Fast Enumeration of Large \(k\)-Plexes

Alessio Conte  
University of Pisa  
conte@di.unipi.it

Donatella Firmani  
Roma Tre University  
donatella.firmani@uniroma3.it

Caterina Mordente  
Be Think Solve Execute  
c.mordente@be-tse.it

Maurizio Patrignani  
Roma Tre University  
patrigna@dia.uniroma3.it

Riccardo Torlone  
Roma Tre University  
torlone@dia.uniroma3.it

ABSTRACT

\(k\)-plexes are a formal yet flexible way of defining communities in networks. They generalize the notion of cliques and are more appropriate in most real cases: while a node of a clique \(C\) is connected to all other nodes of \(C\), a node of a \(k\)-plex may miss up to \(k\) connections. Unfortunately, computing all maximal \(k\)-plexes is a gruesome task and state-of-the-art algorithms can only process small-size networks. In this paper we propose a new approach for enumerating large \(k\)-plexes in networks that speeds up the search by several orders of magnitude, leveraging on (i) methods for strongly reducing the search space and (ii) efficient techniques for the computation of maximal cliques. Several experiments show that our strategy is effective and is able to increase the size of the networks for which the computation of large \(k\)-plexes is feasible from a few hundred to several hundred thousand nodes.

ACM Reference format:
DOI: 10.1145/nnnnnn.nnnnnn

1 INTRODUCTION

In the vast majority of networks representing real-world scenarios the distribution of edges is not uniform and it is often possible to clearly distinguish groups of nodes that are highly connected. The automatic detection of these groups, often called communities, helps to discover fundamental properties of large networks in a variety of different domains. For this reason this problem has been largely investigated [13].

A clique is a set of nodes in a network with all possible edges among them, and is a formal and strict way of defining a community. So strict, in fact, that cliques are generally thought to be too rigid to be used in practice [16]. A more appropriate notion in many practical cases is the \(k\)-plex: a set of nodes such that each of them has edges with all the others, with the possible exception of up to \(k\) missing neighbors (including itself). So, for example, for \(k = 1\), \(k\)-plexes are cliques, for \(k = 2\), each node may miss one edge, etc. Hence, \(k\)-plexes are a simple and intuitive generalization of cliques.

Unfortunately, the detection of all maximal \(k\)-plexes in a network is impractical being hindered by two main problems: (i) maximal \(k\)-plexes are even more numerous than maximal cliques, even if most \(k\)-plexes are small and not significant; (ii) the most efficient algorithms in the literature for computing maximal \(k\)-plexes can only be used on small-size graphs: we show in our experiments that the largest networks we were able to analyze with the algorithm in [5] have a few hundred nodes.

In this paper we propose a solution to the first issue that is also a solution to the second one. Namely, if we restrict the search to large \(k\)-plexes, which are the most meaningful in practice, we can devise efficient algorithms to detect them.

Indeed, computing all maximal \(k\)-plexes does not make sense when the purpose is that of detecting communities. In this respect, it is useful to focus on the relationship between \(s\), the size of a \(k\)-plex, and \(k\) itself. Starting from \(k = 1\), which corresponds to cliques, if we increase the value of \(k\), we obtain progressively sparser communities that are clearly less interesting in practice. In addition, there is a dramatic effect on small \(k\)-plexes: it is trivial that if \(s \leq k\) a \(k\)-plex can be composed of isolated nodes, but it is possible to show that even if \(s < 2k\), the \(k\)-plex can be disconnected (see Section 5). Hence, small \(k\)-plexes do not correspond to communities. In particular, in order to avoid finding the degenerate \(k\)-plexes mentioned above, it is natural to impose at least that \(s \geq 2k\).

In this framework, our strategy for finding large \(k\)-plexes relies on two main observations. First, the complexity of the problem can be reduced in the vast majority of cases on the basis of certain properties of large \(k\)-plexes that can be efficiently checked and that allows us to filter out a large portion of the network before starting their search. The second consideration is that, differently to what happens for \(k\)-plexes, the state-of-the-art techniques to compute all maximal cliques are able to scale up to millions of nodes by decomposing the network into small blocks [7, 11]. Unfortunately, the decomposition approach cannot be easily adapted to the detection of \(k\)-plexes. However, we demonstrate that we can find all \(k\)-plexes non-smaller than \(m\) by looking in the neighborhood of cliques of a size that depends on \(k\) and \(m\). Hence, it turns out that the knowledge of maximal cliques in a network provides a hint for finding all the significant \(k\)-plexes.

In sum, our contributions are the following.

- We present techniques to efficiently compute all maximum \(k\)-plexes of a network in the hypothesis that they are greater than a fixed threshold, which is adequate to obtain significant results. Our approach is based on the intuition
As mentioned in the introduction, our approach is based on two main ideas: (i) starting the search of \(k\)-plexes, we can filter out a relevant portion of the network in which necessary conditions for the presence of large \(k\)-plexes do not hold, and (ii) in large networks, cliques can drive the search of \(k\)-plexes. While the first point provides an effective way to simplify the problem at hand, the second can lead to an efficient strategy for finding \(k\)-plexes.

