Aggregating maximal cliques in real-world graphs

Noga Alon
Princeton University
nalon@math.princeton.edu

Sabyasachi Basu
Microsoft Research
sabyasachi.basu@microsoft.com

Shweta Jain
University of Utah
shweta.jain@utah.edu

Haim Kaplan
Tel Aviv University, Google Research
haimk@tauex.tau.ac.il

Jakub Łącki
Google Research
jlacki@google.com

Blair D. Sullivan
University of Utah
sullivan@cs.utah.edu

ABSTRACT

Maximal clique enumeration is a fundamental graph mining task, but its utility is often limited by computational intractability and highly redundant output. To address these challenges, we introduce *ρ-dense aggregators*, a novel approach that succinctly captures maximal clique structure. Instead of listing all cliques, we identify a small collection of clusters with edge density at least ρ that collectively contain every maximal clique.

In contrast to maximal clique enumeration, we prove that for all $\rho < 1$, every graph admits a ρ-dense aggregator of *sub-exponential* size, $n^{O(\log_{1/\rho} n)}$, and provide an algorithm achieving this bound. For graphs with bounded degeneracy, a typical characteristic of real-world networks, our algorithm runs in near-linear time and produces near-linear size aggregators. We also establish a matching lower bound on aggregator size, proving our results are essentially tight. In an empirical evaluation on real-world networks, we demonstrate significant practical benefits for the use of aggregators: our algorithm is consistently faster than the state-of-the-art clique enumeration algorithm, with median speedups over 2.5× for $\rho = 0.1$ (and over 300× in an extreme case), while delivering a much more concise structural summary.

PVLDB Reference Format:
Noga Alon, Sabyasachi Basu, Shweta Jain, Haim Kaplan, Jakub Łącki, and Blair D. Sullivan. Aggregating maximal cliques in real-world graphs. PVLDB, 19(9): 2183-2195, 2026.
doi:10.14778/3819518.3819543

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/google/graph-mining/tree/main/in_memory/clustering/clique_aggregator.

1 INTRODUCTION

Finding maximal cliques is a fundamental graph mining task with many applications in social network analysis [24, 43], community detection [20, 51], bioinformatics [47, 53], analysis of financial networks [9, 10] among other areas. However, despite its wide applicability, its practical adoption is impeded by two significant challenges.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819543

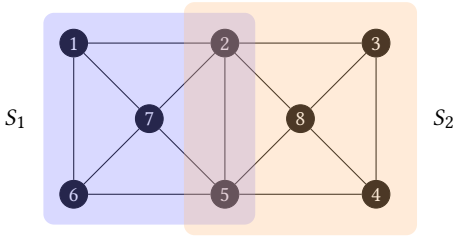

Figure 1: A graph with 8 maximal cliques (each maximal clique is a triangle), and an 0.8-dense aggregator $S = \{S_1, S_2\}$, where $S_1 = \{1, 2, 5, 6, 7\}$ and $S_2 = \{2, 3, 4, 5, 8\}$. $density(S_1)$ and $density(S_2) \geq 0.8$, and for every clique C in the graph, either $C \subseteq S_1$ or $C \subseteq S_2$.

The first is computational feasibility. Finding the largest size clique, or even approximating it within a factor of $n^{1-\epsilon}$ is NP-hard [26, 29, 54], and for a given k, even detecting if a graph has a clique of size k is W[1]-hard [18, 19]. Note that both of these problems are trivially solved if one can list all maximal cliques. Furthermore, the number of maximal cliques can grow exponentially with the number of vertices, making complete enumeration intractable for many graphs. A classic example is the Moon-Moser graph, which contains $3^{n/3}$ maximal cliques. The influential work of Eppstein, Löffler and Strash [23] showed that maximal clique enumeration is feasible in multiple real-world sparse graphs with millions of edges. However, the problem's difficulty is highly sensitive to the graph's structure and so it remains intractable for other graphs of similar size and density.

The second challenge relates to the utility of the output. Even when enumeration is computationally tractable, the resulting list of maximal cliques may be highly redundant and fail to provide a succinct structural summary. A single large, dense region in a graph can give rise to a vast number of highly overlapping maximal cliques and a single vertex may be contained in hundreds of thousands or even millions of maximal cliques. For applications, a more useful output might be a single cluster representing the union of these highly overlapping cliques, rather than an exhaustive and repetitive list of all of them. Although numerous relaxations of cliques and dense subgraph mining have been studied in the literature, these methods typically suffer from poor scalability and do not guarantee concise representation of all cliques [12, 14, 21, 27, 37, 38].

Our contributions. In this work, we address these limitations by introducing a new problem that formalizes the idea of succinctly capturing the maximal clique structure. Instead of listing every maximal clique, we aim to find a small collection of dense clusters

that collectively contain all of them. We formalize this notion using the following definition.

Definition 1.1. A ρ-dense aggregator is a set of clusters where every maximal clique is fully contained within at least one cluster, and every cluster has edge density of at least ρ.

Definitin 1.1 is illustrated in Figure 1. Here, we consider the normalized notion of *edge density* which for a subgraph S is defined as $|E(S)|/\binom{|V(S)|}{2}$.

The problem we consider in this paper is how to find, for a given density ρ, a small ρ-dense aggregator. Note that when the density threshold is set to $\rho = 1.0$, our problem becomes equivalent to maximal clique enumeration, as every cluster in the aggregator must be a clique.

To illustrate the utility of our new formulation, consider the extreme case of a Moon-Moser graph [35] – an n vertex graph that has $3^{n/3}$ maximal cliques, each of size $n/3$. This graph is obtained from a complete graph by removing n edges[1]. The output of maximal clique enumeration is both exponential in size, and highly redundant, as two randomly chosen cliques have, in expectation, $n/9$ vertices in common. Yet, the density of the entire graph is $1 - 2/(n-1)$. So for any $\rho \leq 1 - 2/(n-1)$ there exists a trivial ρ-dense aggregator with a single cluster containing all vertices.

Theoretical guarantees: While the Moon-Moser graph is admittedly very different from real-world graphs, the fact that the aggregator size is much smaller than the number of maximal cliques is a more general phenomenon. Namely, we show that for any constant $\rho < 1$ every graph admits a ρ-dense aggregator of sub-exponential size upper bounded by $n^{O(\log_{1/\rho} n)}$. In fact, in Section 3 we also give an algorithm for computing such an aggregator whose running time satisfies the above bound.

We also show that the running time of our algorithm (and also the maximum aggregator size) is almost-linear for graphs with polylogarithmic *degeneracy* (Corollary 3.4). The degeneracy of a graph G is the smallest number d, such that every subgraph of G contains a vertex of degree at most d. It has been observed that the degeneracy of real-world graphs is typically much smaller than their maximum degree [32].

To show the tightness of our arguments, we also provide a lower bound on the aggregator size. Namely, we show that for $\rho \geq c/\log n$ (for some constant c), there exist graphs where every ρ-dense aggregator has size $\Omega^*(n^{\log n/\log\log n})$ where Ω^* hides factors sublinear in n (Theorem 4.1). This bound matches our upper bound up to sublinear factors.

Empirical observations: We conduct an empirical evaluation of our algorithm on sixteen real-world graphs and demonstrate the efficiency of our approach in comparison to QuickCliques [23], the state-of-the-art algorithm for listing maximal cliques. Across over 10 datasets we study, we observe that computing a 0.9-dense aggregator is typically over 30% faster than listing maximal cliques. For 0.1-dense aggregators, the median speedup is over 2.5×. In the most extreme case, our clique aggregator is over 170 times faster than the baseline, both at 0.1 and 0.9 density thresholds.

We also observe that the output size of the clique aggregator is consistently smaller than the number of maximal cliques, on average by almost 20× and 2× for the density thresholds of 0.1 and 0.9, respectively.

Finally, we also measure the maximum number of clusters a single vertex is contained in and observe that in the solution produced by our clique aggregator algorithm this quantity decreases by 66×, and 5.2× for the density thresholds of 0.1 and 0.9, respectively.

Overall, our clique aggregator algorithm runs faster than the state-of-the-art algorithm for enumerating maximal cliques, produces a smaller output, and significantly reduces the maximum number of clusters a single vertex is contained in.

To understand how common clustering formulations differ from the task of computing a clique aggregator, we also study how common clustering techniques, such as modularity clustering, perform in terms of finding dense clusters that contain all maximal cliques. We find that, perhaps unsurprisingly, once we tune the methods to find dense clusters, they perform poorly at the coverage task, typically covering 20%–30% of maximal cliques. Finally, we also provide a theoretical justification for this observation. In Theorem 4.2 we show that there exist graphs for which any clustering into non-overlapping clusters of at least constant density fails to cover almost all maximal cliques. This holds even if covering only a constant fraction of a clique is sufficient. Theorem 4.2 formalizes an intuitive fact that covering maximal cliques with dense clusters requires overlapping clustering. Certain proofs and additional figures have been delegated to the arXiv version in the interest of space.

1.1 Related Work

Maximal clique enumeration has a long history, starting with the seminal work of Bron–Kerbosch [11]. Their recursive enumeration algorithm and its pivoting-enhanced variant [46] significantly improved the time required to enumerate maximal cliques, both in theory and practice. More efficient algorithms (and especially output-sensitive algorithms) have since been developed for sparse graphs, with near-optimal bounds relative to the number of maximal cliques [15, 22, 33]. More recent work has even further accelerated maximal clique listing by graph reduction [16] and hybrid branching [50]. Recently, clique enumeration has been shown to be fixed-parameter tractable for c-closed graphs [25]. Despite these advances, maximal clique enumeration remains computationally challenging due to the potential exponential number of maximal cliques, motivating alternative approaches.

One line of work addresses this challenge by observing that many maximal cliques tend to be concentrated in dense regions that resemble cliques with a few missing edges. They thus aim to enumerate relaxations of cliques such as k-plexes, k-clubs, near-cliques, quasi-cliques, etc. However, these methods do not scale well for large values of k, particularly on large graphs [12, 14, 27, 37]. Approaches for enumerating all subgraphs exceeding a specified edge-density threshold have also been proposed [38, 48], but these too are typically effective only within limited density regimes and fail to scale to large graphs.

