KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs

Bintao Sun
The University of Hong Kong
Hong Kong SAR, China
btsun@connect.hku.hk

Maximilien Danisch*
Sorbonne Université, CNRS,
LIP6, F-75005 Paris, France
maximilien.danisch@lip6.fr

T-H. Hubert Chan†
The University of Hong Kong
Hong Kong SAR, China
hubert@cs.hku.hk

Mauro Sozio‡
Télécom Paris, IP Paris
Paris, France
sozio@telecom-paris.fr

ABSTRACT

The problem of finding densest subgraphs has received increasing attention in recent years finding applications in biology, finance, as well as social network analysis. The k-clique densest subgraph problem is a generalization of the densest subgraph problem, where the objective is to find a subgraph maximizing the ratio between the number of k-cliques in the subgraph and its number of nodes. It includes as a special case the problem of finding subgraphs with largest average number of triangles ($k = 3$), which plays an important role in social network analysis. Moreover, algorithms that deal with larger values of k can effectively find quasi-cliques. The densest subgraph problem can be solved in polynomial time with algorithms based on maximum flow, linear programming or a recent approach based on convex optimization. In particular, the latter approach can scale to graphs containing tens of billions of edges. While finding a densest subgraph in large graphs is no longer a bottleneck, the k-clique densest subgraph remains challenging even when $k = 3$. Our work aims at developing near-optimal and exact algorithms for the k-clique densest subgraph problem on large real-world graphs. We give a surprisingly simple procedure that can be employed to find the maximal k-clique densest subgraph in large-real world graphs. By leveraging appealing properties of existing results, we combine it with a recent approach for listing all k-cliques in a graph and a sampling scheme, obtaining the state-of-the-art approaches for the aforementioned problem. Our theoretical results are complemented with an extensive experimental evaluation showing the effectiveness of our approach in large real-world graphs.

1. INTRODUCTION

Algorithms for finding dense subgraphs have emerged as an important subroutine in a wide variety of data mining applications, in particular, in biology [14] and finance [11], while they have been employed for social network analysis [3], reachability and distance query indexing [8, 22] and many other tasks.Many of those applications require finding quasi-cliques, i.e. graphs where "almost" every pair of nodes is connected with an edge. In [30], the author formulated and studied the k-clique densest subgraph problem, where one wishes to find a subgraph with maximum k-*clique* density, defined as the ratio between its number of k-cliques and its number of nodes. Such a problem is a generalization of the *densest subgraph* problem ($k = 2$) and the problem of finding subgraphs with largest average number of triangles ($k = 3$). It has been experimentally demonstrated in [30] and [9] that as k increases, a subgraph with large k-clique density quickly approaches a quasi-clique. Hence, algorithms for finding k-clique densest subgraphs provide powerful tools for social network analysis.

The densest subgraph problem can be solved in polynomial time with algorithms based on maximum flow, linear programming (LP) or a recent approach based on convex optimization [10]. In particular, the last approach can scale to graphs containing up to tens of billions of edges. While finding a densest subgraph in large graphs is no longer a bottleneck, the k-clique densest subgraph remains challenging even when $k = 3$. It can be solved in polynomial time when k is a constant, however, exact algorithms suffer from scalability issues even for relatively small graphs and relatively small values of k. On the other hand, the greedy approximation algorithm for such a problem [30] comes with the somehow disappointing approximation guarantee of $\Theta(k)$. Our work aims at developing exact and near-optimal algorithms for the k-clique densest subgraph problem on large real-world graphs. This might enable a more effective discovery of quasi-cliques which in turn might lead to the discovery of more interesting patterns in the data.

When adapting exact algorithms for the densest subgraph problem to the k-clique densest subgraph problem, one has to either introduce some variables for every k-clique in the graph (in the LP-based approach) or some nodes for each such a clique in the maximum flow network. This is clearly not feasible for large values of k. Our first contribution is a surprisingly simple algorithm for computing the maximal k-clique densest subgraph, requiring linear amount of memory in the size of the graph for any $k \geq 2$. The algorithm for computing the maximal densest subgraph ($k = 2$) proceeds

*Funded by the ANR (French National Agency of Research) under the LiMass JCJC and the FiT LabCom projects.
†T-H. Hubert Chan was partially supported by the Hong Kong RGC under the grants 17200817.
‡This work has been carried out in the frame of a cooperation between Huawei Technologies France SASU and Telecom Paris (Grant no. YBN2018125164).

as follows. We assign a score $r(u)$ to every node u in the graph, which is initialized to 0. Then in each iteration, we process the edges in some arbitrary order. When processing edge uv we add 1 to the lower score between $r(u)$ and $r(v)$, breaking ties arbitrarily. The generalization to compute the maximal k-clique densest subgraph is straightforward and does not require any additional memory: we process all k-cliques in a sequential fashion, each time adding 1 to the minimum score among all r scores of the nodes in the k-clique. We show that when the number of iterations is sufficiently large, the nodes in the maximal k-clique densest subgraph have larger scores than the others. This allows for an efficient extraction of the k-clique maximal densest subgraph. In contrast to the algorithm developed in [10] for the densest subgraph problem, our algorithm is not a gradient-descent-like algorithm, so we need different techniques to prove its convergence. Our algorithm employs KCLIST [9] as a subroutine, which is the state-of-the-art algorithm for listing all k-cliques in a graph, while requiring linear amount of memory in the size of the input graph. We call our algorithm KCLIST++ as it uses the KCLIST algorithm as a building block and the ++ operator. In addition, borrowing ideas from [25], we give an alternative solution to resolve the memory issue by taking only a small fraction of k-cliques into consideration via sampling.

Our theoretical results are complemented with an extensive experimental evaluation showing the effectiveness of our approach in large real-world graphs. Our experimental evaluation shows for example that a parallel version of KCLIST++ can find near-optimal k-clique densest subgraphs in graphs containing up to more than one billion edges and for $k = 9$.

Related Work. Dense subgraphs detection has been widely studied [23]. Such a problem aims at finding a subgraph of a given input graph that maximizes some notion of density. The most common density notion employed in the literature is the average degree. Due to its popularity, the corresponding problem of finding a subgraph that maximizes the average degree has been commonly referred to as the densest subgraph problem. The densest subgraph can be identified in polynomial time by solving a parametric maximum flow problem [15], while a simple greedy algorithm based on core decomposition produces a 2-approximation in linear time [7]. It is worth mentioning that [16] and subsequent work [17, 2] show an interesting connection between load balancing and the densest subgraph problem. An algorithm based on convex optimization has also been introduced in [10] and has been shown to scale to graphs containing tens of billions of edges. The densest subgraph problem has also been studied in evolving graphs [5, 12].

In [30], the author studies the k-clique densest subgraph problem, which is a generalization of the densest subgraph problem and consists of finding a subgraph with maximum ratio between the number of its k-cliques and its number of nodes. In particular, the author generalizes the greedy 2-approximation algorithm to a k-approximation algorithm for the k-clique densest subgraph problem. Later in [25], the authors propose a sampling scheme to deal with a large number of k-cliques.

Other notions of density have also been investigated such as the minimum degree density [28] or density based on triangles [32] which can be solved in polynomial time. Other densities leading to NP-hard problems have also been investigated such as the k-densest subgraph problem, which consists of finding a densest subgraph of k nodes [4] and quasi-clique detection [31, 1].

2. PRELIMINARIES

We are given an unweighted undirected graph $G = (V, E)$, with n, m denoting its number of nodes and edges, respectively. Define $[n] := \{1, 2, \ldots, n\}$. We interpret each edge $e \in E$ as a subset $e \subseteq V$ of nodes. A k-clique of G is a subset of nodes, $C \subseteq V$, such that $|C| = k$ and every two distinct nodes in C are adjacent. Denote $\mathcal{C}_k(G) := \{C \subseteq V : C \text{ is a } k\text{-clique of } G\}$ as the collection of k-cliques of G. Note that $\mathcal{C}_2(G) = E$. The k-clique density of a subgraph $H = (V_H, E_H)$ in G is defined as $\rho_k(H) := \frac{|\mathcal{C}_k(H)|}{|V_H|}$. A subgraph H of G is called a *k-clique densest subgraph* if H has the maximum k-clique density among all subgraphs of G.

Problem Definition. Given an undirected graph $G = (V, E)$ and a positive integer k, find its (exact or approximate) k-clique densest subgraph.

For a non-empty $S \subseteq V$, we define the induced subgraph $G[S] := (S, E(S))$ by S in G, where $E(S) := \{e \in E : e \subseteq S\}$ denotes the set of edges in G contained in S. A set S is called a *k-clique densest subset* in G if it induces a k-clique densest subgraph of G.

If we consider a hypergraph where the set of nodes is V and the set of hyperedges is $\mathcal{C}_k(G)$, then k-clique density in G is equivalent to density in this hypergraph. Lemma 4.1 of [6], which focuses on normal graphs, can be readily generalized to the following fact.

FACT 1. *We define the density of an (edge-)weighted hypergraph $H = (V_H, E_H, w_H)$ as $\frac{\sum_{e \in E_H} w_H(e)}{|V_H|}$. The maximal densest subhypergraph of H is unique. It is induced by some subset of nodes and contains all densest subhypergraphs of H.*

We also briefly mention the notion of *quotient hypergraph* and *k-clique-diminishingly-dense decomposition* generalized from [29, 10].