Let us elaborate on these ideas starting with the problem of finding all \(k\)-plexes of maximum size. Assume that we have computed all the maximal cliques of a network and let \(\omega\) be the size of the maximum clique. Then, a maximum \(k\)-plex has size at least \(\omega\), since cliques are also \(k\)-plexes. For example, suppose we are searching for \(k\)-plexes in the network in Fig. 1a, which we will use as a running example in this section: we have that \(\omega = 5\), since the maximum clique (the blue subgraph on the left hand side) involves five nodes. At this point, it turns out that two filtering criteria can be applied.

1. **Coreness** Our first intuition follows from the very definition of \(k\)-plex: all the nodes of a \(k\)-plex of size \(m\) must have degree non-smaller than \(m - k\). If we know that the size of a maximum \(k\)-plex is at least \(\omega\), this means that we can iteratively filter out any node that has degree lower than \(\omega - k\). This corresponds to computing the coreness of all the nodes of the network (Lemma 5.3), a process that can be executed in linear time [4]. For example, suppose we are searching for 2-plexes in the running example of Fig. 1a for which \(\omega = 5\): we can filter out the three black nodes on the top of the picture since they have coreness 2, which is less than \(\omega - k = 3\). In larger networks we show that this criterion allows us to cut up to 99% of the nodes.

2. **Cliqueness** The second intuition is that any node of a \(k\)-plex of size \(m\) must be included in a clique of a size that depends on \(m\). This is confirmed by Corollary 5.5 stating that any node of a \(k\)-plex larger or equal to \(m\) is included in a clique of size at least \([m/k]\). Then, if the size of the maximum \(k\)-plex is at least \(\omega\), we can cut out all those nodes that do not belong to any clique of size at least \([\omega/k]\). For example, if we are searching for 2-plexes in the network depicted in Fig. 1a, we can filter out all nodes that do not belong to cliques of size at least \([5/2]\) = 3, that is, the pair of black nodes in the bottom of the network. We will show in Section 6 that in larger instances this criterion can be tested efficiently and is able to cut up to 98% of the nodes.

Even if some nodes can be filtered out both because their low cliqueness and low coreness, the network in Fig. 1a shows that the two filtering criteria are indeed independent. When both criteria are applied, the size of the network is reduced of magnitude and standard techniques for finding \(k\)-plexes may become applicable even to very large networks.

We push ahead this approach to devise a technique that allows us to further increase the tractable cases. In fact, a consequence of Corollary 5.5 is that if the largest \(k\)-plex has size \(p\), then the network contains a clique of size at least \(p/k\), which in turn must be less than the size of the maximum clique \(\omega\). This implies that the size \(p\) of a maximum \(k\)-plex in the network cannot exceed \(k \cdot \omega\). Hence, the size \(p\) of the searched \(k\)-plexes is in the interval \([\omega, k \cdot \omega]\). In our running example, this would mean to search in the interval \([5, 10]\). Now we iteratively apply the following algorithm:

1. **Coreness** We make an educated guess of a value of \(p\) in the interval \([\omega, k \cdot \omega]\) that allows us to filter out many nodes based on the cliqueness criterion.

2. **Coreness** We launch a traditional method to find all \(k\)-plexes on such reduced network

3. **Coreness** If we find some \(k\)-plex of size greater or equal than \(p\) we are done (our guess was correct and we found all maximum \(k\)-plexes).

4. **Coreness** If we find only \(k\)-plexes of size \(s \in [\omega, p - 1]\), then we know that our guess was too optimistic and we iterate, using \(p - 1\) as new upper bound and using \(s\) as new lower bound (and filtering out nodes based on it).
This technique is effective at least whenever the size $p$ of the largest $k$-plex is much larger than the size $\omega$ of the largest clique, which is a reasonable property for some networks.

Once we have found the largest $k$-plexes, we may be interested into searching smaller ones. We have noted in the introduction that an exhaustive search does not make much sense, since very small $k$-plexes are not significant, to the point that they may be even disconnected or composed by a set of isolated nodes. Hence, the second problem we tackle is to find all maximal $k$-plexes in the network of size bigger than a threshold $m$.

As mentioned above, our idea is to start from cliques, which are $k$-plexes but not necessarily maximal, and possibly enlarge them to find maximal $k$-plexes. Building on the cliqueness criterion, which ensures that each node of a $k$-plex $C$ of size $s$ is included into a clique of size at least $[s/k]$, we start from each of such cliques $K$. If we set $m\geq k^2$, we have that $|K|\geq [s/k] > k$, which implies that any other node of $C$ must be adjacent to at least one node of $K$ (in other words, $K$ is a dominating set of $C$). Hence, we can search for $C$ restricting to a block including $K$ and all its adjacent nodes. For example, suppose you are searching for all maximal 2-plexes of size at least 5 in the network of the top in Fig. 1b. Consider any clique of size at least $[s/k] = [5/2] = 3$, for example the clique $K = \{a, b, e\}$ (yellow triangle in Fig. 1b). The nodes of any $k$-plex of size at least 5 containing $K$ are adjacent to $K$ (surrounded nodes of Fig. 1b top).

We further reduce the size of the block by proving that $C$ can be obtained by considering only nodes belonging to $K$ and to other cliques of size at least $[s/k]$ intersecting with $K$ (Lemma 5.7). For example, the 2-plex of size 6 on the right of Fig. 1b is all contained into the clique $\{a, b, e\}$ and three other cliques of size 3 intersecting with $K$ (surrounded nodes of Fig. 1b bottom). This gives rise to an efficient searching algorithm that decomposes the network into blocks each composed of one clique as the core, and all intersecting cliques as the boundary. Each block can be separately processed, possibly in a distributed environment.

