CREST: Approximate k-Clique Counting in Real-World Networks via Refinement of Star-Based Sample Space

Yehyun Nam
Seoul National University
yhnam@theory.snu.ac.kr

Jihoon Jang
Seoul National University
jhjang@theory.snu.ac.kr

Kunsoo Park
Seoul National University
kpark@theory.snu.ac.kr

Joong Chae Na
Sejong University
jcna@sejong.ac.kr

Hyunjoon Kim*
Hanyang University
hyunjoonkim@hanyang.ac.kr

ABSTRACT

A k-clique, defined as the set of k pairwise adjacent vertices, plays a fundamental role in the analysis of real-world networks. Many downstream tasks require computing the number of k-cliques, yet listing or exact counting is often computationally prohibitive on massive networks, making approximate counting the only scalable option. Existing algorithms for approximate k-clique counting primarily use the Monte Carlo method. These algorithms construct a *sample space*, which is a collection of k-vertex sets including all k-cliques. They then perform *sample trials*, where each trial consists of selecting a k-vertex set uniformly at random from the sample space and checking whether it forms a k-clique. However, existing algorithms suffer from huge sample spaces and expensive sample trials. In this paper, we present CREST, an efficient Monte Carlo algorithm for approximate k-clique counting. We introduce a suite of novel techniques to address the two main objectives: (1) obtaining a small sample space, and (2) reducing the cost of sample trials. We propose a novel sample space refinement strategy to obtain a smaller sample space, and a star-based sampling approach that addresses both of the main objectives. We also develop a combinatorial method to obtain exact clique counts for certain subgraphs, effectively reducing their sample spaces to the extreme. Moreover, we present a new stopping criterion that satisfies the target accuracy requirement with fewer samples. Extensive experiments on real-world networks demonstrate that CREST outperforms the state-of-the-art algorithm by up to two orders of magnitude in running time, while maintaining the specified accuracy requirement.

PVLDB Reference Format:
Yehyun Nam, Jihoon Jang, Kunsoo Park, Joong Chae Na, and Hyunjoon Kim. CREST: Approximate k-Clique Counting in Real-World Networks via Refinement of Star-Based Sample Space. PVLDB, 19(9): 2331 - 2343, 2026.
doi:10.14778/3819518.3819554

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SNUCSE-CTA/CREST.

*Corresponding author

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819554

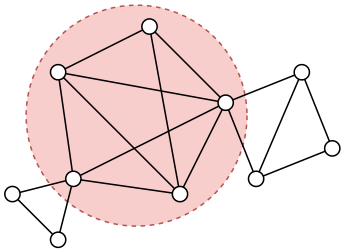


Figure 1: A k-clique densest subgraph for $k = 3$. The highlighted area has the maximum k-clique density of $\frac{7}{5}$.

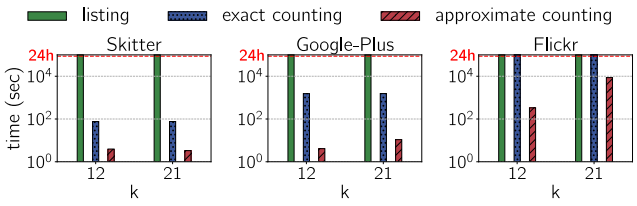


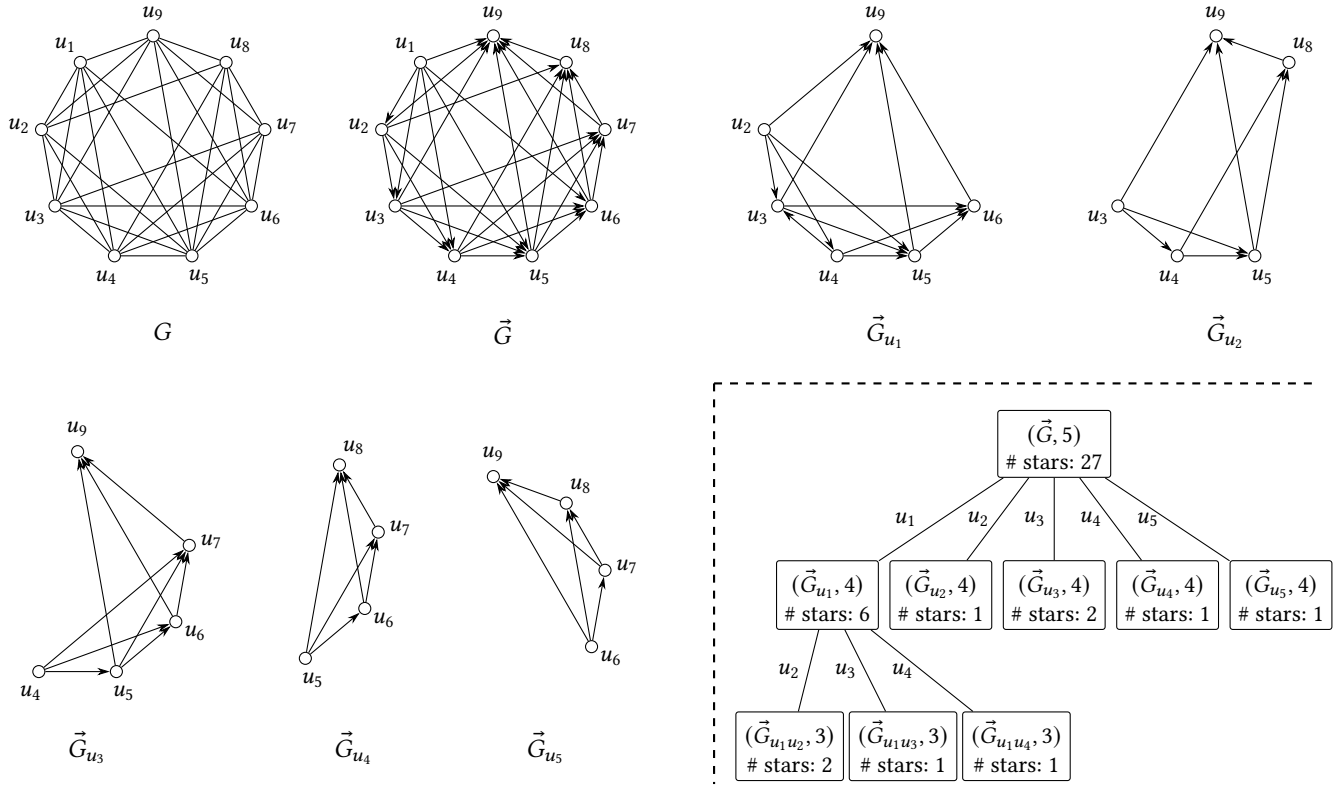
Figure 2: Running times of algorithms for k-clique listing, exact counting, and approximate counting. The y-axis is in a logarithmic scale. Bars hitting the top of the plots indicate that the algorithms failed to complete within the 24-hour time limit.

1 INTRODUCTION

Graphs are widely used to model relationships between entities in various domains, including social network analysis [25], bioinformatics [22, 35], and cheminformatics [12, 27]. Mining cohesive subgraphs in these real-world networks has emerged as a fundamental problem in graph analysis.

A *k-clique*, defined as the set of k pairwise adjacent vertices, is the most fundamental cohesive subgraph [20, 21]. Building upon this definition, various cohesive structures have been proposed, including k-clique communities [26], k-clique densest subgraphs [30, 32], and nucleus decompositions [28]. Additionally, relaxed variants of the clique, such as quasi-cliques [1], defective cliques [13, 41], and plexes [6, 29], are also widely studied.

Many downstream tasks do not require the explicit listing of all cliques. For these tasks, exact or approximate counting is sufficient.



(a) Graph G, DAG $\vec{G}$, and subgraphs induced by out-neighbors of u_1, u_2, u_3, u_4, u_5. All DAGs are oriented according to their degeneracy orderings.

(b) Sample space refinement. DAGs $\vec{G}_{u_1u_2}$, $\vec{G}_{u_1u_3}$, and $\vec{G}_{u_1u_4}$ are the subgraphs of $\vec{G}_{u_1}$ induced by the out-neighbors of u_2, u_3, and u_4, respectively. All DAGs are oriented according to their degeneracy orderings.

Figure 3: Overview of our algorithm

An important application of exact or approximate k-clique counting is k-clique densest subgraph discovery [39, 43]. This problem has various real-world applications, including social community discovery [7, 33] and anomaly detection [18, 31]. A k-clique densest subgraph is a subgraph with the maximum k-clique density, which is defined as the number of k-cliques in the subgraph divided by the number of vertices in the subgraph. Figure 1 illustrates an example of a k-clique densest subgraph. Recently, [43] and [39] proposed algorithms for this problem, utilizing exact k-clique counting and approximate k-clique counting, respectively. Thus, improving the efficiency of exact or approximate k-clique counting directly benefits k-clique densest subgraph discovery algorithms.

Beyond this, exact or approximate k-clique counting has several other important applications. To capture the clustering behavior of complex networks, Yin et al. [40] introduced the concept of the higher-order clustering coefficient, which generalizes the traditional clustering coefficient based on triangle counts to k-clique counts. Independent works by Benson et al. [3] and Tsourakakis et al. [34] also proposed graph clustering methods based on k-clique counts. To enhance the expressive power of Graph Neural Networks, Bouritsas et al. [4] proposed injecting subgraph counts, including k-clique counts, as explicit node features.