DEFINITION 2 (QUOTIENT HYPERGRAPH) .*Given an edge-weighted hypergraph $H = (V, E, w)$ and a subset $B \subseteq V$, the quotient hypergraph of H with respect to B is a hypergraph $H \backslash B = (\widehat{V}, \widehat{E}, \widehat{w})$, which is defined as follows.*

- $\widehat{V} := V \setminus B$.
- $\widehat{E} := \{e \cap \widehat{V} : e \in E, e \cap \widehat{V} \neq \varnothing\}$, *i.e., every hyperedge $e \in E$ not contained in B contributes towards $\widehat{E}$.*
- *For $e' \in \widehat{E}$, $\widehat{w}(e') := \sum_{e \in E : e' = e \cap \widehat{V}} w(e)$.*

DEFINITION 3 (k-CLIQUE-DIMINISHINGLY-DENSE DECOMPOSITION) .*Given an unweighted undirected graph $G = (V, E)$, we define the k-clique-diminishingly-dense decomposition $\mathcal{B}$ of G as the sequence $\varnothing = B_0 \subsetneq B_1 \subsetneq B_2 \subsetneq \cdots \subsetneq B_k = V$ as follows. Consider a weighted hypergraph $H = (V, \mathcal{C}_k(G), w)$ where $w(C) := 1$ for every $C \in \mathcal{C}_k(G)$.*

- *Initially, $B_0 := \varnothing$ and $H_0 := H$.*
- *For $i \geq 1$, if $B_{i-1} = V$, the decomposition is fully defined. Otherwise, let $H_i := H_{i-1} \backslash B_{i-1} = (V_i, E_i, w_i)$ be the quotient graph of H_{i-1} with respect to B_{i-1}. Let S_i be the maximal densest subset in H_i. Define $B_i := B_{i-1} \cup S_i$.*

3. ALGORITHMS

In this section, we present our main algorithms for computing maximal k-clique densest subgraphs. We start by summarizing the Frank-Wolfe based algorithm for finding the maximal densest subgraph which has been developed in [10] (Section 3.1). Then we consider a straightforward generalization, which comes with the drawback of requiring a large amount of memory (Section 3.2): it requires memory proportional to the number of k-cliques in the input graph, making it unfeasible to deal with large graphs and large values of k. Such a problem is tackled by the algorithms presented in Section 3.3 and Section 3.4, which handle the task of computing a near-optimal solution. This represents one of the main contributions of our work. In Section 3.5, we present an algorithm for exactly computing the maximal k-clique densest subgraph for relatively small values of k.

3.1 Frank-Wolfe Based Algorithm

For the sake of self-containment, we summarize the Frank-Wolfe based algorithm for finding the maximal densest subgraph developed in [10]. Such an algorithm maintains two variables α_u^e and α_v^e for each edge $e = uv$, as well as a variable $r(u)$ for each node u. The α_u^e's are initialized to $\frac{1}{2}$. Throughout the execution of the algorithm, $r(u)$ stores the sum over all α_u^e's such that e contains u. At each step, the α's are modified simultaneously as follows. For each edge uv, such that $r(u) \leq r(v)$, let $\widehat{\alpha}_u^e$ and $\widehat{\alpha}_v^e$ be equal to 1 and 0, respectively, breaking ties arbitrarily. A new value for the α's is computed as a convex combination between their value in the previous step and the $\widehat{\alpha}$'s (Line 11 of Algorithm 1). Such an algorithm is an instance of the well known Frank-Wolfe algorithm, which is widely used in convex optimization [19].

Algorithm 1: FW-Based Algorithm for k-clique Densest

Input: $G = (V, E)$, number T of iterations
Output: $\alpha \in \mathbb{R}_+^{\mathcal{C}_k(G) \times V}$ and $r \in \mathbb{R}_+^V$

1 **for** *each k-clique* C **do**
2 $\quad \alpha_u^{C\,(0)} \leftarrow \frac{1}{k}, \forall u \in C$
3 **for** *each* $u \in V$ **do**
4 $\quad r^{(0)}(u) \leftarrow \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^{C\,(0)}$
5 **for** *each iteration* $t = 1, \ldots, T$ **do**
6 $\quad \gamma_t \leftarrow \frac{2}{t+2}$
7 $\quad$ **for** *each k-clique* C **do**
8 $\quad\quad x \leftarrow \arg\min_{v \in C} r^{(t-1)}(v)$
9 $\quad\quad$ **for** *each* $u \in C$ **do**
10 $\quad\quad\quad \widehat{\alpha}_u^C \leftarrow 1$ if $u = x$ and 0 otherwise.
11 $\quad \alpha^{(t)} \leftarrow (1 - \gamma_t) \cdot \alpha^{(t-1)} + \gamma_t \cdot \widehat{\alpha}$
12 $\quad$ **for** *each* $u \in V$ **do**
13 $\quad\quad r^{(t)}(u) \leftarrow \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^{C\,(t)}$
14 **return** $(\alpha^{(T)}, r^{(T)})$

It is shown in [10] that if the number of iterations T is "large enough", then for some $s > 0$, the s nodes with largest r values induce a maximal densest subgraph (see [10, Lemma 4.3 and Corollary 4.9]). Therefore, in that case, the maximal densest subgraph can be found efficiently from the $r(u)$'s. The efficiency of the algorithm has been demonstrated in [10] by an extensive experimental evaluation.

3.2 Large Memory Approximation Algorithm

The algorithm presented in [10] can be generalized so as to compute the k-clique densest subgraph by introducing k variables for each k-clique in the graph and updating them in a similar way. A full pseudocode is shown in Algorithm 1, whose proof of convergence is omitted for space constraints. Observe that Algorithm 1 requires a prohibitively large amount of memory for large values of k, in that, it requires to keep all k-cliques as well as their corresponding variables in main memory. This is clearly not feasible, as the number of k-cliques might grow exponentially in k. For instance, when running Algorithm 1 on some of our datasets (see Table 1), it runs out of memory for Friendster and LiveJournal when $k = 4$ and for Orkut when $k = 5$, while it would require more than 250 terabytes of main memory for Livejournal when $k = 6$.

Therefore, some novel techniques are needed in order to deal with large graphs and large values of k. In Section 3.3 and Section 3.4, we present two variants of Algorithm 1 optimized for memory consumption, which is among the main contributions of our work.

Our algorithm requires a subroutine that list all k-cliques in large input graphs. To this end, we employ kClist [9], which is the state-of-the-art algorithm for such a task. Such an algorithm is appealing in that it requires linear memory in the size of the graph for any $k \geq 2$, listing each k-clique exactly once. Any algorithm ensuring the aforementioned properties can be employed in our approach. Another appealing property of kClist is its running time of $O(k \cdot m \cdot (\frac{c}{2})^{k-2})$, where c is the core value of the graph. This makes it suitable for large real-world graphs which are typically sparse (i.e. their core value is relatively small).

3.3 Linear Memory Approximation Algorithm

We first overcome the memory issue by means of a surprisingly simple algorithm. It processes the k-cliques in the input graph sequentially. At each step, the minimum r value of the nodes in the k-clique is increased by 1 (breaking ties arbitrarily), as if the k-clique is assigned to the node with least "load". By employing kClist [9], we process all k-cliques in the input graph using linear amount of main memory. We call our algorithm kClist++, as its main building blocks are the kClist algorithm and the ++ operator (which increases the value of a variable by 1 in programming languages such as C and Java). A pseudocode of it is shown in Algorithm 2.

Algorithm 2: kClist++

Input: $G = (V, E)$, number T of iterations
Output: $r \in \mathbb{R}_+^V$

1 $r(u) \leftarrow 0, \forall u \in V$
2 **for** $t \leftarrow 0, 1, \ldots, T-1$ **do**
3 $\quad$ **for** *each k-clique* C *in* G **do**
4 $\quad\quad u \leftarrow \arg\min_{v \in C} r(v)$
5 $\quad\quad r(u)$++
6 $r(u) \leftarrow r(u)/T, \forall u \in V$
7 **return** r

After running kClist++ for a sufficiently large number of iterations T, we sort the nodes according to their r values: $r(u_1) \geq r(u_2) \geq \cdots \geq r(u_n)$. In such an ordering, we examine the set of the first s nodes for each $s \in [n]$, and return a set inducing a subgraph with maximum k-clique density. More precisely, we let $y_i := |\{C \in \mathcal{C}_k(G) : i = \max\{j : u_j \in C\}\}|$ for each $i \in [n]$ (as in [10]) and return a value $s \in [n]$ that maximizes $\frac{1}{s}\sum_{i=1}^{s} y_i$. The latter value can be computed by means of another execution of kClist.

We show in Section 4.1 that the graph computed by our algorithm is the maximal k-clique densest subgraph when T is large enough. Additionally, the r values imply an upper bound on the optimal solution; see Section 4.2.

The algorithm stores a linear number of variables in the number of nodes (the $r(u)$'s) without storing any α_u^C's. Furthermore, unlike Algorithm 1, the $r(u)$ variables are updated sequentially. Consequently, any change in the r values is promptly taken into account. This turns out to be more efficient in practice, according to our experimental evaluation. Algorithm 3, an equivalent procedure to Algorithm 2 with α explicitly stored, further illustrates the differences and similarities between the two algorithms. We refer to this variant of the algorithm as Seq-kClist++ or simply as kClist++. For completeness, we consider a variant of kClist++ where the $r(u)$'s are updated simultaneously, which we refer to as Sim-kClist++ (see Algorithm 4 for a pseudocode).