3 FINDING MAXIMUM K-PLEXES

In this section, we show our algorithms for enumerating the maximum $k$-plexes of the input graph $G$. Intuitively, our approach consists of enumerating all the $k$-plexes of a targeted sub-graph of $G$, that we refer to as $H$, and then selecting the largest ones. Clearly, the smaller is $H$ with respect to $G$, the faster is solving the problem, as long as $H$ contains all the maximum $k$-plexes of $G$. Precisely, we aim at extracting a sub-graph $H$ out of $G$, that is:

(1) small enough to make the enumeration fast;
(2) large enough to capture all maximum $k$-plexes of $G$, and allow the computation of a correct solution.

We design two different sub-graph extraction criteria, dubbed coreness and cliqueness. Our criteria are of the form “all the $k$-plexes larger or equal than $m$ consist of nodes with property XYZ”, and have the above desiderata for a suitable choice of $m$. Practically speaking, we make the following observations.

• If $m$ is too small, then the criteria are trivially met by most (or even all) nodes of $G$ (violating the first desiderata);

• If $m$ is too large, then the criteria are met only by few (or even none) nodes of $G$ (violating the second desiderata);

To this end, we show what are the best settings of the threshold $m$ in different scenarios. The theoretical analysis in Section 5 proves that the solution provided by our settings is correct, and the experiments in Section 6 show that the enumeration on the resulting sub-graph is greatly faster than in the input graph.

In the rest of this section, we first describe our criteria and related concepts, then we describe the corresponding algorithms for enumerating maximum $k$-plexes, based on different choices for the threshold $m$.

3.1 Criteria

We now describe our coreness and cliqueness criteria, and how to use them for extracting the sub-graph $H$ out of $G$.

Coreness. Our first criterion is the simplest, and it is based on the intuition that all the nodes of a $k$-plex $C$, with $|C|\geq m$ have degree non-smaller than $m-k$. Clearly, we do not know a priori what are the nodes of $C$. However, we can iteratively filter out any node that has degree lower than $m-k$. Formally, this is equivalent to search for the $(m-k)$-cores of $G$. We define this concept in the following.

Definition 3.1. An $h$-core of $G$ is a maximal connected subgraph of $G$ in which all nodes have degree at least $h$. A node $u$ has coreness $h$ if it belongs to a $h$-core, but not to any $(h+1)$-core.

An $h$-core is one of the connected components of the sub-graph of $G$ formed by repeatedly deleting all nodes of degree less than $h$. We are now ready to state our coreness criterion, as follows.

Criterion 1 (Coreness). All the $k$-plexes of $G$ larger or equal than $m$ consist of nodes having coreness larger or equal than $m-k$.

Cliqueness. Our second criterion is based on the intuition that all the nodes of a $k$-plex $C$, with $|C|\geq m$ form smaller cliques with other nodes of $C$. Informally speaking, if we try to “draw” a $k$-plex by adding one edge at a time, we soon realize that there are no ways of placing edges without forming progressively larger cliques here and there. Lemma 5.4 in Section 5 proves that every node of $C$ participates in a clique non-smaller than $\frac{|C|}{k}$. Therefore, we can filter out any node that only participates in smaller cliques. We define this concept in the following.

Definition 3.2. A node $u$ has cliqueness $h$ if it belongs to a clique of $G$ of size $h$, but not to any clique of $G$ of size $h+1$.

We are now ready to state our cliqueness criterion, as follows.

Criterion 2 (Cliqueness). All the $k$-plexes of $G$ larger or equal than $m$ consist of nodes having cliqueness larger or equal than $\frac{m}{k}$.

Algorithm 1 prune($G, k, m$) algorithm that computes a sub-graph of $G$ according to Criterion 1 and 2.

1: $G'\leftarrow\{v\in G : G.coreness(v)\geq m-k\}$
2: $H\leftarrow\{v\in G' : G'.cliqueness(v)\geq \frac{m}{k}\}$
3: return $H$

Computing the sub-graph. Let $m$ be given as input. The procedure prune($G, k, m$), shown in Algorithm 1, returns the sub-graph
Algorithm 2 $\text{max\_plexes}(G, k)$ algorithm.

1. $K \leftarrow \text{all\_cliques}(G)$
2. $m \leftarrow \max_{K \in \Omega} |K|$
3. $H \leftarrow \text{prune}(G, k, m)$
4. $K \leftarrow \text{all\_plexes}(H, k)$
5. $P \leftarrow \text{argmax}_{K \in \Omega} |K|$
6. return $P$

Algorithm 3 $\text{max\_plexes\_binary}(G, k)$ algorithm.

1. $K \leftarrow \text{all\_cliques}(G)$
2. $m \leftarrow \max_{K \in \Omega} |K|$
3. $LB \leftarrow m$
4. $UB \leftarrow k \cdot m$
5. while $LB \neq UB$ do
   6. $x \leftarrow \frac{LB+UB}{2}$  $\triangleright$ Pivot
   7. $H \leftarrow \text{prune}(G, k, x)$
   8. $K \leftarrow \text{all\_plexes}(H, k)$
   9. $x' \leftarrow \max_{K \in \Omega} |K|$
10. $LB \leftarrow \max(LB, x')$
11. $UB \leftarrow \max(x, x')$
12. end while
13. $P \leftarrow \text{argmax}_{K \in \Omega} |K|$
14. return $P$

$H$ resulting from a combination of coreness and cliqueness. We first compute the $(m-k)$-cores (line 1), and then we filter out all the nodes with low cliqueness in the cores (line 2). Note that the two criteria can be applied sequentially, in any order. Since computing coreness is easy [4], we chose to apply coreness first and compute cliqueness on the smaller graph $G'$ (line 2).

