A Fast Exact Algorithm to Enumerate Maximal Pseudo-cliques in Large Sparse Graphs

Conference Paper · August 2024

DOI: 10.1145/3637528.3672066

4 authors, including:

Ahsanur Rahman
North South University
5 PUBLICATIONS **89** CITATIONS
SEE PROFILE

Kalyan Roy
North South University
3 PUBLICATIONS **22** CITATIONS
SEE PROFILE

Townim Faisal Chowdhury
North South University
11 PUBLICATIONS **85** CITATIONS
SEE PROFILE

A Fast Exact Algorithm to Enumerate Maximal Pseudo-cliques in Large Sparse Graphs

Ahsanur Rahman
ahsanur.rahman@northsouth.edu
North South University
Dhaka, Bangladesh

Kalyan Roy
kalyan.roy@northsouth.edu
North South University
Dhaka, Bangladesh

Ramiza Maliha
ramiza.maliha@northsouth.edu
North South University
Dhaka, Bangladesh

Townim Faisal Chowdhury
townim.chowdhury@adelaide.edu.au
Australian Institute for Machine Learning
University of Adelaide, Adelaide, Australia

Abstract

Pseudo-cliques (subgraphs with almost all possible edges) have many applications. But they do not satisfy the convertible anti-monotone constraint (as we prove here). So, it is hard to reduce the search space of pseudo-cliques and list them efficiently. To our knowledge, only two exact algorithms, namely, ODES and PCE, were proposed for this purpose, but both have high execution times. Here, we present an exact algorithm named *Fast Pseudo-Clique Enumerator (FPCE)*. It employs some pruning techniques we derived to reduce the search space. Our experiment on 15 real and 16 synthetic graphs shows that (i) on real graphs, FPCE is, on average, 38.6 and 6.5 times faster than ODES and PCE, respectively, whereas (ii) on synthetic graphs, FPCE is, on average, 39.7 and 3.1 times faster than ODES and PCE, respectively. We apply FPCE and a popular heuristic method on a PPI network to identify pseudo-cliques. FPCE outputs match with more known protein complexes, are more accurate, and are biologically more significant – suggesting that the exact computation of pseudo-cliques may give better insights. For its speed, FPCE is a suitable choice in such cases.

CCS Concepts

• **Mathematics of computing → Graph algorithms**.

Keywords

graph clustering, pseudo-clique, quasi-clique, dense subgraph

ACM Reference Format:

Ahsanur Rahman, Kalyan Roy, Ramiza Maliha, and Townim Faisal Chowdhury. 2024. A Fast Exact Algorithm to Enumerate Maximal Pseudo-cliques in Large Sparse Graphs. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3672066

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3672066

1 Introduction

Graphs, *a.k.a.* networks, are often used to model real-world objects and their mutual interactions. For example, social networks represent social relations among people; web graphs represent links between web pages; protein-protein interaction (PPI) networks describe mutual interactions among proteins; etc. Densely interconnected subgraphs of such networks are useful to identify social communities from social networks [12], protein complexes from PPI networks [15], link spams from web networks [37], etc. Different research devised different formulations to define such subgraphs and used different terms to indicate them. Unfortunately, the literature uses these terms inconsistently (Section 2). Here, we use the term *dense subgraph* to indicate any definition of such a subgraph.

Motivation: Although a lot of research has been conducted on dense subgraph mining [23], to our knowledge (as shown in Section 2 and as noted by some others [30, 46]), most of them either (i) focus on heuristic algorithms – which do not guarantee to find all dense subgraphs correctly, or (ii) define dense subgraphs in ways that allow them to develop fast exact algorithms for finding those.

To our knowledge, only two exact algorithms employ a natural definition of dense subgraphs: *a subgraph in which almost all possible edges are present* (Figure 1). This definition of dense subgraphs (henceforth called *pseudo-cliques* following [45]) is useful to identify protein complexes [28, 33], co-regulated proteins [22], disease-related circRNAs [50], Schizophrenia risk genes [25], COVID19 drugs [13], etc. In absence of efficient exact algorithms, these works rely on heuristic algorithms to compute pseudo-cliques – which hinders them from judging the actual potential of pseudo-cliques in these applications. Here, we address these issues by (i) developing a fast, exact algorithm to list all maximal pseudo-cliques in a graph and (ii) evaluating its performance and the quality of its outputs.

Contributions. We made the following contributions here.

(1) We prove that "pseudo-cliqueness" is not a convertible anti-monotone property. So, algorithms available for mining patterns with anti-monotone or convertible anti-monotone property are not applicable to pseudo-cliques (Section 4).
(2) We derive some pruning techniques to prune the search-space of pseudo-cliques (Sections 4 and 5, Appendix B).
(3) We integrate these techniques with an algorithm [45] to design an exact algorithm for listing pseudo-cliques. We call it *Fast Pseudo-Clique Enumerator* or *FPCE*, in short (Section 5).

(4) We derive a tighter (than that based on [45]) bound on the time complexity of FPCE. Our analysis is much more detailed and, so, is easier to understand (Section 5, Appendix C).
(5) Our experiment on 16 synthetic and 15 real graphs shows that FPCE is always faster than its competitors (Section 8).
(6) Our ablation study shows that all pruning techniques of FPCE are useful for its speedup (Section 9).
(7) We compare the quality of pseudo-cliques found by FPCE from a PPI network with (i) those found by a popular heuristic approach and (ii) *degree-based quasi-cliques* (another definition of dense subgraphs, see Section 2). FPCE outputs are found to be more biologically interesting (Section 10).

Intuition: Our pruning Techniques are based on our observation that each pseudo-clique has a certain maximum order, contains a certain r-clique, and can be extended from a smaller pseudo-clique only if its edge count and minimum degree are high enough. If any of these conditions are violated, we stop growing the pseudo-clique.

Scope: We focus on sparse graphs only since no exact algorithm can list pseudo-cliques from dense graphs efficiently (Section 8).

2 Related Work

Dealing with Inconsistent Terminology Usage in Literature: Finding works that use our definition of pseudo-cliques is complicated due to the discrepancy of terminology usage in the literature. *Some papers use the term quasi-clique [21, 44], dense-subgraph [26], or near-clique [8, 43] to mean pseudo-cliques while some others use these terms to mean some other definitions of dense subgraphs [18, 32, 37].* For example, different papers use *quasi-clique* to mean at least three different notions: (i) a pseudo-clique [21, 44], (ii) a subgraph induced by a node-set S in which each node's degree is at least $\alpha(|S|-1)$ where α is a user-given threshold [6, 51] (we call it *Degree-based Quasi-clique*, following [34, 41]), and (iii) a subgraph whose average degree is greater than a threshold [37] (we call it *Average-degree-based Quasi-clique*). So, to find all relevant papers, we searched for literature mentioning any of these words and looked at papers citing those works. However, a comprehensive review of all such papers is beyond the scope of this paper (see [16, 23] for this purpose). Here, we only review algorithms to find pseudo-cliques – *no matter what terms were used to denote them in the respective work*. We also discuss some other notable related work to highlight the difference between our and their definitions of dense subgraphs.

Non-exact Algorithms: Abello *et al.* [1] designed a greedy randomized adaptive search procedure (GRASP) to find maximal pseudo-cliques in a graph. A greedy heuristic was proposed in [6] to compute the largest degree-based quasi-clique. Two stochastic local search algorithms were proposed in [9] to find maximal subgraphs that satisfy requirements of both pseudo-cliques and degree-based quasi-cliques. Another local search algorithm was described in [11] to find the maximum pseudo-clique. Tsourakakis *et al.* [44] defined an objective function that generalizes notions of both average-degree-based quasi-cliques and pseudo-cliques as well as devised two non-exact algorithms to find the subgraph achieving the highest value of that function. A distributed algorithm was described in [8] that can find a single pseudo-clique (having a theoretically proven minimum density) in a graph with high probability. But it cannot find pseudo-cliques with a given minimum density and minimum order thresholds. Also, it is not exact and was never implemented.

Exact Algorithms for Finding the Maximum Pseudo-clique: The first exact algorithm to find the largest pseudo-clique in an input graph was proposed by Pattillo *et al.* [35]. They designed two mixed integer programming (MIP) formulations to solve this problem. Four alternative MIP formulations for the same problem were proposed in [46]. It was also solved by Pajouh *et al.* [29] via a DFS-based branch-and-bound algorithm. Finally, Marinelli *et al.* [30] solved it via a branch-and-price algorithm.

Exact Algorithms for Other Related Problems: A fixed-parameter algorithm was proposed in [20] to find the densest subgraph with k nodes. A flow-based algorithm was proposed in [14] to find the largest subgraph with the highest average degree. A DFS-based exact algorithm, called *Quick*, was proposed in [24] that lists all maximal *degree-based quasi-cliques* in a graph. Komusiewicz *et al.* [21] designed a DFS-based branch-and-bound algorithm to find a single pseudo-clique of order k (a user input).

While these are important problems, algorithms to solve them cannot be easily adapted to efficiently find all the maximal pseudo-cliques in a graph — which is the goal of our work.

Exact Algorithms for Finding All Maximal Pseudo-cliques: We found only two exact algorithms for solving this problem for a graph: Overlapping dense subgraph (ODES) [26] and Pseudo Clique Enumerator (PCE) [45]. ODES is a BFS-like search algorithm that maintains a queue of all the order-k pseudo-cliques found so far and tries to grow each of these pseudo-cliques in the next iteration to find all the order-$(k+1)$ pseudo-cliques. ODES continues growing each pseudo-clique in the queue as long as it satisfies the minimum-density constraint. To maintain such a large queue, ODES needs a huge memory. PCE, on the other hand, is a reverse-search [3] based (DFS-like) recursive method that continues growing the current pseudo-clique as long as the minimum-density constraint is satisfied and reports it when it becomes a maximal pseudo-clique.

3 Notations and Definitions

Table 1 lists our notations. We formally define pseudo-clique below.

Definition 1. ***θ-Pseudo-Clique:*** *For a given threshold θ with $0 \leq \theta \leq 1$, a node-set P with $|P| \geq 2$ is called a θ-pseudo-clique if its density, $den(P) \geq \theta$.*

Usually, we are interested in large pseudo-cliques. The following definitions allow us to incorporate these requirements.

Definition 2. ***(ℓ,θ)-Pseudo-Clique:*** *Let θ and ℓ be two given thresholds where $0 \leq \theta \leq 1$ and $\ell \geq 2$. Then P is called a (ℓ,θ)-pseudo-clique if $den(P) \geq \theta$ and $|P| \geq \ell$.*

Definition 3. ***Maximal (ℓ,θ)-Pseudo-Clique:*** *P is called a maximal (ℓ,θ)-pseudo-clique if P is a (ℓ,θ)-pseudo-clique and there is no (ℓ,θ)-pseudo-clique Q, such that $P \subset Q$.*

We aim to design a fast algorithm to solve the problem below.