Moreover, approximate counting is often the only scalable option given the prohibitive computational cost of listing or exact counting on massive networks. Figure 2 compares the running times of state-of-the-art algorithms for k-clique listing [37], exact counting [15], and our approximate counting algorithm on three representative datasets: Skitter, Google-Plus, and Flickr, which vary in size and density. The listing algorithm failed to complete on any of these datasets within the 24-hour time limit. The exact clique counting algorithm processed only the two more tractable datasets Skitter and Google-Plus. (Pivoter is designed to compute exact k-clique counts for all k simultaneously, resulting in a steady running time.) However, our approximate counting algorithm was the only method capable of processing all datasets, achieving speedups of 21× and 153× over the exact counting algorithm on Skitter and Google-Plus, respectively.

Existing approaches and limitations. The problem of k-clique count estimation (i.e., approximate counting) has been extensively studied. Recent works [5, 16, 38] primarily use the Monte Carlo method. Let C be the set of all k-cliques in a graph. The Monte Carlo method first constructs a *sample space* $\mathcal{S}$, which is a collection of k-vertex sets including all k-cliques (i.e., $C \subseteq \mathcal{S}$). The Monte Carlo method then repeats *sample trials*, where each sample trial

consists of selecting a k-vertex set uniformly at random from $\mathcal{S}$ and checking whether the selected k-vertex set indeed forms a k-clique. If r out of t samples form cliques, giving $\frac{r}{t}$ as an estimate of the actual sampling success probability $\frac{|C|}{|\mathcal{S}|}$, then we obtain $\frac{r}{t} \times |\mathcal{S}|$ as an estimate of $|C|$. The accuracy of the estimate improves as more sample trials are performed.

There are two main objectives in designing efficient Monte Carlo algorithms.

(1) *Obtaining a small sample space* $\mathcal{S}$: It is important to reduce the sample space $\mathcal{S}$ while retaining all k-cliques (i.e., $C \subseteq \mathcal{S}$), since a smaller sample space $\mathcal{S}$ increases the sampling success probability $\frac{|C|}{|\mathcal{S}|}$, thereby requiring fewer sample trials for the same accuracy [9].
(2) *Reducing the cost of sample trials*: To perform sample trials efficiently, algorithms require the construction of auxiliary data structures that support the selection of a k-vertex set uniformly at random from $\mathcal{S}$ in $O(k)$ time. The construction of these data structures often incurs non-negligible overhead.

Existing Monte Carlo algorithms for k-clique count estimation reduce the sample space via *refinement*, which is the process of replacing the sample space $\mathcal{S}$ by a set of smaller sample spaces that is equivalent to $\mathcal{S}$. The refinement is based on the following observation: for a directed acyclic graph (DAG) $\vec{G}$, a k-clique in $\vec{G}$ corresponds to a $(k-1)$-clique in $\vec{G}_u$ (subgraph induced by the out-neighbors of a vertex u). For example, in Figure 3, the sample space for estimating 5-cliques in $\vec{G}$ is replaced by the sample spaces for estimating 4-cliques in $\vec{G}_{u_1}, \ldots, \vec{G}_{u_5}$, which are the subgraphs induced by the out-neighbors of $u_1, \ldots, u_5$, respectively. (The subgraphs $G_{u_6}, \ldots, G_{u_9}$ are omitted as they contain fewer than 4 vertices and thus no 4-cliques.) In this example, the 5-clique $\{u_1, u_2, u_3, u_4, u_5\}$ in $\vec{G}$ corresponds to the 4-clique $\{u_2, u_3, u_4, u_5\}$ in $\vec{G}_{u_1}$.

Since the refinement is applied recursively during the sample space construction stage, a careful strategy for choosing which sample space to refine at each recursive step is necessary to efficiently reduce the sample space. However, existing approaches lack an efficient sample space refinement strategy. Turán-Shadow [16] applies refinement to sample spaces in no specific order until the induced subgraphs have a high density[1]. DPColorPath [38] applies refinement only once to the initial sample space and no further. SR-kCCE [5] chooses the sample space with the smallest sampling success probability, but this heuristic is inefficient as it does not directly take into account the actual sample space size.

The sample space size $|\mathcal{S}|$ also depends on the structure of the k-vertex set in $\mathcal{S}$, e.g., an arbitrary k-vertex set, or a k-vertex set that forms a path. Turán-Shadow estimates the number of k-cliques by sampling arbitrary k-vertex sets, i.e., $|\mathcal{S}| = \binom{n}{k}$ where n is the number of vertices in this subgraph. In contrast, DPColorPath and SR-kCCE sample k-paths (i.e., k-vertex sets that form directed paths) within a DAG based on a *color ordering*. While this approach yields a smaller sample space than that of Turán-Shadow, it still remains large in practice. Moreover, sampling k-paths in these algorithms incurs the overhead of constructing a data structure that stores the number of 1-, 2-, …, k-paths starting at each vertex.

[1]The density of a graph is defined as $m/\binom{n}{2}$, where n is the number of vertices and m is the number of edges.

Contributions. In this paper, we present CREST (Clique count estimation via refinement of star-based sample space), a new algorithm for k-clique count estimation, following the Monte Carlo method. Our work introduces a suite of novel techniques to address the two main objectives: (1) obtaining a small sample space, and (2) reducing the cost of sample trials. Moreover, we present a new criterion for stopping sample trials.

- We introduce a novel sample space refinement strategy that allows us to efficiently obtain a smaller sample space. Our refinement strategy explicitly takes into account the sample space size, applying refinement to the sample space that offers the largest potential reduction in the overall sample space. Consider Figure 3. Suppose we applied one refinement to the initial sample space $(\vec{G}, 5)$. We efficiently compute the potential sample space reduction from the refinement of each sample space $(\vec{G}_{u_1}, 4), \ldots, (\vec{G}_{u_5}, 4)$. Based on these, we choose $(\vec{G}_{u_1}, 4)$ for the second refinement, as it offers the largest potential reduction. Our strategy yields a smaller sample space in less time than the state-of-the-art method [5] (see Figure 5).
- We propose a star-based sampling approach, which addresses both of the main objectives. In the star-based sampling approach, each sample is a k-vertex set that forms a *star* within a DAG. This approach incurs significantly lower overhead compared to the previous path-based approaches, thereby reducing the cost of estimation. Moreover, in practice, it yields a sample space that is significantly smaller than that of the path-based approaches (see Figure 6).
- We develop a combinatorial method to obtain exact clique counts for certain sample spaces: if the underlying graph for a sample space is a *split graph* (i.e., a graph in which the vertices can be partitioned into a clique and an independent set), we efficiently compute its exact clique count. This in effect allows us to reduce the sample space to the extreme, i.e., $|\mathcal{S}| = |C|$ for these sample spaces. In Figure 3, our method can efficiently compute the exact number of 4-cliques for the subgraphs $\vec{G}_{u_3}$, $\vec{G}_{u_4}$, and $\vec{G}_{u_5}$. Thus, these sample spaces are no longer considered for further refinement and are excluded from the final sample space used for Monte Carlo estimation.
- We use a new criterion for stopping sample trials as soon as the desired accuracy guarantee is achieved, since the running time of the algorithm is proportional to the number of sample trials. Our criterion terminates sampling significantly earlier than the criterion in prior work, while obtaining the same accuracy guarantees.

Extensive experiments on real-world networks demonstrate the efficiency and effectiveness of our techniques. Our algorithm CREST is up to two orders of magnitude faster than the state-of-the-art algorithm SR-kCCE, while obtaining the same accuracy guarantees.

2 PRELIMINARIES

A *graph* G is a pair (V_G, E_G), where V_G is the set of vertices and E_G is the set of edges. Two vertices u and v are said to be *adjacent* if $(u, v) \in E_G$. We denote by $D_u(G)$ (or simply D_u if the graph is clear from the context) the number of neighbors of u (i.e., the degree of u). We denote by $G[S]$ the subgraph of G induced by a vertex set S. A *directed graph* $\vec{G}$ is a pair $(V_{\vec{G}}, E_{\vec{G}})$, where $V_{\vec{G}}$ is a set of

vertices and $E_{\vec{G}}$ is a set of directed edges. A vertex v is called an *out-neighbor* of a vertex u if $(u, v) \in E_{\vec{G}}$. We denote by $d_u(\vec{G})$ and $N_u(\vec{G})$ (or simply d_u and N_u if the graph is clear from the context) the number of out-neighbors of u (i.e., the out-degree of u) and the set of out-neighbors of u, respectively.

Definition 2.1 (Degeneracy ordering [23]). An ordering $u_1 \prec u_2 \prec \cdots \prec u_n$ of the vertices of a graph G is called a *degeneracy ordering* if u_i has the minimum degree in the subgraph induced by $\{u_i, u_{i+1}, \ldots, u_n\}$ for all $1 \leq i \leq n$.

A degeneracy ordering can be obtained in linear time by repeatedly removing a vertex of minimum degree from the remaining graph until all vertices have been removed [23]. The *degeneracy* is the maximum degree observed during this removal process [11].

Given a graph G, we construct a directed graph $\vec{G}$ by orienting each edge from the vertex that appears earlier in a degeneracy ordering to the one that appears later. Directed graphs constructed in this manner are directed acyclic graphs (DAGs). For a vertex u in a DAG $\vec{G}$, we denote by $\vec{G}_u$ the subgraph of $\vec{G}$ induced by the out-neighbors of u (i.e., $N_u(\vec{G})$), with edges oriented according to a degeneracy ordering of the undirected subgraph $G[N_u(\vec{G})]$. Unless otherwise specified, every DAG constructed in this paper is oriented according to a degeneracy ordering of its underlying undirected graph.