Algorithm 3: kClist++ (with α variables)

Input: $G = (V, E)$, integer T, $\alpha \in \mathbb{R}_+^{\mathcal{C}_k(G)\times V}$ feasible (optional)
Output: $\alpha \in \mathbb{R}_+^{\mathcal{C}_k(G)\times V}$ and $r \in \mathbb{R}_+^V$

if $\alpha = NULL$ **then**
  for *each k-clique* C **do**
    $\alpha_u^C \leftarrow \frac{1}{k}, \forall u \in C$
for *each* $u \in V$ **do**
  $r(u) \leftarrow \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^C$
for *each iteration* $t = 1, \ldots, T$ **do**
  $\gamma_t \leftarrow \frac{1}{t+1}$
  $\alpha \leftarrow (1 - \gamma_t) \cdot \alpha$, $r \leftarrow (1 - \gamma_t) \cdot r$
  for *each k-clique* C **do**
    $u \leftarrow \arg\min_{v \in C} r(v)$
    $\alpha_u^C \leftarrow \alpha_u^C + \gamma_t$, $r(u) \leftarrow r(u) + \gamma_t$
return (α, r)

Algorithm 4: Sim-kClist++

Input: $G = (V, E)$, number T of iterations
Output: $r \in \mathbb{R}_+^V$

$r(u) \leftarrow 0, \forall u \in V$
for $t \leftarrow 0, 1, \ldots, T-1$ **do**
  $s(u) \leftarrow r(u), \forall u \in V$
  for *each k-clique* C *in* G **do**
    $u \leftarrow \arg\min_{v \in C} s(v)$
    $r(u)$++
$r(u) \leftarrow r(u)/T, \forall u \in V$
return r

3.4 Save Memory via Sampling

An alternative approach to save memory, shown in Algorithm 5, is inspired by [25]. In lieu of going through all k-cliques by kClist for many iterations, we just run kClist to count the number of k-cliques (this can also be done by any exact or approximate clique-counting subroutine, such as [13, 20, 21]) and then call it again to sample the k-cliques. Rather than setting the sampling probability directly, we set a parameter σ, the approximate number of k-cliques to be sampled. Once the number of k-cliques M is computed, we set the sampling probability as $p := \min\{\sigma/M, 1\}$, and each k-clique is stored into main memory independently with probability p. As a result, the number of sampled k-cliques will be very close to σ. In the subsequent iterations, we only perform the ++ operator for the sampled k-cliques. We call this variant Seq-Sampling++.

After that, an approximate k-clique densest subset and an upper bound on the optimal solution in the sampled graph can be identified in the same way as described in Section 3.3 (the y_i's corresponding to the sampled graph can be computed by making a pass over the sampled k-cliques stored in main memory). To restore the k-clique density of the subgraph induced by this set in the original graph, we need to call kClist once more on the induced subgraph.

Algorithm 5: Seq-Sampling++

Input: $G = (V, E)$, number T of iterations, rough number σ of cliques to be sampled
Output: $r \in \mathbb{R}_+^V$

$M \leftarrow |\mathcal{C}_k(G)|$
$p \leftarrow \min\{\sigma/M, 1\}$
$S \leftarrow \varnothing$
for *each k-clique* C *in* G **do**
  Include C in S independently with probability p
$r(u) \leftarrow 0, \forall u \in V$
for $t \leftarrow 0, 1, \ldots, T-1$ **do**
  for *each k-clique* C *in* S **do**
    $u \leftarrow \arg\min_{v \in C} r(v)$
    $r(u)$++
$r(u) \leftarrow r(u)/T, \forall u \in V$
return r

3.5 Exact k-Clique Densest Subgraph

In the case when neither k nor the input graph is "too large", we are able to compute an exact solution for the k-clique densest subgraph problem by employing Algorithm 3, which maintains for every k-clique k variables. Similarly to [10], one can derive from the α's an upper bound on the density of a densest subgraph, which in turn can be used to verify whether a given subgraph is densest.

Our exact algorithm (pseudocode shown in Algorithm 6) initializes the α's so that for any k-clique its corresponding variables sum up to 1 (Line 3). We call *feasible* any assignment satisfying such a constraint. Then, it runs T iterations of Algorithm 3 yielding refined values for the α's. Those are then used to tentatively extract a k-clique densest subset S. If S is *stable* (Line 7; see Definition 6), the optimality of S is then tested by means of an improved version of the Goldberg's condition (see [15] and Section 4.3) which states that the densities of two subgraphs cannot be "too close"

(Theorem 17) or a max-flow algorithm (Algorithm 9). If the optimality test fails, we double T and iterate.

The most interesting aspects of Algorithm 6 are perhaps the improved Goldberg's condition (Theorem 17) and the fact that it employs kClist++ where the α variables are kept in main memory (Algorithm 3). The algorithm for tentatively extracting a k-clique densest subgraph (Algorithm 7) is a rather straightforward generalization of the algorithms developed in [10]. We include most pseudocodes, except that of TryDecompose, which is already fully described in Section 3.2 of [10].

Algorithm 6: Exact Algorithm

Input: $G = (V, E)$, k
Output: $S \subseteq V$ inducing a k-clique densest subgraph in G

$T \leftarrow 1$
for *each k-clique C* **do**
  $\alpha_u^C \leftarrow \frac{1}{k}, \forall u \in C$
while *true* **do**
  $(\alpha, r) \leftarrow$ run Algorithm 3 with input (G, T, α)
  $(\alpha, r, S) \leftarrow \mathsf{TryExtractDensest}(G, \alpha, r)$
  if $\min_{u \in S} r(u) > \max_{v \in V \setminus S} r(v)$ **then**
    if *S satisfies condition in Theorem 17* **then**
      return S
    if $\mathsf{IsDensestMF}(G[S])$ **then**
      return S
  $T \leftarrow 2T$

Algorithm 7: TryExtractDdensest

Input: $G = (V, E)$, α, r
Output: α, r, tentative k-clique densest subset S

$(S_1, \ldots, S_l) \leftarrow \mathsf{TryDecompose}(G, r)$
$\alpha \leftarrow \mathsf{Squeeze}(G, k, \alpha, (S_1, \ldots, S_l))$
for $u \in V$ **do**
  $r(u) \leftarrow \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^C$
return (α, r, S_1)

Algorithm 8: Squeeze

Input: $G = (V, E)$, k, α, partition $(S_1, S_2, \ldots, S_l)$
Output: α

for $C \in \mathcal{C}_k(G)$ **do**
  `// Redistribute the weights as ''backward'' as possible`
  $p \leftarrow \max\{1 \le i \le l : C \cap S_i \neq \varnothing\}$
  $s \leftarrow \sum_{u \in C \setminus S_p} \alpha_u^C$
  $\forall u \in C \setminus S_p$, $\alpha_u^C \leftarrow 0$
  $\forall u \in C \cap S_p$, $\alpha_u^C \leftarrow \alpha_u^C + \frac{s}{|C \cap S_p|}$
return α

4. ANALYSIS

Inspired from Charikar's LP relaxation for densest subgraphs [7], Danisch et al. [10] proposed a convex program

Algorithm 9: Optimality Test by Max-Flow: IsDensestMF

Input: $G = (V, E)$
Output: Is G a k-clique densest subgraph in G?

$\mathcal{C}_k \leftarrow \mathcal{C}_k(G)$
$n \leftarrow |V|$, $M \leftarrow |\mathcal{C}_k|$
Construct a network with node set $\{s, t\} \cup V \cup \mathcal{C}_k$
$\forall C \in \mathcal{C}_k$, add an edge with capacity n from s to C
$\forall u \in V$, add an edge with capacity M from u to t
$\forall C \in \mathcal{C}_k$, $\forall u \in C$, add an edge with capacity n from C to u
$f \leftarrow$ maximum flow from s to t
return $f = nM$

to find the so-called *diminishingly-dense decomposition*, the first member of which is indeed the densest subgraph. The generalization to k-clique density is straightforward; we summarize the related results below without proof.

Consider the convex program

$$\mathsf{CP}(G, k) := \min\{Q_{G,k}(\alpha) : \alpha \in \mathcal{D}(G, k)\},$$

where the domain

$$\mathcal{D}(G, k) := \{\alpha \in \prod_{C \in \mathcal{C}_k(G)} \mathbb{R}_+^C : \forall C \in \mathcal{C}_k(G), \sum_{u \in C} \alpha_u^C = 1\}$$

and the objective function

$$Q_{G,k}(\alpha) := \sum_{u \in V} r(u)^2,$$

in which $r(u) = \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^C$, $\forall u \in V$. Intuitively, each k-clique C distributes its unit weight among all nodes in C, and $r(u)$ is the total weight received by u.