Problem: *Enumerate all maximal (ℓ,θ)-pseudo-cliques in a sparse graph for a given density threshold θ and order threshold ℓ.*

Basic Algorithm: Explore all subgraphs in a given graph and report the ones that are maximal (ℓ,θ)-pseudo-cliques.

To systematically enumerate these subgraphs without visiting the same subgraph twice, we want to represent each subgraph by a unique order of its vertices. We define two such orderings below.

Table 1: Notations

Symbol	Meaning/Definition
G	Undirected graph with vertex set V and edge set E
$G[P]$	Subgraph induced by the set of vertices $P \subseteq V$
$E[P]$	Set of edges in $G[P]$
K_r	An r-clique
$d(v)$	Degree of vertex v in the whole graph G
$d_P(v)$	Degree of vertex v in $G[P]$
$dav(P)$	Average vertex-degree of $G[P] = 2\|E[P]\|/\|P\|$
$\delta(P)$	Minimum degree of $G[P] = min\{d_P(v) : v \in P\}$
$N(v)$	Set of all neighbors of v in G
$C_k(G)$	k-core of graph $G = argmax_{P \subseteq V}\{\|P\| : \delta(P) \geq k\}$
$C_k(P)$	k-core of subgraph $G[P]$
$c(v)$	Coreness of vertex v in $G = max\{k : v \in C_k(G)\}$
$c_P(v)$	Coreness of vertex v in $G[P]$
$c(P)$	Coreness of vertex set $P = max\{k : P \subseteq C_k(G)\}$
ξ_G	Degeneracy of $G = max_{v \in V}\{c(v)\} = max_{H \subseteq V}\{\delta(H)\}$
ξ_S	Degeneracy of subgraph $G[S]$
$den(P)$	Density of subgraph $P = E[P] \,/\, \binom{\|P\|}{2}$

Definition 4. ***Degree Ordering:*** *The degree ordering $\succ$ of a subgraph $G[P]$ is a total order of nodes in P s.t. $u \succ v \Leftrightarrow (d_P(u) > d_P(v)) \vee ((d_P(u) = d_P(v)) \wedge (u > v))$.*

For a node-set P, we use $P_\succ$ to denote an ordered tuple of nodes in P ordered by a given ordering $\succ$. For example, consider the subgraph $G[P]$ induced by $P = \{1, 2, 3, 4, 5\}$ in graph G of Figure 1. Assuming $\succ$ denotes the degree ordering of G, $P_\succ =< 5, 4, 3, 1, 2 >$. However, when $\succ$ represents the degree ordering of $G[P]$, $P_\succ =< 4, 3, 2, 1, 5 >$. We use $v^*(P)$ to denote the last element in degree ordering of $G[P]$, *i.e.,* for all $w \in (V[P] - \{v^*(P)\})$, $w \succ v^*(P)$. Here $v^*(P) = 5$.

Definition 5. ***MD Ordering:*** *The MD ordering, a.k.a. reverse degeneracy ordering of a subgraph $G[P]$ for a node-set $P = \{v_1, v_2, \ldots, v_n\}$ is a total order $P_\succ =< v_1, v_2, \ldots, v_n >$ s.t. $\forall v_i \in P : v_i = v^*(P^i)$ where $P^i =< v_1, v_2, \ldots, v_i >$ is the i-length prefix of $P_\succ$.*

For example, if $\succ$ denotes the MD ordering of G, $P_\succ =< 5, 4, 3, 2, 1 >$ but if $\succ$ denotes the MD ordering of $G[P]$, $P_\succ =< 4, 3, 2, 1, 5 >$.

MD ordering sorts the vertices in descending order of their corenesses , *a.k.a.* core numbers [47] where coreness of a vertex is the maximum core it belongs to (see Table 1 and Figure 1).

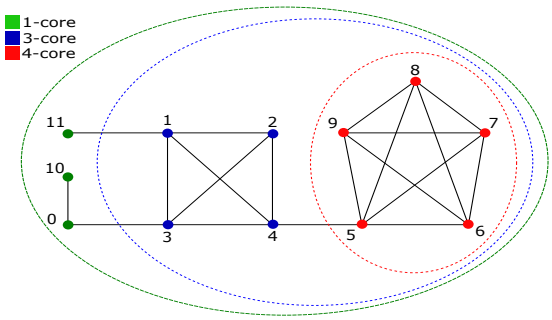

Figure 1: A toy graph. Here both $S_1 = \{1, 2, 3, 4\}$ and $S_2 = \{1, 2, 3, 4, 5\}$ are $(0.7, 4)$-pseudo-cliques. S_2 is a maximal $(0.7, 4)$-pseudo-clique but S_1 is not. Coreness of each green node is one, whereas coreness of each blue (red) node is three (four).

Our algorithm explores subgraphs in MD ordering. To avoid exploring non-pseudo-cliques, it exploits the following constraint.

Definition 6. ***Loosely Anti-monotone (LAM) Constraint:*** *Let an itemset (which is a node-set in our case) S satisfy a constraint p. Then p is called a loosely anti-monotone [7] or a quasi-hereditary constraint [29] if at least one subset of S also satisfies p.*

Every θ-pseudo-clique contains at least one θ-pseudo-clique as a subset, i.e., θ-pseudo-cliques satisfy the LAM constraint [29]. It implies that if no subset of P is a θ-pseudo-clique then P is not a θ-pseudo-clique. Thus, this constraint helps reduce the search space of θ-pseudo-cliques, as we only need to grow a subgraph as long as it remains a θ-pseudo-clique. However, it is less effective for search space reduction than the famous anti-monotone [7] or convertible anti-monotone [36] constraints defined below.

Definition 7. ***Anti-monotone (AM) Constraint:*** *Let an itemset S satisfy a constraint p. Then p is called a hereditary [29] or an anti-monotone [36] constraint if each subset of S also satisfies p.*

Definition 8. ***Convertible Anti-monotone (CAM) Constraint:*** *Let a total order $\succ$ be defined over all the items in the universal set (in our case, V). So the items in any itemset S can be represented as an ordered tuple, $S_\succ =< x_1, x_2, \ldots, x_{|S|} >$. Let S satisfy a constraint p. Then p is called a convertible anti-monotone constraint [36] with respect to the order $\succ$ if every prefix of $S_\succ$ also satisfies p.*

For example, cliques satisfy anti-monotone constraint because every subset of a clique is also a clique. On the other hand, average-degree-based pseudo-cliques satisfy convertible anti-monotone constraint with respect to the degree ordering ($>$) of G because for every prefix P of any node-set $S \subseteq V$ it holds that $dav(P) \geq dav(S)$ [36].

4 Properties of Pseudo-cliques

In this section, we first show that pseudo-cliques may not satisfy the convertible anti-monotone constraint – which explains why search-space reduction for pseudo-cliques is not as easy as that for average-degree-based quasi-cliques. Next, we establish some properties of pseudo-cliques, which we utilize later to prune the search space of pseudo-cliques in our algorithm (Section 5).

Lemma 1. *There exists no ordering of all vertices for which θ-pseudo-cliques satisfy the convertible anti-monotone constraint.*

Proof. Let there be a node-ordering $\succ$ for which θ-pseudo-clique satisfies the convertible anti-monotone constraint. Take the graph of Figure 1 after deleting 9. Consider two 0.7-pseudo-cliques in that graph: $P = \{1, 2, 3, 4, 5\}$ and $Q = \{4, 5, 6, 7, 8\}$. According to the convertible anti-monotone constraint, every prefix of $P_\succ$ (respectively, $Q_\succ$) must be a 0.7-pseudo-clique. But the only subset of P (resp., Q) having $|P| - 1$ (resp., $|Q| - 1$) nodes which is also a 0.7-pseudo-clique is: $P' = P - \{5\}$ (resp., $Q' = Q - \{4\}$). So $P'_\succ$ (resp., $Q'_\succ$) must be a prefix of $P_\succ$ (resp., $Q_\succ$). So 4 (resp., 5) must precede 5 (resp, 4) in the ordering $P_\succ$ (resp., $Q_\succ$) *i.e.,* the order of nodes 4 and 5 differs between $P_\succ$ and $Q_\succ$ (contradiction). □

This lemma shows that no algorithm can compute maximal (ℓ, θ)-pseudo-cliques by iterating over any global ordering of vertices (unlike cliques or average-degree-based quasi-cliques). So we rely on the following idea to solve our problem.

Idea 1. *To find all maximal (ℓ,θ)-pseudo-cliques in G, extend the current subgraph (initially, an empty graph) of G by adding one vertex with the current subgraph at a time (Figure 2) in MD ordering as long as the current subgraph remains a θ-pseudo-clique and finally, report it when it becomes a maximal (ℓ,θ)-pseudo-clique.*

Since there are $O(2^{|V|})$ subgraphs of a graph $G=(V,E)$, the search space of our algorithm is exponential. Below, we put forth some properties of pseudo-cliques that may help us to prune our search space. To save space, we provide all proofs in Appendix B.

The following two lemmas give two different bounds on the maximum possible order of a (ℓ,θ)-pseudo-clique – which are then used to establish our Pruning Technique 1 stated below.

Lemma 2. *If S is a (ℓ,θ)-pseudo-clique then $\xi_S \geq \lceil \theta|S|/2 \rceil$*

Corollary 1. *If S is a θ-pseudo-clique then $|S| \leq \lfloor 2\xi_G/\theta \rfloor$*

Lemma 3. *If S is a θ-pseudo-clique, the largest clique in $G[S]$ is of order ω, and $\theta > (\omega-1)/\omega$ then $|S| \leq \left\lfloor \frac{1}{1-(\omega-1)/\omega\theta} \right\rfloor$*

Corollary 2. *If S is a θ-pseudo-clique with $\theta > \xi_G/(\xi_G+1)$ then*

$$|S| \leq \left\lfloor \frac{1}{1-\xi_G/(\xi_G+1)\theta} \right\rfloor$$

Pruning Technique 1 (Order Bound). *No (ℓ,θ)-pseudo-clique exists if $\ell > \mu$ where μ is the minimum of the two upper-bounds (on the order of a (ℓ,θ)-pseudo-clique) mentioned in Corollaries 1 and 2.*

We also realize that each (ℓ,θ)-pseudo-clique must contain a certain r-clique (Theorem 1) and such cliques must be contained within a certain r'-core of the input graph (Lemma 4). These results are then exploited in our Pruning Technique 2 stated below.

Theorem 1. *A θ-pseudo-clique S with ℓ vertices must contain an r-clique K where $r = \lceil 1/(1-\theta(\ell-1)/\ell) \rceil$.*

Lemma 4. *If G contains a t-clique K, then $K \subseteq C_{t'}[G]$ where $t' = min\{k : k \geq t-1 \text{ and } |C_k| > 0\}$.*

Idea 2. *All r-cliques in G can be computed by enumerating all $(r,1)$-pseudo-cliques in $G[C_{r'}]$ via Idea 1.*

Pruning Technique 2 (Turan Filtering). *Find all r-cliques in G via Idea 2 and extend them to get all maximal (ℓ,θ)-pseudo-cliques.*

5 Method

Our algorithm is a modified version of the PCE algorithm [45], which we briefly introduce below for the reader's convenience.