A line of work that explores coverage strategies and that has inspired our algorithm is the combinatorics literature on the container method. Introduced independently by Balogh, Morris, and Samotij [3], and Saxton and Thomason [42], it allows for the approximation of large families of subgraphs using a much smaller

[1] $n/3$ disjoint triangles to be precise.

collection of "containers" that cover all relevant subsets while satisfying bounded density properties. While foundational, this line of work has been purely theoretical, focusing on combinatorial results rather than practical algorithmic applications. Another line of work explores clique summarization [21], outputting a "meaningful" subset of cliques that preserve certain properties. However, by design, they are lossy and can miss important cliques.

The rest of the paper is organized as follows. We start with some notation and necessary definitions in Section 2. In Section 3 we describe our algorithm for finding clique aggregators and analyze the performance of the algorithm in general graphs as well as in bounded-degeneracy graphs. In Section 4 we describe a lower bound on the worst-case size of an aggregator. In Section 5 we discuss the implementation of our algorithm and the optimizations that we have applied. In Section 6 we describe the results of our experiments on real-world graphs and demonstrate the effectiveness of our algorithm.

2 PRELIMINARIES

Let $G = (V, E)$ be an undirected graph with $n = |V|$ vertices and $m = |E|$ edges. We will use $N_G(v)$ to denote the neighborhood of vertex v in G. A *cluster* in G is a nonempty subset $C \subseteq V$ and a *clustering* of G is a collections of clusters. Note that we allow clusters in a clustering to have nonempty overlap. A clustering in which every vertex belongs to exactly one cluster is called *non-overlapping*. For $C \subseteq V$, we will denote by $G[C]$ the subgraph of G induced by C, and where the meaning is clear from the context, we will abuse notation slightly and use C and $G[C]$ interchangeably. We define the *density* of a graph G to be $m/\binom{n}{2}$, or 1, if the graph has only 1 vertex. We let $0 < \rho \leq 1$ represent a density threshold.

Definition 2.1 ([44]). For a graph $G = (V, E)$, a *degeneracy ordering* of G is a permutation of V given as $v_1, v_2, ..., v_n$ such that: for each $i \leq n$, v_i is the minimum degree vertex in the subgraph induced by $v_i, v_{i+1}, ..., v_n$ (we can enforce uniqueness of the ordering by breaking ties by vertex id). The degree of v_i in $G|_{v_i,...,v_n}$ is the *core number* of v_i. The largest core number is called the *degeneracy* of G, denoted $\alpha(G)$.

It is well known that there exists a linear time procedure to calculate the degeneracy ordering [34] of G. The algorithm simply removes the lowest degree vertex (and its adjacent edges) from the graph iteratively, and the degeneracy ordering is the order in which the vertices are removed from the graph.

Definition 2.2. Let $G = (V, E)$ be an undirected graph. A ρ-dense aggregator of G is a collection of clusters $C_1, \ldots, C_k$, such that:

(1) for any clique H in G, there exists a subset C_i, such that $H \subseteq C_i$, and
(2) for any C_i, the density of $G[C_i]$ is at least ρ.

We will say that an aggregator S is *inclusion maximal* if $\nexists S_i, S_j \in S$ such that $S_i \subseteq S_j$.

Figure 1 gives the example of an inclusion-maximal ρ-dense aggregator for $\rho = 0.8$ and a graph G.

3 MAIN ALGORITHM

In this section, we present our algorithm, Algorithm 1, that outputs an aggregator whose size is subexponential in n. The algorithm is a combination of the algorithm of Chiba-Nishizeki [13], which uses backtracking with degeneracy ordering to recursively enumerate cliques, and the algorithm of Bron-Kerbosch [11], which maintains a set of processed vertices to prune redundant branches. The algorithm also draws inspiration from the graph containers of Kleitman and Winston [30] studied in the context of independent sets, which were recently generalized to hypergraphs [4] and used algorithmically in several lines of work [5, 6, 28, 52].

We first give a high-level description of our procedure. We say that a set of vertices K *extends* a set of vertices $I \subseteq V(G)$ if K is a clique that is disjoint from I, and every vertex of K is adjacent to every vertex of I. We say that a clique K has been *acquired* by an aggregator S if there exists $S_i \in S$ such that $K \subseteq S_i$. Note that if a clique K has been acquired by S, then every clique that is a subset of K has also been acquired by S.

Our algorithm, Algorithm 1, similar to existing clique enumeration algorithms, recursively grows clusters. Each recursive call of our algorithm is associated with a clique C that the algorithm has discovered and is aiming to grow, and a subset $H \subseteq V(G)$ of common neighbors of C which the algorithm can use to grow C. The goal of the recursive call is to return a ρ-dense aggregator that has acquired all the cliques in $G[C \cup H]$.

If the density of $G[C \cup H]$ is at least ρ, we simply add the cluster $\{C \cup H\}$ to the aggregator, acquiring all cliques in $G[C \cup H]$. If the density of $G[C \cup H] < \rho$, then also the density of $G[H] < \rho$ (since C is a clique and each vertex of C has an edge to each vertex of H), and so there exists a vertex $v \in H$ of degree at most $\rho \cdot |H|$. In this case, we recursively acquire cliques of $G[C \cup H]$ as follows.

(a) we make a recursive call with $C := C \cup \{v\}$ and $H := N_H(v)$, the neighborhood of v in H, to find ρ-dense clusters of $G[C \cup H]$ that include v.
(b) we make a recursive call on C and $H \setminus \{v\}$ to find ρ-dense clusters of $G[C \cup H]$ that do not include v.

Since problem (a) is smaller by a factor of ρ, we can derive a much smaller bound on the size of the aggregator, compared to the upper bound on the number of maximal cliques. Note that in our pseudocode (and in practice), we perform (b) iteratively rather than recursively for efficiency. In Figure 2, we provide a worked out recursive step on a small instance to show the two cases for our aggregators.

To further improve the efficiency of our algorithm and ensure that it produces an *inclusion maximal* aggregator, we use the pruning method introduced in the seminal work of Bron and Kerbosch [11]. The implementation of this pruning is marked in teal in CliqueAggRec. In fact, even without the pruning in CliqueAggRec one can show the same running time and correctness guarantees on the algorithm as with pruning, except that the output may not be inclusion maximal anymore.

The algorithm maintains a set X of vertices that are all adjacent to the clique C such that all cliques containing $\{x\} \cup C$, $\forall x \in X$ have already been acquired by the aggregator (note that this implies that all cliques contained in $\{x\} \cup C$ have also been acquired by the aggregator). If at any recursive call there exists a vertex $x \in X$ such that $H \subseteq N_G(x)$, then every clique of $C \cup H$ is contained in a clique of G that also contains x. Since all the cliques containing $C \cup \{x\}$ have been acquired, it must be that all the cliques contained

Algorithm 1 Computing an aggregator. The code in teal ensures that the aggregator is inclusion-maximal

```
1:  function Density(G, H, C)
        return edge density of G[H ∪ C]
2:  function CliqueAggRec(G, H, ρ, C, X)
3:      for each x ∈ X do
4:          if H ⊆ N_G(x) then return ∅
5:      if Density(G, H, C) ≥ ρ then
6:          return {C ∪ H}
7:      S := ∅, Ĥ := H, X̂ := X
8:      Ord := DegeneracyOrdering(Ĥ)
9:      for each v ∈ Ĥ according to Ord do
10:         H_v := N_Ĥ(v)
11:         C_v := C ∪ {v}
12:         X_v := X̂ ∩ N_G(v)
13:         S_v := CliqueAggRec (G, H_v, ρ, C_v, X_v)
14:         S := S ∪ S_v
15:         Ĥ := Ĥ \ {v}
16:         X̂ := X̂ ∪ {v}
17:         for each x ∈ X̂ do
18:             if Ĥ ⊆ N_G(x) then return S
19:         if Density(G, Ĥ, C) ≥ ρ then
20:             return S ∪ {C ∪ Ĥ}
21:     return S
22: function CliqueAgg(G, ρ)        ▷ G is a graph and ρ ∈ (0, 1]
23:     S := CliqueAggRec (G, V(G), ρ, ∅, ∅)
24:     return S
```

in $C \cup H$ have also been acquired. In this case, the algorithm simply returns, thus avoiding adding redundant clusters to the aggregator.

We formally prove the correctness and running time bound of our algorithm below. We first show that CliqueAggRec maintains some invariants.

Lemma 3.1. *For every recursive call of* CliqueAggRec (G, H, ρ, C, X) *made by* CliqueAgg*, the following invariants hold:*

(i) The for *loop of line 9 maintains the invariant that C extends $\hat{X} \cup \hat{H}$.*

(ii) C_v extends $X_v \cup H_v$ for every recursive call CliqueAggRec (G, H_v, ρ, C_v, X_v) *made.*

Proof. We will prove this using induction. Suppose the invariants are true at the start of some recursive call CliqueAggRec (G, H, ρ, C, X). We have that $\hat{H} = H$ and $\hat{X} = X$ and by assumption, C extends $\hat{X} \cup \hat{H}$ at the start of the for loop of line 9. The only modification the loop makes to $\hat{H}$ and $\hat{X}$ in each iteration is to move a vertex (v) from $\hat{H}$ to $\hat{X}$ in lines 15 and 16 . Hence, at every iteration, C extends $\hat{X} \cup \hat{H}$ and invariant (i) holds.

For Invariant (ii), suppose at the start of a recursive call CliqueAggRec (G, H, ρ, C, X), C extends $X \cup H$. Then, by Invariant (i), at every iteration of the for loop of line 9, C extends $\hat{X} \cup \hat{H}$. For a vertex $v \in H$, $C_v = C \cup \{v\}$, $H_v = N_{\hat{H}}(v)$ and $X_v = X \cap N_G(v)$. C_v is a clique and C_v extends $X_v \cup H_v$. Thus, invariant (ii) holds.

