negative log-likelihood

$$cost(C) = -\sum_{v \in V} \log p(\mathrm{adg}(v); \lambda_i),$$

where i is the index of the smallest C_i containing v.

The reason why we write this problem as a minimization problem is because typically the log-likelihood is negative, and in order to establish approximation guarantees we need to have the cost function to be positive.

We are specifically interested in geometric and exponential distributions. Both distributions can be written as $p(x; \lambda) = \exp(-\lambda x - Z(\lambda))$, where $Z(\lambda)$ is the normalization constant[3]. Moreover, smaller values of λ will result in a distribution favoring larger degrees, that is, inner subgraphs should be denser.

6.2 Exact algorithm

In this section we demonstrate how to find an optimal segmentation using locally-dense subgraphs. First we prove the key proposition that states that it is enough to use locally-dense subgraphs when looking for the optimal segmentation.

PROPOSITION 6.3. *Assume that p is either exponential or geometric distribution. Then there is an optimal segmentation $C = (\emptyset = C_0 \subsetneq C_1 \subsetneq \cdots \subsetneq C_k = V)$ such that each C_i is locally-dense.*

To prove the proposition, we need the following technical lemma.

LEMMA 6.4. *Let $C_1, \ldots, C_k$ be the optimal solution, and assume some of the subgraphs are not locally-dense. Then there is C_i that is not locally-dense along with the violating sets X and Y such that $Y \subseteq C_{i+1}$ and $X \cap C_{i-1} = \emptyset$.*

PROOF. Let C_i be a set that is not locally-dense, and let X and Y be the violating sets. Next we argue that we can safely assume that $Y \subseteq C_{i+1}$ and $X \cap C_{i-1} = \emptyset$. We will split the argument in two cases: Case (*i*): $Y \nsubseteq C_{i+1}$ and Case (*ii*): $Y \subseteq C_{i+1}$.

Assume Case (*i*). If $d(C_{i+1} \setminus C_i, C_i) \geq d(X, C_i \setminus X)$, then redefine Y as $C_{i+1} \setminus C_i$. In such case, X and Y are still violating the local density but now we can use Case (*ii*). Assume that $d(C_{i+1} \setminus C_i, C_i) <$

$d(X, C_i \setminus X)$. Define $Y_1 = Y \cap C_{i+1}$ and $Y_2 = Y \setminus Y_1$. Note that $Y_2 \neq \emptyset$. Assume that $d(Y_2, C_i \cup Y_1) \geq d(Y, C_i)$. Then

$$d(Y_2, C_{i+1}) \geq d(Y_2, C_i \cup Y_1) \geq d(Y, C_i) \geq d(X, C_i \setminus X) > d(C_{i+1} \setminus C_i, C_i) \quad .$$

Redefine Y as Y_2, X as $C_{i+1} \setminus C_i$, and increase i by 1. The previous arguments show that new Y and X violate the local density of C_{i+1}, so we repeat our argument with either Case (i) or Case (ii).

Assume now that $d(Y_2, C_i \cup Y_1) < d(Y, C_i)$. This forces $Y_1 \neq \emptyset$. Since $d(Y, C_i)$ is a weighted average of $d(Y_2, C_i \cup Y_1)$ and $d(Y_1, C_i)$, we have $d(Y, C_i) \leq d(Y_1, C_i)$. Redefine Y as Y_1, and apply Case (ii).

Assume Case (ii). Write $X_1 = X \cap C_{i-1}$ and $X_2 = X \setminus X_1$. If $X_1 = \emptyset$, then we are done; assume otherwise. If $d(C_i \setminus C_{i-1}, C_{i-1}) \leq d(Y, C_i)$, then we can replace X with $C_i \setminus C_{i-1}$ to complete the argument. Assume that $d(C_i \setminus C_{i-1}, C_{i-1}) > d(Y, C_i)$.

Assume $X_2 \neq \emptyset$. If $d(X_2, C_i \setminus X_2) \leq d(X, C_i \setminus X)$, then we can replace X with X_2 to complete the argument. Assume $d(X_2, C_i \setminus X_2) > d(X, C_i \setminus X)$. Note that $d(X, C_i \setminus X)$ is a weighted average of $d(X_2, C_i \setminus X_2)$ and $d(X_1, C_i \setminus X)$. This implies that $d(X_1, C_i \setminus X) < d(X, C_i \setminus X)$.