Example 2.2. Consider the graph G shown in Figure 3a, which has a degeneracy ordering of $u_1 \prec u_2 \prec \cdots \prec u_9$. The DAG $\vec{G}$ in Figure 3a is oriented according to this ordering. Similarly, $\vec{G}_{u_1}$, the subgraph induced by the out-neighbors of u_1, is oriented according to its degeneracy ordering $u_2 \prec u_4 \prec u_3 \prec u_5 \prec u_6 \prec u_9$.

Definition 2.3 (k-clique). A *k-clique* in a graph G is a set of k vertices that are pairwise adjacent.

Definition 2.4 (k-star). A *k-star* in a directed graph $\vec{G}$ is a set of k vertices that consists of a center vertex v and $k-1$ distinct out-neighbors of v.

For notational convenience, we refer to this structure as a k-star, rather than a $(k-1)$-star, as is common in some prior work. Notably, a set of k vertices in a DAG forms a k-clique only if it forms a k-star. This property ensures that the set of k-stars constitutes a valid sample space for estimating the number of k-cliques.

Example 2.5. Consider the DAG $\vec{G}$ in Figure 3a. The set $\{u_1, u_2, u_3, u_4, u_5\}$ forms a 5-clique, and it also forms a 5-star centered at u_1, as u_2, u_3, u_4 and u_5 are out-neighbors of u_1.

Problem statement. Given a graph G and an integer k, the goal is to estimate the number of k-cliques in G with an approximation guarantee. Specifically, we aim to compute an estimator $\hat{c}(G, k)$ for the number of k-cliques in G, denoted by $c(G, k)$, such that

$$\Pr\left(\frac{\left|\hat{c}(G,k) - c(G,k)\right|}{c(G,k)} > \varepsilon\right) \leq \delta.$$

Table 1 lists the notations frequently used in this paper.

Table 1: Frequently used notations

Notation	Meaning
$c(G, k)$	number of k-cliques in G
$\hat{c}(G, k)$	estimate of the number of k-cliques in G
$s(\vec{G}, k)$	number of k-stars in $\vec{G}$
$D_u(G)$ or D_u	number of neighbors of u in G
$d_u(\vec{G})$ or d_u	number of out-neighbors of u in $\vec{G}$
$N_u(\vec{G})$ or N_u	out-neighbors of u in $\vec{G}$
$\vec{G}_u$	subgraph of $\vec{G}$ induced by N_u and oriented according to degeneracy ordering of $G[N_u]$
α	degeneracy of the input graph

2.1 Related work

***k*-clique listing.** Chiba and Nishizeki [8] introduced the first practical algorithm for k-clique listing. This was improved by Danisch et al. [10] using a DAG based on a degeneracy ordering, where k-cliques are listed by finding $(k-1)$-cliques within out-neighbors of each vertex. Subsequent improvements include [19, 24, 37, 42].

***k*-clique counting.** While listing algorithms can count k-cliques, they must explicitly visit every k-clique. This becomes computationally prohibitive for larger k, as the number of k-cliques can grow exponentially. To address this, Jain and Seshadhri [15] proposed Pivoter, which computes exact k-clique counts without explicit enumeration. However, for larger k where exact counting becomes computationally prohibitive, approximate counting is necessary.

***k*-clique count estimation.** Jain and Seshadhri [16] proposed Turán-Shadow, which estimates counts via sampling vertex sets from a collection of induced subgraphs with density above the Turán threshold. Ye et al. [38] proposed DPColorPath, which computes exact counts for sparse regions using Pivoter and estimates counts for dense regions via *path sampling*. Although both Turán-Shadow and DPColorPath offer approximation guarantees in theory, they often require too many sample trials to be practical. Consequently, their implementations use a fixed number of sample trials, thereby forfeiting the theoretical guarantee [5]. Utilizing the *stopping rule theorem* [9], Chang et al. [5] proposed SR-kCCE, which is the first algorithm to provide approximation guarantee in practice.

3 OUR APPROACH

3.1 Overview

In this subsection, we present an overview of our approach (Algorithm 1).

Our algorithm consists of two stages, following the Monte Carlo method. In Stage 1, we construct a sample space $\mathcal{S}$ for Stage 2. In Stage 2, we estimate the number of k-cliques using the Monte Carlo method over the sample space $\mathcal{S}$.

The more effort we invest in Stage 1, the smaller the sample space for Stage 2 becomes. To balance the running times of Stage 1 and Stage 2, Chang et al. [5] proposed stopping Stage 1 once the expected running time of Stage 2 becomes smaller than the time already spent in Stage 1. We adopt the same criterion for determining when to stop Stage 1.

However, it is important to note that the overall efficiency of the algorithm depends not just on how much time is spent in Stage 1, but on how effectively that time is used to reduce the sample space. Our sample space reduction strategy in Stage 1 (detailed in Section 3.2) produces a smaller sample space in less time than the strategy used by the state-of-the-art algorithm SR-kCCE, thereby improving the efficiency of Stage 2 while keeping the overhead of Stage 1 low.

Furthermore, the structure of the k-vertex set in the sample space significantly impacts the overall efficiency of both stages. We construct our sample space using stars, in contrast to the paths used by DPColorPath and SR-kCCE or the arbitrary vertex sets in Turán-Shadow. Using stars has two key advantages. First, in practice, stars yield a significantly smaller sample space than paths or arbitrary vertex sets. Second, stars are much simpler and more lightweight to sample than paths. This simplicity is crucial, as the sampling procedure is executed repeatedly throughout the algorithm.

Example 3.1. Consider the DAG $\vec{G}$ in Figure 3a. There are $\binom{9}{5} = 126$ 5-vertex sets in $\vec{G}$. Observe that $\{u_1, u_2, u_3, u_4, u_5\}$ forms both a star and a directed path, whereas $\{u_2, u_3, u_4, u_5, u_6\}$ forms a directed path but not a star due to the missing edge from u_2 to u_6. In contrast, $\{u_1, u_2, u_3, u_4, u_9\}$ forms a star but not a path due to the missing edge from u_4 to u_9. In fact, out of 126 5-vertex sets, only 27 form stars, while 119 form directed paths.

Stage 1: Sample space construction. Both existing algorithms and our algorithm CREST share a key underlying observation for sample space construction: there is a one-to-one correspondence between k-cliques in a DAG $\vec{G}$ and $(k-1)$-cliques in the subgraphs induced by the out-neighbors of each vertex. Specifically, for any vertex u, a $(k-1)$-clique in $\vec{G}_u$ combined with u forms a k-clique in $\vec{G}$. Furthermore, since $\vec{G}$ is a DAG, there is exactly one such vertex u, which has all the other vertices in the k-clique as its out-neighbors.

Example 3.2. Consider $\vec{G}$ and the subgraphs induced by the out-neighbors of each vertex, as shown in Figure 3a. (The subgraphs $\vec{G}_{u_6}, \ldots, \vec{G}_{u_9}$ are omitted, as they contain fewer than 4 vertices and therefore no 4-cliques.) Here, adding vertex u_1 to $\{u_2, u_3, u_4, u_5\}$ (i.e., a 4-clique in $\vec{G}_{u_1}$), yields $\{u_1, u_2, u_3, u_4, u_5\}$ (i.e., a 5-clique in $\vec{G}$). Similarly, adding vertex u_3 to $\{u_4, u_5, u_6, u_7\}$ (i.e., a 4-clique in $\vec{G}_{u_3}$) yields $\{u_3, u_4, u_5, u_6, u_7\}$ (i.e., a 5-clique in $\vec{G}$). Observe that $\vec{G}$ contains 8 5-cliques, and the subgraphs $\vec{G}_{u_1}, \vec{G}_{u_2}, \vec{G}_{u_3}, \vec{G}_{u_4}, \vec{G}_{u_5}$ contain 4, 0, 2, 1, 1 4-cliques, respectively, which add up to 8.

This observation enables the Monte Carlo estimation over a reduced sample space. Specifically, one can estimate the number of k-cliques in the input graph instead by repeatedly sampling a $(k-1)$-vertex set from one of the out-neighbor-induced subgraphs and checking whether the selected vertices form a $(k-1)$-clique. Each time we sample a $(k-1)$-vertex set from $\vec{G}_u$, we interpret it as implicitly containing the vertex u, thereby forming a k-vertex set in the input graph. A sampled $(k-1)$-vertex set in $\vec{G}_u$ forms a $(k-1)$-clique in $\vec{G}_u$ if and only if the $(k-1)$-vertex set together with the vertex u forms a k-clique in the input graph. We refer to this process of replacing a sample space by a set of smaller sample spaces as *refinement*.

Example 3.3. Consider Figure 3a. We can estimate the number of 5-cliques in $\vec{G}$ by instead applying the Monte Carlo method over the sample space consisting of 4-vertex sets from $\vec{G}_{u_1}, \ldots, \vec{G}_{u_5}$. The DAG $\vec{G}$ contains 27 5-stars, whereas $\vec{G}_{u_1}, \ldots, \vec{G}_{u_5}$ contain 6, 1, 2, 1, and 1 4-stars, respectively, adding up to only 11. In this way, the sample space size is reduced by 16. In this case, a 4-star sampled from an out-neighbor-induced subgraph corresponds to a 5-vertex set in the input graph.