Since, for the initial call, CliqueAggRec $(G, V(G), \rho, \emptyset, \emptyset)$, $C = \emptyset$ and $X = \emptyset$, Invariants (i) and (ii) vacuously hold true for the initial call to CliqueAggRec and as a result, Lemma 3.1 is true for every recursive call of CliqueAggRec made by CliqueAgg. □

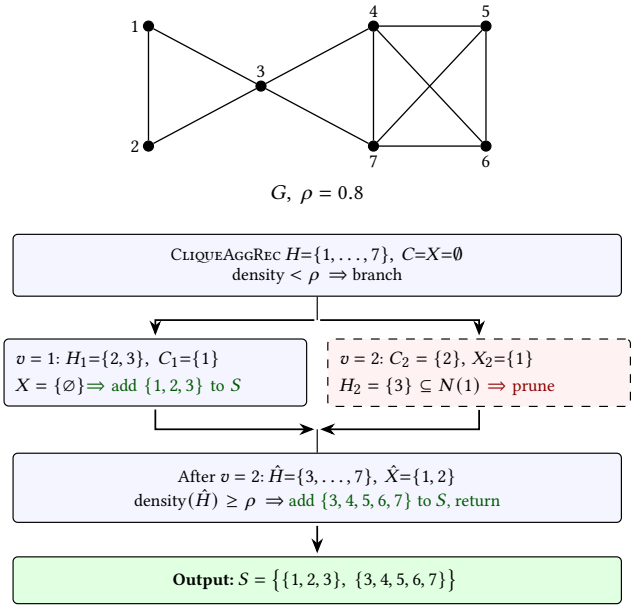

Figure 2: Illustrative execution of CliqueAgg with $\rho = 0.8$ on the graph G. The triangle $\{1, 2, 3\}$ is discovered at the first recursive branch, while the cluster $\{3, 4, 5, 6, 7\}$ satisfies the density check and leads to early termination.

We now prove our main claim that algorithm CliqueAgg outputs an inclusion-maximal ρ-dense clique aggregator of G. We use induction on the number of vertices in the set H to demonstrate that every recursive call of CliqueAggRec (G, H, ρ, C, X) returns an inclusion-maximal set S of ρ-dense clusters such that every clique $K \in C \cup H$ is either acquired by a ρ-dense cluster S_i that is added to S, or extended by a vertex x from the set X. Since, for the initial call to CliqueAggRec made by CliqueAgg, $X = C = \emptyset$ and $H = V(G)$, the S returned by that call (and hence by CliqueAgg) is an inclusion-maximal ρ-dense clique aggregator of G.

Theorem 3.2. CliqueAgg (G, ρ) *outputs an inclusion-maximal ρ-dense clique aggregator of G.*

Proof sketch. We first show that for the set S returned by any recursive call CliqueAggRec (G, H, ρ, C, X) made by CliqueAgg, the following three properties hold:

(i) For every clique K in $G[H \cup C]$, either $\exists x \in X$ such that $K \cup \{x\}$ is also a clique in G, or $\exists S_i \in S$ such that $K \subseteq S_i$.
(ii) For every $S_i \in S$, $G[S_i]$ is ρ-dense.
(iii) S is inclusion-maximal.

Note that these properties imply that the S returned by CliqueAggRec $(G, V(G), \rho, \emptyset, \emptyset)$ and hence by CliqueAgg (G, ρ) is an inclusion-maximal ρ-dense clique aggregator for G.

Property (ii) follows directly from the conditions leading to adding a cluster in lines 5 and 19. Towards Property (i), for a recursive call CliqueAggRec (G, H, ρ, C, X), we say that a clique K

is a *violating clique* if $K \subseteq C \cup H$, there exists no $x \in X$ that extends K and no $S_i \in S = \textsc{CliqueAggRec}\ (G, H, \rho, C, X)$ such that $K \subseteq S_i$ (i.e. K is not acquired by $S = \textsc{CliqueAggRec}\ (G, H, \rho, C, X)$). Then showing Property (i) is equivalent to showing that there is no violating clique for $\textsc{CliqueAggRec}\ (G, H, \rho, C, X)$.

We prove this claim as well as Property (iii) by induction on the number of vertices in H. The complete proof, which is conceptually straightforward, but quite technical is included in the supplementary material. □

We now bound the size of the aggregator output by CliqueAgg and the running time of the algorithm.

Theorem 3.3. *Algorithm* CliqueAgg *outputs a ρ-dense aggregator of size $O(n^{(1+\log_{1/\rho} n)/2})$ in time $O((m+n)n^{(1+\log_{1/\rho} n)/2})$. For a graph with degeneracy α, algorithm* CliqueAgg *outputs a ρ-dense aggregator of size $O(n\cdot\alpha^{(1+\log_{1/\rho} \alpha)/2})$ in time $O(m\alpha+n\alpha^{(3+\log_{1/\rho} \alpha)/2})$ for $\rho \in [1/n, 1)$.*

We note that the assumption that $\rho \geq 1/n$ is not limiting. Whenever $\rho \leq 1/n$, there exists a trivial clique aggregator consisting of the connected components of the graph.

Proof. Let us first bound the number of recursive calls of the algorithm. In the proof, we consider a function $T : \mathbb{R}_{\geq 0} \to \mathbb{N}$, which satisfies the following three properties. First, $T(0) = T(1) = 1$. Second, $T(i)$ is an upper bound on the number of recursive calls of $\textsc{CliqueAggRec}(G, H, \rho, C, X)$ assuming that the size of H is i. Third, T is nondecreasing.

From the pseudocode and monotonicity of T, we obtain

$$T(n) \leq \sum_{i=1}^{n} T(\rho \cdot (n-i)) \leq n \cdot T(\rho \cdot n).$$

Using this property, we can prove by induction that

$$T(n) \leq n^{(1+\log_{1/\rho} n)/2}.$$

Indeed,

$$\begin{aligned} T(n) &\leq n \cdot T(\rho \cdot n) \leq n \cdot (\rho \cdot n)^{(1+\log_{1/\rho}(\rho \cdot n))/2} \\ &= n \cdot (\rho \cdot n)^{(\log_{1/\rho} n)/2} = n \cdot n^{-1/2} n^{(\log_{1/\rho} n)/2} \\ &= n^{(1+\log_{1/\rho} n)/2}. \end{aligned}$$

Thus, the algorithm makes at most $n^{(1+\log_{1/\rho} n)/2}$ recursive calls. We now bound the cost of each recursive call. We call the time spent within a function, excluding the recursive calls it makes, the *non-cumulative running time.*

Finding the degeneracy ordering, calculating the density, constructing the neighborhoods of the nodes, and maintaining the set X can be done in $O(m+n)$ time by making a constant number of passes over the vertices and edges in the graph and counting/adding them in the appropriate sets. Hence, the non-cumulative time spent at every recursive call is $O(m+n)$. Since there are at most $O(n^{(1+\log_{1/\rho} n)/2})$ recursive calls made, the total running time of the algorithm is $O((m+n)n^{(1+\log_{1/\rho} n)/2})$.

However, similar to [22] we can give a more fine-grained bound in terms of the degeneracy, α. For the sake of exposition, in Algorithm 1, we have passed G to every recursive call, along with the corresponding sets X, C and H. However, all the tasks that contribute to the non-cumulative running time such as constructing the neighborhoods of the nodes, and maintaining the set X can be performed more efficiently if we pass the subgraph $G[H \cup X]$ instead (in addition to C). This is because the vertices and edges not in $G[H \cup X \cup C]$ are immaterial to constructing the inclusion-maximal ρ-dense aggregator of $G[H \cup C]$. To process a vertex v in CliqueAggRec then, it suffices to look at v's neighbors in $G[H \cup X]$.

Consider the first call to CliqueAggRec made by CliqueAgg. For any $v \in V$, the sets X_v and H_v, and the subgraph $G[H_v \cup X_v]$ are constructed in time $O(\alpha(|H_v| + |X_v|)$ (Lemma 3 and Theorem 2 in [22]). Thus, cumulatively, the first call to CliqueAggRec takes time at most $\sum_v (\alpha(|H_v| + |X_v|)) = m\alpha$ to construct the subgraphs $G[H_v \cup X_v]$ for all $v \in V$. Moreover, the use of degeneracy ordering in CliqueAggRec implies that for each of the recursive calls, H has size at most α. Hence, the size of the recursion tree created by each recursive call is at most $\alpha^{(1+\log_{1/\rho} \alpha)/2}$. The non-cumulative time for every recursive call in these trees is $O(m+n) = O(n\alpha)$. Thus, the total time of each recursion tree is at most $O(n\alpha \cdot \alpha^{(1+\log_{1/\rho} \alpha)/2}) = O(n\alpha^{(3+\log_{1/\rho} \alpha)/2})$. Combining with the $O(m\alpha)$ taken by the first call of CliqueAggRec, the total running time of Algorithm CliqueAgg is $O(m\alpha + n\alpha^{(3+\log_{1/\rho} \alpha)/2})$. □

We note that since $\alpha \leq n-1$, Theorem 3.3 implies that a ρ-dense aggregator of size quasi-polynomial in n always exists. This is remarkable because the number of maximal cliques can be exponential ($\Theta(3^{n/3})$ [11]) in the worst case.

Theorem 3.3 implies the following useful lemma.

Corollary 3.4. *Let $\alpha = 2^{o(\sqrt{\log n})}$. For $\rho = 1 - \Omega(1)$, algorithm* CliqueAgg *outputs a ρ-dense aggregator of size $O(n^{1+o(1)})$ in time $O(n^{1+o(1)})$ and for $\rho \geq c/\log n$ for some constant c, algorithm* CliqueAgg *outputs a ρ-dense aggregator of size $n^{O(\frac{\log n}{\log\log n})}$ in time $n^{O(\frac{\log n}{\log\log n})}$.*

Proof. We note that $\alpha = 2^{o(\sqrt{\log n})} = n^{o(1/\sqrt{\log n})} = n^{o(1)}$ and $\log \alpha = o(\sqrt{\log n})$. Hence, if $\rho = 1 - \Omega(1)$ then by substituting for α and ρ in Theorem 3.3

$$\begin{aligned} \alpha^{(1+\log_{1/\rho} \alpha)/2} &= \alpha^{O(\log \alpha)} = (2^{o(\sqrt{\log n})})^{o(\sqrt{\log n})} \\ &= 2^{o(\log n)} = n^{o(1)}. \end{aligned}$$

Similarly, when $\rho \geq c/\log n$, the asymptotic running time and size of aggregator are $n^{O(\frac{\log n}{\log\log n})}$. As we will show in Theorem 4.1, our lower bound matches this upper bound up to constant factors. □

4 LOWER BOUND

In this section, we will show that there exist graphs for which clique aggregators whose size is polynomial in n do not exist, and we give a lower bound for the size of an aggregator for these graphs that matches the upper bound of Corollary 3.4 up to sublinear factors.