Definition 4 (Level Sets [10]) . *The collection of the level sets of a vector $r \in \mathbb{R}_+^V$ is defined as $\{S_\rho : \exists u \in V, r(u) = \rho\}$, where each level set is $S_\rho := \{u \in V : r(u) \ge \rho\}$.*

Proposition 5 ([10, Corollary 4.4]) . *Suppose an optimal solution α of $\mathsf{CP}(G, k)$ induces the vector $r \in \mathbb{R}_+^V$ via $r(u) = \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^C$, $\forall u \in V$. Then the level sets of r give the exact k-clique-diminishingly-dense decomposition of G. In particular, let $\rho^* := \max_{u \in V} r(u)$. Then S_{ρ^*} induces the maximal k-clique densest subgraph of G with k-clique density ρ^*.* □

Definition 6 (Stable Subset) . *A non-empty subset $B \subseteq V$ is stable with respect to $\alpha \in \mathcal{D}(G, k)$ if the following conditions hold.*

- *$\forall u \in B$ and $v \in V \setminus B$, $r(u) > r(v)$.*
- *$\forall C \in \mathcal{C}_k(G)$ s.t. C intersects both B and $V \setminus B$, $\forall u \in C \cap B$, $\alpha_u^C = 0$.*

Proposition 7 ([10, Lemma 4.11]) . *Suppose a non-empty subset $B \subseteq V$ is stable with respect to some $\alpha \in \mathcal{D}(G, k)$. Then B is a member of the k-clique-diminishingly-dense decomposition. In particular, the maximal k-clique densest subgraph of G is contained in $G[B]$.* □

4.1 Convergence Rate of kClist++

In this section, we study the convergence rate of kClist++. Recall that such an algorithm updates its variables sequentially and it is not a gradient-descent-like algorithm. As a result, we need different techniques from the one used in [10] to study its convergence rate.

From now on, we use the simplified notation $Q(\alpha) := Q_{G,k}(\alpha)$, $\mathcal{D} := \mathcal{D}(G,k)$ and $K := |\mathcal{C}_k(G)|$. Let

$$\Delta := \max_{v \in V} |\{C \in \mathcal{C}_k(G) : v \in C\}|$$

be the maximum k-clique degree.

The convergence rate will be represented by the *curvature* of the objective function [19], which is defined as

$$\xi_Q := \sup_{\substack{\alpha, s \in \mathcal{D}, \\ \gamma \in (0,1], \\ y = \alpha + \gamma(s-\alpha)}} \frac{2}{\gamma^2}(Q(y) - Q(\alpha) - \langle y - \alpha, \nabla Q(\alpha)\rangle).$$

It has been observed in [10, Lemma 4.7] that $\xi := 2\Delta K$ is an upper bound on ξ_Q.

Notice that the vector α, although discarded in the algorithm, is still lying behind: we pretend that we maintain r and α simultaneously and when we perform $r(u)$++ to the node $u = \arg\min_{v \in C} r(v)$, we see it as if α_u^C is also increased by 1. We use $r^{(t)}$ and $\alpha^{(t)}$ to denote vector r and the implicit α in Algorithm 2 when t rounds of iteration are completed, respectively.

Now suppose having completed t iterations, we pick the first clique C^1 from the stream in the $(t+1)$-st iteration. Formally speaking, we find the node u with minimum $r^{(t)}$ value within C^1 and let $s^{(t,1)} \in \mathbb{R}_+^{C^1}$ be the vector with value 1 at coordinate u and 0 elsewhere. Inductively, for the i-th clique C^i, $s^{(t,i)} \in \mathbb{R}_+^{C^i}$ is the vector with value 1 at coordinate u and 0 elsewhere, where u is the node within C^i with minimum r value induced by $\alpha^{(t,i)} := \alpha^{(t)} + \sum_{j=1}^{i-1} s^{(t,j)}$ (here we slightly abuse the operator $+$ by embedding each $s^{(t,j)}$ into $\mathcal{D} \subseteq \prod_{C \in \mathcal{C}_k(G)} \mathbb{R}_+^C$). Finally, letting $s^{(t)} := \sum_{i=1}^K s^{(t,i)} \in \mathcal{D}$, we have $\alpha^{(t+1)} = \alpha^{(t)} + s^{(t)}$.

Note that the r values between consecutive iterations in Algorithm 2 satisfies $\frac{r^{(t+1)}}{t+1} = \frac{1}{t+1}\left(r^{(t)} + \left(r^{(t+1)} - r^{(t)}\right)\right) = \frac{t}{t+1} \cdot \frac{r^{(t)}}{t} + \frac{1}{t+1}\left(r^{(t+1)} - r^{(t)}\right)$. Rewriting it in terms of $\alpha^{(t+1)}$, $\alpha^{(t)}$ and $s^{(t)}$, we have $\frac{\alpha^{(t+1)}}{t+1} = \frac{t}{t+1} \cdot \frac{\alpha^{(t)}}{t} + \frac{1}{t+1} \cdot s^{(t)}$. We aim to build the following variant of [19, Theorem 1]. To this end, Lemma 9 is the most technical part.

Theorem 8 (Convergence Rate) . *For each $t \geq 1$, $\alpha^{(t)}$ computed as above satisfies*

$$Q\left(\frac{\alpha^{(t)}}{t}\right) - Q(\alpha^*) \leq O\left(\frac{\log t}{t}\right) \cdot \xi(1+\delta),$$

where $\alpha^* \in \mathcal{D}$ is an optimal solution to $\mathsf{CP}(G,k)$, and $\delta = \sqrt{32k}$.

Lemma 9 (Approximate Linear Minimizer) . *Given any $t \geq 1$, denote*

$$\widehat{s} := (\widehat{s}^1, \ldots, \widehat{s}^K) := \arg\min_{\widetilde{s} \in \mathcal{D}} \left\langle \widetilde{s}, \nabla Q\left(\alpha^{(t)}\right)\right\rangle,$$

$\delta := \sqrt{32k}$, $\gamma_t := \frac{1}{t+1}$ and $\xi := 2\Delta K$. Then $s^{(t)}$ is an approximate linear minimizer, i.e.,

$$\left\langle s^{(t)}, \nabla Q\left(\frac{\alpha^{(t)}}{t}\right)\right\rangle \leq \left\langle \widehat{s}, \nabla Q\left(\frac{\alpha^{(t)}}{t}\right)\right\rangle + \frac{1}{2}\delta\gamma_t\xi.$$

Proof. We use $\nabla_C Q(\alpha)$ to denote the projection of $\nabla Q(\alpha)$ onto $\mathbb{R}^C$. By a straightforward calculation [10, proof of Lemma 4.5], The (C,u)-coordinate of $\nabla Q(\alpha)$ is

$$2r(u) = 2\sum_{\widetilde{C} \in \mathcal{C}_k(G) : u \in \widetilde{C}} \alpha_u^{\widetilde{C}}.$$

It has been noted in the proof of Lemma 4.5 in [10] that one can consider each k-clique $C \in \mathcal{C}_k(G)$ independently and actually $s^{(t,i)} = \arg\min_{\widetilde{s} \in \mathbb{R}^{C^i} : \sum_{u \in C^i} \widetilde{s}_u = 1} \left\langle \widetilde{s}, \nabla_{C^i} Q\left(\alpha^{(t,i)}\right)\right\rangle$. Therefore, for $t \geq 1$, we have

$$\begin{aligned}
&\left\langle s^{(t)}, \nabla Q\left(\frac{\alpha^{(t)}}{t}\right)\right\rangle - \left\langle \widehat{s}, \nabla Q\left(\frac{\alpha^{(t)}}{t}\right)\right\rangle \\
&= \frac{1}{t}\left\langle s^{(t)} - \widehat{s}, \nabla Q\left(\alpha^{(t)}\right)\right\rangle \\
&= \frac{1}{t}\sum_{i=1}^K \left\langle s^{(t,i)} - \widehat{s}^i, \nabla_{C_i} Q\left(\alpha^{(t)}\right)\right\rangle \\
&= \frac{1}{t}\sum_{i=1}^K \left\langle s^{(t,i)} - \widehat{s}^i, \nabla_{C_i} Q\left(\alpha^{(t,i)}\right)\right\rangle \\
&\quad + \sum_{i=1}^K \left\langle s^{(t,i)} - \widehat{s}^i, \nabla_{C_i} Q\left(\alpha^{(t)} - \alpha^{(t,i)}\right)\right\rangle \\
&\leq \frac{1}{t}\sum_{i=1}^K \left\langle s^{(t,i)} - \widehat{s}^i, \nabla_{C_i} Q\left(\alpha^{(t)} - \alpha^{(t,i)}\right)\right\rangle \\
&= \frac{1}{t}\left\langle s^{(t)} - \widehat{s}, \nabla Q\left(\alpha^{(t)}\right) - \left(\nabla_{C_1} Q\left(\alpha^{(t,1)}\right), \ldots, \right.\right. \\
&\qquad\qquad \left.\left., \ldots, \nabla_{C_K} Q\left(\alpha^{(t,K)}\right)\right)\right\rangle \\
&\leq \frac{1}{t}\left\|s^{(t)} - \widehat{s}\right\| \cdot \left\|\nabla Q\left(\alpha^{(t)}\right) - \left(\nabla_{C_1} Q\left(\alpha^{(t,1)}\right), \ldots, \right.\right. \\
&\qquad\qquad \left.\left., \ldots, \nabla_{C_K} Q\left(\alpha^{(t,K)}\right)\right)\right\|.
\end{aligned}$$

Now observe that $\frac{1}{t} \leq 2\gamma_t$ and $\|s^{(t)} - \widehat{s}\| \leq \sqrt{2K}$. Also, $\frac{\partial Q(\alpha)}{\partial \alpha_u^C} = 2r(u)$. In each iteration, $r(u)$ will be increased by at most Δ, so the absolute value of each coordinate of

$$\nabla Q\left(\alpha^{(t)}\right) - \left(\nabla_{C_1} Q\left(\alpha^{(t,1)}\right), \ldots, \nabla_{C_K} Q\left(\alpha^{(t,K)}\right)\right)$$

is at most 2Δ. Since the dimension of α is kK, we have

$$\left\|\nabla Q\left(\alpha^{(t)}\right) - \left(\nabla_{C_1} Q\left(\alpha^{(t,1)}\right), \ldots, \nabla_{C_K} Q\left(\alpha^{(t,K)}\right)\right)\right\| \leq 2\Delta\sqrt{kK}.$$

Therefore, $\left\langle s^{(t)}, \nabla Q\left(\frac{\alpha^{(t)}}{t}\right)\right\rangle - \left\langle \widehat{s}, \nabla Q\left(\frac{\alpha^{(t)}}{t}\right)\right\rangle$ can be upper bounded by $\frac{1}{t}\sqrt{2K} \cdot 2\Delta\sqrt{kK} \leq 4\gamma_t\Delta K\sqrt{2k} = \frac{1}{2}\delta\gamma_t\xi$. □

Proof of Theorem 8. Now that we have Lemma 9, by the same argument as in [19], we can establish a variant of [19, Lemma 5] and in turn deduce the claimed guarantee. □

We complete the analysis on convergence rate as in [10] by making a connection between error in Q and error in r.