PCE Algorithm: PCE recursively grows θ-pseudo-cliques. The following holds for the corresponding recursion tree [45].

Property 1. *$Q = P \cup \{u\}$ is a child of a θ-pseudo-clique P if and only if (i) Q is a θ-pseudo-clique and (ii) $u = v^*(Q)$.*

From this, we have derived the following corollary.

Corollary 3. *A node-set $Q = P \cup \{u\}$ is a child of a θ-pseudo-clique P if and only if (i) Q is a θ-pseudo-clique and (ii) $P_{\succ}$ is a prefix of $Q_{\succ}$ where $\succ$ is the MD ordering of Q.*

It implies that PCE explores pseudo-cliques using Idea 1. Note that Q has only one parent $(Q - \{v^*(Q)\})$ since $v^*(Q)$ is unique for Q. This property ensures the tree structure of the search-space of PCE. PCE traverses this tree (*a.k.a., reverse search tree* [3]) in a DFS manner to visit all θ-pseudo-cliques. It starts from each vertex and continues growing the current θ-pseudo-clique P until a maximal θ-pseudo-clique is found (Figure 2). $P \cup \{u\}$ becomes a child of P if it satisfies the conditions of Property 1. These conditions are equivalent to the following three conditions [45].

Lemma 5. *If P is a θ-pseudo-clique and node $u \notin P$ then $P \cup \{u\}$ is a child of P if and only if the following conditions hold: (i) $d_P(u) \geq \theta\binom{|P|+1}{2} - E[P]$, (ii) $(d_P(u),u) < (\delta(P)+1, v^*(P))$), (iii) $v \prec u \Rightarrow v \in N(u)\ \forall v \in P$ where $\succ$ is the degree ordering of $G[P]$.*

Corollary 4. *If $P, P \cup \{u\}$ are θ-pseudo-cliques and $P \cup \{u\}$ is a child of P then $\theta\binom{|P|+1}{2} - E[P] \leq \delta(P)+1$*

Our Algorithm: Our approach, namely FPCE, is a significant modification of the PCE algorithm. Specifically, we added some pruning techniques with PCE to reduce its search space. We discussed all but one of these techniques in Section 4. The last one is based on the following lemmas (see proofs in Appendix B).

Lemma 6. *If S, P are θ-pseudo-cliques such that $|S| = \ell$ and S is a descendant of P then $\theta\binom{\ell}{2} \leq E[P] + (\ell - |P|)\,(\delta(P) + (\ell - |P| + 1)/2)$.*

Lemma 7. *Let P, S are θ-pseudo-cliques, $|S| = \ell$, $\tau_P = \delta(P) + \frac{1}{2}$, $\eta(P) = min\{c(P), \delta(P) + \ell - |P|\}$, and P is an ancestor of S in the reverse search tree, then*

$$\theta\binom{\ell}{2} \leq \begin{cases} E[P] + (\ell - |P|)\,\delta(P), & \text{if } c(P) = \delta(P) \\ E[P] + (\ell - P + \tau_P)\eta(P) - (\delta(P)+1)\tau_P, & \text{otherwise} \end{cases}$$

Pruning Technique 3 (Edge Bound). *If a θ-pseudo-clique P does not satisfy the inequalities in Lemma 6 or Lemma 7 then it cannot be extended to obtain a θ-pseudo-clique of order ℓ.*

Algorithm 1 illustrates our overall idea. To compute core numbers of nodes in the input graph G, we implement an O(E) time algorithm [5] (line 1) that iteratively deletes the first node from a sorted (in ascending order of degrees) list L of nodes. Whenever a node is deleted, its core number is set to its current degree, its neighbors' degrees are updated, and they are re-positioned in L to maintain their degree-wise ordering. We use these core numbers to apply the *Order Bound* (lines 2 – 4) and to compute $C_{r'}[G]$ (line 6). We call a recursive method named *recur* on each node of $C_{r'}[G]$. It takes a pseudo-clique P and either (i) outputs P (if it is a maximal pseudo-clique) or (ii) extends P to get each child $Q = P \cup \{u\}$ of P (otherwise) – where Q is either a clique (when $|P| < r$; lines 12 – 14) or a pseudo-clique (when $|P| \geq r$; line 20). In case (ii), *recur* calls itself on each child Q (lines 21 – 23).

We design an algorithm called *getChildrenClq* to generate each clique $P \cup \{u\}$ from a clique P. It relies on the following property.

Property 2. *$P \cup \{u\}$ is a clique having at least two nodes u, v if and only if P is a clique, $u \in N(v)$, and $P - \{v\} \cup \{u\}$ is a clique.*

This property allows us to exploit the already known child-cliques of $P - \{v\}$ in a reverse-search tree. We use a compressed bitset [10] to store $N(v)$ for each node v, which lets us check conditions in this Property in $O(lg\Delta)$ time to find each child of P.

We rely on the *getChildrenPseudoClq* method of PCE (line 20) to generate children of the current pseudo-clique P. Briefly, it returns nodes satisfying all conditions of Lemma 5.

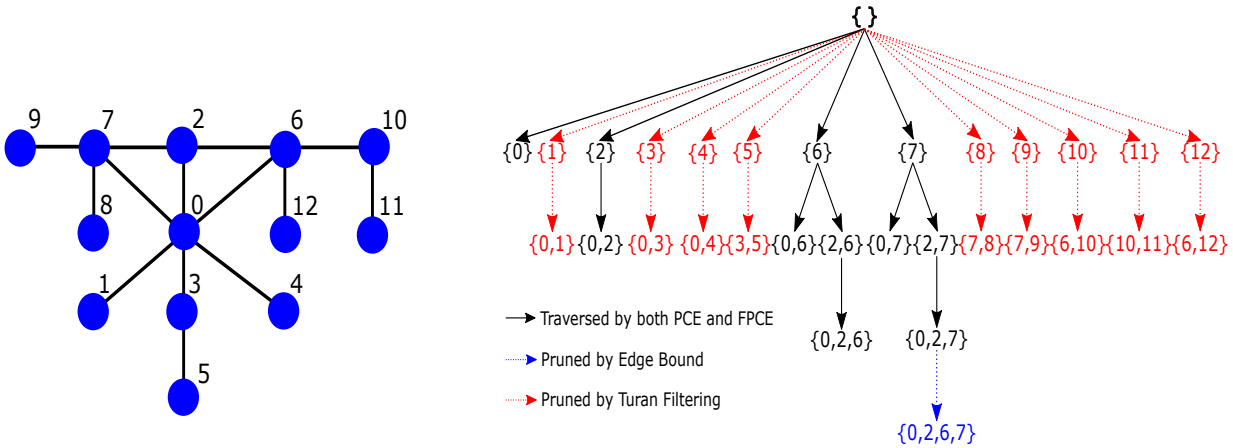

Figure 2: A graph (left) and the reverse search tree (right) traversed by PCE and FPCE to search (in vain) for all maximal $(5, 0.8)$-pseudo-cliques in it. This figure shows that FPCE significantly reduces the search space of PCE even for this small graph.

Figure 2 shows a reverse-search tree depicting *recur* calls on 0.8-pseudo-cliques to search for maximal $(5, 0.8)$-pseudo-cliques in a graph. Here $r = 3$, so FPCE only searches for 3-cliques in the 2-core ($\{0, 2, 6, 7\}$) of this graph, thereby pruning nodes outside that core (Turan Filtering). It also detects that no child of $\{0, 2, 6\}$ or $\{0, 2, 7\}$ can exist and thereby saves time by skipping *getChildrenPseudoClq* calls on the corresponding tree-nodes (Edge Bound).

Time Complexity: We put a detailed time complexity analysis in Appendix C. Briefly, FPCE time mainly depends on the time taken by *recur* calls. The bottleneck of *recur* is *getChildrenPseudoClq*, which takes $O(\Delta lgV + min\{\Delta^2, V + E\})$ time, as per [45]. We exploit properties of pseudo-cliques to derive a tighter time complexity of $O(\Delta(lgV + \xi_G))$ for it. This is also the running time of *recur*. Since *recur* is called once for each pseudo-clique, the time complexity of FPCE is $O(2^{|V|}\Delta(lgV + \xi_G))$, but typically it takes much less time.

Time Reduction: Although, time complexity of FPCE is the same as PCE, FPCE often prunes many branches of the reverse-search tree. If it prunes k branches or detects k nodes to be leaves (for e.g. in Figure 2, Edge Bound detects $\{0, 2, 6\}$ to be a leaf), it saves $O(k\Delta(lgV + \xi_G))$ time. When $|P| \leq r$, it saves time by calling *getChildrenClq* instead of slower *getChildrenPseudoClq*. When Order Bound holds, it prunes the entire search space. So, FPCE should be faster than PCE – which is evident from our results, too (Section 8).

6 Datasets

Synthetic Datasets: We used Python Networkx library to create two types of undirected graphs: (i) Scale-free (SF) graphs and (ii) Small-world (SW) graphs because real graphs tend to have these types of structures [48]. To generate SF graphs, we used a slightly modified version of Barabási–Albert (BA) preferential attachment model [4] with parameters (n, m). Starting with an initial graph (by default, a star graph of m+1 nodes), it incrementally adds new nodes one by one. A new node u is connected with k pre-existing nodes in such a way that the probability of a pre-existing node v being a neighbor of u is proportional to $d(v)$. In the BA model, $k = m$, making it prone to generate graphs with an unrealistically high minimum degree. To fix this issue, we modified the BA model so that it chooses k uniformly at random from $[1, m]$ in each iteration.

Input: Graph G, minimum order ℓ, minimum density θ
Output: All maximal (ℓ, θ)-pseudo-cliques in G

```
1  c = getCoreNumbers(G)  //array of core numbers
2  ξ = max(c)  //compute ξ_G
3  μ = orderBound(θ, ξ)
4  if ℓ > μ then return  //Pruning Tech. 1
5  r = ⌈1/(1 − θ(ℓ − 1)/ℓ)⌉
6  C_r' = getCore(G, c, r)
7  forall v ∈ C_r' do
8      recur(G, ∞, {v}, v, {})
9
10 Procedure recur (G, cp, P, v, A)
11     cp = min(cp, c[v]) //compute c(P)
12     if |P| < r then
13         D = getChildrenClq(G, P, v, A)
14         Go to line 21  //Pruning Tech. 2
15     if P is a maximal (ℓ, θ)-pseudo-clique then
16         output P
17         return
18     if edgeBound(ℓ, cp, P) then return //Pruning Tech. 3
19     if θ(C(|P|+1, 2)) − E[P] > δ(P) + 1 then return //Corollary 4
20     D = getChildrenPseudoClq(G, P, v)
21     forall u ∈ D do
22         Q = P ∪ {u}
23         recur(G, cp, Q, u, D)
24
25 Procedure getChildrenClq (G, P, v, A)
26     if |P| = 1 then return {u : u < v and u ∈ N(v)}
27     else return ∪_{u∈A}{u : u < v and u ∈ N(v)}
```

Algorithm 1: FPCE(G, ℓ, θ) algorithm.