On the other hand, if $X_2 = \emptyset$, then $X_1 = X$, and $d(X_1, C_i \setminus X) = d(X, C_i \setminus X)$.

Combining everything gives us

$$d(X_1, C_{i-1} \setminus X_1) \leq d(X_1, C_i \setminus X) \leq d(X, C_i \setminus X) \leq d(Y, C_i) < d(C_i \setminus C_{i-1}, C_{i-1}) \quad .$$

Redefine X as X_1, Y as $C_i \setminus C_{i-1}$ and decrease i by one, and repeat Case (ii).

Note that we do first at most k repetitions of Case (i), and then at most k repetitions of Case (ii). After a finite numer of repetitions we end up with C_i that satisfies the conditions. This completes the proof. □

PROOF OF PROPOSITION 6.3. Both geometric and exponential distributions can be written as $p(x; \lambda) = \exp(-\lambda x - Z)$, where Z is the normalization constant (depending on λ).

Write $B_i = C_i \setminus C_{i-1}$. We can write the optimization function as

$$cost(C) = \sum_{i=1}^{k} \sum_{v \in B_i} Z_i + \lambda_i adg(v; C_i) = \sum_{i=1}^{k} |B_i|(Z_i + 2\lambda_i d(B_i, C_{i-1})),$$

where Z_i is the normalization constant for the parameter λ_i.

Assume that C_i is not locally-dense, that is, there is X and Y that violate the local density. Lemma 6.4 states that we can safely assume that $Y \subseteq C_{i+1}$ and $X \cap C_{i-1} = \emptyset$. This allows us to either remove X from C_i or add Y to C_i without changing the other sets.

The cost of the ith and the $i + 1$th segment is equal to

$$|B_i|Z_i + |B_{i+1}|Z_{i+1} + 2\lambda_i E_\Delta(B_i, C_{i-1}) + 2\lambda_{i+1} E_\Delta(B_{i+1}, C_i) \quad .$$

Let us define $W = B_i \cup B_{i+1}$. Due to the equality

$$E_\Delta(I, A) - E_\Delta(J, A) = E_\Delta(I \setminus J, A \cup J), \quad \text{for} \quad J \subset I, \tag{1}$$

the cost can be rewritten as

$$|B_i|(Z_i - Z_{i+1}) + |W|Z_{i+1} + 2\lambda_i E_\Delta(B_i, C_{i-1}) + 2\lambda_{i+1}(E_\Delta(W, C_{i-1}) - E_\Delta(B_i, C_{i-1}))$$

by setting $I = W$, $J = B_i$ and $A = C_{i-1}$. We would like to vary B_i while keeping the remaining variables constant; let us define

$$g(U) = |U|(Z_i - Z_{i+1}) + |W|Z_{i+1} + 2\lambda_i E_\Delta(U, C_{i-1}) + 2\lambda_{i+1}(E_\Delta(W, C_{i-1}) - E_\Delta(U, C_{i-1}))$$

$$= |U|(Z_i - Z_{i+1}) - 2(\lambda_{i+1} - \lambda_i)E_\Delta(U, C_{i-1}) + |W|Z_{i+1} + 2\lambda_{i+1}E_\Delta(W, C_{i-1}) \quad .$$

Note that the last two terms do not depend on U. Due to optimality of C_i, we have $g(B_i) \leq g(B_i \cup Y)$, or

$$0 \leq g(B_i \cup Y) - g(B_i)$$
$$= |Y|(Z_i - Z_{i+1}) - 2(\lambda_{i+1} - \lambda_i)(E_\Delta(B_i \cup Y, C_{i-1}) - E_\Delta(B_i, C_{i-1}))$$
$$= |Y|(Z_i - Z_{i+1}) - 2(\lambda_{i+1} - \lambda_i)E_\Delta(Y, C_i),$$

where the last equality is due to Eq. 1. We can rewrite the inequality as

$$Z_i - Z_{i+1} \geq 2(\lambda_{i+1} - \lambda_i)d(Y, C_i) \geq 2(\lambda_{i+1} - \lambda_i)d(X, C_i \setminus X),$$

where the last inequality follows from the fact that X and Y violate the local density of C_i, and since $\lambda_{i+1} \geq \lambda_i$. We can rewrite the left-hand side and the right-hand side as

$$0 \leq |X|(Z_{i+1} - Z_i) - 2(\lambda_{i+1} - \lambda_i)E_\Delta(X, C_i \setminus X) = g(B_i) - g(B_i \setminus X),$$

or $g(B_i \setminus X) \leq g(B_i)$.