LEMMA 10. *Suppose $\alpha \in \mathcal{D}$ induces r such that $\|r - r^*\|_2 \geq \epsilon > 0$ where r^* is induced by an optimal α^*. Then, $Q(\alpha) - Q(\alpha^*) \geq \epsilon^2$.* □

COROLLARY 11. *For any $\epsilon > 0$, if $t > \Omega\left(\frac{\log \epsilon}{\epsilon^2} \cdot \Delta K \sqrt{k}\right)$, then we have $\|r^{(t)} - r^*\|_2 < \epsilon$.* □

REMARK 12. *The step size of $\frac{2}{t+2}$ chosen in [19] gives an error of $O(1/t)$ after t iterations. As stated in Theorem 8, a step size of $\frac{1}{t+1}$ gives a slightly worse error of $O(\frac{\log t}{t})$. This in turn introduces an extra factor of $\Theta(\log \epsilon)$ in the number of steps required in Corollary 11. However, we will see in Section 5 that our algorithms are efficient in practice for such a step size.*

4.2 Upper Bound

In Section 4.2 and Section 4.3, we consider any *density vector* $r \in \mathbb{R}_+^V$ induced via $r(u) = \sum_{C \in \mathcal{C}_k(G): u \in C} \alpha_u^C$, $\forall u \in V$ for some $\alpha \in \mathcal{D}(G, k)$.

The vector r returned by Algorithm 2 implies an upper bound on the maximum k-clique density of the input graph.

LEMMA 13. *Given a density vector $r \in \mathbb{R}_+^V$ on G supposing $r(u_1) \geq r(u_2) \geq \cdots \geq r(u_n)$. Fix $i \in [n]$. Then any i nodes induces a subgraph with k-clique density at most $\min\left\{\frac{1}{i}\binom{i}{k}, \frac{1}{i}\sum_{j=1}^{i} r(u_j)\right\}$.*

PROOF. As the number of k-cliques among any i nodes is at most $\binom{i}{k}$, the k-clique density of an i-node subgraph is at most $\frac{1}{i}\binom{i}{k}$. In addition, suppose there are M k-cliques in the induced subgraph of some subset $\{v_1, v_2, \ldots, v_i\}$ of i nodes. Then each of the M cliques will distribute all its weight among the nodes in it. As a result, the sum of r values of these i nodes is at least M. Therefore, the k-clique density $\frac{M}{i} \leq \frac{1}{i}\sum_{j=1}^{i} r(v_j) \leq \frac{1}{i}\sum_{j=1}^{i} r(u_j)$, where the last inequality follows from the fact that $u_1, u_2, \ldots, u_i$ are the i nodes with greatest r values. □

COROLLARY 14 (UPPER BOUND ON MAXIMUM k-CLIQUE DENSITY) . *Given a vector $r \in \mathbb{R}_+^V$ returned by Algorithm 2 and suppose $r(u_1) \geq r(u_2) \geq \cdots \geq r(u_n)$. Then the maximum k-clique density of G is at most*

$$\max_{1 \leq i \leq n} \min\left\{\frac{1}{i}\binom{i}{k}, \frac{1}{i}\sum_{j=1}^{i} r(u_j)\right\}. \quad \square$$

Consider the hypergraph with hyperedge set $\mathcal{C}_k(G)$. Sampling with some probability p preserves k-clique densities of subgraphs due to Chernoff bounds [25], so an upper bound on the maximum k-clique density of the sampled hypergraph implies an upper bound on that of the original hypergraph. For $U \subseteq V$, we use $\rho(U)$ and $\widetilde{\rho}(U)$ to denote the k-clique density of the subhypergraph induced by U in the original hypergraph and the sampled hypergraph, respectively. The following theorem is adapted from [25, Theorem 4] and we omit the proof.

THEOREM 15. *Let $\epsilon > 0$ be an accuracy parameter. Suppose we sample each k-clique independently with probability $p \geq \frac{6 \log n}{\epsilon^2 D}$ where $D \geq \log n$ is the density threshold, then with probability $1 - O(\frac{1}{n})$, for all $U \subseteq V$ such that $\rho(U) \geq D$, we have $\widetilde{\rho}(U) \geq (1 - \epsilon)pD$.* □

COROLLARY 16. *Suppose a real number $\widetilde{D}$ satisfies that $\forall U \subseteq V$, $\widetilde{\rho}(U) < \widetilde{D}$ and $\widetilde{D} > 6 \log n$. Then with probability $1 - O(\frac{1}{n})$, for all $U \subseteq V$, $\rho(U) < \frac{\widetilde{D}}{\left(1 - \sqrt{\frac{6 \log n}{\widetilde{D}}}\right)p}$.*

PROOF. This follows directly from Theorem 15 by setting $\epsilon := \sqrt{\frac{6 \log n}{\widetilde{D}}}$ and $D := \frac{\widetilde{D}}{\left(1 - \sqrt{\frac{6 \log n}{\widetilde{D}}}\right)p}$. □

4.3 Exact Algorithm

THEOREM 17 (OPTIMALITY TEST BY IMPROVED GOLDBERG'S CONDITION) . *Given a density vector $r \in \mathbb{R}_+^V$ on G and suppose $r(u_1) \geq r(u_2) \geq \cdots \geq r(u_n)$. Let $K := |\mathcal{C}_k(G)|$. If $\forall i \in [n-1]$,*

$$\min\left\{\frac{1}{i}\binom{i}{k}, \frac{1}{i}\sum_{j=1}^{i} r(u_j)\right\} - \rho_k(G) < \max\left\{\frac{1}{ni}, \frac{1}{i}\left(\left\lceil\frac{iK}{n}\right\rceil - \frac{iK}{n}\right)\right\},$$

then G is the maximal k-clique densest subgraph of itself.

PROOF. Suppose by contradiction that there exist $i < n$ nodes with strictly higher k-clique density than G. Let M be the number of k-cliques among these i nodes. Then $M/i > \rho_k(G) = K/n$.

Since M, i, K and n are all integers, we have $\frac{M}{i} - \frac{K}{n} \geq \frac{1}{ni}$ and $\frac{M}{i} - \frac{K}{n} = \frac{1}{i}\left(M - \frac{iK}{n}\right) \geq \frac{1}{i}\left(\left\lceil\frac{iK}{n}\right\rceil - \frac{iK}{n}\right)$. By Lemma 13, $\min\left\{\frac{1}{i}\binom{i}{k}, \frac{1}{i}\sum_{j=1}^{i} r(u_j)\right\}$ is an upper bound on M/i, which leads to a contradiction. □

THEOREM 18 (OPTIMALITY TEST BY MAX-FLOW) . *Algorithm 9 returns true if and only if G is the maximal k-clique densest subgraph of itself.*

PROOF. Suppose Algorithm 9 returns true, i.e., there exists a feasible flow with value nM in the constructed network. The feasible flow induces a vector $\alpha \in \mathcal{D}(G, k)$: for each $u \in C \in \mathcal{C}_k(G)$, α_u^C is the flow on the edge from u to C divided by n. This α induces $r \in \mathbb{R}_+^V$ where $r(u) = M/n$, $\forall u \in V$. Lemma 13 implies that there is no subgraph with strictly higher k-clique density.

Conversely, if G is the maximal k-clique densest subgraph of itself, the k-clique-diminishingly-dense decomposition of G will simply be $\emptyset = B_0 \subsetneq B_1 = V$. By Proposition 5, V is the only level set of the density vector $r \in \mathbb{R}_+^V$ induced from the optimal solution α of $\mathsf{CP}(G, k)$, i.e., $r(u) = M/n$, $\forall u \in V$. From α we can construct a feasible flow with value nM by setting the flow from C to u to be $n\alpha_u^C$, $\forall u \in C \in \mathcal{C}_k(G)$. □

After performing Squeeze in Line 2 of Algorithm 7, if set S satisfies the condition in Line 7 of Algorithm 6, then S is stable with respect to α. By Proposition 7, the maximal k-clique densest subgraph is included in $G[S]$. Hence, once S passes either one of the two optimality tests, we can confirm that S is the maximal k-clique densest subgraph of G. In addition, it is guaranteed by Theorem 8 and Corollary 11 that after sufficiently many iterations, the maximal k-clique densest subset will emerge as S_1 in Algorithm 7.

5. EXPERIMENTS

We design our experiments so as to answer the following questions:

Q1. Convergence: How fast does kClist++ converge as a function of the number of iterations? Do the sequential updates lead to much faster convergence than the simultaneous ones? And what about Seq-Sampling++?

Q2. Comparison: Does our algorithms allow better tradeoffs between time and quality of results than existing algorithms?