### 3.2 Algorithms for maximum $k$-plexes

Let $\omega$ be the size of any maximum clique in $G$. Criterion 2 shows that the size of any maximum $k$-plex lies in the range $[\omega, k \cdot \omega]$. In principle, for the problem at hand, $m$ can be set anywhere in the interval $[\omega, k \cdot \omega]$. With our desiderata in mind (complete solution and fast enumeration), we identify two alternative strategies that make sense with no prior knowledge on the maximum $k$-plexes, that we refer to as the cautious choice and the greedy choice.

- **Cautious choice.** We set $m = \omega$. The pros are that we cannot miss the maximum $k$-plexes. The cons are that if the size of the maximum $k$-plex is closer to $k \cdot \omega$ than to $\omega$, $H$ may be much larger than needed.
- **Greedy choice.** We set $m$ in the middle. The pros are that $H$ is very small. The cons are that we may miss the maximum $k$-plex. To this end, we can iterate in a binary search fashion.

Let us introduce the following useful methods.

- $\text{all\_cliques}(G)$: The method returns all the maximal cliques of $G$.
- $\text{all\_plexes}(H, k)$: The method returns all the maximal $k$-plexes of the sub-graph $H$, for a given $k$, with any appropriate state-of-art method.

**Cautious choice.** Our first strategy is illustrated in Algorithm 2, that we refer to as $\text{max\_plexes}()$. The procedure is very simple. We first enumerate all the maximal cliques of $G$ (line 1), and then extract the sub-graph $H$ using our criteria and the maximum clique size as threshold (line 2–3). Finally, we enumerate all the maximal $k$-plexes of the sub-graph $H$, and return the maximum ones (line 4–5). Note that some cliques can be included in the solution. We observed that some instances in our experiments only have one maximum clique $C_{\max}$, that also corresponds to the maximum $k$-plex. In such extreme cases, the sub-graph $H$ only consists in $C_{\max}$, and we can even do without $\text{all\_plexes}()$.

**Greedy choice.** Our second strategy is illustrated in Algorithm 3, that we refer to as $\text{max\_plexes\_binary}()$. We first enumerate all the maximal cliques of $G$ (line 1), as in $\text{max\_plexes}()$. After that, rather than using the maximum clique size as threshold, the algorithm attempts to extract a smaller sub-graph by setting $m$ to the middle point of the range $[\omega, k \cdot \omega]$ (line 6–7). Then, we enumerate all the maximal $k$-plexes of the sub-graph $H$, and compute the maximum size $x'$ (line 9). If $H = \emptyset$ we set $x' = 0$. The following scenarios are possible.

- If $x' \in [0, x)$, then there are no $k$-plexes larger than the pivot $x$ and the search continues. The lower-bound does not change (line 10), and the pivot becomes the new upper-bound (line 11).
- If $x' \geq x$, then we found $k$-plexes non-smaller than the pivot $x$. The search comes to an end, since the value $x'$ becomes both the new lower-bound (line 10) and the new upper-bound (line 11).

In particular, if $x' \geq x$, then $x'$ is also the maximum $k$-plex size of $G$ by our criteria. Equivalently, all the maximum $k$-plexes of the input graph $G$ must be included in $K$ (line 8). Therefore, at the end of the search, we return the maximum elements of $K$ (line 13–14).

Note that if we replace the selection of $x = \frac{LB+UB}{2}$ with $x = \omega$ we obtain the same behavior than $\text{max\_plexes}()$ algorithm.

### 4 FINDING LARGE $k$-PLEXES

Let $\omega_k$ be the size of any maximum $k$-plex of $G$, as computed for instance with the algorithm $\text{max\_plexes}()$ in the earlier section. A natural next step is to find all the $k$-plexes within a range $(1 - \epsilon)\omega_k$. One may be tempted to re-use the approach of $\text{max\_plexes}()$ as a template, and change line 2 of Algorithm 2 by setting $m = (1 - \epsilon)\omega_k$.

While this is valid in principle, we observed in practice that large $k$-plexes are more numerous that maximum $k$-plexes (that in most cases only consist of a single max $k$-plex) even if $\epsilon$ is small. On the one hand, this confirms our initial intuition that enumerating all the $k$-plexes of $G$ is impractical. On the other, this calls for the design of an efficient algorithm specific for the large $k$-plexes problem at hand. To this end, we introduce a third criterion, that we refer to as OVERLAPPINGCLIQUE, which enables the enumeration of large $k$-plexes in networks that are orders of magnitude larger than previously considered.

**Overlapping cliques criterion.** This is an advanced application of the cliqueness criterion. Let $C$ be a $k$-plex non-smaller than $m$. Consider any maximal clique $K \in C$. We know from the cliqueness criterion that $|K| \geq \frac{m}{k}$. 