Theorem 4.1. *There exists a constant c and a family of graphs $\mathcal{G}$ for which every ρ-dense clique aggregator for $\rho \geq c/\log n$ has size $\Omega^*(n^{\frac{3\log n}{8\log\log n}})$ (Ω^* hides factors sublinear in n).*

Proof. We show this by obtaining a lower bound for the number of clusters required to acquire every maximum clique in the graph. Since every maximum clique is also a maximal clique, clearly, this is a lower bound for the size of the aggregator. Consider the following family $\mathcal{G}$ of random graphs. V consists of n vertices partitioned into k vertex-disjoint subsets $V_1, \ldots, V_k$ each consisting of n/k vertices[2] for $k = \frac{1}{2}\frac{\log n}{\log\log n}$. For each $i = 1, \ldots, k$, the vertex set V_i is independent. Edges between vertices $u \in V_j$ and $v \in V_{j'}$ for distinct (j, j') are added independently with probability $p = \frac{1}{\log n}$.

Our proof now splits into two parts. We first show that any cluster of at least $\log^2 n$ vertices in a graph from $\mathcal{G}$ has density smaller than $c/\log n$ for some constant c with high probability (i.e. with probability that (polynomially) goes to 0 as n goes to infinity). Then we show that a graph from $\mathcal{G}$ has $\Omega^*(n^{\frac{3\log n}{8\log\log n}})$ cliques of size $k = \frac{1}{2}\frac{\log n}{\log\log n}$ with high probability. Consequently, in almost every graph from $\mathcal{G}$, the maximum cliques must be covered by clusters of size less than $\log^2 n$. Furthermore, each such cluster covers at most $(\log^2 n)^k = O(n)$ maximum cliques. So our cover must use $\Omega^*(n^{\frac{3\log n}{8\log\log n}})$ clusters.

(A) Clusters of size at least $\log^2 n$ have density $O(1/\log n)$.

Consider a cluster C of ℓ vertices in a graph from $\mathcal{G}$. The number of edges in the subgraph induced by this cluster is a binomial random variable $X = Binomial(m, p)$ where $m = \binom{\ell}{2}$. Clearly $\mu = E[X] = mp = \binom{\ell}{2}\frac{1}{\log n}$. Fix $\rho = \frac{c}{\log n}$ for some large constant c. We bound the probability of having at least $s = \rho\binom{\ell}{2} = c\mu$ edges in $G[C]$. A standard tail bound on a binomial random variable gives that for any $\delta > 0$:

$$\Pr(X \geq (1+\delta)\mu) \leq \exp\left(-\frac{\mu\delta^2}{2+\delta}\right) .$$

Substituting $\delta = c - 1$ and using the fact that $\mu = \binom{\ell}{2}\frac{1}{\log n}$, we get that

$$\Pr(X \geq c\mu) \leq e^{-\frac{\mu(c-1)^2}{c+1}} = e^{-\frac{\binom{\ell}{2}\frac{1}{\log n}(c-1)^2}{(c+1)}} .$$

There are $\binom{n}{\ell}$ clusters of ℓ vertices. So by the union bound the probably that at least one of them has density larger than $\frac{c}{\log n}$ is at most

$$\binom{n}{\ell} e^{-\frac{\binom{\ell}{2}\frac{1}{\log n}(c-1)^2}{(c+1)}} \leq e^{(\ln n)\cdot\ell - \frac{\ell(\ell-1)(c-1)^2}{2(c+1)\log n}} .$$

For $\ell = \Omega(\log^2 n)$ and a reasonably large c this is $e^{-O(\ell^2/\log n)}$. We conclude that all clusters of size $\Omega(\log^2 n)$ have density smaller than $O(1/\log n)$ with high probability.

(B) The number of cliques of size k. By the structure of the graph, these are maximum cliques. A cluster of k vertices could be a clique only if each vertex is from a different V_i. It follows that there are $s = (\frac{n}{k})^k$ such clusters, $C_1, \ldots, C_s$. Let A_i, $i \in [s]$, be an indicator random variable of the event that C_i is a clique. We have for all $i \in [s]$

$$\Pr(A_i) = \left(\frac{1}{\log n}\right)^{\binom{k}{2}} .$$

[2]We assume that k divides n.

Let $Y = \sum_{i\in[s]} A_i$ be the number of k-cliques in the graph. Then

$$E[Y] = \left(\frac{n}{k}\right)^k \Pr(A_i) = \left(\frac{n}{k}\right)^k \left(\frac{1}{\log n}\right)^{\binom{k}{2}}$$
$$= \left(\frac{n}{k}\right)^k 2^{-\log\log n\cdot k(k-1)/2} .$$

Substituting $k = \frac{1}{2}\frac{\log n}{\log\log n}$ we get that

$$E[Y] \approx n^{\frac{3k}{4}} = n^{\frac{3\log n}{8\log\log n}},$$

where this approximation is tight up to a polynomial factor smaller than n.

To show that Y is concentrated around its mean, we follow the second moment method described in Alon and Spencer [1], and bound the variance of Y. By definition,

$$Var[Y] = E[Y] + \sum_{i\neq j} Cov[A_i, A_j]$$

where

$$Cov[A_i, A_j] = E[A_iA_j] - E[A_i]E[A_j] .$$

Now, if C_i and C_j are edge-disjoint then $Cov[A_i, A_j] = 0$. Otherwise,

$$Cov[A_i, A_j] \leq E[A_iA_j] = \Pr(A_i \wedge A_j) .$$

We conclude that

$$Var[Y] \leq E[Y] + \sum_{i\sim j} \Pr(A_i \wedge A_j), \tag{1}$$

where $i \sim j$ means that C_i and C_j have at least one edge in common.

We can further write

$$\sum_{i\sim j} \Pr(A_i \wedge A_j) = \sum_i \Pr[A_i] \sum_{j\sim i} \Pr[A_j \mid A_i] = E[Y]\Delta^*, \tag{2}$$

where $\Delta^* = \sum_{j\sim i} \Pr[A_j \mid A_i]$ - which is in fact independent of i by symmetry. Fix C_i, there are $\binom{k}{2}\left(\frac{n}{k}-1\right)^{k-2}$ clusters C_j that share two vertices (a single edge) with C_i. For each such cluster we have

$$\Pr[A_j \mid A_i] = \left(\frac{1}{\log n}\right)^{\left(\binom{k}{2}-1\right)} .$$

Similarly we have $\binom{k}{3}\left(\frac{n}{k}-1\right)^{k-3}$ clusters C_j that share three vertices with C_i. For each such cluster we have

$$\Pr[A_j \mid A_i] = \left(\frac{1}{\log n}\right)^{\left(\binom{k}{2}-3\right)} .$$

In general for $1 < \beta < k$ we have $\binom{k}{\beta}\left(\frac{n}{k}-1\right)^{k-\beta}$ clusters C_j that share β vertices with C_i. For each such cluster we have

$$\Pr[A_j \mid A_i] = \left(\frac{1}{\log n}\right)^{\left(\binom{k}{2}-\binom{\beta}{2}\right)} .$$

It follows that the total contribution of clusters C_j such that $|C_j \cap C_i| = \beta$ to Δ^* is

$$\binom{k}{\beta}\left(\frac{n}{k}-1\right)^{k-\beta}\left(\frac{1}{\log n}\right)^{\left(\binom{k}{2}-\binom{\beta}{2}\right)} .$$

Dividing this by $E[Y]$ we get

$$\frac{\binom{k}{\beta}\left(\frac{n}{k}-1\right)^{k-\beta}\left(\frac{1}{\log n}\right)^{\left(\binom{k}{2}-\binom{\beta}{2}\right)}}{\left(\frac{n}{k}\right)^{k}\left(\frac{1}{\log n}\right)^{\binom{k}{2}}} \leq \frac{k^{\beta}(\log n)^{\binom{\beta}{2}}}{\left(\frac{n}{k}\right)^{\beta}}$$

$$= k^{2\beta}2^{\log\log n\beta(\beta-1)/2-\log n\cdot\beta} .$$

Since $\beta < k = \frac{1}{2}\frac{\log n}{\log\log n}$ we get that this ratio is at most

$$k^{2\beta}2^{\log n(\beta-1)/4-\log n\cdot\beta} \leq k^{2\beta}n^{-\frac{3\beta}{4}} ,$$

which is $O(n^{1-\frac{3\beta}{4}})$. So for any $1 < \beta < k$ we conclude that $\Delta^* = o(E(Y))$. Combining this with Equations (1) and (2) yields

$$Var[Y] = o((E[Y])^2) .$$

Recall that the Chebyshev Inequality says that for any $\varepsilon > 0$

$$\Pr(|Y - E[Y] \geq \varepsilon E[Y]) \leq \frac{Var[Y]}{\varepsilon^2(E[Y])^2},$$

so it follows that

$$\Pr(|Y - E[Y] \geq \varepsilon E[Y]) \underset{n\to\infty}{\longrightarrow} 0 ,$$

and Y is within a constant factor of its mean with high probability. □

4.1 Coverage of non-overlapping clustering

In this section we show that there exist graphs, for which any non-overlapping clustering completely fails to capture the structure of their dense subgraphs.

Let G be an undirected graph and $\mathcal{C}$ be a non-overlapping clustering of G. Also, let D be a clique in G. We say that $\mathcal{C}$ *partially covers* D if there exists a cluster $C \in \mathcal{C}$ such that $|C \cap D| = \Omega(|D|)$, that is there is a cluster containing a constant fraction of the vertices of D.

Theorem 4.2. *For any sufficiently large n, there exists a graph G_n, such that for any non-overlapping clustering $\mathcal{C}$ of G_n, only a $o(1)$-fraction of all maximal cliques of G_n are partially covered by $\mathcal{C}$.*

To prove the above theorem we are going to use a graph, whose properties are given in the following lemma.

Lemma 4.3. *For any sufficiently large n and any $t \in [1, n^{0.1}]$ there exists an undirected graph $H_{n,t}$ with the following two properties.*

(1) $H_{n,t}$ has n vertices, each of degree $\Theta(t)$.
(2) $H_{n,t}$ has no cycles of length ≤ 8.

To prove Theorem 4.2 we fix n and t and consider a graph $H^2_{n,t}$, which is a *square* of $H_{n,t}$. That is, $H^2_{n,t}$ has the same vertex set as $H_{n,t}$ but a different set of edges. Namely, there is an edge uv in $H^2_{n,t}$ if and only if there is a path of length 2 between u and v in $H_{n,t}$. The rest of the proof is described in the supplementary material.