Q3. Degree of parallelism and running time: Does the kClist++ algorithm have a good degree of parallelism? How fast is our algorithm and what can it do within a reasonable amount of time?

Q4. Exact solution: Can our exact algorithm compute the exact densest subgraph for large values of k on large graphs when existing methods fail?

5.1 Experimental Setup

Machine. We carried out our experiments on a Linux machine equipped with 4 processors Intel Xeon CPU E5-4617 2.90GHz with 6 cores each (that is a total of 24 threads) and with 64G of RAM DDR4 2133 MHz. We will use 1 thread for most of the experiments and use up to 24 threads to evaluate the degree of parallelism.

Datasets. We consider several large real-world graphs from SNAP [24] of different nature (social networks, co-authorship and communication networks, etc.) containing up to two billion edges, while exhibiting a wide range of core values. In particular, the core value of the input graph has an important impact on the overall running time of kClist. Statistics of the datasets are summarized in Table 1.

Table 1: Our set of large real-world graphs from [24]. The last three columns specify the core value c, the size $k_{\max}$ of a maximum clique (computed by [26]) and the number s of nodes in the maximal densest subgraph (computed by [10]), respectively.

Network	n	m	c	$k_{\max}$	s
road-CA	1,965,206	2,766,607	3	4	62
Amazon	334,863	925,872	7	7	97
soc-pokec	1,632,803	22,301,964	47	29	8,485
loc-gowalla	196,591	950,327	51	29	560
Youtube	1,134,890	2,987,624	51	17	1,959
zhishi-baidu	2,140,198	17,014,946	78	31	8,993
DBLP	425,957	1,049,866	113	114	115
WikiTalk	2,394,385	4,659,565	131	26	1,384
Wikipedia	2,080,370	42,336,692	208	67	1,238
Orkut	3,072,627	117,185,083	253	51	25,969
Friendster	124,836,180	1,806,067,135	304	129	49,092
LiveJournal	4,036,538	34,681,189	360	327	440

Competitors. We consider the following existing approaches to compare against:

1. **Greedy:** The algorithm which removes in each round a vertex belonging to the minimum number of k-cliques and returns the subgraph that achieves the largest k-clique density, is a k-approximation for the k-clique densest subgraph problem. It also produces the so-called *k-clique core decomposition* [30, 9].
2. **Batch:** The algorithm which removes in each round the set of vertices belonging to less than $k(1+\epsilon)\rho_k$ k-cliques and returns the subgraph that achieves the largest k-clique density, where ρ_k is the k-clique density of the subgraph at that round. Such an algorithm gives a $k(1+\epsilon)$-approximation, while it terminates in $O(\frac{\log n}{\epsilon})$ rounds, for any $\epsilon > 0$ [30, 9].
3. **MaxFlow:** The exact Goldberg's max-flow based binary search algorithm generalized to the case when $k \geq 3$ by Tsourakakis [30] and Hu et al. [18].
4. **MaxFlow-Sampling:** The max-flow based binary search algorithm equipped with sampling, proposed by Mitzenmacher et al. [25]. It differs from our Seq-Sampling++ only in that it finds a k-clique densest subgraph via a series of max-flow computations, rather than iterations of sequential update.

Implementation. We implemented all our algorithms in C++, including the max-flow based exact algorithm. We used OpenMP for parallelization and made our implementations publicly available[1]. To compute a maximum flow, we used the implementation available in Boost library [27]. In particular, we used the push-relabel algorithm which turned out to be faster than the Boykov-Kolmogorov algorithm for our settings. For the competitors we used the publicly available implementations[2] of the algorithms presented in [9]. We set $\sigma := 10^7$ for Seq-Sampling++ and $\sigma := 10^5$ for MaxFlow-Sampling to make both the running time and the obtained solution comparable.

Implementation Details of k-Cliques Ordering. It turns out that the order by which the k-cliques are processed affects the convergence rate of our sequential algorithms. In particular, if the k-cliques that contain a node u come in batch too early, then $r(u)$ will not adjust later during that iteration over the k-cliques. Ideally, the k-cliques should appear in a random order, but this does not seem to be possible under the framework of kClist. We specify the order of k-clique enumeration as follows. Recall that kClist is a recursive algorithm performed on a directed version of the input graph G based on some total ordering on the nodes $\eta : V \to [n]$, where each edge uv is oriented from u to v such that $\eta(u) < \eta(v)$. In each recursive step, kClist picks the nodes in the current graph one by one to recurse in the subgraph induced by the out-neighbors of each node ([9, Line 9–10, Algorithm 2]). The nodes can be picked in arbitrary order for the purpose of k-clique listing. We pick the nodes in *reverse order*: in Line 9 of [9], we process the nodes in decreasing order of η value, that is, in the order $u_1, u_2, \ldots, u_n$ such that $\eta(u_1) > \eta(u_2) > \cdots > \eta(u_n)$. For Seq-Sampling++ we simply shuffle the sampled cliques.

5.2 Q1. Convergence

Figure 1 shows the relative error of kClist++ as a function of the number of the iterations. The relative error is defined as $(ub_k - \rho_k)/\rho_k$ where ub_k is the upper bound on the k-clique density, while ρ_k is the k-clique density of the subgraph detected. We can see that we obtain a relative error of 10^{-3} within 1,000 iterations in all graphs.

Figure 2 compares the two variants of kClist++, namely with sequential updates (Seq-kClist++) and simultaneous

[1] https://github.com/btsun/kClistpp

[2] https://github.com/maxdan94/kClist

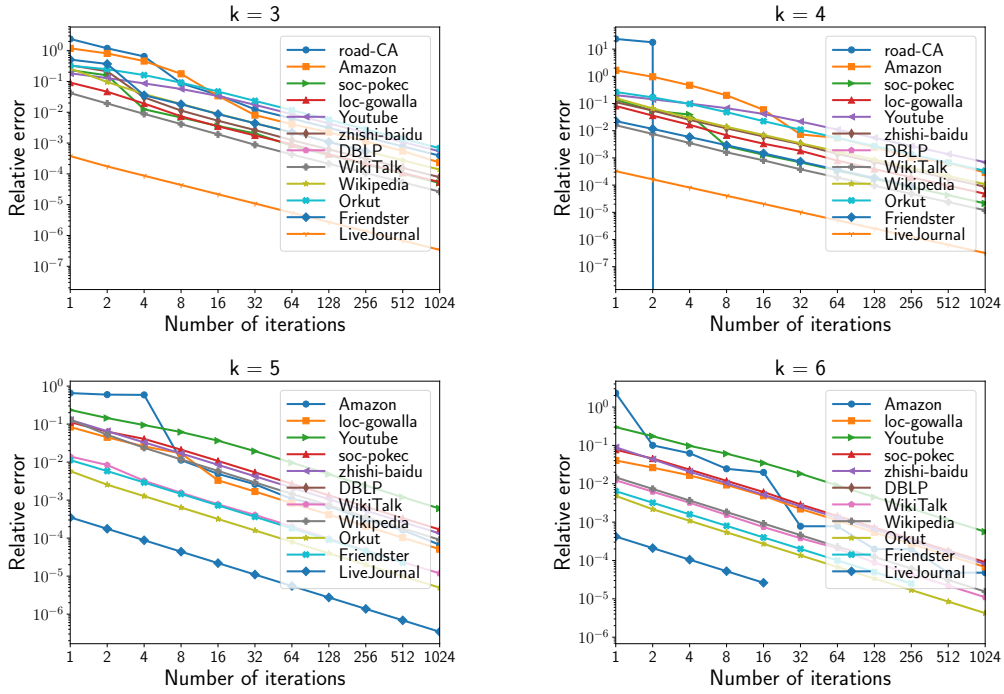

Figure 1: Relative error of kClist++ as a function of the number of iterations.

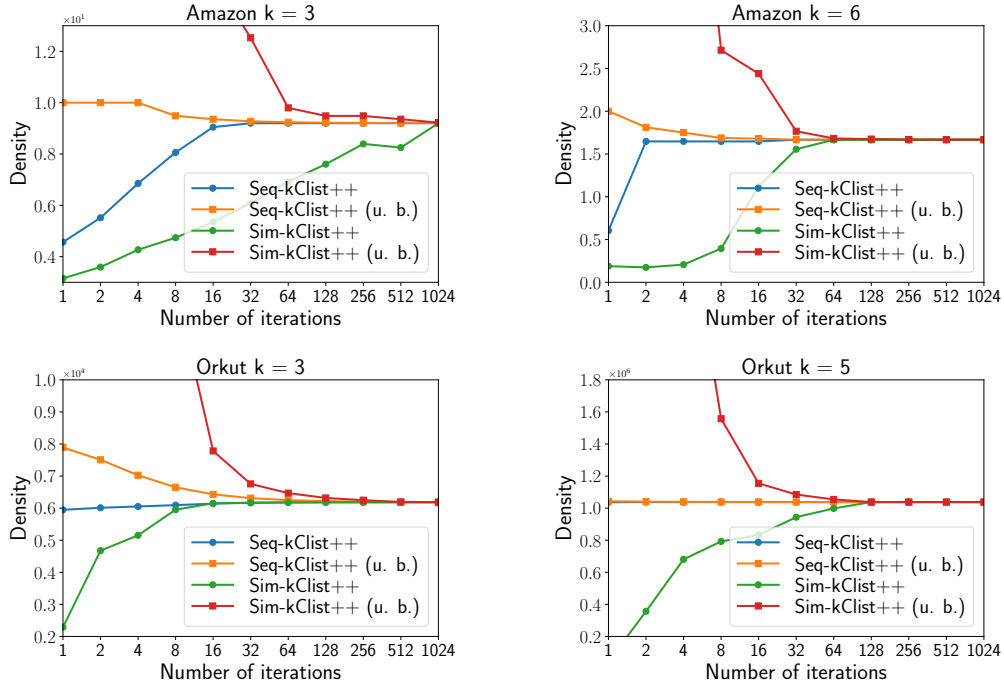

Figure 2: Comparison of the convergence rate of Seq-kClist++ and Sim-kClist++. "Seq-kClist++ (u.b.)" and "Sim-kClist++ (u.b.)" denote the upper bound on the optimum density obtained by Seq-kClist++ and Sim-kClist++, respectively.