Algorithm 4 large_plexes(G, k, m) algorithm.
1: \( H \leftarrow \text{prune}(G, k, m) \)
2: \( K \leftarrow \text{all_cliques}(H) \)
3: for \( K \in \mathcal{K} \) do
4: \( B \leftarrow \text{boundary}(K) \)
5: \( P \leftarrow \text{all_plexes}(H[B], k) \) \( \implies H[B] \) is the subgraph of \( H \)
induced by \( B \subseteq V \).
6: for \( P \in P, |P| \geq m \) do
7: \( \text{yield } P \) \( \implies \) duplicates admitted
8: end for
9: end for

Algorithm 5 boundary(K) algorithm.
1: \( B \leftarrow \emptyset \)
2: for \( K' \in \mathcal{K} \) do
3: if \( K' \cap K \neq \emptyset \) then
4: \( B \leftarrow B \cup K' \)
5: end if
6: end for
7: return \( B \)

- If \( \frac{k - 1}{2} \geq k \), every node of \( C \setminus K \) must be adjacent to at least
one node \( v \in K \), since every node of a \( k \)-plex can miss up
to \( k - 1 \) neighbors.
- Every node of \( C \setminus K \) must itself participate in a clique \( K' \),
at least as large as \( \frac{k - 1}{2} \).

Intuitively, for large enough \( k \)-plexes with \( m \geq k^2 \), all the nodes of
\( C \) not in \( K \) participate in cliques overlapping with \( K \), that is, sharing
at least one node with \( K \). (We provide a proof of this statement in
Lemma 5.7.) This allows us to target our search for large \( k \)-plexes
to the neighborhood of each clique \( K' \). More formally, to all the
cliques overlapping with \( K \), that we refer to as its boundary. We
state the overlapping cliques criterion as follows.

**Criterion 3 (OverlappingCliqu es).** All the \( k \)-plexes of \( G \) with
size \( m \geq k^2 \) consist of nodes either belonging to a clique \( K \) s.t. \( |K| \geq \frac{m}{k} \), or to overlapping cliques \( K' \), s.t. \( |K'| \geq \frac{k}{2} \) and \( K \cap K' \neq \emptyset \).

Algorithm for large \( k \)-plexes. Our algorithm for large \( k \)-plexes
is illustrated in Algorithm 4, that we refer to as large_plexes().
The procedure uses the same auxiliary methods than max_plexes() (see Section 3), and the boundary(K) primitive in addition.

- boundary(K): The method returns all the nodes included
in a given clique \( K \), or in overlapping cliques.

The large_plexes() algorithm first extracts the sub-graph \( H \)
using our criteria and the input threshold \( m = (1 - \epsilon)\omega_k \) (line 1).
Note that prune() selects all the nodes of \( G \), except those that do not
participate in any \( k \)-plex larger or equal than \( m \). Then, it enumerates
all the maximal cliques of the sub-graph \( H \) (line 2). The resulting set
\( \mathcal{K} \) (line 2) only consists of cliques that are larger than \( \frac{m}{k} \)
(because of the constructive process of \( H \)) and thus can be used as “seeds”
for growing large \( k \)-plexes, according to the OVERLAPPINGCliqu es
criterion. Finally, the large_plexes() algorithm iterates over \( \mathcal{K} \)
(lines 3–9) and for each clique considers its boundary (line 4). Let \( B \)
the current set returned by boundary(). We first enumerate all the
maximal \( k \)-plexes of the sub-graph of \( H[B] \) of \( H \) induced by \( B \)
(line 5). Then, we return only \( k \)-plexes non-smaller than \( m \) (lines
6–8), and proceed with the next clique.

In Algorithm 5 we show a simple implementation of the boundary() algorithm. At line 2, the data structure \( \mathcal{K} \) is shared with the caller method (i.e., large_plexes()).

**Duplicates.** Note that the above method can return the same \( k \)-plex multiple times (line 7). To this end, we design a clever test,
that guarantees that every \( k \)-plex is returned at most once. Let \( C \)
be any \( k \)-plex computed by all_plexes(H[B], k). We define the concept of parent \( P(C) \), and return \( C \) only when the current
clique is equal to \( P(C) \). Specifically, let

- \( \text{min}(C) \) be the node \( u \) in \( C \) with smallest id;
- \( \text{complete}(X, Y) \) be a method that iteratively adds to the
clique \( X \) the next minimum node in the set \( Y \), \( s.t. \) \( X \) is still a
clique;

We define \( P(C) \) by construction as in the following equation.
\[
P(C) = \text{complete}(\text{complete}(\text{min}(C)), C, H) \quad (1)
\]
Practically speaking, we start from the node \( u \) in \( C \) with smallest
id. Then, the process of construction has two phases. In the first
phase, we extend \( u \) within \( C \) in increasing order of id. Then we
keep extending by selecting nodes from the whole \( H \).

We can thus rewrite line 6 as “if \( P(P) = K \) then yield \( P \)”:
this way each \( k \)-plex is returned exactly once.

5 **THEORETICAL BASIS**

A \( k \)-plex \( C = (V, E) \) with \( s \) nodes, \( s \geq 2k \), is a graph such that
every node is adjacent at least to all other nodes in \( C \) except \( k \). For the sake
of simplicity \( C \) may refer to both the set of nodes it contains and to
the induced graph. We refer to nodes adjacent to a given node \( u \),
as neighbors of \( u \). A clique can be thought of as a 1-plex. Any subset
of a \( k \)-plex is also a \( k \)-plex, and a \( k \)-plex is also a \( k + 1 \)-plex. We
need to prove the three criteria described in the previous sections,
dubbed coreness, cliqueness and OVERLAPPINGCliqu es.