5 IMPLEMENTATION AND OPTIMIZATIONS

In this section we describe our C++ implementation of the clique aggregator algorithm and the optimizations that we have used. Algorithm 2 gives the pseudocode of the algorithm that we implement, which, as we argue in this section, is equivalent to Algorithm 1. The code is available at https://github.com/google/graph-mining/tree/main/in_memory/clustering/clique_aggregator. The main recursive function in Algorithm 2 is CliqueAggImpl.

Algorithm 2 Pseudocode of our C++ implementation The code in teal ensures that the aggregator is inclusion-maximal

```
1:  function Density(G, C)
        return density of V(G) ∪ C where C extends V(G)
2:  function RecX({X_1, ..., X_k}, v, G⃗)
3:      H_v := N_{G⃗}(v)                          ▷ The set we recurse on
4:      X_out := {X_i ∩ H_v | 1 ≤ i ≤ k} ∪ {N_{G⃗}(w) ∩ H_v | w ∈ N^{in}_{G⃗}(v)}
5:      if |H_v| > 0 then
6:          Remove empty sets from X_out
7:      return X_out
8:  function CliqueAggRecImpl(G, ρ, C, 𝒳)
9:      for each X_i ∈ 𝒳 do
10:         if X_i = V(G) then return ∅
11:     if Density(G, C) ≥ ρ then
12:         return {C ∪ V(G)}
13:     S := ∅
14:     Ord := DegeneracyOrdering(G)
15:     G⃗ := Direct(G, Ord)
16:     for each v ∈ G⃗ according to Ord do
17:         G_v := G[N_{G⃗}(v)], C_v = C ∪ {v}
18:         S := S ∪ CliqueAggRecImpl(G_v, ρ, C_v, RecX(𝒳, v, G⃗))
19:         remove v from G
20:         if Density(G, C) ≥ ρ then
21:             if ∃X ∈ 𝒳 s.t. V(G) ⊆ X then return S
22:             P := vertices of G⃗, who are not later than v in Ord
23:             if ∃w ∈ P s.t. V(G) ⊆ N_{G⃗(w)} then return S
24:             return S ∪ {C ∪ V(G)}
25:     return S
26: function CliqueAggImpl(G, ρ)   ▷ G is a graph and ρ ∈ (0, 1]
27:     S := CliqueAggRecImpl (G, ρ, ∅, ∅)
28:     return S
```

Equivalence of Algorithms 1 and 2. We start by discussing how Algorithm 2 works. In Algorithm 1, the main recursive function CliqueAgg takes as arguments a graph G and a set of vertices $H \subseteq V(G)$ and operates on a graph $G[H]$. In Algorithm 2 this works a bit differently: each recursive call is simply given $G[H]$ as an argument (in Algorithm 2 it is denoted by G). Materializing the induced subgraph enables several optimizations that we discuss later on.

However, this also requires some extra work of actually computing the induced subgraph in Line 17. To this end, Algorithm 2 computes not only the degeneracy ordering, but also a directed version $\vec{G}$ of G, where each edge is directed towards the endpoint that comes later in the ordering. This orientation is a standard tool for efficient triangle listing, as finding all triangles involving a vertex v requires finding all edges between its neighbors [13]. Note that this is the exact operation needed to construct the induced subgraph for the next recursive step.

In Algorithm 1, as vertices are processed in degeneracy order, they are notionally removed from the graph. Consequently, any subsequent operation on a vertex v only considers neighbors that appear later in the ordering. Algorithm 2 achieves this efficiently by

using the directed graph $\vec{G}$; the relevant neighborhood of a vertex v is simply its out-neighborhood $N_{\vec{G}}(v)$. This approach avoids the more complex operation of dynamically removing vertices from an undirected graph and updating the adjacency lists of its neighbors.

The second difference is in how the set X used for pruning is represented. In Algorithm 1, the pruning set X contains vertices that have already been processed. Its only purpose is to check if the current candidate set H is a subset of a previously processed vertex's neighborhood (i.e., $H \subseteq N_G(x)$). Algorithm 2 makes this check more direct. Instead of storing a vertex x in a set X, our collection $\mathcal{X}$ contains the precomputed neighborhoods of such vertices restricted to the current graph. Namely, instead of storing a vertex x_i in X, Algorithm 2 stores a set X_i, where $X_i = N_G(x_i) \cap H$. This allows us to check for inclusion directly without referencing the original graph.

The set $\mathcal{X}$ is also computed a bit differently from X. When Algorithm 1 recurses on the neighborhood of a vertex v, it considers a set $\hat{X}$ which is the union of two sets:

(1) Set X provided as an argument to CliqueAgg, and
(2) all vertices that came before v in the degeneracy ordering. These are added to $\hat{X}$ in Line 16 in Algorithm 1.

Then, this set is intersected with $N_G(v)$ to obtain the set X for the recursive call. In Algorithm 2, this is achieved by the function RecX, which computes an analogous set $\mathcal{X}$. Specifically, $\{X_i \cap H_v \mid 1 \leq i \leq k\}$ is a direct counterpart of (1), while $\{N_{\vec{G}}(w) \cap H_v \mid w \in N^{in}_{\vec{G}}(v)\}$ is responsible for (2). Observe that $w \in N^{in}_{\vec{G}}(v)$ is exactly the set of vertices that come in the degeneracy ordering before v and have an edge to v.

Optimized data representation. It has been previously observed that set intersection is a dominant operation in maximal clique enumeration [7]. Thus, we optimize our data representation in order to support fast intersections.

We use two different graph representations. For large graphs, we simply put the neighbors of each vertex in a flat array (`std::vector`). For small graphs, we use a bit matrix representation. That is, an n-vertex graph is represented with an $n \times n$ array A of bits, where $A[i][j] = 1$ iff there is an edge from vertex i to j. We store the rows of this array packed into 64-bit integers (`uint64_t`).

In our implementation we use the flat array representation in the outermost recursive call, and the bit array representation in every other call. This is because the size of the graph in each recursive call is at most the graph degeneracy, which for real-world graphs is typically small. For example, among over 80 datasets studied in [23] the maximum degeneracy is 266. For graphs of this size, the bit matrix representation uses at most five 64-bit variables to store each adjacency list.

Let us now roughly compare the space usage of both representations. Consider an n-vertex graph with average outdegree is 10. We use 4 bytes to store each neighbor, and so the total space usage, ignoring the additional `std::vector` overhead is $40 \cdot n$ bytes. For the bit array representation, we get $n^2/8$ bytes. Under these assumptions the bit array representation is smaller as long as $n < 320$.

The bit matrix representation allows us to efficiently iterate through the neighbors of a given node by using bit operations, i.e., finding the index of the lowest nonzero bit. More importantly, it allows us to efficiently intersect two adjacency lists using bitwise AND, which we use for example when computing the induced subgraph in Line 17. We also use a bit array representation to store elements of $\mathcal{X}$. Observe that in our algorithm each element of $\mathcal{X}$ is a subset of vertices of a graph, whose size is at most the degeneracy of the input graph. Hence, a similar reasoning shows that representing each element of $\mathcal{X}$ uses at most a handful of 64-bit integers. This combination of bit-level representations for the graph and the elements of $\mathcal{X}$ allows us to very efficiently compute set intersection and set containment in lines 4, 10, 17, 21 and 23.

6 EMPIRICAL EVALUATION

In this section we present the result of a series of experiments performed to study the performance of our algorithm. We use an AMD EPC 7003 virtual machine with 32 vCPUs and 128 GB of RAM. The github code in Section 5 points to the repository: the code was compiled using `GCC 12.2.0` with the `O3` optimization. The implementation has some parameters: for the rest of this section, we treat the default as the setting that does allow Bron Kerbosch pruning, allows disconnected clusters and uses the bitmapped representation discussed above. In Section 6.3, study how the running time of the algorithm changes when some of these parameters are toggled.

Datasets. We carry out extensive experiments on 16 publicly available graph datasets shown in Table 1. Fourteen of these datasets come from the SNAP repository [31]. In order to choose challenging datasets, we picked graphs for which the number of maximal cliques is at least half the number of edges. In addition, we also use the the hollywood and soc-twitter datasets, which were taken from the network repository [40]. These graphs have higher degeneracy than the graphs available in SNAP, which makes them particularly hard for the task of listing maximal cliques.

All datasets were treated as undirected graphs, and were preprocessed to remove parallel edges and self loops. In addition, for the soc-twitter dataset, we use the 3-core of the original dataset, that is, we (iteratively) remove vertices of degree at most 2. This reduces the number of edges in the graph from 1.5 billion to about 256 million. We note that this preprocessing does not remove any maximal clique of size at least 4.

We perform experiments to compare our method with several baselines in terms of the running time, and the properies of the output, specifically, the number of clusters, their density and the vertex membership, that is the number of clusters a vertex is contained in. For every run, we set a timeout of 5 hours.

6.1 Comparison with listing maximal cliques

We perform a comprehensive comparison of our algorithm with maximal clique enumeration. We use CliqueAggImpl to refer to the implementation of our algorithm described in Section 5 or CliqueAggImpl (ρ) to refer to the algorithm run with a density threshold of ρ. We use Eppstein et al.'s implementation of maximal clique enumeration via the QuickCliques package [23], and the more recent RMCE algorithm [16]. We refer to these implementations as QC and RMCE for the rest of the text. For QC we use the `hybrid` algorithm, which was typically the fastest, and we compare with the sequential version of RMCE.

Dataset	Vertices	Edges	Degen.
euemail	1K	16K	34
Wiki-Vote	8K	101K	53
citHepTh	28K	352K	37
soc-Epinions1	76K	406K	67
Slashdot	82k	948k	55
web-BerkStan	685K	8M	201
hollywood	1M	56M	2208
youtube	1M	3M	51
pokec	2M	22M	47
skitter	2M	11M	111
wiki	2M	25M	99
WikiTalk	2M	5M	131
orkut	3M	117M	253
cit-Patents	4M	17M	64
livejournal	4M	35M	360
soc-twitter	21M	265M	1695

Table 1: Datasets used in our experiments, along with their sizes and degeneracies. The soc-twitter dataset is the 3-core of the publicly available twitter dataset.