Algorithm 1: Our algorithm

Input : Graph G, integer k, error tolerance ε, failure probability δ

Output : An estimate of the number of k-cliques in G with an (ε, δ)-approximation guarantee

```
/* Stage 1: Sample space construction                    */
```
Initialize $\mathcal{Q}$ with $(\vec{G}, k)$.
Initialize $C_{\text{exact}} := 0$.
while *time spent in Stage 1 < expected runtime for Stage 2* **do**
  Pop $(\vec{G}, h)$ from $\mathcal{Q}$ that offers the largest potential reduction in the sample space. // (Sec. 3.2)
  foreach $u \in V_{\vec{G}}$ **do**
    If $\vec{G}_u$ is a split graph, increment C_{exact} by the exact number of $(h-1)$-cliques in $\vec{G}_u$. // (Sec. 3.4)
    Otherwise, push $(\vec{G}_u, h-1)$ into $\mathcal{Q}$.

```
/* Stage 2: Monte Carlo estimation                       */
```
Initialize $t := 0$ and $r := 0$.
while *approximation guarantee not achieved* // (Sec. 3.5)
do
  Select $(\vec{G}, h) \in \mathcal{Q}$ with probability proportional to $s(\vec{G}, h)$, and sample an h-star from $\vec{G}$. // (Sec. 3.3)
  If the sampled h-star forms an h-clique, increment r.
  Increment t.
return $\frac{r}{t} \times \sum_{(\vec{G},h)\in\mathcal{Q}} s(\vec{G}, h) + C_{\text{exact}}$

The refinement can be applied recursively, and this recursive process generates a search tree. Each node in the search tree is denoted by a pair $(\vec{G}, h)$, which we call a *sample instance*, that represents a sample space which is the set of h-stars in $\vec{G}$. The sample spaces represented by the leaf nodes of this search tree collectively constitute the overall sample space $\mathcal{S}$ used for the Monte Carlo estimation.

Example 3.4. Figure 3b shows an example search tree resulting from two refinements: we first apply refinement to $(\vec{G}, 5)$, and then to $(\vec{G}_{u_1}, 4)$. The overall sample space consists of four 3-stars in $\vec{G}_{u_1u_2}$, $\vec{G}_{u_1u_3}$, and $\vec{G}_{u_1u_4}$, together with five 4-stars in $\vec{G}_{u_2}, \ldots, \vec{G}_{u_5}$, adding up to a total size of 9.

Consider a leaf node $(\vec{G}', h)$, where $\vec{G}'$ is the subgraph of $\vec{G}$ induced by the out-neighbors of vertices $u_1, \ldots, u_{k-h}$ on the path from the root to the leaf node. If an h-star in $\vec{G}'$ forms an h-clique, this h-star corresponds to a k-clique, consisting of the vertices of this h-star together with $u_1, \ldots, u_{k-h}$. For example, the 4-star $\{u_4, u_5, u_6, u_7\}$ in $\vec{G}_{u_3}$ is a 4-clique, and it corresponds to the 5-clique $\{u_3, u_4, u_5, u_6, u_7\}$ in $\vec{G}$. Within a single leaf node $(\vec{G}', h)$, no two

h-stars consist of the same vertex set. Moreover, for any two distinct leaf nodes, the vertices $u_1, \ldots, u_{k-h}$ on the path from the root are distinct due to the acyclic orientation. Thus, if an h-star in the overall sample space forms an h-clique, it corresponds to a unique k-clique in $\vec{G}$. Therefore, the collection of h-stars in the leaf nodes of this search tree constitute the overall sample space $\mathcal{S}$ for the Monte Carlo estimation.

In Stage 1, we progressively reduce the sample space by repeatedly selecting a sample instance and applying refinement. It is important to carefully decide which sample instance to refine, as our goal in Stage 1 is to obtain a sufficiently small sample space as quickly as possible, improving efficiency in Stage 2 while keeping Stage 1 overhead low. Section 3.2 describes how we select sample instances for refinement.

Example 3.5. Consider Figure 3b. By selecting $(\vec{G}_{u_1}, 4)$ for the second refinement (according to our refinement strategy), we reduce the sample space by 2. In contrast, had we selected $(\vec{G}_{u_2}, 4)$ instead, the reduction would have been only 1.

We manage the search space refinement process in Stage 1 by maintaining a set Q of the leaf nodes of the search tree (lines 1–7 of Algorithm 1). We select from Q a sample instance $(\vec{G}, h)$ that offers the largest potential reduction in the sample space (line 4), and replace it by new sample instances $(\vec{G}_u, h-1)$ for each vertex u in $\vec{G}$ (lines 5–7). This strategy is designed to reduce the sample space as efficiently as possible with a small number of refinements. During refinement, if the underlying graph $\vec{G}_u$ for a new sample instance $(\vec{G}_u, h-1)$ is a *split graph*, we compute the exact number of $(h-1)$-cliques using our combinatorial counting technique, and directly add this count to the final estimate (line 6). Otherwise, the sample instance $(\vec{G}_u, h-1)$ is added to Q (line 7). When Stage 1 finishes, the sample space $\mathcal{S}$ for Stage 2 is given by the union of all h-stars in $\vec{G}$ for all remaining sample instances $(\vec{G}, h)$ in Q. The total number of stars in the sample space $\mathcal{S}$ will be called the *sample space size* $|\mathcal{S}|$.

Stage 2: Monte Carlo estimation. In Stage 2, we estimate the number of k-cliques using the Monte Carlo method on the sample space constructed during Stage 1. Specifically, we repeatedly sample a star from the sample space and check whether it forms a clique.

During Stage 2 (lines 8–14 of Algorithm 1), we maintain two variables, namely t and r, representing the number of sampled stars and the number of sampled stars that form cliques, respectively. To sample a star uniformly at random from the sample space, we first select a sample instance $(\vec{G}, h)$ from Q with probability proportional to the number of h-stars in $\vec{G}$, and then sample an h-star from $\vec{G}$ (line 11). We stop Stage 2 once a sufficient number of samples have been drawn to ensure an (ε, δ)-approximation guarantee(line 9). To this end, we adopt the theorem proposed by Audibert et al. [2]. The final estimate is the approximate count (i.e., $\frac{r}{t}$ multiplied by the sample space size) plus the exact count computed in Stage 1 (line 14).

3.2 Sample space refinement strategy

Stage 1 aims to construct a small sample space to ensure high sampling success probability $\frac{|C|}{|\mathcal{S}|}$, i.e., the number of cliques divided by the sample space size, while keeping its own running time low by avoiding excessive refinements. To achieve this, we must carefully decide which sample instances to refine further.

If we could tell how much reduction in sample space the refinement of each sample instance would yield, a natural strategy would be to refine the sample instance that yields the greatest reduction (recall Example 3.5). However, finding such a sample instance would be costly in practice, as it requires constructing the subgraph induced by the out-neighbors of each vertex and computing the number of $(h-1)$-vertex sets in that subgraph with the specific structure (to be contained in the sample space) for every sample instance $(\vec{G}, h)$ in Q.

To avoid this overhead, we introduce an efficient heuristic. For each sample instance $(\vec{G}, h)$ in Q, we approximately compute the number of h-stars that do not form cliques, i.e.,

$$s(\vec{G}, h) - \hat{c}(\vec{G}, h), \tag{1}$$

where $\hat{c}(\vec{G}, h)$ is an estimate of the number of h-cliques in $\vec{G}$, and then simply apply refinement to the sample instance that maximizes this value. Formula (1) provides an approximate upper bound on the reduction in sample space, allowing our strategy to mimic the ideal one without expensive overhead. When a sample instance is refined into smaller sample instances, the number of sample structures (in our case, stars) may decrease, while the number of cliques remains unchanged. Since each clique has a corresponding sample structure, the number of sample structures after refinement must be at least the number of cliques. Hence, *the difference between the number of sample structures and the number of cliques serves as an upper bound on the potential reduction in sample space.* Of course, the actual number of cliques in a subgraph is unknown. To address this, we obtain a rough estimate of the number of cliques $\hat{c}(\vec{G}, h)$ in the subgraph by sampling a small number of stars within the subgraph. This estimate does not need to be highly accurate, as it is used solely to guide refinement decisions, rather than to guarantee correctness.

We explain the sampling budget (i.e., the number of sample trials to obtain the rough estimate) for a sample instance $(\vec{G}, h)$. Performing sample trials on $\vec{G}$ requires first constructing the adjacency matrix of $\vec{G}$, which takes $O(n^2)$ time where n is the number of vertices in $\vec{G}$. In CREST, the sampling budget for a sample instance $(\vec{G}, h)$ is set to $\frac{n^2}{h^2}$. That is, we use an adaptive sampling budget that increases as the size n of the subgraph grows, and decreases as the size h of cliques grows. Since checking whether a sampled h-star forms an h-clique takes $O(h^2)$ time, performing a total of $\frac{n^2}{h^2}$ sample trials incurs $O(n^2)$ time, which is subsumed by the $O(n^2)$ time complexity already paid to construct the adjacency matrix of $\vec{G}$.

Comparison with existing approaches. The state-of-the-art algorithm SR-kCCE refines a sample instance with the (roughly estimated) minimum sampling success probability. This metric does not directly account for the actual reduction in sample space size; consequently, the refinement process does not necessarily yield a smaller sample space. For instance, SR-kCCE may prioritize refining an already small sample instance simply because it exhibits a low success probability, providing negligible benefits toward sample space reduction. In contrast, our strategy achieves more direct reduction by explicitly accounting for the sample space size. In fact, on most benchmark datasets, SR-kCCE often yields a sample space even larger than that produced by using a simple FIFO strategy with the same number of refinements, although SR-kCCE incurs additional overhead for maintaining a priority queue.

3.3 Star-based sampling approach

We explain how to compute the number $s(\vec{G}, h)$ of h-stars in $\vec{G}$ and sample h-stars uniformly at random.

For a vertex u in a DAG $\vec{G}$, there are $\binom{d_u}{h-1}$ h-stars centered at u, where d_u is the out-degree of u, since u together with any subset of $h-1$ distinct out-neighbors forms an h-star. Thus, the total number of h-stars in $\vec{G}$ is $s(\vec{G}, h) = \sum_{u \in V_{\vec{G}}} \binom{d_u}{h-1}$.