We have shown that if there is C_i that is not locally-dense, we can delete some vertices from C_i without sacrificing the quality. We continue this until all C_i are locally-dense; the process must end because at each step we reduce the size of some C_i. □

The proposition gives us means to compute the optimal segmentation. First we discover locally-dense decomposition, say, $\mathcal{L}$. If the number of subgraphs is less or equal than k, we are done. Otherwise, we group subgraphs until we reach k. The optimal grouping can be done with a dynamic program. Write $o[i, j]$ to be the cost of partial j-segmentation using only $L_0, \ldots, L_i$. We have the identity

$$o[i, j] = \min_{\ell < i} c[\ell, i] + o[\ell, j - 1], \quad \text{where} \quad c[\ell, i] = -\sum_{v \in L_i \setminus L_\ell} \log p(\mathrm{adg}(v) ; \lambda)$$

and λ is the optimal parameter for modeling $L_i \setminus L_\ell$. This identity allows us to compute $o[n, k]$ recursively with a dynamic program. Note that the monotonicity of the segmentation—that is, the inner subgraphs should be more dense—is automatically guaranteed. We will refer to this algorithm as SEGMENT($\mathcal{L}, k$).

Computing $c[\ell, i]$ can be done in constant time. To see this, let $r = |L_i \setminus L_\ell|$ be the number of nodes in $L_i \setminus L_\ell$. Let also

$$q = \sum_{v \in \in L_i \setminus L_\ell} \mathrm{adg}(v) = 2E_\Delta(L_i \setminus L_\ell, L_\ell)$$

be the sum of all adjusted degrees in $L_i \setminus L_\ell$. Note r and q can be maintained in constant time. Then the corresponding costs for the geometric and exponential distributions are

$$c_{geo}[i, \ell] = -r \log \frac{r}{r + q} - q \log \frac{q}{r + q} \quad \text{and} \quad c_{exp}[i, \ell] = r + r \log \frac{q}{r} \quad .$$

Let us consider computational complexity. Discovering locally-dense decomposition can be done in $O(n^2 m)$ time, whereas the actual segmentation can be done in $O(\ell^2 k) \subseteq O(n^2 k)$ time, where ℓ is the number of subgraphs in locally-dense decomposition. In practice, $\ell \ll n$ so the segmentation step is relatively cheap. However, if ℓ is large, it is possible to achieve $(1 + \epsilon)$ approximation for the segmentation in linear time [18, 28].

6.3 Approximation algorithm

As pointed out above, the bottleneck of the exact algorithm is the locally-dense decomposition step. For large graphs we can significantly speed-up this step by using the faster algorithm GREEDYLD. The next proposition shows that this yields 2-approximation guarantee, if we use the geometric distribution.

PROPOSITION 6.5. *Let p be the geometric distribution. Let C be the optimal segmentation, and let $O =$ SEGMENT(GREEDYLD(G), k) be the optimal segmentation using the sets obtained from GREEDYLD. Then $cost(O) \leq 2cost(C)$.*

Before proving the result, we need to introduce some notation. The geometric distribution can be written as

$$- \log p(x; \lambda) = \lambda x + Z(\lambda),$$

where $Z(\lambda) \geq 0$ is the normalization constant.

To prove the result let us enumerate the vertices, that is, $V = \{v_i\}_{i=1}^n$, and assume that this order respects the optimal segmentation $\{C_i\}$, $v_i \in C_j$ implies that $v_{i-1} \in C_j$. Let $\{\lambda_i\}$ be the optimal parameters for $\{C_i\}$. We write η_i to be the parameter λ_j that is used to model $adg(v_i; C_j)$, where C_j is the smallest subgraph containing v_i. Write $Z = \sum Z(\eta_i)$ to be the sum of normalization constants. Note that $Z \geq 0$. Given a sequence $X = x_1, \ldots, x_n$, we define

$$f(X) = Z + \sum_{i=1}^n x_i \eta_i \quad .$$

Define A with $a_i = adg(v_i)$. Note that $f(A) = cost(C)$.