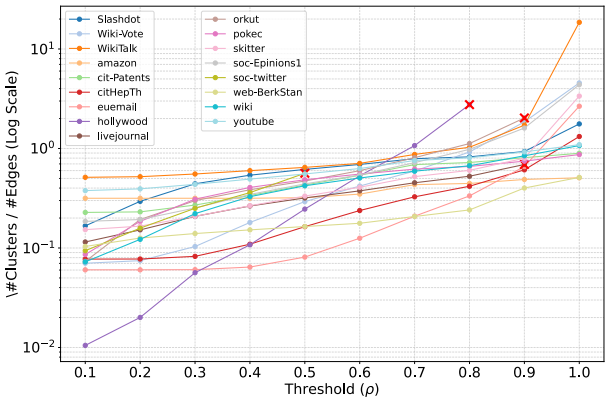

Figure 3: Aggregator sizes as the density threshold increases; we normalize the cluster count by the number of edges.

To the best of our knowledge, this is the most well-engineered and popular (serial) maximal clique enumeration package which is publicly available. We note that with the threshold ρ set to 1.0, CliqueAggImpl produces maximal cliques, even though it is not engineered to shine at this threshold.

6.1.1 Aggregator sizes. We first compare the size of the clique aggregator computed by our algorithm (that is, the number of clusters) by gradually decreasing the density threshold from 1.0 to 0.1. The theoretical upper bound on the number of clusters computed by CliqueAggImpl (ρ) for $\rho \leq 0.9$ is much smaller than the number of maximal cliques (quasi-polynomial vs exponential). The purpose of this experiment is to verify whether a similar phenomenon is also visible in real-world graphs.

In Figure 3 report aggregator sizes divided by the number of edges—this normalization puts the numbers from the graphs we studied on a similar scale.

We observe a sharp change in the number of clusters as the threshold decreases. In several cases, there is an order of magnitude difference, even when the density decreases from 1.0 to 0.9. Typically, at 0.1, the size of the aggregator is significantly smaller than the number of maximal cliques. The geometric means of the reduction in aggregator sizes over the number of maximal cliques (barring the datasets where maximal clique enumeration did not terminate) is almost 25× for $\rho = 0.1$, and over 5× at 0.5. While for $\rho = 0.9$ this reduction is not as pronounced, it is still over 2× on average. Even in a logarithmic scale, the sharp decrease for densities < 1.0 is noticeable. Noticeably, for orkut and WikiTalk, even the number of clusters in the 0.9-dense aggregator is just about 10% of the number of maximal cliques.

Our results align with earlier findings that in multiple real-world graphs the number of maximal cliques is far smaller than what the exponential upper bound admits [22]. We observe that a similar phenomenon extends to clique aggregators: the number of clusters in the aggregator is much smaller than the theoretical upper bound. The graphs we study admit a clique aggregator using many fewer clusters than the number of maximal cliques.

6.1.2 Running time. We now compare the running time of CliqueAggImpl with QC and RMCE to understand the scalability of our algorithm and implementation. The results are presented in Table 2. When measuring the running time of the algorithms, we exclude the I/O time, that is the time needed to read the input graph and to write the output to disk.

We consistently see that the time taken to produce aggregators at low thresholds is often orders of magnitudes lower than the time taken by QC to compute all maximal cliques. RMCE is conisderably faster on some datasets, especially the orkut dataset, where neither CliqueAggImpl not QC terminate in 5 hours but RMCE terminates in just over half an hour. In several others, the gap is more modest, but especially for the large graphs, CliqueAggImpl at a threshold of 0.9 is typically faster than RMCE. At 0.1, the geometric mean of the speedups is over 5×, and the arithmetic mean over 25×. We note that when computing these speedups we assumed a running time of 5 hours for each RMCE run which did not finish within this time limit, which means that the actual speedups are larger. In an extreme case, of the livejournal dataset, CliqueAggImpl (0.9) finished in about 2 minutes, while RMCE did not finish in 5 hours, which corresponds to a speedup of at least 170×.

On the other hand, we observe that CliqueAggImpl (1.0) is usually slower than both QC and RMCE. The reason for this slowdown is likely the fact that QC and RMCE are optimized for this exact setting. Still, this observation can also be viewed positively. Namely, it suggests that the speedups we observe for $\rho < 1$ should not be attributed to implementation-level optimizations, but rather to the algorithmic difference between the algorithms.

Overall, the results indicate that a clique aggregator can be obtained much more efficiently than listing maximal cliques. The most notable improvement is for $\rho = 0.9$. We can produce aggregators of extremely high density that are computed in minutes even when maximal clique enumeration does not terminate in hours.

Networks	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	QC	RMCE	S/U(0.1)	S/U(0.9)
euemail	15ms	16ms	17ms	16ms	17ms	19ms	24ms	34ms	50ms	200ms	74ms	34 ms	2.3×	0.7×
Wiki-Vote	123 ms	124 ms	133 ms	154 ms	186 ms	225 ms	273 ms	397 ms	672 ms	2 s	860 ms	366 ms	3.0×	0.5×
citHepTh	319 ms	321 ms	345 ms	382 ms	423 ms	487 ms	582 ms	659 ms	906 ms	2 s	1 s	510 ms	1.6×	0.6×
soc-Epinions1	511 ms	485 ms	547 ms	615 ms	683 ms	795 ms	1 s	1 s	2 s	8 s	4 s	1 s	2.8×	0.7×
Slashdot	453 ms	508 ms	553 ms	575 ms	620 ms	627 ms	674 ms	702 ms	848 ms	5 s	2 s	714 ms	1.6×	0.8×
web-BerkStan	6 s	6 s	6 s	6 s	6 s	6 s	7 s	7 s	8 s	14 s	13 s	13 s	2.4×	1.6×
hollywood	3 min	4 min	5 min	6 min	8 min	12 min	22 min	62 min	-	-	-	-	88.6×+	-
youtube	3 s	3 s	3 s	4 s	4 s	4 s	4 s	5 s	5 s	7 s	6 s	4 s	1.3×	0.7×
pokec	33 s	34 s	38 s	39 s	41 s	43 s	45 s	46 s	51 s	58 s	1 min	39 s	1.2×	0.8×
skitter	11 s	11 s	12 s	13 s	14 s	15 s	17 s	19 s	26 s	13 min	2 min	46 s	4.2×	1.8×
wiki	34 s	37 s	41 s	44 s	47 s	49 s	53 s	57 s	1 min	1 min	2 min	1 min	2.1×	1.1×
WikiTalk	4 s	5 s	5 s	6 s	7 s	8 s	10 s	16 s	40 s	13 min	4 min	1 min	15.3×	1.63×
orkut	4 min	5 min	6 min	6 min	7 min	8 min	10 min	12 min	21 min	-	-	32 min	7.3×	1.5×+
cit-Patents	19 s	19 s	20 s	21 s	22 s	23 s	24 s	24 s	26 s	28 s	27 s	19 s	1.0×	0.7×
livejournal	51s	54 s	57 s	1 min	1 min	1 min	1 min	1 min	2 min	-	-	-	350×+	173.6×+
soc-twitter	22 min	31 min	43 min	70 min	168 min	-	-	-	-	-	-	-	13.4×+	-

Table 2: The running time of CliqueAggImpl for different density thresholds, compared to the running time of QC, and RMCE The final two columns show the speedup of CliqueAggImpl over RMCE for $\rho \in \{0.1, 0.9\}$. The timing numbers are rounded, but the speedups are calculated over the exact values. Entries marked with '-' refer to methods that do not terminate in 5 hours, and OOM refers to the case where the algorithm crashes due to memory constraints.

6.1.3 Cluster Membership. In this section we study the degree to which the clusters overlap by considering two different overlap measures. The first one is *membership distribution*: the fraction of vertices belonging to at most x clusters. While in a disjoint partition *all* vertices belong to exactly one cluster, a vertex can be part of several clusters when overlaps are allowed. Figure 4 compares the membership distribution of maximal cliques (equivalent to CliqueAggImpl (1.0)) with CliqueAggImpl (ρ) for $\rho \in \{0.1, 0.5, 0.9\}$ for two datasets. For the two datasets presented in 4 we observe that only about 20% of vertices are contained in at most 1 maximal clique. In contrast, the number of vertices contained in at most 1 cluster in the result of CliqueAggImpl (0.1) is significantly higher: almost 60% for skitter and over 40% for web-BerkStan.

We also observe that difference between the membership distributions of the results of CliqueAggImpl (0.9) and CliqueAggImpl (1.0) is quite small. What does differ significantly, however, is the *maximum* number of cluster a vertex belongs to. To study this phenomenon further, for any clustering C of the vertices of a graph $G = (V, E)$, we define the *maximum membership* of C as

$$\mathcal{M}(C) = \max_{v \in V} \sum_{C \in C} \mathbb{1}_{v \in C}. \tag{3}$$

Note that the maximum membership can be defined for any clustering and is exactly 1 for any non-overlapping clustering. To compare our results to maximal clique enumeration we define *the relative maximum membership* at threshold ρ as

$$\mathcal{R}_{MC}(\rho) = \frac{\mathcal{M}(CliqueAggImpl(\rho))}{\mathcal{M}(MC)}. \tag{4}$$

We slightly abuse notation to refer to the clustering induced by the respective methods on the RHS. In Table 3, we present relative maximum membership for three different density thresholds and the maximum membership of maximal cliques. Datasets for which we did not get the maximal clique count are excluded.

We note that in over half of the datasets we worked with, the maximum membership at threshold of 0.1 is two orders of magnitude lower than the maximal membership of maximal cliques. Even at threshold 0.9 this drop is still clearly visible, as the mean reduction is over 5.2×. Overall, the results show that a clique aggregator indeed reduces the cluster overlap compared to maximal clique enumeration. For $\rho = 0.1$ this is visible across the membership distribution. For 0.9 the difference is less pronounced. However, the maximum number of clusters a vertex belongs to is up to 100× less.

Dataset	$\mathcal{R}_{MC}(0.1)$	$\mathcal{R}_{MC}(0.5)$	$\mathcal{R}_{MC}(0.9)$	$\mathcal{M}$ (MC)
euemail	0.3%	1.8%	14.5%	16.0k
Wiki-Vote	0.3%	2.5%	29.3%	172k
cit-HepTh	6.3%	10.3%	43.2%	37.7k
soc-Epinions1	0.5%	3.0%	23.0%	518k
Slashdot	0.7%	1.6%	2.6%	322k
web-BerkStan	13.9%	54.2%	72.6%	801k
youtube	12.0%	24.5%	63.6%	237k
pokec	25.3%	34.9%	40.4%	57.8k
skitter	0.2%	0.5%	0.9%	15.6M
wiki	13.3%	56.1%	87.7%	1.79M
WikiTalk	0.3%	0.3%	4.7%	32.3M
cit-Patents	0.2%	0.7%	28.7%	345k

Table 3: The maximum membership of the maximal clique induced clustering, and the relative maximum membership of our aggregators at different thresholds, in percentages.