Example 3.6. Consider the DAG $\vec{G}_{u_1}$ in Figure 3a. Since the out-degree of u_2 is 4, there are $\binom{4}{3} = 4$ 4-stars centered at u_2 in $\vec{G}_{u_1}$. Additionally, each of u_3 and u_4 has out-degree 3, contributing $\binom{3}{3} = 1$ 4-star. Thus, the total number of 4-stars in $\vec{G}_{u_1}$ is $4 + 1 + 1 = 6$.

For a given DAG $\vec{G}$, uniform sampling of an h-star is performed in two steps:

(1) Select a center vertex v with probability proportional to the number of h-stars centered at v.
(2) Select $h-1$ distinct out-neighbors of v uniformly at random.

For step (1), the probability that a vertex v is selected as the center is

$$\frac{\binom{d_v}{h-1}}{\sum_{u \in V_{\vec{G}}} \binom{d_u}{h-1}},$$

because there are $\binom{d_u}{h-1}$ h-stars centered at each vertex u. Since the number of h-stars centered at each vertex can be computed directly from its degree, computing this distribution takes $O(|V_{\vec{G}}|)$ time. To efficiently select a center vertex according to this distribution, we adopt the alias method [36], which enables selecting a center vertex in $O(1)$ time after constructing an alias structure in $O(|V_{\vec{G}}|)$ time and space. Thus, preprocessing for our star-based sampling consists of (i) counting the number of h-stars centered at each vertex u, and (ii) constructing an alias structure. This preprocessing requires $O(|V_{\vec{G}}|)$ time and space.

For step (2), we perform a partial Fisher-Yates shuffle (adapted from Algorithm P [17]). Let the d_v out-neighbors of the center vertex v be stored in an array A in arbitrary order. We first swap $A[1]$ with a randomly chosen vertex from $A[1], \ldots, A[d_v]$, then swap $A[2]$ with a randomly chosen vertex from $A[2], \ldots, A[d_v]$, and so on. After completing $h-1$ iterations, the first $h-1$ vertices in A, together with the center vertex s, form the selected h-star. Each sampling of an h-star centered at v is thus performed in $O(h)$ time.

The edge orientation determines the out-degrees of the vertices, and thereby the total number of h-stars. To reduce the number of h-stars, we orient the edges of $\vec{G}$ according to a degeneracy ordering. In such an orientation, each vertex has out-degree at most the *degeneracy* [11], which is small in real-world networks. This prevents vertices from having disproportionately high out-degrees. Consequently, a DAG oriented according to a degeneracy ordering contains a small number of h-stars in practice. In our implementation, CREST simply breaks ties based on vertex IDs when computing degeneracy orderings.

Comparison with existing approaches. Ye et al. [38] proposed a method for sampling h-paths (i.e., a directed path consisting of h vertices) from a DAG based on a color ordering, and the state-of-the-art algorithm SR-kCCE [5] adopts this method.

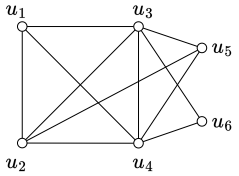


Figure 4: Example split graph

Sampling h-stars is more effective than sampling h-paths for the clique count estimation problem. In a DAG with maximum out-degree Δ, the number of h-paths starting at any vertex u is bounded by Δ^{h-1}, whereas the number of h-stars centered at u is bounded by $\binom{\Delta}{h-1} \approx \frac{\Delta^{h-1}}{(h-1)!}$. In practice, most subgraphs contain fewer h-stars than h-paths, resulting in a smaller sample space for sampling h-stars (as demonstrated in our experiments). Consequently, a sampled h-star is more likely to form an h-clique than a sampled h-path. This reduces the number of required samples to estimate the number of h-cliques with a given approximation guarantee.

Furthermore, compared to sampling h-paths, sampling h-stars is significantly simpler and more lightweight. Using a lightweight sampling method is crucial, as sampling is performed repeatedly on numerous subgraphs throughout the algorithm. As explained previously, preprocessing for our star-based sampling approach requires only $O(|V_G|)$ time and space. In contrast, uniform sampling of h-paths requires a more costly preprocessing, which involves counting the number of 1-paths, 2-paths, ..., h-paths starting at each vertex via dynamic programming. This requires $O(h \cdot c \cdot |V_G| + |E_G|)$ time and $O(h \cdot |V_G| + |E_G|)$ space [38], where c denotes the maximum color value.

3.4 Combinatorial counting

In this subsection, we describe how to efficiently compute the exact number of cliques within certain subgraphs.

Recall that in Stage 1, we maintain a set $\mathcal{Q}$ of sample instances that will collectively form the sample space for Stage 2. If the h-clique count for a sample instance $(\vec{G}, h)$ in $\mathcal{Q}$ can be computed efficiently, we can remove it from $\mathcal{Q}$ and add the count to our final estimate later. This allows us to perform the Monte Carlo estimation on a reduced sample space in Stage 2.

Definition 3.7 (Split graph [14]). A *split graph* is a graph in which the vertices can be partitioned into a clique and an independent set.

Example 3.8. Figure 4 shows a split graph, where the vertex set is partitioned into a clique $\{u_1, u_2, u_3, u_4\}$ and an independent set $\{u_5, u_6\}$.

For a sample instance $(\vec{G}, h)$ in $\mathcal{Q}$, if the undirected underlying graph of $\vec{G}$ is a *split graph*, we directly compute the exact number of h-cliques in $\vec{G}$ using a combinatorial method and remove the sample instance from $\mathcal{Q}$. As a result, the sample instance is no longer considered for refinement in Stage 1.

In case the underlying graph of $\vec{G}$ is a split graph (which can be verified based on the vertex degrees [14] in $O(|E_G|)$ time), we directly compute the exact number of h-cliques in $\vec{G}$ as follows. Suppose we have a split graph G whose vertex set is partitioned

Table 2: Time and space complexities of each stage for CREST and SR-kCCE. Time complexities are expressed in terms of the number of subgraphs considered during Stage 1 ($A_{\mathbf{CR}}$ for CREST and $A_{\mathbf{SR}}$ for SR-kCCE), the number of alias constructions during Stage 2 ($B_{\mathbf{CR}}$ for CREST and $B_{\mathbf{SR}}$ for SR-kCCE), and the number of sample trials during Stage 2 ($C_{\mathbf{CR}}$ for CREST and $C_{\mathbf{SR}}$ for SR-kCCE). Space complexities are expressed in terms of the maximum size of priority queue ($M_{\mathbf{CR}}$ for CREST and $M_{\mathbf{SR}}$ for SR-kCCE). $|E|$ is the number of edges in the input graph, and α is the degeneracy of the input graph.

	Time Complexity		Space Complexity	
	CREST	**SR-kCCE**	**CREST**	**SR-kCCE**
Stage 1	$O(\alpha^2 A_{\mathrm{CR}})$	$O(\alpha^2 k A_{\mathrm{SR}})$	$O(\alpha M_{\mathrm{CR}} + \lvert E\rvert)$	$O(\alpha M_{\mathrm{SR}} + \lvert E\rvert)$
Stage 2	$O(\alpha^2 B_{\mathrm{CR}} + k^2 C_{\mathrm{CR}})$	$O(\alpha^2 k B_{\mathrm{SR}} + k^2 C_{\mathrm{SR}})$	$O(\alpha M_{\mathrm{CR}} + \lvert E\rvert)$	$O(\alpha M_{\mathrm{SR}} + \lvert E\rvert)$

Table 3: The number of subgraphs constructed during Stage 1 ($A_{\mathbf{CR}}$ for CREST and $A_{\mathbf{SR}}$ for SR-kCCE), the number of alias constructions during Stage 2 ($B_{\mathbf{CR}}$ for CREST and $B_{\mathbf{SR}}$ for SR-kCCE), the number of sample trials during Stage 2 ($C_{\mathbf{CR}}$ for CREST and $C_{\mathbf{SR}}$ for SR-kCCE), and the maximum size of priority queue ($M_{\mathbf{CR}}$ for CREST and $M_{\mathbf{SR}}$ for SR-kCCE)

Dataset	k	Stage 1 # Subgraphs		Stage 2 # Alias		Stage 2 # Samples		Max Queue Size	
		A_{CR}	A_{SR}	B_{CR}	B_{SR}	C_{CR}	C_{SR}	M_{CR}	M_{SR}
Skitter	12	$\mathbf{2.69 \times 10^5}$	8.32×10^5	$\mathbf{4.27 \times 10^3}$	4.11×10^4	$\mathbf{2.49 \times 10^7}$	3.59×10^7	$\mathbf{3.72 \times 10^4}$	1.42×10^5
Skitter	21	$\mathbf{1.09 \times 10^5}$	5.64×10^5	$\mathbf{1.86 \times 10^4}$	2.73×10^4	$\mathbf{9.11 \times 10^6}$	5.82×10^7	$\mathbf{1.53 \times 10^4}$	1.92×10^5
Google-Plus	12	$\mathbf{4.76 \times 10^4}$	8.25×10^5	$\mathbf{4.29 \times 10^4}$	9.55×10^4	$\mathbf{1.32 \times 10^7}$	5.12×10^7	$\mathbf{2.99 \times 10^4}$	4.64×10^5
Google-Plus	21	$\mathbf{1.32 \times 10^5}$	1.84×10^6	8.69×10^4	$\mathbf{6.09 \times 10^4}$	$\mathbf{2.29 \times 10^7}$	2.63×10^8	$\mathbf{9.48 \times 10^4}$	7.58×10^5
Flickr	12	$\mathbf{5.30 \times 10^5}$	4.12×10^7	$\mathbf{3.29 \times 10^5}$	1.16×10^6	$\mathbf{7.35 \times 10^8}$	3.77×10^9	$\mathbf{3.98 \times 10^5}$	2.20×10^7
Flickr	21	$\mathbf{2.07 \times 10^7}$	-	$\mathbf{1.42 \times 10^7}$	-	$\mathbf{1.47 \times 10^{10}}$	-	$\mathbf{1.93 \times 10^7}$	-

into a clique C and an independent set I. Then, any h-clique in G either (1) lies entirely within C, or (2) contains exactly one vertex u from I and contains $h-1$ vertices in C that are adjacent to that vertex. For case (1), there are $\binom{|C|}{h}$ such h-cliques. For case (2), there are $\binom{D_u}{h-1}$ such h-cliques for each vertex $u \in I$, where D_u denotes the number of neighbors of u, since any neighbor of u must be contained in C. In this way, the exact number of h-cliques in a split graph can be computed in $O(|V_G|)$ time.