To generate SW graphs, we used Watts–Strogatz (WS) (n, m, p) model [49]. Starting with a cycle graph of n nodes, it connects each node with its m nearest neighbors to form a *ring lattice*. Then with probability p, it replaces each edge (u, v) in that lattice with another edge (u, w) where w is chosen uniformly at random from all nodes while avoiding the creation of any self-loop/multi-edge.

We created eight instances of each of these two types of graphs by varying parameter m in $\{5, 10, 15, \ldots, 40\}$. We used $n = 100,000$ to get sufficiently large graphs. For the WS model, we chose $p = 0.2$.

Table 2: Properties of synthetic graphs.

	Scale Free (SF) Graphs			Small World (SW) Graphs		
m	$\|E\|$	$den(G)$	ξ_G	$\|E\|$	$den(G)$	ξ_G
5	299,799	5.99e-5	5	200,000	4.00e-5	3
10	550,036	1.10e-4	8	500,000	1.00e-4	7
15	798,375	1.59e-4	11	700,000	1.40e-4	10
20	1,046,002	2.09e-4	15	1,000,000	2.00e-4	15
25	1,300,746	2.60e-4	18	1,200,000	2.40e-4	18
30	1,549,292	3.09e-4	22	1,500,000	3.00e-4	24
35	1,802,411	3.60e-4	25	1,700,000	3.40e-4	28
40	2,047,534	4.09e-4	28	2,000,000	4.00e-4	33

Real Datasets: We collected 18 real graphs from [40] and removed self-loops and multi-edges from them. Last 3 of them are dense and the rest are sparse (Table 3). We mostly considered sparse graphs since organically formed networks tend to be sparse [48].

Table 3: Features of real graphs. Starred ones are dense.

ID	Graph (G) Name	$\|V\|$	$\|E\|$	$den(G)$	ξ_G
1	bio-grid-human	9,436	31,181	7.00e−4	12
2	web-webbase-2001	16,062	25,592	1.98e−4	32
3	scc_retweet-crawl	17,151	24,014	1.63e−4	19
4	soc-gplus	23,628	39,193	1.40e−4	12
5	dictionary28	39,327	89,037	1.15e−4	25
6	tech-internet-as	40,164	85,122	1.05e−4	23
7	PROTEINS-full	43,466	81,043	8.57e−5	4
8	soc-douban	154,908	327,161	2.72e−5	15
9	citationCiteseer	268,495	1,156,646	3.20e−5	15
10	ca-MathSciNet	332,689	820,643	1.48e−5	24
11	com-amazon	334,863	925,871	1.65e−5	6
12	TWITTER-Partial	580,768	717,557	4.25e−6	9
13	delaunay_n23	8,388,608	25,165,783	7.15e−7	4
14	inf-europe_osm	50,912,018	54,054,659	4.17e−8	3
15	socfb-konect	59,216,215	92,522,017	5.27e−8	16
16	bn-mouse-brain*	213	16,242	7.19e−1	112
17	school-proximity*	242	8,317	2.85e−1	47
18	p-hat700-3*	700	183,010	7.48e−1	426

7 Experiments

Investigation: We compare the performance of FPCE with two exact algorithms: PCE and ODES. These algorithms were implemented in C/CPP – just like our FPCE code. ODES computes connected pseudo-cliques only. So, to fairly compare all algorithms, we modified PCE and FPCE codes so that they only compute connected pseudo-cliques. We achieved this goal by noting that, for connected pseudo-cliques, condition (i) of Lemma 5 for $P \cup \{v\}$ being a child of P becomes: $d_P(v) \geq max\left(\theta\binom{|P|+1}{2} - E[P], 1\right)$ while the other two conditions remain the same. We downloaded the PCE code from https://research.nii.ac.jp/~uno/codes.htm and collected a single-threaded implementation of ODES from [38]. To show the effectiveness of our algorithm for finding large dense subgraphs, we executed each algorithm with parameters: $\ell = 10, \theta = 0.9$.

Evaluation: We define following performance metrics for FPCE.
(i) Speedup w.r.t. algorithm $A = \frac{\text{Execution Time of A}}{\text{Execution Time of FPCE}}$
(ii) Call reduction = $N_{PCE} - N_{FPCE}$

Here A denotes PCE or ODES algorithm and N_{PCE} (resp., N_{FPCE}) denotes the number of *recur* calls made by PCE (resp., FPCE).

Experimental Setup: We conducted our experiments on a workstation with AMD Ryzen 9 5900X 3.7 GHz 12 Core 24 thread CPU and 64 GB RAM running on a 64-bit Ubuntu 20.04 LTS O/S. To ensure fair comparison among algorithms, we compiled all codes using the same GCC flag, turned off the 'Core Performance Boost' and 'Frequency Scaling' features, and fixed all processors' frequencies to 3.7 GHz. We implemented FPCE in C and wrote Python scripts to preprocess graphs and post-process results. Our code/data are shared on our suppl. website: https://github.com/ahsanur-research/FPCE

8 Results

(i) *No algorithm finished executing any of our three dense graphs* within five days, even though all of these graphs are quite small. So, our results are based on the remaining 15 graphs.

(ii) *FPCE time tends to increase with the number of pseudo-cliques and with the order of a graph* (Figures 3 and 4) – which is expected from its time complexity (Section 5).

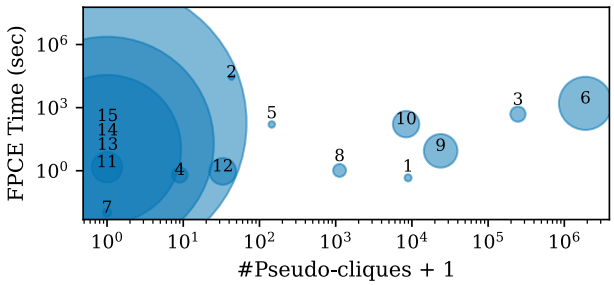

Figure 3: Bubble chart showing how FPCE time depends on the number of pseudo-cliques in real graphs. Disk labels are graph IDs (Table 3). Disk area varies with call reduction.

(iii) *FPCE makes a lot fewer recursive calls than PCE for all graphs, especially for large graphs.* Increasing m increases both the size and density of synthetic graphs (Table 2) and exponentially increases both the number of pseudo-cliques and the time taken by FPCE to enumerate them. But call reduction also increases almost exponentially with m (Figure 4) because larger graphs tend to contain many more *unpromising subgraphs*: subgraphs that can never be extended to any (ℓ, θ)-pseudo-cliques. Similarly, real graphs 13, 14, and 15 (respectively, graphs 1 – 5) exhibit much larger (resp., smaller) call reduction than others (Figure 3). These results indicate that FPCE is more effective than PCE on larger graphs.

(iv) *FPCE is always faster than PCE and ODES* (Figure 5). For real graphs, FPCE is, on average, 38.6 and 6.5 times faster than ODES and PCE, respectively. For synthetic graphs, FPCE is, on average, 39.7 and 3.1 times faster than ODES and PCE, respectively. Among synthetic graphs, SW graphs yield higher speedup (of FPCE w.r.t. PCE) than SF graphs. The average speedup of FPCE w.r.t. PCE on SW graphs (resp., on SF graphs) is 4.23 (respectively, 1.88). This is expected as SW graphs exhibit higher call reduction (Figure 4).

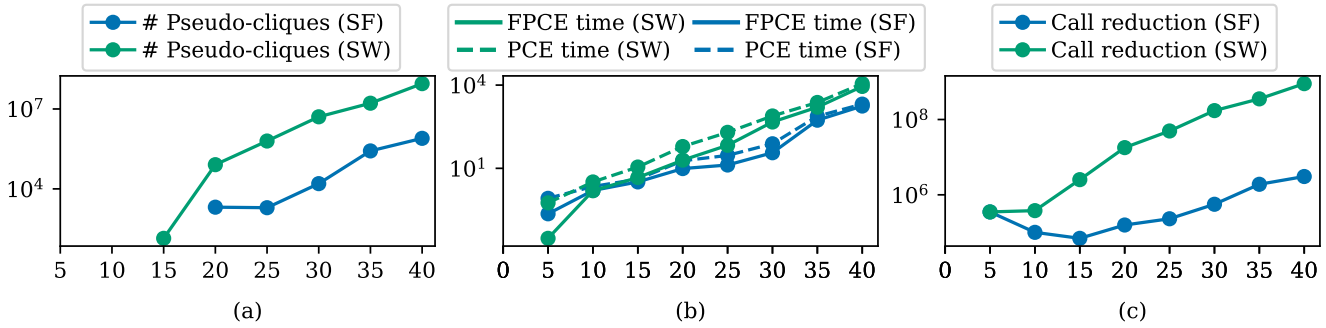

Figure 4: (a) Number of pseudo-cliques, (b) PCE and FPCE execution times (in sec), and (c) Call reduction of FPCE for Small World (SW) and Scale Free (SF) graphs. Here, X-axis denotes values of m used to generate these synthetic graphs (see Section 6).

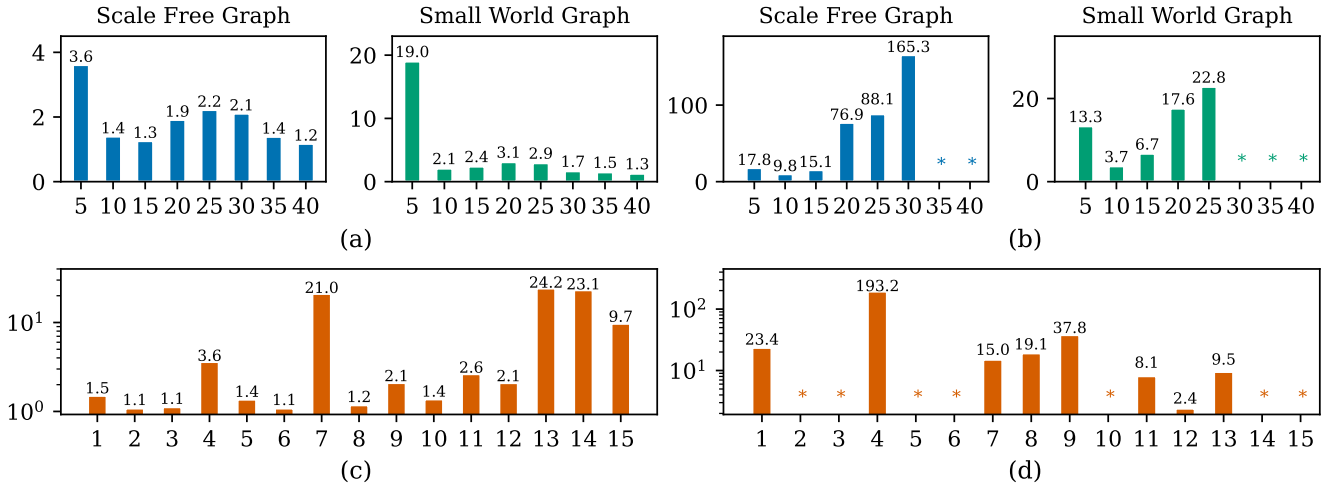

Figure 5: Speedup of FPCE w.r.t. (a) PCE and (b) ODES for synthetic graphs. Speedup of FPCE w.r.t. (c) PCE and (d) ODES for real graphs. X-axis labels in (a) – (b) denote values of m used for synthetic graphs (see Section 6) and those in (c) – (d) indicate real graphs' IDs (see Table 3). Asterisks in (b) and (d) indicate real graphs on which ODES terminated prematurely.