6.1.4 Pseudocliques. A *pseudoclique* is a cluster that is almost a clique, i.e., it may be missing a few edges. The exact definition of a pseudoclique varies depending on the application. We attempted to compare with the very recent FPCE algorithm by Rahman et al. [39] which produces maximal subgraphs of density at least θ, a user-specified threshold. On the majority of datasets tested, FPCE does not terminate in five hours, completing only on our smallest graphs: euemail. Even in this tiny graph, FPCE produces *millions* of maximal pseudocliques at density 0.9; in comparison, the number of maximal cliques is about 43 thousand. While it remains algorithmically interesting, this demonstrates that maximal pseudoclique enumeration on real-world social networks is a prohibitively expensive task. We note that with the density threshold set to 1.0, QC, CliqueAggImpl, and FPCE produce the same output. However, as we decrease the threshold, the task solved by FPCE becomes

Dataset	#MC	CliqueAggImpl (0.1)		CliqueAggImpl (0.5)		CliqueAggImpl (0.9)		Louvain			LambdaCC			AGM		
		#Cl.	$\hat{\rho}$	#Cl.	$\hat{\rho}$	#Cl.	$\hat{\rho}$	#Cl.	%Cov'd	$\hat{\rho}$	#Cl.	%Cov'd	$\hat{\rho}$	#Cl.	%Cov'd	$\hat{\rho}$
euemail	43k	969	0.69	1301	0.76	11k	0.95	20	7%	0.40	246	46%	0.64	5	6%	0.20
Wiki-Vote	459k	7k	0.64	29k	0.84	187k	0.95	60	15%	0.71	3k	27%	0.48	2	82%	0.12
citHepTh	465k	27k	0.59	58k	0.79	215k	0.96	2k	53%	0.68	4k	26%	0.60	418	26%	0.12
soc-Epinions1	1.8M	75k	0.80	179k	0.85	651k	0.95	11k	51%	0.73	26k	58%	0.58	827	4%	0.09
Slashdot	890k	84k	0.65	311k	0.91	470k	0.96	8k	61%	0.71	21k	53%	0.50	609	8%	0.11
web-BerkStan	3.4M	689k	0.84	1.1M	0.86	2.7M	0.98	41k	27%	0.71	178k	12%	0.54	5k	26%	0.21
hollywood	DNF	591k	0.68	14M	0.73	OOM	OOM	41k	-	0.97	130k	-	0.82	16k	-	0.53
youtube	3.3M	1.1M	0.84	1.7M	0.90	2.8M	0.96	172k	61%	0.74	479k	28%	0.59	6k	44%	0.04
pokec	19.4M	1.9M	0.49	11M	0.93	17M	0.97	22k	68%	0.78	316k	14%	0.48	1701	54%	0.13
skitter	37.3M	1.7M	0.65	3.7M	0.85	8.9M	0.96	90k	44%	0.67	520k	34%	0.63	8k	25%	0.08
wiki	27.2M	1.8M	0.45	11M	0.89	21M	0.97	5k	53%	0.68	417k	9%	0.51	2k	35%	0.09
WikiTalk	86.3M	2.4M	0.90	3M	0.91	8M	0.94	54k	3%	0.61	2.2M	88%	0.59	8k	3%	0.02
orkut	DNF	8.6M	0.74	55M	0.87	238M	0.95	4k	-	0.54	380k	-	0.51	4k	-	0.16
cit-Patents	14.8M	3.8M	0.57	7.1M	0.81	13M	0.96	432k	57%	0.58	680k	39%	0.46	25k	90%	0.04
livejournal	DNF	4M	0.67	11M	0.87	24M	0.96	243k	-	0.68	1M	-	0.57	38k	-	0.12
soc-twitter	DNF	24.9M	0.74	149M	0.78	DNF	DNF	2.3M	-	0.76	7.1M	-	0.57	DNF	DNF	DNF

Table 4: Comparison of CliqueAggImpl with Louvain, LambdaCC, and AGM: cluster count (#Cl.), fraction of covered cliques (%Cov'd) and average density in output sets of size at least 3. #MC is the number of maximal cliques. Note that for CliqueAggImpl we do not report the fraction of covered cliques, since it's 100% in each case. DNF means that the computation did not finish within 5 hours. For the cases where we did not successfully compute the number of maximal cliques, we could not compute the fraction of covered cliques (marked with an -). For hollywood, CliqueAggImpl (0.9) runs into an out of memory error on our machine, marked with an OOM. For orkut, RMCE estimated over 2.2 billion cliques, and we were unable to compute individual coverage numbers with the constraints on disk space and memory on our machine.

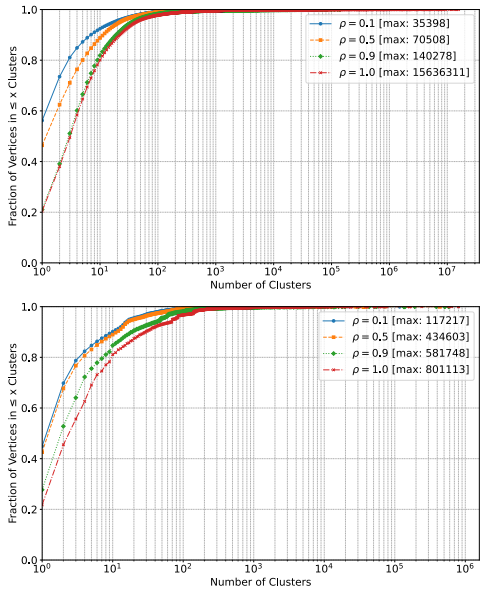

Figure 4: The number of clusters each vertex belongs to (defined as membership distributions in Section 6.1.3) in aggregators of different thresholds for two datasets: skitter (top) and web-BerkStan (bottom).

harder, while computing a clique aggregator becomes significantly easier. This points to the advantages of our model over vanilla enumeration techniques.

6.2 Comparisons with clustering techniques

In this section we study the difference between the commonly used clustering approaches and the task of computing an aggregator. Since clustering algorithms typically attempt to build dense clusters, we want to understand how well they cover maximal cliques. We consider three scalable clustering algorithms:

- Louvain [8]: An extremely popular clustering algorithm used for community detection, which uses a local-search approach maximize the celebrated modularity objective [36].
- LambdaCC [49]: An algorithm optimizing a generalized correlation clustering objective called LambdaCC. Given a graph G and a parameter $\lambda > 0$, the objective of a non-overlapping clustering $\{C_1, \ldots, C_k\}$ is:[3]

$$\sum_{i=1}^{k} |E(G[C_i])| - \lambda \binom{C_i}{2}.$$

  The algorithm uses a similar local search to Louvain. The algorithm provides a (soft) guarantee on edge density of each cluster. Namely, for a given value of λ, the objective of a cluster is positive iff the density of the cluster is $> \lambda$. For the experiments, we use a scalable parallel implementation [45].
- AGM [2]: The algorithm uses personalized PageRank to find initial clusters, which are then combined to minimize edges between them. This approach is for distributed computation, allowing graph problems to be solved on separate machines with reduced communication.

We use the default parameter settings for the Louvain algorithm suggested in their high-performance implementation. For LambdaCC, we set $\lambda = 0.1$, in order to obtain clusters of density at least 0.1. For AGM, we set the parameter to be 1000; to the best of our knowledge, there is no real analogue for density given the drastically different use case.

The results are presented in Table 4. We observe that the average density of the clusters computed by CliqueAggImpl is much higher

[3]Note that for the purpose of this paper we can state a simplified formula, since we work solely with unweighted graphs.

than the provided density threshold. For example, for $\rho = 0.1$ the average cluster density is at least 0.45 on each dataset. We also observe that CliqueAggImpl uses orders of magnitude more clusters than the clustering baselines. The clustering methods cover on average 30% (Louvain or LambdaCC) or 20% (AGM) of all the cliques. The lower coverage of AGM is despite the fact that it uses very sparse clusters, which in addition may overlap.

The large gap in the number of clusters between our method and the clustering baselines raises an interesting question whether the aggregator sizes could be further reduced.

6.3 Ablation Studies

In this section, we evaluate the influence of different components of our algorithm on the performance. Namely, we focus on the following two factors:

(1) *Bron-Kerbosch Pruning:* Our implementation supports a pruning strategy, derived from the Bron-Kerbosch algorithm to ensure that the computed aggregator is inclusion-maximal. The corresponding code is marked in teal in Algorithm 1 and Algorithm 2. While pruning is standard in most clique enumeration techniques, it can be turned off without affecting correctness. By default, it is turned on.

(2) *Bitmap:* As described in Section 5 our implementation by default uses a bit-matrix to represent graphs in the recursive calls of the algorithm. We compare this setting a more standard approach of using a compressed sparse row representation [17, 41]. We keep this on by default.

To examine the effects of these, we perform an ablation study and compare the time taken with and without each of these optimizations. The results, averaged over all graphs (when completed) and density thresholds $\rho \in \{0.1, 0.5, 0.9\}$ using the geometric mean are presented in Table 5.

Our experiments show that the bitmap representation makes the biggest difference in our performance. The settings with the bitmap off and pruning turned on are the slowest: it is on average about 1.5 times slower than the default settings we choose. The setting with both the bitmap and pruning turned off typically takes a little bit more than the default as well, but it is quite close.

Configuration	0.1	0.5	0.9
Bitmap Off, Pruning On	1.35× (0.02)	1.45× (0.01)	1.67× (0.02)
Bitmap Off, Pruning Off	0.98× (0.03)	1.12× (0.01)	1.36× (0.02)
Bitmap On, Pruning Off	0.65× (0.01)	0.71× (0.00)	0.70× (0.01)
Bitmap On, Pruning On	1.00×	1.00×	1.00×

Table 5: The geometric mean of the relative slowdown at each density threshold. In brackets, we include the variance over these slowdown factors.