**Coreness criterion.** Let \( \Delta(C) \) denote the diameter of \( C \), that is,
the largest number of nodes which must be traversed in order to travel
from one node to another. While a clique, or a 1-plex, has diameter
equals 1, \( k \)-plexes with \( k > 1 \) come in a variety of forms and can have
arbitrarily high diameter (which is not a desirable property for
a community). However, for \( k \leq \frac{3}{2} \) – which means that every
node is adjacent more than half nodes in \( C \) – the diameter is only
at most 2. This is proven in the following.

**Lemma 5.1.** If \( s \geq 2k \) then \( \Delta(C) \leq 2 \).

**Proof.** If \( C \) has diameter larger than 2, there are at least two
nodes \( u \) and \( v \) at distance larger than 2. Since \( u \) is missing at most
\( k \) edges, it has at least \( s - k \) neighbors. However, neither \( u \) nor
its neighbors are connected to \( v \) and therefore \( v \) missing at least
\( s - k + 2 \) edges, which is larger than \( k \) if \( k \leq \frac{3}{2} \).

**Corollary 5.2.** If \( s \geq 2k \) then \( C \) is connected.

We are now ready for proving the coreness criterion.

**Lemma 5.3 (Coreness).** Every node in \( C \) has coreness at least \( s - k \).

**Proof.** Let \( \delta(u) \) denote the degree of a node \( u \), that is, the
number of nodes of \( C \) adjacent to \( u \). It easy to verify that for every node
c ∈ C, δ(u) ≥ s − k. A h-core of C is a connected subgraph of C in which all nodes have degree at least h [19]. Since C is connected by Corollary 5.2, C is a s − k-core. □

Other criteria. We give the technical lemma below, that we use for deriving the cliqueness and adjacency criteria, and the advanced search principle.

Lemma 5.4. Every clique $X \subseteq C$, s.t. $|X| < \frac{4}{k}$, is included in a bigger clique $X_{big}$, s.t. $|X_{big}| \geq \frac{4}{k}$.

Proof. Let $X \subseteq C$ be any clique of $C$, s.t. $|X| < \frac{4}{k}$. Let $N \subseteq C$ be the set of nodes which are not adjacent to all nodes of $X$, that is, that are adjacent from 0 to $|X| - 1$ nodes of $X$. By picking any node $u' \in C \setminus (X \cup N)$, we have that $X' = X \cup \{u'\}$ is a clique of size $|X| + 1$. Since every $u \in X$ can miss at most $k$ neighbors including itself, $|N \cup X| \leq |X| (k - 1) + |X| = k|X|$. This means that at most $k|X|$ nodes are excluded for the selection of $u'$. Let $N'$ be the nodes not adjacent to all nodes of $X'$. We can repeat the process and grow $X'$, by picking any node $u'' \in C \setminus (X' \cup N')$, until we run out of nodes. Note that the newly-excluded nodes for selecting $u''$ are $u'$ and its missing neighbors. Such a clique-growing process can be thought of as an iterative process starting from a node and growing a clique, as if $X$ itself were grown after $X$ steps of the process -- and excluding at most $k$ nodes at a time. Therefore, the process will run at least $\left\lceil \frac{4}{k}\right\rceil$ steps, after which $X$ has been grown to $\frac{4}{k}$ nodes. □

Note that, in case $\frac{4}{k}$ is not integer, the proof yields $|X_{big}| \geq \left\lceil \frac{4}{k}\right\rceil$.

The cliqueness and the adjacency criteria directly follow.

Corollary 5.5 (cliqueness). Every node in $C$ has cliqueness at least $\left\lceil \frac{4}{k}\right\rceil$.

Lemma 5.6. Consider a clique $X \subseteq C$, s.t. $|X| \geq \frac{4}{k}$. If $s \geq k^2$, every node in $C$ either belongs to $X$ or has a neighbors in $X$.

Proof. We know that such a clique always exists from Lemma 5.4. Since its size is at least $k$ by assumption, then every $u \notin C \setminus X$ has to be adjacent to at least one node in $X$. □

Our last criterion is a stricter form of the above lemma, which we formalize as follows.

Lemma 5.7 (overlapping cliques). Consider a clique $X \subseteq C$, s.t. $|X| \geq \frac{4}{k}$. If $s \geq k^2$, every node in $C$ either belongs to $X$ or to an overlapping clique $X'$, s.t. $|X'| \geq \frac{4}{k}$ and $X \cap X' \neq \emptyset$.

Proof. Let $u$ be any node of $C \setminus X$. We know from Lemma 5.6 that exists a node $v \in X$ adjacent to $u$. Since $\{u, v\}$ is a clique of size 2, we can apply Lemma 5.4 and conclude that both nodes belong to a clique $X'$ larger or equal than $\frac{4}{k}$. Finally, $v \in X \cup X'$. □

6 EXPERIMENTS

In this section, we compare our and previous algorithms over different real-world networks, and show the advantages and limitations of our approach.

Datasets. We considered a mix of real-world networks with various sizes and characteristics. All our networks are publicly available on the LASAGNE meta-repository [1], and come from different human activities. Specifically, we consider:

- small to large collaboration networks, where nodes represent authors of published papers or books, and edges represent co-authorship;
- a large citation network, where nodes represent published papers or books, and edges represent citations;

The networks considered in our experiments, together with their number of nodes ($n$), density, maximum clique size ($\omega$) and type, are listed in Table 1. In the table, we also list the file name of every network in the repository, for sake of reproducibility.