Define an order for vertex indices $(o_i)_{i=1}^n$, vertices with high degree first, that is, $\deg(v_{o_i}) \geq \deg(v_{o_{i+1}})$. Define a sequence T with $t_i = \deg(v_{o_i})$.

LEMMA 6.6. $f(T) \leq cost(C)$.

PROOF. Define T' as $t'_i = \deg(v_i)$. We argue first that $f(T') \leq cost(C) = f(A)$. We can rewrite

$$f(T') = Z + \sum_{(v_i, v_j) \in E} \eta_i + \eta_j \quad \text{and} \quad f(A) = Z + \sum_{(v_i, v_j) \in E} 2\eta_{\max(i,j)} \quad .$$

Since $\eta_i \leq \eta_{i+1}$, we have $f(T') \leq f(A)$. To prove $f(T) \leq f(T')$, note that

$$x\alpha + y\beta \leq y\alpha + x\beta, \quad \text{for} \quad \alpha \leq \beta, \ x \geq y \quad .$$

That is, let (q_i) be *any* vertex order, and let X be the degree sequence $x_i = \deg(v_{q_i})$. Then sorting the vertices with bubble sort from (q_i) to (o_i) will not increase the sum in f at any step. Consequently, $f(T) \leq f(X)$. Since this holds for any order, $f(T) \leq f(T')$, which proves the lemma. □

Let $(g_i)_{i=1}^n$ be the reverse order of indices in which GREEDYLD removes the vertices, and let s_i be the degree of v_{g_i} during its removal.

LEMMA 6.7. $s_i \leq t_i$.

PROOF. Consider two sets $P = \{v_{g_1}, \ldots, v_{g_{i-1}}\}$ and $Q = \{v_{o_1}, \ldots, v_{o_{i-1}}\}$. Assume that $P \neq Q$ when treated as sets, that is, there are indices j and ℓ with $j < i \leq \ell$ such that $g_j = o_\ell$. Let h be the degree of v_{g_j} when deleting v_{g_i}. Since GREEDYLD deletes the vertex with the smallest degree, $s_i \leq h$. Consequently, $s_i \leq h \leq \deg(v_{g_j}) = t_\ell \leq t_i$.

Assume the opposite case: $P = Q$. Due to pigeonhole principle, there is $\ell \geq i$ such that $o_\ell = g_i$. Thus, $s_i \leq \deg(v_{g_i}) = t_\ell \leq t_i$. □

PROOF OF PROPOSITION 6.5. Define B as $b_i = \text{adg}(v_{g_i})$. Note that $\text{adg}(v_{g_i}) \leq 2s_i$. Thus,

$$2f(T) = 2Z + 2\sum_{v_i \in V} \eta_i t_i \geq 2Z + 2\sum_{v_i \in V} \eta_i s_i \geq 2Z + \sum_{v_i \in V} \eta_i b_i \geq Z + \sum_{v_i \in V} \eta_i b_i = f(B) \quad .$$

Consider a segmentation $\mathcal{G}$ respecting the order g_i and having the same sizes as C, $|C_i| = |G_i|$. The value $f(B)$ corresponds to the log-likelihood of $\mathcal{G}$ and the parameters $\lambda_1, \ldots, \lambda_k$, and $cost(\mathcal{G})$ corresponds to the log-likelihood of $\mathcal{G}$ and the optimized parameters. Thus, $cost(\mathcal{G}) \leq f(B)$.

We have shown that there is a segmentation respecting the order chosen by GREEDYLD that is at most $2cost(C)$. Thus, the optimal segmentation respecting the order is also at most $2cost(C)$. The argument in the proof of Proposition 6.3 can be now used to show that we can safely assume that the segmentation uses sets returned by GREEDYLD. □

We can show a similar result for the exponential distribution as long as the original graph does not have any singletons.