Quite surprisingly, Bron Kerbosch pruning makes our algorithm slower. However, pruning effectively reduces the sizes of our aggregators: this is particularly noticeable at the higher thresholds. The geometric mean of the increase in aggregator sizes without pruning (over settings that allow pruning) ranges from less than 2% of the size of the aggregator for $\rho = 0.1$, but it climbs to over 12% for $\rho = 0.9$. One notable exception is the hollywood dataset,

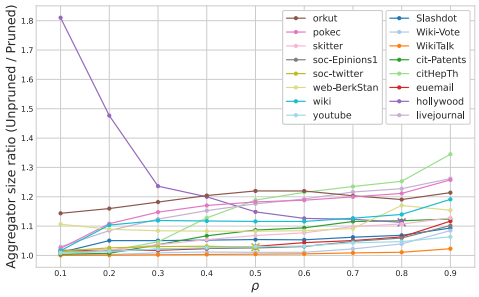

Figure 5: The increase in cluster count in our aggregators at different thresholds when pruning is turned off.

where the number of clusters in the aggregator is over 80% higher when there is no pruning at $\rho = 0.1$ and this gradually drops to around 10%. We refer the reader to Figure 5 for a thorough analysis. Barring hollywood, the gentle climb in the cluster count without pruning is visible in most other datasets.

7 DISCUSSION AND FUTURE WORK

We introduced *ρ-dense aggregators* as a formulation for summarizing the structure of maximal cliques and showed that, contrary to a list of maximal cliques, clique aggregators of sub-exponential size always exist. We gave an algorithm that finds such aggregators and demonstrated that it is significantly faster than existing methods for enumerating maximal cliques while reducing redundancy.

There are several interesting lines for further research. It would be interesting to establish better bounds on the size of *ρ-dense aggregators* for special graph families, in particular for families that model social networks such as c-closed graphs [25]. On the practical side a natural question is whether the aggregator size could be further reduced, especially given that the average density of the clusters computed by our algorithm is much higher than the prescribed density threshold. Another challenge is to design and analyze a parallel version of our algorithm and evaluate its performance on real datasets. One can also study other forms of clique aggregators, for example, by relaxing the requirement of having a cluster that contains each clique, possibly just requiring a large intersection with each clique or covering a substantial fraction of the maximal cliques.

ACKNOWLEDGMENTS

SB was supported by NSF DMS-2023495 and CCF-1839317 while a PhD student at UC Santa Cruz for the majority of the duration of this project. SJ was supported by the NSF under grant no. 2127309 to the Computing Research Association for the CIFellows 2021 Project. HK was partially supported by ISF grant no.1156/23 and the Blavatnik Research Foundation. BS acknowledges a 2024 Google gift to the University of Utah supporting her work on the theoretical foundations of finding dense subgraphs. We also thank Sayan Bhattacharya for helpful discussions. Experiments were run courtesy of a GCP research credits grant.

REFERENCES

[1] Noga Alon and Joel H Spencer. 2016. *The probabilistic method*. John Wiley & Sons.

[2] Reid Andersen, David F. Gleich, and Vahab Mirrokni. 2012. Overlapping clusters for distributed computation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*.

[3] József Balogh, Robert Morris, and Wojciech Samotij. 2015. Independent sets in hypergraphs. *Journal of the American Mathematical Society* 28, 3 (2015), 669–709.

[4] József Balogh, Robert Morris, and Wojciech Samotij. 2018. The method of hypergraph containers. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*. World Scientific, 3059–3092.

[5] Eric Blais and Cameron Seth. 2023. Testing Graph Properties with the Container Method. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. 1787–1795.

[6] Eric Blais and Cameron Seth. 2024. New Graph and Hypergraph Container Lemmas with Applications in Property Testing. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing (STOC 2024)*. 1793–1804.

[7] Jovan Blanuša, Radu Stoica, Paolo Ienne, and Kubilay Atasu. 2020. Manycore clique enumeration with fast set intersections. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2676–2690.

[8] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (Oct. 2008).

[9] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. 2005. Statistical analysis of financial networks. *Computational statistics & data analysis* 48, 2 (2005), 431–443.

[10] Vladimir Boginski, Sergiy Butenko, and Panos M Pardalos. 2006. Mining market data: A network approach. *Computers & Operations Research* 33, 11 (2006), 3171–3184.

[11] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[12] Zi Chen, Long Yuan, Li Han, and Zhengping Qian. 2021. Higher-order truss decomposition in graphs. *IEEE Transactions on Knowledge and Data Engineering* 35, 4 (2021), 3966–3978.

[13] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223.

[14] Alessio Conte, Daniele De Sensi, Roberto Grossi, Andrea Marino, and Luca Versari. 2018. Discovering k-trusses in large-scale networks. In *2018 IEEE high performance extreme computing conference (HPEC)*. IEEE, 1–6.

[15] Mina Dalirrooyfard, Surya Mathialagan, Virginia Vassilevska Williams, and Yinzhan Xu. 2024. Towards optimal output-sensitive clique listing or: Listing cliques from smaller cliques. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*. 923–934.

[16] Wen Deng, Weiguo Zheng, and Hong Cheng. 2024. Accelerating Maximal Clique Enumeration via Graph Reduction. *Proc. VLDB Endow.* 17, 10 (June 2024), 2419–3431. https://doi.org/10.14778/3675034.3675036

[17] Laxman Dhulipala, Jessica Shi, Tom Tseng, Guy E. Blelloch, and Julian Shun. 2020. The Graph Based Benchmark Suite (GBBS). In *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA'20)*. Association for Computing Machinery, Article 11. https://doi.org/10.1145/3398682.3399168

[18] Rodney G. Downey and Michael R. Fellows. 1995. Fixed-parameter tractability and completeness II: On completeness for W[1]. *Theoretical Computer Science* 141, 1-2 (1995), 109–131. https://doi.org/10.1016/0304-3975(94)00097-3

[19] Rodney G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity*. Springer, New York, NY, USA. https://doi.org/10.1007/978-1-4612-0515-9

[20] Marco D'Elia, Irene Finocchi, and Maurizio Patrignani. 2023. Clique-TF-IDF: a new partitioning framework based on dense substructures. In *International Conference of the Italian Association for Artificial Intelligence*.

[21] Marco D'Elia, Irene Finocchi, and Maurizio Patrignani. 2025. Maximal cliques summarization: Principles, problem classification, and algorithmic approaches. *Computer Science Review* 58 (2025), 100784.

[22] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing all maximal cliques in sparse graphs in near-optimal time. *International Symposium on Algorithms and Computation* (2010), 403–414.

[23] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)* 18 (2013), 3–1.

[24] Katherine Faust and Stanley Wasserman. 1995. *Social network analysis: Methods and applications*. Vol. 1. Cambridge university press Cambridge.

[25] Jacob Fox, Tim Roughgarden, C Seshadhri, Fan Wei, and Nicole Wein. 2020. Finding cliques in social networks: A new distribution-free model. *SIAM J. Comput.* 49, 2 (2020), 448–464.

[26] Johan Håstad. 1999. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica* 182, 1 (1999), 105–142. https://doi.org/10.1007/BF02392825

[27] Shweta Jain and C Seshadhri. 2020. Provably and efficiently approximating near-cliques using the Turán shadow: PEANUTS. In *Proceedings of the Web Conference 2020*. 1966–1976.

[28] Matthew Jenssen, Will Perkins, and Aditya Potukuchi. 2023. Approximately counting independent sets in bipartite graphs via graph containers. *Random Structures & Algorithms* 63, 1 (2023), 215–241.

[29] Richard M Karp. 2009. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*. Springer, 219–241.

[30] Daniel J. Kleitman and Kenneth J. Winston. 1982. On the number of graphs without 4-cycles. *Discrete Mathematics* 41, 2 (1982), 167–172.

[31] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[32] Irene Malvestio, Alessio Cardillo, and Naoki Masuda. 2020. Interplay between k-core and community structure in complex networks. *Scientific reports* 10, 1 (2020), 14702.

[33] George Manoussakis. 2018. An output sensitive algorithm for maximal clique enumeration in sparse graphs. In *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 27–1.

[34] 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.

[35] John W Moon and Leo Moser. 1965. On cliques in graphs. *Israel Journal of Mathematics* 3, 1 (1965), 23–28.

[36] M. E. J. Newman. 2006. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences* 103, 23 (2006).

[37] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. 2011. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks: Communication and Social Networks*. Springer, 143–162.

[38] 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*. 2479–2490.

[39] 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*. 2479–2490.

[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*. http://networkrepository.com

[41] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems* (2nd ed.). Society for Industrial and Applied Mathematics, USA.

[42] David Saxton and Andrew Thomason. 2015. Hypergraph containers. *Inventiones mathematicae* 201, 3 (2015), 925–992.

[43] John Scott. 1992. *Network analysis: A handbook*. Sage Publications.

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

[45] Jessica Shi, Laxman Dhulipala, David Eisenstat, Jakub Łącki, and Vahab Mirrokni. 2021. Scalable community detection via parallel correlation clustering. *Proc. VLDB Endow.* 14, 11 (July 2021), 2305–2313.

[46] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical computer science* 363, 1 (2006), 28–42.

[47] Armin Töpfer, Tobias Marschall, Rowena A Bull, Fabio Luciani, Alexander Schönhuth, and Niko Beerenwinkel. 2014. Viral quasispecies assembly via maximal clique enumeration. *PLoS computational biology* 10, 3 (2014), e1003515.

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

[49] Nate Veldt, David F. Gleich, and Anthony Wirth. 2018. A Correlation Clustering Framework for Community Detection. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. 439–448.

[50] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2025. Maximal Clique Enumeration with Hybrid Branching and Early Termination . In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 599–612. https://doi.org/10.1109/ICDE65448.2025.00051

[51] Xuyun Wen, Wei-Neng Chen, Ying Lin, Tianlong Gu, Huaxiang Zhang, Yun Li, Yilong Yin, and Jun Zhang. 2016. A maximal clique based multiobjective evolutionary algorithm for overlapping community detection. *IEEE Transactions on Evolutionary Computation* 21, 3 (2016), 363–377.

[52] Or Zamir. 2023. Algorithmic Applications of Hypergraph and Partition Containers. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*. 985–998.

[53] Bing Zhang, Byung-Hoon Park, Tatiana Karpinets, and Nagiza F Samatova. 2008. From pull-down data to protein interaction networks and complexes with biological relevance. *Bioinformatics* 24, 7 (2008), 979–986.

[54] David Zuckerman. 2006. Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, Jon M. Kleinberg (Ed.). ACM, 681–690. https://doi.org/10.1145/1132516.1132612