(v) *ODES is the worst of all.* It terminates prematurely (showing a memory overflow error) on many graphs and when it works, it always takes much longer time than both PCE and FPCE (Figures 4 (b) and (d)). So we focus on comparing FPCE with PCE from now.

(vi) *Among graphs with a similar number of pseudo-cliques, sparser ones tend to yield higher call reduction and speedup w.r.t PCE.* Real graphs in each of the following groups have a similar number of pseudo-cliques: $\{1, 10\}$, $\{2, 12\}$, $\{7, 11, 13, 14, 15\}$. In terms of density: $1 > 10, 2 > 12, 7 > 11 > 13 > 14 \approx 15$. In terms of call reduction (resp., speedup): $1 < 10, 2 < 12, 7 < 11 < 13 < 14 < 15$ (resp., $1 \approx 10, 2 < 12, 7 > 11 < 13 \approx 14 > 15$). Among synthetic graphs without pseudo-cliques (SW graphs for m = 5, 10 and SF graphs for m = 5, 10, 15), the denser the graph is, the lower the speedup and call reduction of FPCE is. The same observation holds for SW graph m = 25 and SF graph m = 40, which have a similar number of pseudo-cliques (Tables 2 and 3, Figures 3– 5). FPCE's call reduction and speedup exhibit this behavior because if a subgraph is not embedded in a dense enough supergraph, then FPCE identifies and prunes it as it is an unpromising subgraph.

(vii) FPCE's speedup w.r.t. PCE mostly depends on call reduction, *i.e.,* it is mainly the effect of our pruning techniques (Appendix A).

9 Ablation Study

When order-bound's condition is met (line 4, Algorithm 1), no (ℓ, θ)-pseudo-clique can exist – which FPCE reports and terminates early. Thus, we know that order-bound holds on real graphs 7,13,14, and SW graph m=5. So, we see a high speedup (19 – 24.2) of FPCE w.r.t. PCE on these graphs (Figures 5(a),(c)). Since order-bound prunes the entire search tree, FPCE skips the other two pruning techniques, acting as *FPCE-with-Order-Bound-only* on these four graphs.

We created a version of FPCE called *FPCE-EB* which lacks Edge Bound. We found that on our graphs other than these four graphs:

- *FPCE-EB* is faster than PCE on almost all these graphs, having an average speedup (w.r.t. PCE) of 2.2 and 1.6 on these real and synthetic graphs, respectively. It is the effect of Turan Filtering alone as order-bound didn't hold on these graphs.
- FPCE is faster than FPCE-EB on almost all these graphs, with an average speedup (w.r.t. FPCE-EB) of 1.1 and 1.25 on these real and synthetic graphs, respectively, *i.e.,* adding Edge Bound with FPCE-EB yields even more speedup.

These results imply that all pruning techniques of FPCE positively contribute to its efficiency.

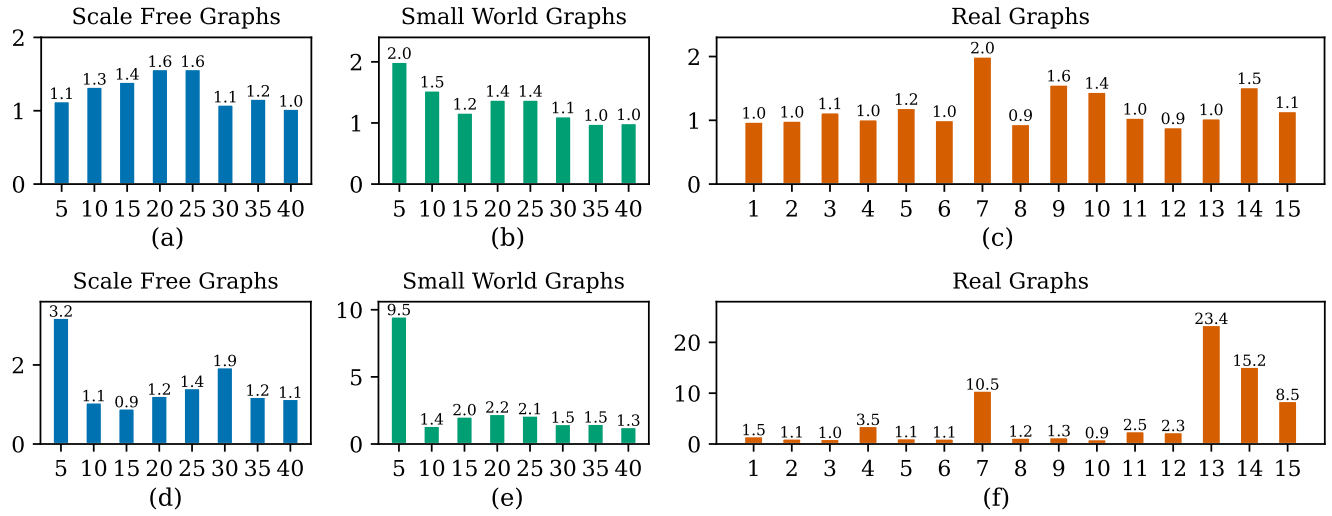

Figure 6: Speedup of FPCE w.r.t. FPCE-EB for (a) – (b) synthetic and (c) real graphs (top figure) and speedup of FPCE-EB w.r.t PCE for (d) – (e) synthetic and (f) real graphs (bottom figure).

10 Application and Comparison

We found a popular heuristic[1] method called ClusterONE [33] (henceforth called CL1) that computes pseudo-cliques in PPI networks to identify protein complexes. So, we applied both CL1 and FPCE on a PPI graph to see if the exact computation of pseudo-cliques give any extra insights. We also applied an exact algorithm called *FastQC* [51] to compute degree-based quasi-cliques (henceforth *DQCs*) in that graph using the same parameters as FPCE. An (ℓ, θ)-DQC is a connected subgraph $G[P]$ s.t. $|P| \geq \ell$ and $\delta(P) \geq \theta(|P|-1)$ *i.e.,* it is always an (ℓ, θ)-pseudo-clique but the opposite may not hold. We did not compare pseudo-cliques with other types of dense subgraphs because there is no such correspondence between their parameters and parameters of pseudo-cliques.

Dataset: To form a PPI network, we combined two PPI datasets on *Escherichia coli* from [2] and [39]. We removed multi-edges and self-loops from it. The resultant graph has 2119 nodes and 3778 edges. To prepare a gold standard, we downloaded known *E. coli* protein complexes from ECOCYC[2]. Since it is easier to find very small complexes, we only kept complexes with at least 5 proteins. There are 28 such complexes. These form our gold-standard dataset.

Investigation: We applied FPCE and CL1 (respectively, FastQC) on our PPI network to find all maximal $(5, \theta)$-pseudo-cliques (respectively, $(5, \theta)$-DQCs) with $\theta \in \{0.9, 0.8, 0.7, 0.6\}$.

Evaluation: We evaluate FPCE and CL1 based on:
(i) Geometric Accuracy: *Geometric Accuracy, a.k.a. Accuracy* [33] is a popular metric in protein complex prediction research. To define it, let C_i be the i-th complex in a gold standard, n be the number of such complexes, P_j be the j-th complex predicted (which is a pseudo-clique/DQC in our experiment) by an algorithm, and m be the number of such complexes. Then *accuracy* is the geometric mean of *Sensitivity (SNS)* and *Positive Predictive Value (PPV)* , where

$$SNS = \frac{\sum_{i=1}^{n} max_j\left(|C_i \cap P_j|\right)}{\sum_{i=1}^{n}|C_i|} \text{ and } PPV = \frac{\sum_{j=1}^{m} max_i\left(|C_i \cap P_j|\right)}{\sum_{j=1}^{m}\sum_{i=1}^{n}|C_i \cap P_j|}$$

(ii) Matching Complexes: If a pseudo-clique/DQC shares at least λ proteins with a known complex, we say that the *complex matches* that pseudo-clique/DQC. We find complexes matching with pseudo-cliques/DQCs (computed using FPCE/CL1/FastQC) for $\lambda \in \{1, 3\}$.

(iii) Biological Significance: As protein complex databases are incomplete, a predicted complex not matching any known complex may still be a valid complex or, at least, may form parts of the same cellular component. So, researchers often estimate the biological significance of their predicted complexes by Gene Ontology[3] (GO) term[4] *enrichment* via Fisher's exact test, which takes a *population* of genes/proteins (in our case, all proteins in our PPI network) and a study set (in our case, a pseudo-clique/DQC) to compute a p-value for each GO term. If a term has a low p-value, that means the fraction of proteins annotated with that term in the study set is significantly (statistically) higher than the fraction of proteins annotated with that term in the population. In such a case, we say that the study set is *enriched* in that term [19]. But this test is not meaningful/applicable to our pseudo-cliques/DQCs directly, due to their small sizes and high overlaps among them [27, 42, 52]. So, we combine pseudo-cliques/DQCs in the following way. We build a graph whose nodes represent pseudo-cliques/DQCs and place an edge between two nodes if their corresponding pseudo-cliques/DQCs overlap. We merge the pseudo-cliques/DQCs in each connected component of this graph to get some *clusters*. We use GOATOOLS [19] to find *GO Cellular Component* (CC) terms that are enriched in those clusters.

Results: (i) *FPCE outperforms both CL1 and FastQC* in terms of SNS, PPV, and accuracy for almost all values of θ (Figure 7(a)).
(ii) *All algorithms executed in < 1 sec for each density threshold (θ).*

[1]Another heuristic was proposed to compute pseudo-cliques [1], but its code is not publicly available and its authors could not provide it when we contacted them.
[2]https://ecocyc.org/group?id=:ALL-PROTEINS-2&orgid=ECOLI
[3]https://geneontology.org/
[4]A GO term denotes a function/component in a cell shared by a set of genes/proteins.