PROPOSITION 6.8. *Let p be the exponential distribution. Assume that G has no singletons. Let C be the optimal segmentation, and let $O = \text{SEGMENT}(\text{GREEDYLD}(G), k)$ be the optimal segmentation using the sets obtained from GREEDYLD. Then $cost(O) \leq 2cost(C)$.*

PROOF. Similarly to the geometric distribution, exponential distribution can be written as

$$-\log p(x; \lambda) = \lambda x + Z(\lambda) \quad .$$

Let Z be as defined in proof of Proposition 6.5, that is, it is total sum of the normalization constants. To prove the result we only need to show that $Z \geq 0$, and we can use the proof of Proposition 6.5. Note that $Z(\lambda) = -\log \lambda$, and the optimal λ for a segment C_i is $1/[2d(C_i \setminus C_{i-1}, C_{i-1})]$. This leads to

$$Z = \sum_{i=1}^{k} |C_i| \log 2d(C_i \setminus C_{i-1}, C_{i-1}) \quad .$$

To prove the result we will show that $d(C_i \setminus C_{i-1}, C_{i-1}) \geq 1/2$. It is enough to prove the case $i = k$ as due to Proposition 6.3 the densities are monotonic.

Let X be any subset of vertices. As there are no singletons, $\deg(v) \geq 1$. This leads to

$$d(X, V \setminus X) \geq \frac{1}{2|X|} \sum_{v \in X} \deg(v) \geq \frac{1}{2} \quad .$$

Set $X = C_k \setminus C_{k-1}$ to complete the proof. □

We should point out that these results also work if the graph has weights on the edges. However, in such a case, Proposition 6.8 requires weights to be larger than or equal to 1.

7 RELATED WORK

This paper is an extension of previously published work [30], and in this extension we introduce the segmentation problem, where we constrain the number of subgraphs. Danisch et al. [15] introduced an alternative iterative technique for computing locally-dense decomposition that scales well in practice.

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 [20]. 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 [27]. 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 [24]. 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 [34]. 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 [20].

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 [31]. 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. [32], 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 [21]. Such a definition could serve for defining decompositions of directed graphs, which is also left for future work.

Hierarchical communities. A classic technique for modelling hierarchical nature of communities is with a hierarchical blockmodel [14]. Here we are given a tree, where the leaves are the vertices of the original graph and each vertex in a tree is given a probablility. We then model an edge (u, v) with a probability given to the lowest common ancestor of u and v. Tatti and Gionis [29] studied a restricted version of this problem where the tree yields a nested structure; inner communities being denser. Unfortunately, no exact polynomial-time algorithm is known for the restricted or general problem. On other hand, in the segmentation problem we based the model on degrees and not individual edges. This allowed to us to solve the problem exactly.

8 EXPERIMENTAL EVALUATION

We will now present our experimental evaluation. We tested 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.

8.1 Experimental setup

We performed our evaluation on 13 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.
- `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.
- `airports`: US flight traffic in January 2016[4], where vertices represent airports and weighted edges flight routes. The weights represent the number of flights between two airports.
- `trains`: UK train routes.[5] The vertices represent medium or large exchange points (stations), while the weighted edges represent scheduled routes. The weights represent the number of routes in a single week.

The first three datasets were obtained from UCIrvine Network Data Repository,[6] and the remaining datasets, except for `airports` and `trains`, were obtained from Stanford SNAP Repository.[7]

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

8.2 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 $O(n^2 m)$. 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 $O(k)$ 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 $O(nm)$ 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.

---

[4]http://www.transtats.bts.gov/

[5]http://data.atoc.org/

[6]http://networkdata.ics.uci.edu/index.php

[7]http://snap.stanford.edu/data

[8]The implementation is available at
https://version.helsinki.fi/dacs

Table 1. Basic characteristics of the datasets and the running times of the algorithms.

| Name | n | m | running time | | |
			CORE	GREEDYLD	EXACTLD
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	2 766 607	7s	8s	1m6s
skitter	1 696 415	11 095 298	21s	21s	1m46s
airports	294	3 995	11ms	10ms	27ms
trains	363	1 357	7ms	7ms	23ms

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

| Name | $r(C, \mathcal{B})$ | | | $d(C_1) / d(B_1)$ | |
	CORE	GREEDYLD		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
airports	0.75	0.90		0.93	1.00
trains	0.60	0.84		0.82	0.96

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

$$r(C, \mathcal{B}) = \min_i \frac{p(i; C)}{p(i; \mathcal{B})}, \tag{2}$$

where $\mathcal{B}$ is the locally-dense decomposition and C is obtained by either from GREEDYLD or CORE. These ratios are shown in Table 2. We also compare $p(1; C) / p(1; \mathcal{B})$, that is, the ratio of density

Fig. 2. Profile functions for `lesmis`, `fb1912`, `astro`, and `hepph`.