For subgraphs identified as split graphs, we directly add their exact clique counts to C_{exact} (line 6 of Algorithm 1). Since this portion of the total count is error-free, we may increase the relative error tolerance ε for Stage 2. The derivation of this relaxed relative error tolerance is detailed in [5].

3.5 Stopping criterion for Stage 2

In this subsection, we explain our stopping criterion for Stage 2, which guarantees an (ε, δ)-approximation.

Recall that in Stage 2 we maintain two variables: the total number t of sampled stars, and the number r of stars that form cliques among the sampled stars. To ensure an (ε, δ)-approximation guarantee, we stop sampling once the following condition is satisfied:

$$\sqrt{\frac{r}{t}\left(1-\frac{r}{t}\right)\frac{2\ln(3/\delta)}{t}} + \frac{3\ln(3/\delta)}{t} < \frac{\varepsilon}{1+\varepsilon}\cdot\frac{r}{t} \tag{2}$$

This stopping criterion is derived from the theorem proposed by Audibert et al. [2].

The key advantage of our criterion (2) is that it directly uses the observed sampling success probability $\frac{r}{t}$, which gets more accurate with every new sample. In contrast, the criterion used by the state-of-the-art algorithm SR-kCCE relies only on the number of successful samples r and does not directly take into account the updated sampling success probability. Experimental results show that our criterion allows us to stop Stage 2 earlier than SR-kCCE in practice while providing the same approximation guarantee.

Theorem 3.9. *Stopping Stage 2 with criterion* (2) *provides an* (ε, δ)*-approximation guarantee.*

3.6 Theoretical analysis

In this subsection, we analyze the time and space complexities of CREST. Table 2 summarizes the time and space complexities of each stage for CREST and SR-kCCE.

Time complexity. In Table 2, the time complexity of Stage 1 is expressed in terms of the number of subgraphs constructed during Stage 1 (A_{CR} for CREST and A_{SR} for SR-kCCE). Similarly, the time complexity of Stage 2 is expressed in terms of (1) the number of alias structure constructions during Stage 2 (B_{CR} for CREST and B_{SR} for SR-kCCE), and (2) the number of sample trials during Stage 2 (C_{CR} for CREST and C_{SR} for SR-kCCE). The cost for one subgraph is $O(\alpha^2)$ for CREST and $O(\alpha^2 k)$ for SR-kCCE in both Stage 1 and Stage 2. The cost for one trial in Stage 2 is k^2, which is the time to verify whether a sampled vertex set forms a clique or not. Therefore, we get the time complexities in Table 2.

Table 3 reports the actual values for these parameters: A_{CR}, B_{CR}, C_{CR} for CREST and A_{SR}, B_{SR}, C_{SR} for SR-kCCE. A_{CR} and C_{CR} are consistently smaller than A_{SR} and C_{SR}, respectively, while B_{CR} is smaller than or comparable to B_{SR}. This trend also holds for datasets not shown in Table 3.

We analyze the time complexities of Stages 1 and 2 of CREST as follows.

Theorem 3.10. *Stage 1 of CREST runs in* $O(\alpha^2 A_{CR})$ *time, and Stage 2 of CREST runs in* $O(\alpha^2 B_{CR} + k^2 C_{CR})$ *time.*

Table 4: Datasets listed by degeneracy α, which is a standard measure of sparsity [11]. Higher α typically indicates denser regions and more cliques, making instances harder. For the top six datasets, the k-clique counts are computed using Pivoter. For the bottom six, Pivoter failed to finish (exceeding a 30-day time limit or 256 GB of memory).

dataset	# vertices	# edges	α
Google	875,713	4,322,051	44
Gowalla	196,591	950,327	51
Brightkite	58,228	214,078	52
Stanford	281,903	1,992,636	71
Skitter	1,696,415	11,095,298	111
Google-Plus	211,186	1,141,650	135
Livejournal	4,846,609	42,851,237	372
Flickr	1,715,255	15,555,041	568
UK-2005	39,454,463	783,027,125	588
Twitter	21,297,772	265,025,545	1,695
IT-2004	41,290,548	1,027,474,947	3,224
SK-2005	50,646,059	1,810,063,330	4,510

Space complexity. In Table 2, the space complexity of CREST (and SR-kCCE) is expressed in terms of M_{CR} (and M_{SR}), the maximum size of the priority queue (Q in Algorithm 1). Table 3 shows the actual values of these parameters M_{CR} and M_{SR}.

Theorem 3.11. *Stage 1 and Stage 2 of CREST require $O(\alpha M_{CR} + |E|)$ space.*

The preprocessing space complexity of CREST is $O(\alpha)$, where α is the degeneracy of the input graph, because the same memory space is reused for all alias constructions. Thus it is negligible compared to total space complexity $O(\alpha M_{\mathrm{CR}} + |E|)$ of CREST.

4 EXPERIMENTS

In this section, we conduct experiments to demonstrate the efficiency of our algorithm CREST.

4.1 Experimental setup

Algorithms. We compare our algorithm CREST against the state-of-the-art algorithm SR-kCCE [5]. We do not include earlier methods in our comparison because (1) SR-kCCE was shown to significantly outperform them, and (2) their implementations do not provide an approximation guarantee, as noted in [5]. The source code for CREST[2] and SR-kCCE[3] is publicly available.

Datasets. We obtained 12 real-world networks from the Network Repository[4], with sizes up to 1.8 billion edges. Table 4 lists the datasets used in the experiments. To obtain the exact number of k-cliques in each dataset, we used the state-of-the-art algorithm Pivoter [15] for the exact k-clique counting problem. However, on harder datasets, Pivoter did not finish within a 30-day time limit (Livejournal, Flickr, UK-2005, Twitter) or exceeded the available memory of 256 GB (IT-2004, SK-2005).

[2]https://github.com/SNUCSE-CTA/CREST
[3]https://github.com/LijunChang/kCliqueCountEstimation
[4]https://networkrepository.com

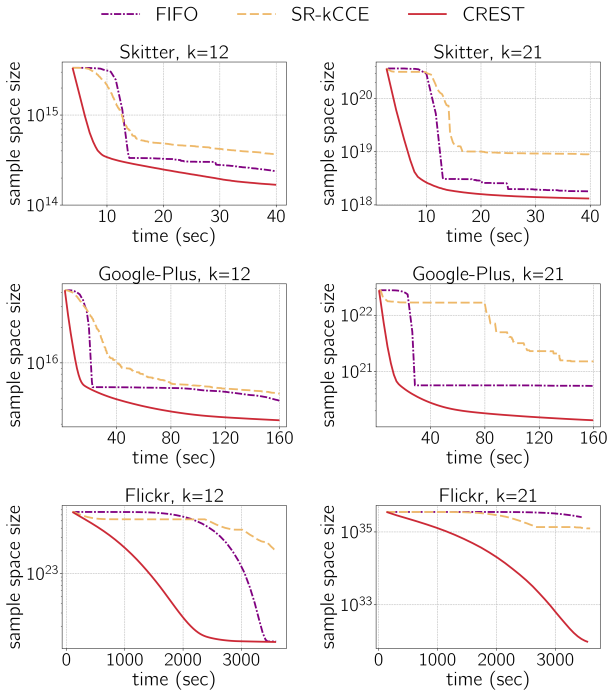


Figure 5: Comparison of different refinement strategies and their impact on sample space size over time. The x-axis shows the time spent on Stage 1 (sample space construction), while the y-axis shows the sample space size. The y-axis is in a logarithmic scale. The refinement strategy of CREST (solid red) is compared against that of SR-kCCE (dashed yellow) and a simple baseline FIFO (dash-dotted purple).

Experimental settings. The experiments were conducted on a Linux machine with two Intel Xeon E5-2680 v3 (2.50 GHz) CPUs and 256 GB of RAM. We set the relative error tolerance to $\varepsilon = 0.001$ and the failure probability to $\delta = 0.01$, adopting the default parameter values from prior work [5]. For each dataset, we ran experiments for clique sizes $k \in \{6, 9, 12, 15, 18, 21\}$, a range consistent with prior work [5] (which varied k up to 20). Reported running times include both Stage 1 (sample space construction) and Stage 2 (Monte Carlo estimation of the k-clique count). We set a time limit of 24 hours per instance. Reported averages are geometric means.

4.2 Comparison of individual techniques

Prior to the main experiment, we compare each individual technique of CREST against the corresponding technique of the state-of-the-art algorithm SR-kCCE, since each one has a direct counterpart in SR-kCCE. This helps us better understand the performance gains observed in the main experiment (Section 4.3).