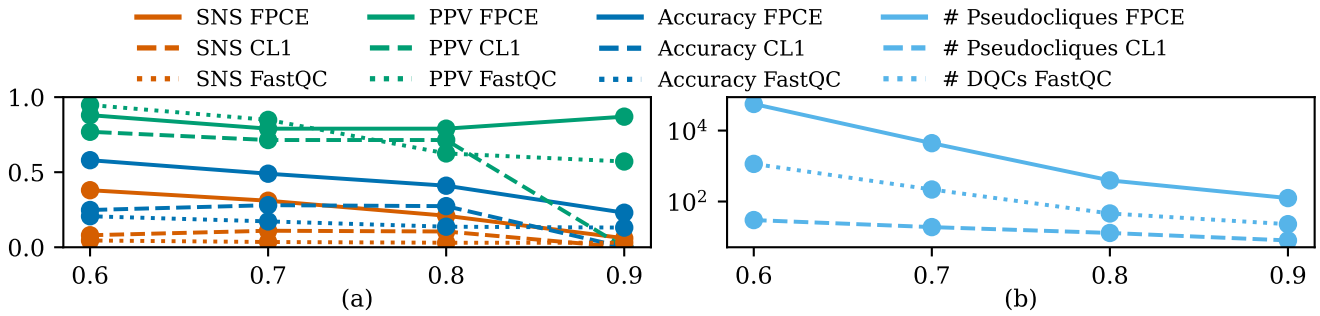

Figure 7: (a) Performance comparison between FPCE vs. ClusterONE (denoted CL1) and FastQC on protein-complex prediction and (b) number of $(5, \theta)$-pseudo-cliques/DQCs computed by them for this purpose. In both figures, X-axis indicates values of θ.

Table 4: Protein complexes that match (for $\lambda = 3$) with a FPCE pseudo-clique but do not match with any DQCs (starred ones match with neither any DQCs nor any CL1-pseudo-cliques). Results for $\lambda = 1$ are put in suppl. website to save space.

Complex Name	**Complex Members**, C	$\|C\|$	**Best Matched Pseudo-clique**, P	$\|P\|$	$P \cap C$	$\|P \cap C\|$
ATP synthase F1 complex*	atpA, atpC, atpD, atpG, atpH	5	atpA, atpC, atpD, atpG, atpH	5	atpA, atpC, atpD, atpG, atpH	5
degradosome*	eno, pnp, ppk, rhlB, rne	5	pnp, rhlB, rluB, rne, srmB, yfgB	6	pnp, rhlB, rne	3
ATP synthase Fo complex*	atpA, atpB, atpC, atpD,atpE, atpF, atpG, atpH	8	atpA, atpC, atpD, atpE, atpG	5	atpA, atpC, atpD, atpE, atpG	5
DNA polymerase III, holoenzyme	dnaE, dnaN, dnaQ, dnaX, holA, holB, …, holE	9	dnaE, dnaN, dnaQ, dnaX, holA, holB, holC, holD	8	dnaE, dnaN, dnaQ, dnaX, holA, …, holD	8
hydrogenase 3	hycB, hycC …, hycG	6	fdhF, hycB, hycE, hycF, hycG	5	hycB, hycG	2
NADH:quinone oxidoreductase I	nuoA, nuoB, nuoC, nuoE, nuoF …, nuoN	13	nuoJ, nuoK, nuoL, nuoM, nuoN	5	nuoJ, nuoK, nuoL, nuoM, nuoN	5
formate hydrogenlyase complex	fdhF, hycB, hycC, hycD, hycE, hycF, hycG	7	fdhF, hycB, hycE, hycF, hycG	5	fdhF, hycB, hycE, hycF, hycG	5

(iii) *Number of pseudo-cliques decreases with the increase of θ and as such SNS and accuracy also decreases.* We chose $\theta = 0.8$ for our next analyses since it gives high enough accuracy (0.41) and decreasing θ further increases accuracy a little bit (0.49 for $\theta = 0.7$) but results in many more pseudo-cliques (Figure 7 (b)).

(iv) For $\lambda = 3$, only 4 and 0 (respectively, 7) known complexes match with some CL1-pseudo-cliques and FastQC-DQCs (resp., FPCE pseudo-cliques), respectively. If we lower λ to 1, then 7 and 5 (resp., 11) complexes match with some CL1 pseudo-cliques and DQCs, respectively (resp., FPCE pseudo-cliques). For both values of λ (resp., for $\lambda = 3$), each complex that matches with some CL1-pseudo-cliques (resp., DQCs), also matches with some FPCE pseudo-cliques (for $\lambda = 1$, two complexes exist that match with some DQCs but not with any FPCE pseudo-cliques – which may be the effect of the small value of λ). These results imply that *CL1/FastQC is less useful compared to FPCE alone* here. For $\lambda = 3$ (resp., for $\lambda = 1$), we found 3 (resp., 4) complexes that match with some FPCE pseudo-cliques but do not match with any CL1 pseudo-cliques (starred complexes in Table 4), which implies that *FPCE may unearth complexes missed by a heuristic pseudo-clique miner.* Similarly, For $\lambda = 3$ (resp., for $\lambda = 1$), we found 7 (resp., 8) complexes that match with some FPCE pseudo-cliques but do not match with any DQCs (Table 4), which implies that *FPCE may unearth complexes that cannot be found by computing DQCs.*

(v) We got 18, 13, and 8 clusters by merging overlapping pseudo-cliques/DQCs got from FPCE, CL1, and FastQC, respectively. Among those, 13 FPCE-derived clusters, 8 CL1-derived clusters, and 6 FastQC-derived clusters are enriched in some GO CC terms. FPCE, CL1, and FastQC-derived clusters are enriched in 64, 42 and 35 unique GO CC terms, respectively. Among these terms, 28 (resp., only 6) are enriched in some FPCE clusters (resp, CL1 clusters) but are not enriched in any CL1 (resp. FPCE) clusters; whereas 40 (resp., only 11) terms are enriched in some FPCE clusters (resp, FastQC clusters) but are not enriched in any FastQC (resp., FPCE) clusters. All these results indicate that *FPCE outputs are biologically more interesting than those of CL1/FastQC.*

11 Conclusion and Future Directions

We present an exact algorithm to compute pseudo-cliques in a graph. We show that it is the fastest one on our synthetic and real graphs. But no exact algorithm (including ours) could enumerate pseudo-cliques from dense graphs in a reasonable time. Other pruning techniques may need to be devised to achieve that goal. More speedup can be achieved via parallelization. Our ideas can also be extended to find pseudo-cliques in weighted graphs.

Acknowledgments

This work was funded by NSU grant CTRG-20/SEPS/19. We thank Osama Nadeem, Shahriar Kamal, Khan Asfi Reza for helping us.

References

[1] James Abello, Mauricio G.C. Resende, and Sandra Sudarsky. 2002. Massive quasi-clique detection. In *Lecture Notes in Computer Science*, Vol. 2286. Springer, 598–612.

[2] Mohammad Arifuzzaman, Maki Maeda, Aya Itoh, Kensaku Nishikata, Chiharu Takita, Rintaro Saito, Takeshi Ara, Kenji Nakahigashi, Hsuan-Cheng Huang, Aki Hirai, et al. 2006. Large-scale identification of protein–protein interaction of Escherichia coli K-12. *Genome research* 16, 5 (2006), 686–691.

[3] David Avis and Komei Fukuda. 1996. Reverse search for enumeration. *Discrete Applied Mathematics* 65, 1-3 (1996), 21–46.

[4] Albert László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* (1999).

[5] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *CoRR* cs.DS/0310049 (2003).

[6] Malay Bhattacharyya and Sanghamitra Bandyopadhyay. 2009. Mining the largest quasi-clique in human protein interactome. In *Proceedings of the 2009 International Conference on Adaptive and Intelligent Systems, ICAIS 2009*. IEEE, 194–199.

[7] Francesco Bonchi and Claudio Lucchese. 2005. Pushing tougher constraints in frequent pattern mining. In *Lecture Notes in Computer Science*. Springer, 114–124.

[8] Zvika Brakerski and Boaz Patt-Shamir. 2011. Distributed discovery of large near-cliques. *Distributed Computing* 24 (2011), 79–89.

[9] Mauro Brunato, Holger H. Hoos, and Roberto Battiti. 2008. On effectively finding maximal quasi-cliques in graphs. In *Lecture Notes in Computer Science*, Vol. 5313 LNCS. 41–55.

[10] Samy Chambi, Daniel Lemire, Owen Kaser, and Robert Godin. 2016. Better bitmap performance with roaring bitmaps. *Software: practice and experience* 46, 5 (2016), 709–719.

[11] Jiejiang Chen, Shaowei Cai, Shiwei Pan, Yiyuan Wang, Qingwei Lin, Mengyu Zhao, and Minghao Yin. 2021. NuQClq: An effective local search algorithm for maximum quasi-clique problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 12258–12266.

[12] Jie Chen and Yousef Saad. 2012. Dense Subgraph Extraction with Application to Community Detection. *IEEE Transactions on Knowledge and Data Engineering* 24, 7 (2012), 1216–1230.

[13] Serena Dotolo, Anna Marabotti, Angelo Facchiano, and Roberto Tagliaferri. 2020. A review on drug repurposing applicable to COVID-19. *Briefings in Bioinformatics* 22, 2 (2020), 726–741.

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

[15] Mokhtarul Haque, Rosy Sarmah, and Dhruba K. Bhattacharyya. 2018. A common neighbor based technique to detect protein complexes in PPI networks. *Journal of Genetic Engineering and Biotechnology* 16, 1 (2018), 227–238.

[16] Steve Harenberg, Gonzalo Bello, L. Gjeltema, Stephen Ranshous, Jitendra Harlalka, Ramona Seay, Kanchana Padmanabhan, and Nagiza Samatova. 2014. Community detection in large-scale networks: A survey and empirical evaluation. , 426–439 pages.

[17] Shweta Jain and C. Seshadhri. 2017. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *Proceedings of the 26th International Conference on World Wide Web*. IW3C2, 441–449.

[18] Shweta Jain and C. Seshadhri. 2020. Provably and Efficiently Approximating Near-cliques using the Turán Shadow: PEANUTS. In *Proceedings of The Web Conference 2020 (WWW '20)*. 1966–1976.

[19] DV Klopfenstein, Liangsheng Zhang, Brent S Pedersen, Fidel Ramírez, Alex Warwick Vesztrocy, Aurélien Naldi, Christopher J Mungall, Jeffrey M Yunes, Olga Botvinnik, Mark Weigel, et al. 2018. GOATOOLS: A Python library for Gene Ontology analyses. *Scientific reports* 8, 1 (2018), 10872.

[20] Christian Komusiewicz and Manuel Sorge. 2015. An algorithmic framework for fixed-cardinality optimization in sparse graphs applied to dense subgraph problems. *Discrete Applied Mathematics* 193 (2015), 145–161.