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	Core	GreedyLD	ExactLD	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
airports	221	200	219	0.99	0.99	0.996
trains	187	59	156	0.87	0.89	0.98

for the inner most subgraph in $\mathcal{C}$ against the density of $\mathcal{B}_1$, the densest subgraph. Propositions 4.7 and 5.1 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.

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 1.1 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 4.6. We illustrate several prototypical examples of such profile functions in Figure 2. 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.

In Figure 3 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 are merged and the 7th subgraph (in ExactLD) lends vertices to the 8th 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 protagonist of the book, Jean Valjean, is not placed into the first shell by Core.

Next, we present our result with segmentation. First we computed the cost of optimal segmentation as a function of the number of segments k. Here, we used exponential distribution as the underlying model. The normalized scores are shown in left plot of Figure 4. The scores behave similarly for all datasets: they improve quickly at the very beginning (for $k = 1, \ldots, 10$), after which they settle to a relatively stable value. This value depends on the dataset.

Next, we study how well can approximate the segmentation by using GreedyLD instead of the exact solution. The results are shown in the right plot of Figure 4. Here, we plot the relative difference between the approximate solution and the optimal solution. Ideally, the difference should be 0, and Proposition 6.8 states that it is at most 1. We see that in practice the estimates are really close to each other: all differences are within 0.006. The approximation is better for smaller k. This is a natural result as there is less room for disagreement in more coarse segmentations.

Finally, let us look on segmentations obtained from trains and airports data. Our goal is to discover which locations, that is, train stations or airports, are central. Here, by centrality we mean that a central location is well-connected with others central locations. To quantify this notion we use locally-dense subgraphs. Note that the number of locally-dense subgraphs is relatively large in these graphs; this is due to the fact that the graphs are weighted. We were interested to group the locations in 4 categories. So to reduce the the size of decomposition, we solved segmentation problem with $k = 4$ and the exponential model. The results are shown in Figure 5–7.

The discovered trains segmentation shows that the densest segment occurs in the vicinity of London, as expected. There is also a strong concentration of the second densest segment around Manchester/Liverpool area while the stations in Scotland, apart from the capital Edinburgh, are in outer segments. For airports, we see that the inner segments consists of large well-connected airports, such as JFK, DFW, ATL, or ORD, while the smaller, regional, airports are assigned to the outer segments.

Fig. 3. Decompositions of the `lesmis` dataset. The upper three graphs show the decompositions of each method using numbers and colors. The lower graph shows the abbreviated names; the table provides mapping from abbreviations to full names.

9 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 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

Fig. 4. Ratios of segmentation costs as a function of k the number of segments. Here, O is the optimal solution, C is the approximate solution using GREEDYLD, and $cost(\{V\})$ is the cost of having just one segment. The left figure shows the optimal costs normalized by $cost(\{V\})$. The right figure shows how well we approximate the exact solution by using GREEDYLD, the lower the better, ideally at 0.

Fig. 5. `airports` data, 4-segmentation using exponential distribution and EXACTLD based on traffic data. Segments are indicated by color and shapes. Groups with smaller indices consists of central airports that are connected to other central airports.

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.

Finally, we considered a constrained version of the problem, where we restrict the number of subgraphs. We do this by designing a model based on segmentation. The likelihood of this model is

Fig. 6. `trains` data, 4-segmentation using exponential distribution and ExactLD based on traffic data. Segments are indicated by color and shapes. To avoid clutter, London area is not annotated; see Figure 7 for the zoom-in. Groups with smaller indices consists of central stations that are connected to other central stations.

then optimized, and we show that we can do this either exactly or estimate this efficiently by a factor of 2.

Fig. 7. London area zoom-in of `trains` data, 4-segmentation using exponential distribution and EXACTLD. Segments are indicated by color and shapes. For the whole map, see Figure 6.

REFERENCES

[1] James Abello, MauricioG.C. Resende, and Sandra Sudarsky. 2002. Massive Quasi-Clique Detection. In *LATIN 2002: Theoretical Informatics*. 598–612.