Sample space refinement strategy. Figure 5 compares different refinement strategies and their effect on sample space size over time. We compare (1) the refinement strategy of CREST, (2) the refinement strategy of SR-kCCE, and (3) FIFO, a simple baseline that refines sample instances in a first-in-first-out manner without any priority. We provide the results for three representative datasets, Skitter, Google-Plus, and Flickr, which vary in size and

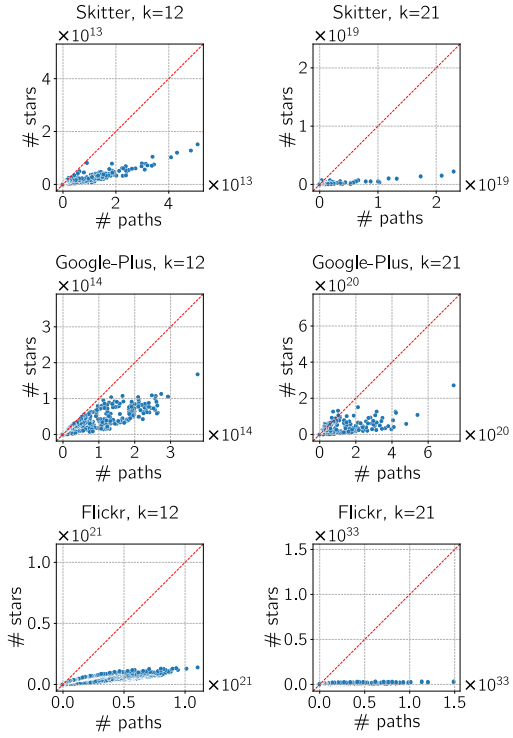


Figure 6: Comparison of the number of stars versus paths for induced subgraphs. Each data point represents a single induced subgraph; the x-axis value is the number of paths and the y-axis value is the number of stars. The red line represents the line $y = x$, which serves as a reference to show the general trend.

density. Overall, the refinement strategy of CREST achieves a sharp reduction in sample space size from the start and consistently obtains the smallest sample space throughout Stage 1. This is because it directly prioritizes sample instances expected to yield the largest reduction. In contrast, the refinement strategy of SR-kCCE initially performs reasonably well compared to FIFO, but is eventually outperformed by FIFO. This occurs when the refinement strategy of SR-kCCE repeatedly selects small sample instances with very low success probabilities, resulting in little reduction in sample space despite additional refinements.

Star-based sampling. Figure 6 compares the number of stars (constituting the sample space of CREST) and the number of paths (constituting the sample space of SR-kCCE and DPColorPath) for each induced subgraph considered during Stage 1. In Figure 6, each data point represents a single induced subgraph considered during Stage 1 (sample space construction) with the number of paths as the x-axis value and the number of stars as the y-axis value. Nearly all data points are below the red reference line $y = x$, indicating that, for nearly all induced subgraphs, the number of stars is lower than the number of paths. This gap widens for larger k values. Specifically, on Flickr for $k = 21$, the rightmost data point represents an induced subgraph containing 1.49×10^{33} paths, versus only 2.42×10^{31} stars, demonstrating a reduction of nearly two orders of magnitude.

Table 5: Proportion of exact counting in total count. Columns show number of k-cliques exactly counted during Stage 1 (C_{exact}), number of k-cliques estimated during Stage 2 (C_{est}), and total count ($C_{\text{exact}}+C_{\text{est}}$). The values in the parentheses for C_{exact} show the proportion of exact counting in total count.

Dataset	k	Algorithm	C_{exact}	C_{est}	Total count
Google	12	SR-kCCE	3.404×10^{10} (36 %)	5.965×10^{10}	9.369×10^{10}
		CREST	6.251×10^{10} (67 %)	3.118×10^{10}	9.369×10^{10}
Google	21	SR-kCCE	3.214×10^{12} (41 %)	4.693×10^{12}	7.907×10^{12}
		CREST	7.311×10^{12} (92 %)	5.953×10^{11}	7.907×10^{12}
UK-2005	12	SR-kCCE	6.107×10^{25} (52 %)	5.705×10^{25}	1.181×10^{26}
		CREST	1.079×10^{26} (91 %)	1.023×10^{25}	1.181×10^{26}
UK-2005	21	SR-kCCE	1.314×10^{39} (48 %)	1.451×10^{39}	2.765×10^{39}
		CREST	2.512×10^{39} (91 %)	2.532×10^{38}	2.765×10^{39}
IT-2004	12	SR-kCCE	8.302×10^{33} (66 %)	4.323×10^{33}	1.262×10^{34}
		CREST	1.262×10^{34} (100 %)	-	1.262×10^{34}
IT-2004	21	SR-kCCE	4.211×10^{54} (62 %)	2.543×10^{54}	6.755×10^{54}
		CREST	6.755×10^{54} (100 %)	-	6.755×10^{54}
SK-2005	12	SR-kCCE	2.638×10^{35} (99 %)	1.091×10^{33}	2.649×10^{35}
		CREST	2.649×10^{35} (100 %)	-	2.649×10^{35}
SK-2005	21	SR-kCCE	1.731×10^{57} (99 %)	1.278×10^{55}	1.744×10^{57}
		CREST	1.744×10^{57} (100 %)	-	1.744×10^{57}

Combinatorial counting. Table 5 shows the impact of our combinatorial counting technique. During Stage 1, SR-kCCE obtains exact counts for a sample instance $(\vec{G}, h)$ when (i) $\vec{G}$ is itself a clique (so the count is simply $\binom{|V_{\vec{G}}|}{h}$), or (ii) $\vec{G}$ is small enough to invoke an exact clique-counting algorithm. We provide results for Google, UK-2005, IT-2004, and SK-2005, as these are the datasets where exact counting contributes the highest percentage of total count in both CREST and SR-kCCE. On these datasets, the proportion of exact counting in the total count is high for CREST. Moreover, CREST's proportion is larger than SR-kCCE's. Notably, on IT-2004 and SK-2005 for $k > 6$, CREST requires no sample trials in Stage 2, as our combinatorial counting yielded a large number of exact k-clique counts, leaving only a negligible sample space. For instance, on IT-2004 for $k = 12$, CREST obtained $C_{\text{exact}} = 1.262 \times 10^{34}$ exact k-clique counts, and the sample space size for Stage 2 was $|\mathcal{S}| = 4.016 \times 10^{28}$. Since $|\mathcal{S}| < 0.001 \times C_{\text{exact}}$, the exact count C_{exact} alone guarantees the estimate within $\varepsilon = 0.001$ relative error tolerance, regardless of the number of k-cliques remaining in $\mathcal{S}$. On the other hand, SR-kCCE was required to perform sample trials in Stage 2 to obtain the remaining $C_{\text{est}} = 4.323 \times 10^{33}$ k-cliques.

4.3 Main experiment

Figure 7 shows our main experimental results. On instances where the exact number of k-cliques could be found with Pivoter, we confirmed that both CREST and SR-kCCE produced estimates within $\varepsilon = 0.001$ relative error tolerance. Across all datasets, CREST outperformed SR-kCCE in terms of running time while providing the same (ε, δ)-approximation guarantee. Specifically, CREST was up to 179.10× faster than SR-kCCE (on Brightkite for $k = 21$), achieving an average speedup of 12.68× across all datasets. On the most challenging instances (on Flickr and Twitter for $k > 12$), CREST was the only algorithm to finish within the 24-hour time limit.

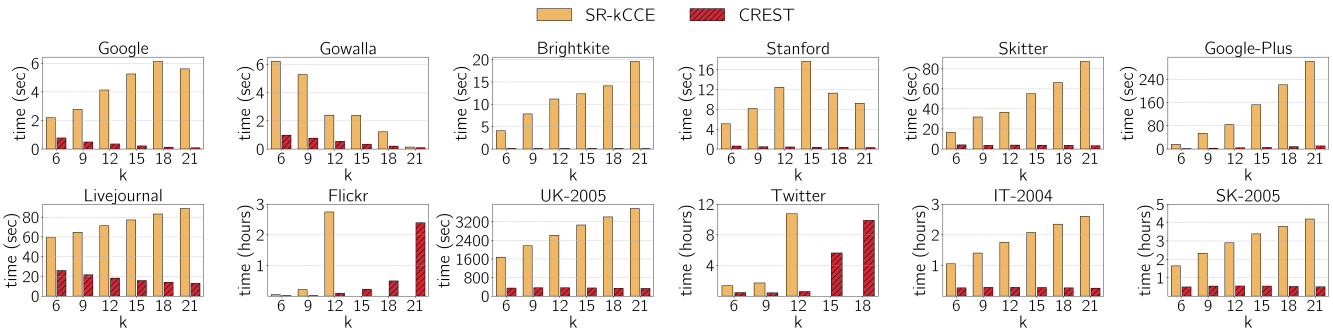


Figure 7: Running time on real-world networks with varying clique sizes k. Our algorithm CREST (hatched red) is compared against the state-of-the-art algorithm SR-kCCE (solid yellow).