[21] Christian Komusiewicz, Manuel Sorge, and Kolja Stahl. 2015. Finding Connected Subgraphs of Fixed Minimum Density: Implementation and Experiments. In *Proceedings of the 14th International Symposium on Experimental Algorithms - Volume 9125*. Springer-Verlag, 82–93.

[22] Georg Kustatscher, Martina Hödl, Edward Rullmann, Piotr Grabowski, Emmanuel Fiagbedzi, Anja Groth, and Juri Rappsilber. 2023. Higher-order modular regulation of the human proteome. *Molecular Systems Biology* 19, 5 (2023), e9503.

[23] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. 2010. A Survey of Algorithms for Dense Subgraph Discovery. Springer, 303–336.

[24] Guimei Liu and Limsoon Wong. 2008. Effective pruning techniques for mining quasi-cliques. In *Lecture Notes in Computer Science*, Vol. 5212 LNAI. Springer B.H., Berlin, Heidelberg, 33–49.

[25] Jiewei Liu, Ming Li, Xiong-Jian Luo, and Bing Su. 2018. Systems-level analysis of risk genes reveals the modular nature of schizophrenia. *Schizophrenia Research* 201 (2018), 261–269.

[26] James Long and Chris Hartman. 2010. ODES: An overlapping dense sub-graph algorithm. *Bioinformatics* 26, 21 (2010), 2788–2789.

[27] Weijun Luo, Michael S Friedman, Kerby Shedden, Kurt D Hankenson, and Peter J Woolf. 2009. GAGE: generally applicable gene set enrichment for pathway analysis. *BMC bioinformatics* 10 (2009), 1–17.

[28] Xiuli Ma, Guangyu Zhou, Jingbo Shang, Jingjing Wang, Jian Peng, and Jiawei Han. 2017. Detection of Complexes in Biological Networks Through Diversified Dense Subgraph Mining. *Journal of computational biology : a journal of computational molecular cell biology* 24 (2017), 923–941. Issue 9.

[29] Foad Mahdavi Pajouh, Zhuqi Miao, and Balabhaskar Balasundaram. 2014. A branch-and-bound approach for maximum quasi-cliques. *Annals of Operations Research* 216, 1 (2014), 145–161.

[30] Fabrizio Marinelli, Andrea Pizzuti, and Fabrizio Rossi. 2020. LP-based dual bounds for the maximum quasi-clique problem. *Discrete Applied Mathematics* (2020).

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

[32] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. 2015. Scalable Large Near-Clique Detection in Large-Scale Networks via Sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 815–824.

[33] Tamás Nepusz, Haiyuan Yu, and Alberto Paccanaro. 2012. Detecting overlapping protein complexes in protein-protein interaction networks. *Nature Methods* (2012).

[34] Grigory Pastukhov, Alexander Veremyev, Vladimir Boginski, and Oleg A Prokopyev. 2018. On maximum degree-based-quasi-clique problem: Complexity and exact approaches. *Networks* 71, 2 (2018), 136–152.

[35] Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. 2013. On the maximum quasi-clique problem. *Discrete Applied Mathematics* 161, 1-2 (2013), 244–257.

[36] Jian Pei, Jiawei Han, and Laks V.S. Lakshmanan. 2001. Mining frequent itemsets with convertible constraints. *Proceedings - International Conference on Data Engineering* (2001), 433–442,.

[37] Lu Qin, Rong-Hua Li, Lijun Chang, and Chengqi Zhang. 2015. Locally Densest Subgraph Discovery. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 965–974.

[38] Ahsanur Rahman, Steve T. K. Jan, Hyunju Kim, B. Aditya Prakash, and T. M. Murali. 2016. Unstable Communities in Network Ensembles. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 504–512.

[39] Seesandra V Rajagopala, Patricia Sikorski, Ashwani Kumar, Roberto Mosca, James Vlasblom, Roland Arnold, Jonathan Franca-Koh, Suman B Pakala, Sadhna Phanse, Arnaud Ceol, et al. 2014. The binary protein-protein interaction landscape of Escherichia coli. *Nature biotechnology* 32, 3 (2014), 285–290.

[40] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 4292 – 4293.

[41] Seyed Vahid Sanei-Mehri, Apurba Das, and Srikanta Tirthapura. 2018. Enumerating Top-k Quasi-Cliques. In *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*. IEEE, 1107–1112.

[42] Cedric Simillion, Robin Liechti, Heidi EL Lischer, Vassilios Ioannidis, and Rémy Bruggmann. 2017. Avoiding the pitfalls of gene set enrichment analysis with SetRank. *BMC bioinformatics* 18, 1 (2017), 1–14.

[43] Shu Tadaka and Kengo Kinoshita. 2016. NCMine: Core-peripheral based functional module detection using near-clique mining. *Bioinformatics* 32, 22 (2016), 3454–3460.

[44] 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 *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 104–112.

[45] Takeaki Uno. 2010. An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica (New York)* 56, 1 (2010), 3–16.

[46] Alexander Veremyev, Oleg A. Prokopyev, Sergiy Butenko, and Eduardo L. Pasiliao. 2016. Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications* 64, 1 (2016), 177–214.

[47] Jose L Walteros and Austin Buchanan. 2020. Why is maximum clique often easy in practice? *Operations Research* 68, 6 (2020), 1866–1895.

[48] Xiao Fan Wang and Guanrong Chen. 2003. Complex networks: small-world, scale-free and beyond. *IEEE circuits and systems magazine* 3, 1 (2003), 6–20.

[49] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.

[50] Qiu Xiao, Haiming Yu, Jiancheng Zhong, Cheng Liang, Guanghui Li, Pingjian Ding, and Jiawei Luo. 2020. An in-silico method with graph-based multi-label learning for large-scale prediction of circRNA-disease associations. *Genomics* 112, 5 (2020), 3407–3415.

[51] Kaiqiang Yu and Cheng Long. 2023. Fast Maximal Quasi-clique Enumeration: A Pruning and Branching Co-Design Approach. *Proc. ACM Manag. Data* 1, 3, Article 211 (2023), 26 pages.

[52] Qi Zheng and Xiu-Jie Wang. 2008. GOEAST: a web-based software toolkit for Gene Ontology enrichment analysis. *Nucleic acids research* 36, suppl_2 (2008), W358–W363.

A Supplementary Results

FPCE's speedup varies with call ratio (ratio of no. of recursive calls of PCE vs. FPCE), *i.e.,* its speedup is mainly the effect of its pruning techniques, not merely that of implementation tactics (Figure 8).

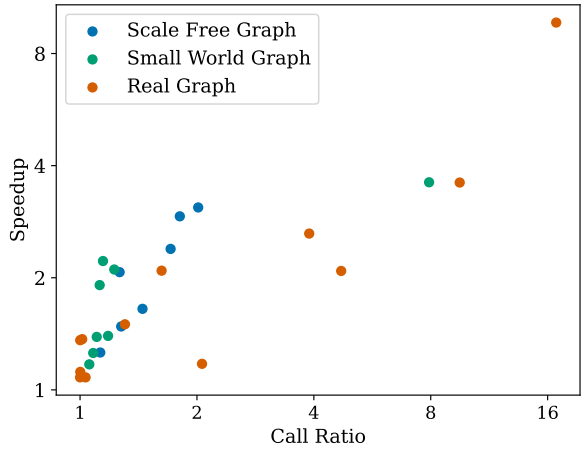

Figure 8: Call ratio vs. speedup of FPCE w.r.t. PCE.

On real graphs, FPCE and FPCE-EB achieve an average speedup of 6.5 and 5, respectively. Both use an average of 2.9 times more memory than PCE (Table 5). For space limitation, we put other results for $(\ell, \theta) = (10, 0.9)$ and those for $(10, 0.8)$ in our suppl. website.

Table 5: Time (in sec) and Memory (Resident Set Size, in MB) of PCE, FPCE-EB, and FPCE on real graphs. Memory consumptions of FPCE and FPCE-EB are almost the same.

Graph ID	Time			Memory	
	PCE	FPCE-EB	FPCE	PCE	FPCE
1	0.69	0.45	0.46	3.1	5.8
2	29,713.24	27,345.89	27,498.36	3.6	7.1
3	548.66	552.35	490.87	3.6	7.2
4	2.27	0.64	0.63	4.0	9.2
5	215.52	189.23	158.51	5.5	14.5
6	1,727.02	1,604.64	1,598.60	5.6	14.6
7	0.21	0.02	0.01	5.6	13.3
8	1.21	0.97	1.03	14.2	50.9
9	18.45	13.78	8.83	27.5	127.9
10	224.50	237.39	164.08	29.0	124.0
11	3.89	1.54	1.48	29.7	127.7
12	1.94	0.83	0.93	43.1	160.3
13	260.63	11.13	10.79	706.2	3,003.3
14	453.52	29.86	19.64	3,521.9	12,184.8
15	1,797.16	212.36	185.51	4,322.4	19,220.3

B Proofs

At first, we prove Lemmas 8 to 10 and derive some corollaries from them. Then we use those results to prove Lemmas 2 to 7 and Theorem 1 – which have already been stated in Sections 4 and 5.

Lemma 8. $c(P) = min_{v \in P} c(v)$

Proof. $c(P) = max_{P \subseteq C_k(G)}\{k\} = t \implies c(v) \geq t \; \forall v \in P$ because some vertices $v \in P$ may belong to a $(t+x)$-core (for some $x > 0$) – which must be a subgraph of the t-core [31]. On the other hand, there must be a vertex $v \in P$ s.t. $c(v) = t$ because if $\forall v \in P: \; c(v) > t$, then $c(S) > t$. □

Corollary 5. $c(P) \leq c(v) \quad \forall v \in P$

Lemma 9. *For any subset of vertices P,* $\delta(P) \leq c(P)$.

Proof. Let $\delta(P) > c(P)$. So $\forall v \in P$: $d_P(v) \geq \delta(P) > c(P)$. So $P \subseteq C_t(G)$ where $t \geq \delta(P) > c(P) \implies c(P) = t > c(P)$ (contradiction). □

Corollary 6. $\delta(P) \leq c(P) \leq c(v) \quad \forall v \in P$

The last inequality follows from Corollary 5.

Lemma 10. $P \subseteq V \implies c_P(v) \leq c(v) \quad \forall v \in P$

Proof. Let v be any node in P and $c_P(v) = k$ *i.e.,* $v \in C_k(P) \subseteq P$. According to Corollary 6, $c(v) \geq \delta(C_k(P)) = k = c_P(v)$ □

Corollary 7. $P \subseteq S \implies c_P(v) \leq c_S(v) \quad \forall v \in P$

Lemma 2: *If S is a* (ℓ, θ)*-pseudo-clique then* $\xi_S \geq \lceil \theta |S|/2 \rceil$.