Implementation details. We implement the auxiliary methods used by our algorithms, with the most recent methods in literature for the corresponding tasks, to the best of our knowledge.

In all $\text{cliques}(G)$, we use the algorithm in [11] for enumerating all the maximal cliques of the input graph $G$.

In all $\text{plexes}(H, k)$, we use the algorithm in [5] for enumerating all the maximal $k$-plexes of the sub-graph $H$.

We implement $\text{coreness}$ and $\text{cliqueness}$ criteria, respectively, with the method in [4] and the already mentioned algorithm in [11]. Our code is publicly available [2].

Methodology. We compare the execution of our methods for enumerating targeted $k$-plexes, with the most recent method for enumerating all $k$-plexes of the input graph [5]. In other words, we compare the algorithms $\max\text{plexes}(G, k)$, $\max\text{plexes}_\text{binary}(G, k)$, $\max\text{plexes}(G, k, m)$ with the algorithm $\text{all}\text{plexes}(G, k)$ over the same input graph, for different values of $k$ and threshold $m$. To the best of our knowledge, indeed, there are no faster approaches than [5] that are specific for maximum $k$-plexes and $k$-plexes non-smaller than a threshold. Targeted enumeration is indeed our contribution.

Test environment. Our experiments were performed on a machine with two CPU Intel Xeon E5-520 units with 4 cores each, running at 2.26GHz, with 8MB of cache and 32GB RAM. The operating system was Linux CentOS 6.7, with kernel version 2.6.32, Java Virtual Machine version 1.8.0_111 (64-Bit) and Python version 2.6.6 (64-Bit). All our executions have a reasonable 6 hours timeout, after which they are interrupted. In the experiments, we show that even the smallest networks with few thousands nodes timeout with traditional methods. Our algorithms, instead, can process networks up to hundreds of thousands of nodes.

6.1 Results

In Figure 2, we show the number of nodes in the sub-graph $H$ computed by the algorithm $\text{prune}(G, k, m)$, with different values of...
Figure 2: Number of nodes (log scale) in the sub-graph $H$.

Figure 3: Nodes of $G$ surviving the CORENESS criteria.

Figure 4: Nodes of $G$ surviving the CLIQUENESS criteria.

$k$ and $m = \omega$, i.e., the maximum clique size. As frame of comparison, we show the number of nodes in $G$ (i.e., $n$). The figure shows that, except for newm where $\omega = 3$, the sub-graph $H$ is order of magnitudes smaller than $G$. Notably, for different networks (jazz, geom, hepPh, newm, and dblp), $H$ is left with only the nodes of the maximum $k$-plex. In such lucky cases we can even skip the execution of the enumeration step $\text{all}_k\text{plexes}(H, k)$. Since the sub-graph produced by a given $k$ is included in the sub-graph produces by $k + 1$, is not surprising that higher values of $k$ yield more nodes in $H$. (Remember that for $k = n$ we have $G = H$.) However, for most instances, the sub-graph is small with respect to $G$ (thus allowing for faster enumeration of $k$-plexes) for different values of $k$.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$k$</th>
<th>Full Enum Time</th>
<th>Our Approach Time</th>
<th>Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>jazz</td>
<td>2</td>
<td>&gt; 6h</td>
<td>&gt; 6h</td>
<td>-</td>
</tr>
<tr>
<td>grQc</td>
<td>2</td>
<td>4.85 s</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>hepPh</td>
<td>2</td>
<td>&gt; 6h</td>
<td>&gt; 6h</td>
<td>-</td>
</tr>
<tr>
<td>newm</td>
<td>4</td>
<td>&gt; 6h</td>
<td>&gt; 6h</td>
<td>-</td>
</tr>
<tr>
<td>astroPh</td>
<td>4</td>
<td>&gt; 6h</td>
<td>&gt; 6h</td>
<td>-</td>
</tr>
<tr>
<td>mathSci</td>
<td>2</td>
<td>2.75 s</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>dblp</td>
<td>4</td>
<td>&gt; 6h</td>
<td>&gt; 6h</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2: Running time for finding all the largest 2-plexes.

Table 3: Running time for finding all $k$-plexes larger than 80% of the maximum clique.

Figures 3 and 4 report the fraction of nodes of $G$ residual after the CORENESS and CLIQUENESS criteria, applied separately. Figure 5 shows the same fraction after both criteria, applied together as in our prune() algorithm. For the considered networks, most nodes are filtered out by CORENESS. Then, the structures that are too connected to be filtered by CORENESS but too small to play a role in the search for $k$-plexes non-smaller than $\omega$, are filtered out by CLIQUENESS. Such an additional CLIQUENESS step has bigger impact in advogato and patents.

Maximum $k$-plexes. In Table 2, we show running times for different steps of $\text{max}_k\text{plexes}(G, 2)$, compared to the time required for enumerating all 2-plexes (column "FULL ENUM"), over the same input graph.

- column "core" is the time needed for applying CORENESS criterion, and computing an intermediate sub-graph $G'$;
- column "clique" is the time needed for applying CLIQUENESS criterion over the intermediate sub-graph $G'$, and computing $H$;
column "enum" is the execution time of `large_plexes()` over H.