[2] Alan Agresti. 2010. *Analysis of Ordinal Categorical Data* (2nd ed.). John Wiley & Sons.

[3] J. Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. 2005. k-core decomposition: a tool for the visualization of large scale networks. *CoRR* abs/cs/0504107 (2005).

[4] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 1996. Greedily finding a dense subgraph. *Scandinavian Workshop on Algorithm Theory (SWAT)* (1996), 136–148.

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

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

[7] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V. Hicks. 2011. Clique Relaxations in Social Network Analysis: The Maximum k-Plex Problem. *Operations Research* 59, 1 (2011), 133–142.

[8] Béla Bollobás. 1984. The evolution of random graphs. *Trans. Amer. Math. Soc.* 286, 1 (1984), 257–274.

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

[10] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding All Cliques of an Undirected Graph. *Commun. ACM* 16, 9 (1973), 575–577.

[11] Toon Calders, Nele Dexters, Joris J. M. Gillis, and Bart Goethals. 2014. Mining frequent itemsets in a stream. *Information Systems* 39 (2014), 233–255.

[12] Shai Carmi, Shlomo Havlin, Scott Kirkpatrick, Yuval Shavitt, and Eran Shir. 2007. A model of Internet topology using k-shell decomposition. *Proceedings of the National Academy of Sciences* 104, 27 (2007), 11150–11154.

[13] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. *APPROX* (2000).

[14] Aaron Clauset, Cristopher Moore, and Mark EJ Newman. 2008. Hierarchical structure and the prediction of missing links in networks. *Nature* 453, 7191 (2008), 98–101.

[15] Maximilien Danisch, T-H Hubert Chan, and Mauro Sozio. 2017. Large Scale Density-friendly Graph Decomposition via Convex Programming. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 233–242.

[16] Werner Dinkelbach. 1967. On Nonlinear Fractional Programming. *Management Science* 13, 7 (1967), 492–498.

[17] Andrew V Goldberg. 1984. Finding a maximum density subgraph. *University of California Berkeley Technical report* (1984).

[18] Sudipto Guha, Nick Koudas, and Kyuseok Shim. 2006. Approximation and streaming algorithms for histogram construction problems. *ACM Transactions on Database Systems* 31, 1 (2006), 396–438.

[19] Patric Hagmann, Leila Cammoun, Xavier Gigandet, Reto Meuli, Christopher J. Honey, Van J. Wedeen, and Olaf Sporns. 2008. Mapping the Structural Core of Human Cerebral Cortex. *PLoS, Biology* 6, 7 (2008), 888–893.

[20] Johan Håstad. 1996. Clique is Hard to Approximate Within $n^{1-\epsilon}$. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. 627–636.

[21] Samir Khuller and Barna Saha. 2009. On Finding Dense Subgraphs. In *Automata, Languages and Programming*, Vol. 5555. 597–608.

[22] 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.

[23] David Matula and Leland Beck. 1983. Smallest-last Ordering and Clustering and Graph Coloring Algorithms. *J. ACM* 30, 3 (1983), 417–427.

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

[25] James Orlin. 2013. Max flows in $O(nm)$ time, or better. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*. 765–774.

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

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

[28] Nikolaj Tatti. 2019. Strongly polynomial efficient approximation scheme for segmentation. *Inform. Process. Lett.* 142 (2019), 1–8.

[29] Nikolaj Tatti and Aristides Gionis. 2013. Discovering Nested Communities. In *Machine Learning and Knowledge Discovery in Databases—European Conference, ECML PKDD 2013*. 32–47.

[30] Nikolaj Tatti and Aristides Gionis. 2015. Density-friendly Graph Decomposition. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. 1089–1099. https://doi.org/10.1145/2736277.2741119

[31] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015*. 1122–1132.

[32] Charalampos E. Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A. Tsiarli. 2013. 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*. 104–112.

[33] Johan Ugander, Lars Backstrom, Cameron Marlow, and Jon Kleinberg. 2012. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences* 109, 16 (2012), 5962–5966.

[34] Takeaki Uno. 2010. An Efficient Algorithm for Solving Pseudo Clique Enumeration Problem. *Algorithmica* 56, 1 (2010), 3–16.