Proof. Each vertex in the MD ordering of $G[S]$ is adjacent to at most ξ_S preceding nodes [47]. So

$$|E[S]| \leq 0 + 1 + 2 + \ldots + (\xi_S - 1) + (|S| - \xi_S)\xi_S = \left(|S| - \frac{\xi_S + 1}{2}\right)\xi_S$$

Since S is a (ℓ, θ)-pseudo-clique of order $|S|$,$|E[S]| \geq \theta\binom{|S|}{2}$.

$$\therefore \theta\binom{|S|}{2} \leq \left(|S| - \frac{\xi_S + 1}{2}\right)\xi_S \implies \xi_S \geq \lceil \theta |S|/2 \rceil$$

It follows from the solution of the quadratic inequality in the LHS, which was simplified by $\xi_S \geq 1$ (since $G[S]$ is connected). □

Lemma 3: *If S is a* θ*-pseudo-clique, the largest clique in* $G[S]$ *is of order* ω*, and* $\theta > (\omega - 1)/\omega$ *then* $|S| \leq \left\lfloor \frac{1}{1-(\omega-1)/\omega\theta} \right\rfloor$.

Proof. Let $|S| = n$. Since the maximum clique of G has ω nodes, G is $K_{\omega+1}$-free. So by Turan's theorem [17], $|E[S]| \leq (1 - 1/\omega)\, n^2/2$. Since S is a θ-pseudo-clique $|E[S]| \geq \theta\binom{n}{2}$. So $\theta\binom{n}{2} \leq (1 - 1/\omega)\, n^2/2$. Simplifying this, we get this upper bound on $|S| = n$. □

Theorem 1: *A* θ*-pseudo-clique S with* ℓ *vertices must contain an r-clique K where* $r = \lceil 1/(1 - \theta(\ell - 1)/\ell) \rceil$.

Proof. According to Turan's theorem, if a subgraph $G[S]$ contains no K_{j+1} then $|E[S]| \leq (1 - 1/j)\ell^2/2$. Since S is a θ-pseudo-clique with ℓ vertices, $|E[S]| \geq \theta\binom{\ell}{2}$. So if K_{j+1} is not a subgraph of $G[S]$ then $j \geq \lceil 1/(1 - \theta(\ell - 1)/\ell) \rceil$. Taking contrapositive: $G[S]$ must contain a j-clique for each value of $j \in \{1, 2, \ldots r\}$ where $r = \lceil 1/(1 - \theta(\ell - 1)/\ell) \rceil$. □

Lemma 4: *If G contains a t-clique K, then* $K \subseteq C_{t'}[G]$ *where* $t' = min\{k : k \geq t - 1 \text{ and } |C_k| > 0\}$.

Proof. Let $< v_1, v_2, \ldots, v_{|V|} >$ be the MD ordering of G and v_n be the last vertex of K in that ordering. Let $P = \{v_1, v_2, \ldots v_n\}$. Clearly, $K \subseteq P$. According to Corollary 6, $c(v) \geq \delta(P) = d_P(v_n) \geq d_K(v_n) \geq t - 1$ for all $v \in P$. Therefore $K \subseteq P \subseteq C_{t'}[G]$. □

Lemma 6: *If S, P are θ-pseudo-cliques such that $|S| = \ell$ and S is a descendant of P then $\theta\binom{\ell}{2} \leq E[P] + (\ell - |P|)\,(\delta(P) + (\ell - |P| + 1)/2)$*

Proof. By condition (ii) of Lemma 5, if $P \cup \{u\}$ is a child of P then $d_{P\cup\{u\}}(u) = \delta(P \cup \{u\}) \leq \delta(P) + 1$. By condition (i) of the same lemma, $\theta\binom{|P|+1}{2} \leq E[P] + \delta(P) + 1$. Generalizing this condition for a descendant S of P we get $\theta\binom{|S|}{2} \leq E[P] + \sum_{i=\delta(P)+1}^{\delta(P)+\ell-|P|} i = E[P] + (\ell - |P|)\left(\delta(P) + \frac{\ell-|P|+1}{2}\right)$ □

Lemma 7: *Let P, S are θ-pseudo-cliques, $|S| = \ell$, $\tau_P = \delta(P) + \frac{1}{2}$, $\eta(P) = min\{c(P), \delta(P) + \ell - |P|\}$, and P is an ancestor of S in the reverse search tree, then*

$$\theta\binom{\ell}{2} \leq \begin{cases} E[P] + (\ell - |P|)\,\delta(P), & \text{if } c(P) = \delta(P) \\ E[P] + (\ell - P + \tau_P)\eta(P) - (\delta(P) + 1)\tau_P, & \text{otherwise} \end{cases}$$

Proof. Let S^i be the i-length prefix of $S_{\succ}$ and v_i be the last node of S^i. By Lemma 9 and Corollary 7: $\delta(S) \geq \delta(P)$. When $\delta(S) = \delta(P)$, each node $v_i \in S - P$ has degree $\delta(P)$ in $G[S^i]$. So, $\theta\binom{\ell}{2} \leq \psi_1(P) = E[P] + (\ell - |P|)\delta(P)$. When $\delta(S) > \delta(P)$, $\theta\binom{\ell}{2} \leq E[P] + \sum_{i=\delta(P)+1}^{\delta(S)-1} i + (\ell - |P| - (\delta(S) - \delta(P) - 1))\delta(S)$. As $\delta(S) \leq \eta(P)$, $\theta\binom{\ell}{2} \leq \psi_2(P) = E[P] + (\ell - |P| + \tau_P)\eta(P) - (\delta(P) + 1)\tau_P$. Altogether $\theta\binom{\ell}{2} \leq max(\psi_1(P), \psi_2(P))$. If $c(P) = \delta(P)$, $\psi_1(P) > \psi_2(P)$. Otherwise, *i.e.,* when $c(P) > \delta(P)$, $\psi_2(P) > \psi_1(P)$ for both possible values of $\eta(P)$. Putting these together, we get this bound. □

C Time Complexity Analysis

The PCE algorithm (Algorithm 2) is similar to our FPCE algorithm except that it does not apply our pruning techniques. As compared to PCE, FPCE takes an extra $O(E)$ time during preprocessing for core calculation (line 1 of Algorithm 1) and an extra $O(1)$ time during each call of *recur* for applying Edge Bound. When Turan Filtering is applied, FPCE takes less time than PCE because in that case, it calls *getChildrenClq* instead of slower *getChildrenPseudoClq*. So, the time complexity of FPCE remains the same as PCE. Since our pruning techniques do not incur any extra time complexity, the worst case time complexity of FPCE is the same as that of PCE, even though in reality, significant speedup is achieved via FPCE, as shown in Section 8. Therefore, to derive the time complexity of FPCE, it suffices to derive the time complexity of the PCE algorithm – which we do below, along with a discussion of its working principle.

To explore the reverse search tree, PCE calls a recursive subroutine called *recur-basic* on each pseudo-clique of order one, *i.e.,* on each node in $V[G]$. This subroutine relies on a procedure called *getChildrenPseudoClq* (line 9 of Algorithm 2) to generate child pseudo-cliques of the current pseudo-clique P. This procedure returns a set D of nodes that satisfy the conditions of Lemma 5. Note that, condition (iii) of this lemma is equivalent to the following condition: either (a) $u \prec v^*(P)$, or (b) $u \prec \gamma(u, P)$ where $\gamma(u, P)$ is the last node in degree ordering of P which is not in $N(u)$ (if u is a neighbor of all nodes in P, then $u \prec \gamma(u, P)$, by default) [45]. To find such nodes efficiently, PCE maintains a binary tree T (henceforth called *degree tree*) that stores all nodes in $V[G]$ at its leaves in the degree ordering $\succ$ of $G[P]$. It exploits this tree to find nodes $u \notin P$ satisfying the first two conditions of Lemma 5 and condition (a) or (b), *i.e.,* it generates nodes having either of the following properties:

```
Input: Graph G, minimum order ℓ, minimum density θ
Output: All maximal (ℓ, θ)-pseudo-cliques in G
1  forall v ∈ V[G] do
2  |  recur − basic(G, {v}, v, {})
3
4  Procedure recur-basic (G, P, v, A)
5  |  if P is a maximal (ℓ, θ)-pseudo-clique then
6  |  |  output P
7  |  |  return
8  |  if θ(|P|+1 choose 2) − E[P] > δ(P) + 1 then return
9  |  D = getChildrenPseudoClq(G, P, v)
10 |  forall u ∈ D do
11 |  |  Q = P ∪ {u}
12 |  |  recur − basic(G, Q, u, D)
```

Algorithm 2: PCE(G, ℓ, θ) algorithm.

(1) Nodes satisfying the aforementioned condition (a): for such a node u, $Q = P \cup \{u\}$ must be a child of P (by Lemma 5). So *getChildrenPseudoClq* simply returns all such nodes and then *recur-basic* is called on each of them iteratively. Finding each such node from T requires $O(lgV)$ time.
(2) Nodes satisfying condition (b): Such a node must be a neighbor of $v^*(P)$ [45]. But if a node $v \in N(v^*(P))$ satisfies condition (ii) of Lemma 5, then $\gamma(v, P)$ must be in the first $d(v) + 1 = O(\Delta)$ vertices in the degree ordering of P in T. So searching for $\gamma(v, P)$ requires $O(min\{\Delta, |P|\})$ time for each $v \in N(v^*(P))$. Since $|P| \leq 2\Delta/\theta$ [20], this time is equivalent to $O(|P|)$. So the total time to compute all such neighbors of $v^*(P)$ is $O(\Delta|P|)$. This time is equivalent to $O(\Delta\xi_G)$ since $|P|$ is $O(\xi_G)$ as per Corollary 1. Thus, the time required to find each such node u is $O(\Delta\xi_G/k)$ where k is the number of such nodes. Since $k \geq 1$ (because otherwise P would be a maximal pseudo-clique), we simply count the upper bound $O(\Delta\xi_G)$ as the time spent to find each such node.

The *recur-basic* algorithm takes each of these nodes and calls itself on each extension $Q = P \cup \{u\}$ of P (lines 10 – 12 of Algorithm 2). Note that it is computationally expensive to copy a set of nodes and its corresponding degree tree. So, in the actual implementation, a global set of nodes P and its degree tree T is maintained. Whenever $recur-basic(G, Q, u, D)$ is called (respectively, returned), u is added with (respectively, deleted from) P, and its neighbors' degrees are updated in the degree tree T. No update is required for all the other nodes $w \notin N(u)$ since $deg_P(w)$ remains the same. Thus, this update operation takes $O(\Delta lgV)$ time in each call of *recur-basic*.

So, the total time spent by *recur-basic* algorithm for generating (done by *getChildrenPseudoClq*) and processing each pseudo-clique is $O(\Delta(lgV + \xi_G))$. Assuming there are N θ-pseudo-cliques in the input graph (therefore, N treenodes in the reverse-search tree), the total time complexity of PCE is $O(N\Delta(lgV + \xi_G))$ which is $O(2^{|V|}\Delta(lgV + \xi_G))$, in the worst case.