updates (Sim-kClist++). It shows the k-clique density as well as the upper bound on the optimal solution as a function of the number of iterations. We can see that in all cases, Seq-kClist++ converges faster. In particular for Orkut and $k = 5$, Seq-kClist++ computes a nearly optimal solution within one iteration, while the Sim-kClist++ needs more than 100 iterations to achieve a similar result. This is expected because in Seq-kClist++ any change in the

Table 2: The results (density, upper bound, and the relative error implied) given by Seq-Sampling++ after 1 and 256 rounds of computation. The density and the upper bound after 256 iterations are presented as the difference (Δ) between the value and that produced after 1 iteration.

Network	k	# iterations	Density	Upper bound	Relative error
loc-gowalla	5	1	1.38×10^4	1.51×10^4	9.71%
		256	$\Delta \approx 0.35$	$\Delta \approx -8.11$	9.65%
zhishi-baidu	5	1	7.99×10^3	9.58×10^3	19.84%
		256	$\Delta = 0$	$\Delta \approx -5.19$	19.77%
DBLP	5	1	1.29×10^6	1.34×10^6	4.07%
		256	$\Delta = 0$	$\Delta \approx -6.57$	4.07%
Orkut	5	1	1.04×10^6	1.65×10^6	58.65%
		256	$\Delta = 0$	$\Delta \approx -2718$	58.39%
LiveJournal	5	1	1.69×10^8	1.91×10^8	12.97%
		256	$\Delta = 0$	$\Delta \approx -24\,621$	12.95%

Table 3: Relative error and number of iterations over all k-cliques after one hour and one day of computations using kClist++ with 20 threads.

		One hour		One day	
Network	k	Rel. err.	# iter.	Rel. err.	# iter.
Orkut	7	2×10^{-1}	16	2×10^{-3}	256
	8	3×10^{-1}	2	1×10^{-2}	64
Friendster	7	4×10^{-3}	2	3×10^{-4}	64
	8	5×10^{-3}	1	6×10^{-4}	32
LiveJournal	5	1×10^{-3}	32	3×10^{-5}	1024
	6	3×10^{-3}	1	2×10^{-3}	16

minimum r values are promptly taken into account, while in Sim-kClist++ a given $r(u)$ might be increased even if it is no longer the minimum because of the simultaneous updates. We observe similar results for other graphs, which are omitted due to space constraints.

As for Seq-Sampling++, Table 2 shows the approximate solution and the upper bound computed on part of the datasets (other results are similar and omitted due to space constraints). We observe that the quality of the approximate solution converges quite fast as the density given after 256 iterations does not significantly differ from that given after 1 iteration. However, the relative error is not ideal compared with kClist++. This is because the term $1 - \sqrt{\frac{6 \log n}{\widetilde{D}}}$ is inevitable when estimating the upper bound on the original hypergraph (see Corollary 16), even though the upper bound on the sampled hypergraph converges.

5.3 Q2. Comparison Against State of the Art

Figure 3 shows the k-clique density of the subgraph produced by the algorithms as a function of time. Here, we consider the approximation algorithms, namely kClist++, Seq-Sampling++, MaxFlow-Sampling, Batch and Greedy. We also include the upper bound computed by kClist++ to examine the quality of the solutions. We remark that Greedy, Batch and MaxFlow-Sampling produce exactly *one* solution, while our algorithms provide gradually improving results. In practice, when running kClist++ or Seq-Sampling++, one can stop whenever needed depending on the available computation resources or time.

We can see that kClist++ and Seq-Sampling++ outperform their competitors almost all the time. When executed long enough, kClist++ computes a near-optimal k-clique densest subgraph, while Seq-Sampling++ is even faster than kClist++ in producing a good approximation. Batch turns out to be faster than Greedy but often leads to much worse results. As expected, MaxFlow-Sampling is less effective, due to the fact that computing a maximum flow is usually more time-consuming than iterating through k-cliques in main memory[3]. Additionally, the upper bound given by kClist++ also helps estimating the quality of solution produced by other algorithms in absence of an exact solution. For example, Greedy turns out to achieve an approximation ratio close to 1 relatively often, as opposed to a k-approximation guarantee without such an upper bound. Similar results can be observed in other graphs and for other values of k, which are omitted for space constraints.

5.4 Q3. Degree of Parallelism

Figure 4 shows the running time and the speedup of kClist++ as a function of the number of threads. We can see that the algorithm has a very good degree of parallelism, as using 24 threads allows to reduce the running time by a factor of nearly 24 in most settings for large value of k. When k is small, say 5 or less, there is a relatively small number of k-cliques. In such a case, the speedup is sub-optimal (e.g. Friendster $k = 4$ has a speedup of nearly 10 using 24 threads) due to the non-negligible cost of thread coordination. Nevertheless, in these settings kClist++ is already very fast.

We pushed kClist++ to its limit to see what we are able to compute using 20 threads within one hour and one day of computations on our largest graphs and for large values of k. We found that kClist++ is able to compute approximate k-clique densest subgraphs with a relative error of at most 3×10^{-1} within one hour and at most 1×10^{-2} within one day; see Table 3. We conclude that kClist++ boasts an excellent degree of parallelism, allowing the computation of nearly optimal k-clique densest subgraphs in large graphs for unprecedentedly large values of k.

5.5 Q4. Exact Algorithms

We compare the algorithms for computing the maximal k-clique densest subgraphs, namely MaxFlow and the exact version of kClist++ (Algorithm 6) on several datasets and

[3]A comparison between the running time of a Frank-Wolfe based algorithm and a max-flow based algorithm is shown in [10, Table 2].

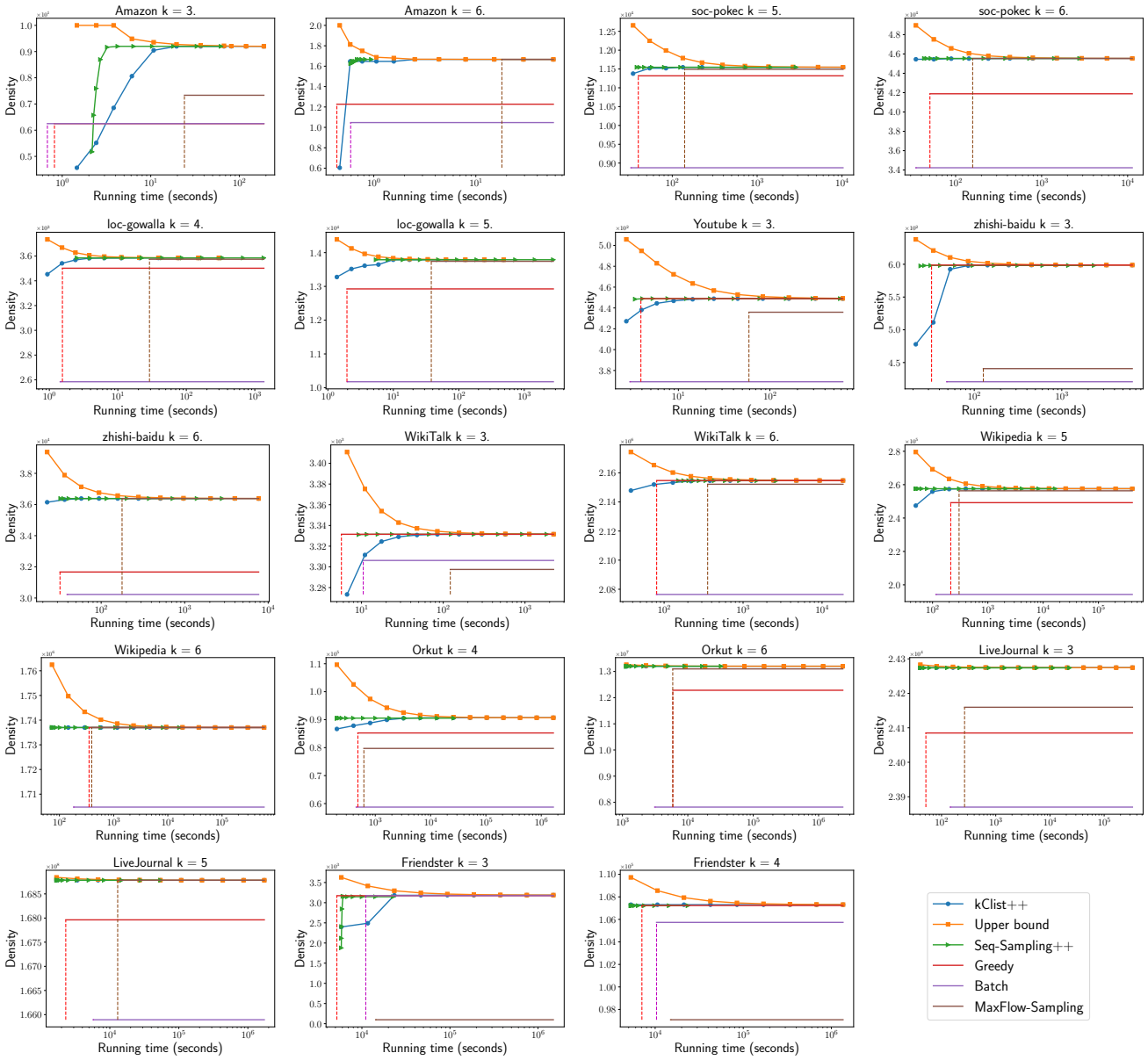

Figure 3: Comparison of kClist++ and Seq-Sampling++ against the three state-of-the-art approximation algorithms, namely Greedy, Batch and MaxFlow-Sampling.