The time for computing our criteria has little impact on the overall running time, which is dominated by the enumeration of 2-plexes of the residual sub-graph when necessary. As a consequence, in all the networks where H is left with only the nodes of the maximum clique, which is in turn also the maximum 2-plex, the computation of `max_plexes()` ends successfully after fractions of seconds. For such networks, we write "no need" in the "enum" column. As frame of comparison, full enumeration of 2-plexes (i.e., as in [5]) requires hours even on our smallest network (jazz), and times out on the other networks.

When H is not as minimal as in the networks with no need for `all_plexes(H, k)`, the running time of `max_plexes()` still ranges from fractions of seconds (most networks) to 2 h, proportionally to the size of H (see Figure 2 for comparison). In general, we can conclude that the running time of our algorithm mostly depends on the size of the sub-graph computed by `prune()`.

We observed that the results for higher values of k, namely k = 3 and k = 4, are similar. This is because the sub-graph H produced with higher value of k contains only few more nodes than the sub-graph produced with k = 2, as shown in Figure 2.

**Binary search.** Our "cautious choice" strategy for the maximum k-plexes is able to process quickly all the networks in Table 2, except `newm` and patents. For the `newm` network, we observe that \( \omega = 3 \) (see Table 1) and the size of the maximum 2-plex is \( \omega_2 = 5 \), that is, \( \omega_2 \) is closer to \( 2 \cdot \omega \) (i.e., the theoretical maximum size for a 2-plex) than to \( \omega \). This makes the `newm` network a good candidate for a "greedy choice" as the `max_plexes_binary()` algorithm described in Section 3.2. In practice, the value of \( \omega_2 \) is not known a priori. However, we can decide for the greedy choice also by observing that \( \omega \) is small. We observed experimentally for the `newm` network that the computation of `max_plexes_binary(G, 2)` terminates after only one iteration in less than our time out. Instead, the network patients is hard to process even with the greedy choice. To improve on this result is a challenging future task.

**Large k-plexes.** In Table 3, we show the overall running time of `large_plexes(G, k, m)` on different networks in our dataset, for different values of k. For this experiment, we set \( m = 0.8 \omega_2 \), where \( \omega_2 \) is the maximum k-plex size, as computed by `max_plexes(G, k)`. The time required for enumerating all k-plexes (column "full enum") of such networks is always larger than our timeout (6 hours). The table also show the number of k-plexes returned (column "found"). All the networks considered contain less than a dozen k-plex non-smaller than \( 0.8 \omega_2 \), which are quickly found by our algorithm in most cases. Note that in this experiment we call `prune(G, k, 0.8 \omega_2)`, that is, we compute the sub-graph H using a different threshold than in Figure 2 (possibly \( 0.8 \omega_2 \geq \omega \)).

The `astroPh` network requires much more time than other networks. To this end, for this specific network, we plot in Figure 6a the overall running time of `large_plexes(G, 2, m)` (i.e., for 2-plexes), for m ranging in \([0, \omega_2] \). We computed using `max_plexes(G, 2)` that \( \omega_2 = 57 \). On the one side of the spectrum, for \( m = 0 \), all the 2-plexes ought to be returned (i.e., the same result than `all_plexes()`). On the other side, for \( m = \omega_2 \) only a single 2-plex – the maximum 2-plex – is returned. The plot shows that the running time quickly decreases from hours to seconds as m approaches to \( \omega_2 \). The data-point close to 5 hours corresponds to \( m = 0.8 \omega_2 \), as in Table 3. We compare these results with the cumulative distribution function (CDF) of 2-plexes of `astroPh`, as reported in Figure 6b. As frame of comparison, we also show CDF of cliques (that are 1-plexes). As expected, most 2-plexes have size smaller than \( 10^2 \). For \( m > 0.8 \omega_2 \), which represents the 99th percentile for 2-plexes, the running time of `large_plexes()` is even smaller than half an hour, confirming the effectiveness of our targeted enumeration system.

### 7 RELATED WORKS

In the field of network analysis, dense substructures in graphs (aka dense subgraphs) are associated with communities, or more in general sets of closely related elements [13, 16]. The problem of finding these substructures has been extensively studied for decades, and continues to be the object of cutting edge research. The simplest and most rigorous definition of dense subgraph is the clique, i.e., a subgraph in which all nodes are pairwise connected. Many algorithms for finding all maximal cliques have been developed, most of them being inspired to the Bron-Kerbosh algorithm [6], such as [12, 18] or to the more recent paradigm of reverse search [3], such as [9, 10, 14]. The definition of clique may be too strict in...
We have proposed a novel approach to the enumeration of large \( k \)-plexes, a formal and meaningful way to define interesting communities in real-world networks that generalizes the notion of clique.

Two main clues have driven our solution: (i) a relevant portion of the network can be filtered out well before starting the detection of large \( k \)-plexes and (ii) cliques, which are more restricted but can be computed efficiently, can be used as starting points for the search of \( k \)-plexes in the network. The efficiency of the approach over state-of-the-art algorithms has been confirmed by our experiments.

In the future, we intend to further extend the applicability of our approach and tackle the problem of computing large \( k \)-plexes on real world networks with millions of nodes.

ACKNOWLEDGMENTS

We thank the authors of [5] for providing us with their code and for discussions on the subject of this paper.

REFERENCES