for different values of k. Recall that kClist++ might also run a max-flow algorithm, however, on a subgraph which is possibly much smaller than the input graph. We summarize the main results in Table 4. We observe that MaxFlow runs out of memory in many cases, namely in WikiTalk for $k \geq 5$, zhishi-baidu for $k \geq 9$, etc., which is expected as MaxFlow introduces one node for every k-clique in the graph. In most of those cases, kClist++ can compute the k-clique densest subgraph within 60–90 minutes. Moreover, in almost all other cases (with very few exceptions), kClist++ is faster than MaxFlow by several orders of magnitude (up to a factor of 50–100). Finally, in a few cases kClist++ does not require any maximum flow computation at all, namely in DBLP for $k \in \{3, 4, 5\}$.

6. CONCLUSION

We developed algorithms for computing a near-optimal solution for the k-clique densest subgraph problem, which has received increasing attention in recent years. Our algorithms are appealing in that they require small amount of memory even when the number of k-cliques residing in the input graph is huge, which allows to deal with large real-world graphs for relatively large values of k. In contrast with state-of-the-art approaches, our algorithms update their variables sequentially, which turns out to be more efficient in practice than similar algorithms based on gradient descent. Our study of the convergence rate requires different techniques and it is interesting per se. Seq-Sampling++ is faster than kClist++ in producing a near-optimal solution, while kClist++ additionally gives a good upper bound on the

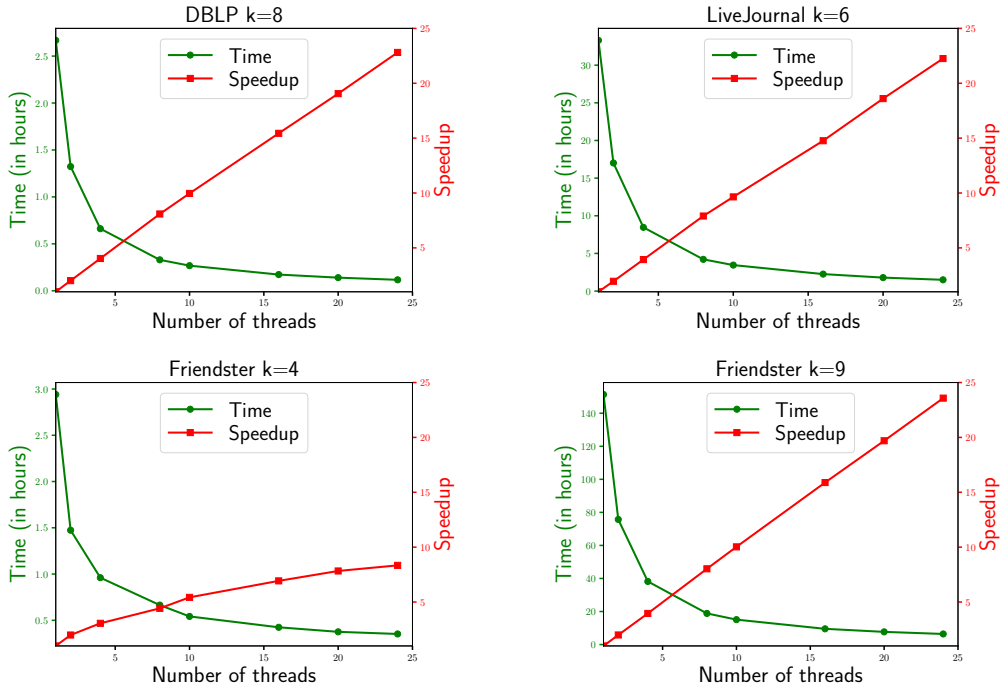

Figure 4: Running time and speedup as a function of the number of threads for one iteration of kClist++.

Table 4: Comparison of efficiency between exact algorithms.

		Running time (seconds)		# max-flow calls		# iterations
Network	k	MaxFlow [30]	kClist++	MaxFlow [30]	kClist++	kClist++
WikiTalk	3	669	44	37	1	16
	4	13954	384	42	1	2
	5	Run out of memory	4099	-	1	8
zhishi-baidu	8	18672	187	39	1	1
	9	Run out of memory	537	-	1	1
	10	Run out of memory	2010	-	1	1
soc-pokec	8	38647	261	47	1	1
	9	Run out of memory	418	-	1	1
	10	Run out of memory	1013	-	1	1
loc-gowalla	9	15795	316	32	1	1
	10	46070	733	33	1	1
	11	Run out of memory	1967	-	1	1
DBLP	3	35	1	28	0	1
	4	1480	4	34	0	1
	5	Run out of memory	64	-	0	1
Wikipedia	3	970	260	37	2	8
	4	15306	19320	44	7	1024
	5	Run out of memory	5740	-	1	1
Orkut	3	9432	11262	44	1	256
	4	Run out of memory	Run out of memory	-	-	-
LiveJournal	3	8312	206	42	1	1
	4	Run out of memory	Run out of memory	-	-	-

maximum k-clique density. We also developed an exact algorithm, which is shown to be able to deal with relatively large graphs and values of k. We demonstrated the effectiveness of our algorithms by means of an extensive experimental evaluation on graphs containing up to two billion edges and for values of k up to 10. The evaluation shows that our algorithms outperform state-of-the-art approaches and boast an excellent degree of parallelism.

7. REFERENCES

[1] J. Abello, M. G. Resende, and S. Sudarsky. Massive quasi-clique detection. In *Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer, 2002.

[2] V. Anantharam, J. Salez, et al. The densest subgraph problem in sparse random graphs. *The Annals of Applied Probability*, 26(1):305–327, 2016.

[3] A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012.

[4] Y. Asahiro, R. Hassin, and K. Iwama. Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, 121(1):15–26, 2002.

[5] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and MapReduce. *PVLDB*, 5(5):454–465, 2012.

[6] O. D. Balalau, F. Bonchi, T. Chan, F. Gullo, and M. Sozio. Finding subgraphs with maximum total density and limited overlap. In *WSDM*, pages 379–388, 2015.

[7] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[8] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 937–946, 2002.

[9] M. Danisch, O. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 589–598, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee.

[10] M. Danisch, T. H. Chan, and M. Sozio. Large scale density-friendly graph decomposition via convex programming. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017* [10], pages 233–242.

[11] X. Du, R. Jin, L. Ding, V. E. Lee, and J. H. T. Jr. Migration motif: a spatial - temporal pattern mining approach for financial markets. In *SIGKDD*, pages 1135–1144, 2009.

[12] A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015.

[13] I. Finocchi, M. Finocchi, and E. G. Fusco. Clique counting in MapReduce: algorithms and experiments. *Journal of Experimental Algorithmics (JEA)*, 20:1–7, 2015.

[14] E. Fratkin, B. T. Naughton, D. L. Brutlag, and S. Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006.

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

[16] B. Hajek. Performance of global load balancing by local adjustment. *IEEE Transactions on Information Theory*, 36(6):1398–1414, 1990.

[17] B. Hajek et al. Balanced loads in infinite networks. *The Annals of Applied Probability*, 6(1):48–75, 1996.

[18] S. Hu, X. Wu, and T. H. Chan. Maintaining densest subsets efficiently in evolving hypergraphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 929–938, 2017.

[19] M. Jaggi. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *ICML*, pages 427–435, 2013.

[20] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Proceedings of the 26th International Conference on World Wide Web*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017.

[21] S. Jain and C. Seshadhri. The power of pivoting for exact clique counting. In *WSDM*, pages 268–276, 2020.

[22] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 813–826, 2009.

[23] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[24] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[25] M. Mitzenmacher, J. Pachocki, R. Peng, C. Tsourakakis, and S. C. Xu. 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*, pages 815–824. ACM, 2015.

[26] R. A. Rossi, D. F. Gleich, and A. H. Gebremedhin. Parallel maximum clique algorithms with applications to network analysis. *SIAM Journal on Scientific Computing*, 37(5):C589–C616, 2015.

[27] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *Boost Graph Library: User Guide and Reference Manual, The.* Pearson Education, 2001.

[28] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*, pages 939–948, 2010.

[29] N. Tatti and A. Gionis. Density-friendly graph decomposition. In *WWW*, pages 1089–1099, 2015.

[30] C. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th international conference on world wide web*, pages 1122–1132. International World Wide Web Conferences Steering Committee, 2015.

[31] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *SIGKDD*, pages 104–112, 2013.

[32] N. Wang, J. Zhang, K.-L. Tan, and A. K. Tung. On triangulation-based dense neighborhood graph discovery. *PVLDB*, 4(2):58–68, 2010.