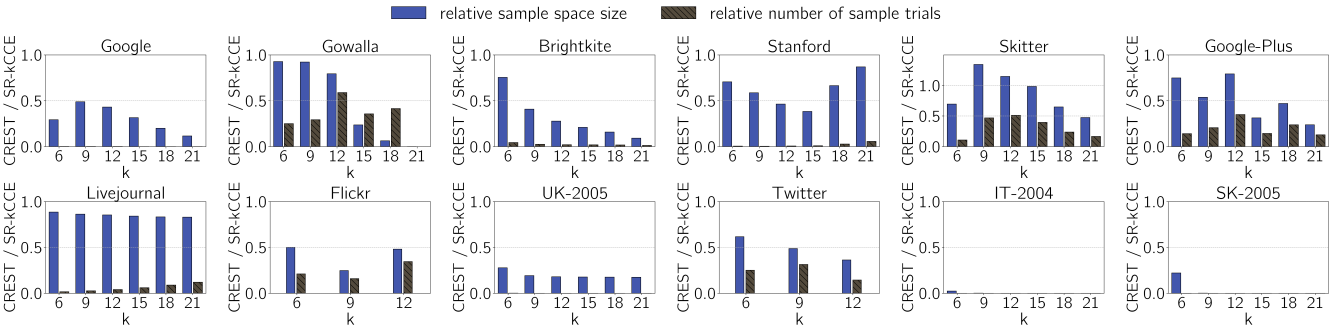


Figure 8: Relative sample space size constructed in Stage 1 (solid blue) and relative number of sample trials in Stage 2 (hatched gray). Each bar shows the ratio of CREST to SR-kCCE, and values below 1.0 indicate CREST outperforms SR-kCCE. On Flickr and Twitter for $k > 12$, bars are omitted because SR-kCCE did not finish within the 24-hour time limit. On IT-2004 and SK-2005 for $k > 6$, the bars for relative number of sample trials have zero height because CREST required no sample trials in Stage 2.

Figure 8 provides an analysis of (1) sample space sizes and (2) the number of sample trials for CREST and SR-kCCE (values below 1.0 indicate CREST outperforms SR-kCCE).

(1) CREST consistently constructed a smaller sample space than SR-kCCE, except for $k = 9$ and $k = 12$ on Skitter. On average, CREST constructed a sample space 221.90× smaller than that of SR-kCCE, while CREST's time for Stage 1 was only 14.83 % of SR-kCCE's time for Stage 1. In many datasets, the relative sample space size decreases as k grows. This trend is also visible in Skitter, where CREST constructed a sample space 1.34× larger than that of SR-kCCE for $k = 9$, but 2.11× smaller for $k = 21$. This efficiency owes to two key factors: the effectiveness of our refinement strategy and the compactness of our star-based sample space.
   - First, as demonstrated in Figure 5, the refinement strategy of CREST achieves a sharp reduction in sample space size with significantly fewer refinements. In Figure 5, on Skitter for $k = 21$, SR-kCCE took about 40 seconds to reach a sample space size of 9.35×10^{18}, at which point it proceeded to Stage 2. In contrast, our refinement strategy reached the same sample space size in about 6 seconds. Similarly, on Google-Plus for $k = 21$, SR-kCCE took about 160 seconds to reach a sample space size of 1.47×10^{21}, at which point it proceeded to Stage 2, whereas our refinement strategy reached the same sample space size in less than 10 seconds.
   - Second, as demonstrated in Figure 6, sample spaces consisting of stars are more compact than sample spaces consisting of paths. In Figure 6, on Skitter for $k = 21$, most induced subgraphs contain fewer stars than paths; the number of stars is on average 8.16× smaller than paths. Similarly, on Google-Plus for $k = 21$, the number of stars is on average 7.55× smaller than paths.

   A smaller sample space reduces the number of required sample trials, which in turn improves running time. As a result, CREST achieved speedups of 26.30× and 27.87× over SR-kCCE on Skitter and Google-Plus, respectively, for $k = 21$, as shown in Figure 7. The only exceptions where CREST constructed larger sample spaces were on Skitter for $k = 9$ and $k = 12$. In these instances, CREST performed fewer refinements in Stage 1 (CREST spent only about 10 % of SR-kCCE's time spent on Stage 1) and moved to Stage 2 earlier because our star-based sampling is sufficiently lightweight that the expected running time for Stage 2 was already small despite a larger sample space.
(2) Across all instances, CREST required fewer sample trials to obtain the same approximation guarantee. As demonstrated in Table 5, on IT-2004 for $k \in \{9, \ldots, 21\}$ and on SK-2005 for $k \in \{6, \ldots, 21\}$, CREST required no sample trials at all. In these instances, our combinatorial counting yielded a large number of exact k-clique counts and only a small sample space was left, so that the exact count alone guaranteed the estimate within

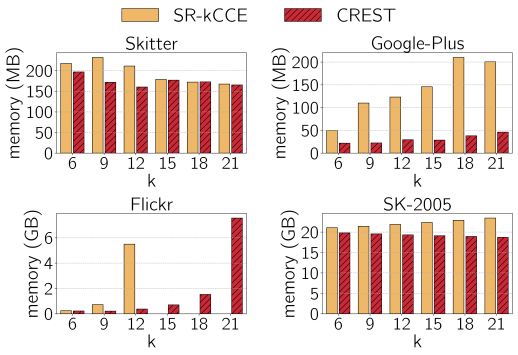


Figure 9: Peak memory usage of CREST and SR-kCCE on real-world networks for varying clique sizes k

the relative error tolerance ε. Moreover, the stopping criterion of CREST consistently requires fewer sample trials than that of SR-kCCE. In Figure 8, on Skitter for $k = 12$, CREST required about half the number of sample trials compared to SR-kCCE, even though CREST constructed a 1.14× larger sample space. The reduced number of sample trials directly translates to improved running time. Consequently, CREST achieved a speedup of 11.38× and 9.29× over SR-kCCE, on Skitter for $k = 12$, as shown in Figure 7.

4.4 Memory usage

We conducted an experiment to measure the peak memory usage of CREST and SR-kCCE (Figure 9). In most cases (including ones not shown in Figure 9), CREST achieves lower peak memory usage than SR-kCCE. Notably, on Google-Plus and Flickr, CREST uses significantly less memory than SR-kCCE. The peak memory consists of two factors: (i) the input graph and (ii) the additional memory required by the algorithm, of which the main contributor is the priority queue (Q in Algorithm 1) storing the subgraphs (i.e., sample instances). On many datasets, factor (i) dominates the memory usage. On Skitter, for example, about 150 MB of memory is already required for the input graph data structures, making the impact of factor (ii) relatively small. In contrast, on Google-Plus and Flickr, factor (ii) dominates the memory usage, where the input graph requires about 20 MB and 190 MB, respectively. Table 3 shows the maximum size of the priority queue (M_{CR} for CREST and M_{SR} for SR-kCCE). On Flickr for $k = 12$, CREST maintained up to 3.98×10^5 subgraphs in the priority queue during Stage 1, whereas SR-kCCE maintained up to 2.20×10^7 subgraphs.

4.5 Convergence

We conducted an experiment to compare how quickly the estimates produced by CREST and SR-kCCE converge to the actual number of k-cliques. As shown in Figure 10, the estimate produced by CREST converges to the actual number of k-cliques much faster than that by SR-kCCE. Notably, on Google, the estimates of CREST are already within $\varepsilon = 0.001$ relative error at the beginning of Stage 2. Also, the estimate lies within $\varepsilon = 0.001$ relative error well before CREST terminates (marked with •). On Skitter for $k = 21$, the estimate of CREST entered the 0.001 relative-error bound only after 1.9×10^5 sample trials and remained within this range thereafter. Similar results were observed on other datasets for which the actual number of k-cliques is known.

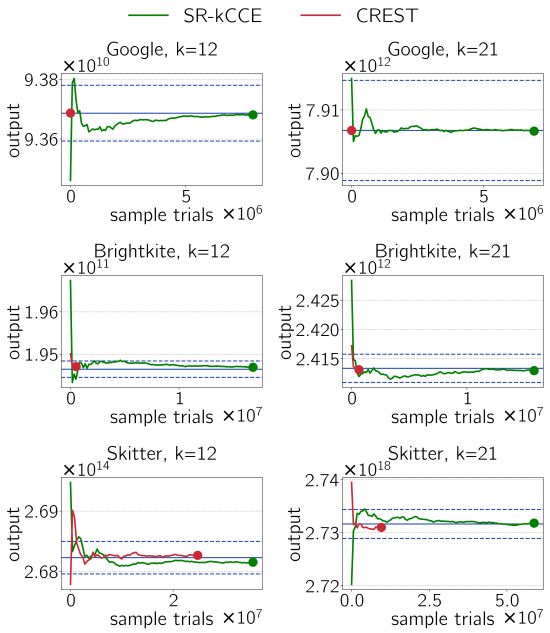


Figure 10: Estimates produced by CREST and SR-kCCE as the number of sample trials increases. The termination points of CREST and SR-kCCE are marked with •. The solid blue line indicates the actual number C_k of k-cliques, and the dashed blue lines indicate the bounds that satisfy the relative error guarantee for $\varepsilon = 0.001$, i.e., $(1 \pm 0.001) \times C_k$.

5 CONCLUSION

In this paper, we presented CREST, an efficient algorithm for approximate k-clique counting. Our approach integrates (i) a novel sample space refinement strategy, (ii) a star-based sampling approach, (iii) a combinatorial method for exact counting, and (iv) a new stopping criterion. Extensive experiments on real-world networks demonstrate that CREST significantly outperforms the state-of-the-art algorithm in terms of running time, while satisfying the same target accuracy requirements.

ACKNOWLEDGMENTS

Y. Nam, J. Jang, and K. Park were supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00551, Framework of Practical Algorithms for NP-hard Graph Problems). J. C. Na was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2020R1F1A1068873). H. Kim was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2020-II201373, Artificial Intelligence Graduate School Program (Hanyang University)).

REFERENCES

[1] James Abello, Panos M. Pardalos, and Mauricio G. C. Resende. 1998. On Maximum Clique Problems in Very Large Graphs. In *External Memory Algorithms*, James Abello and Jeffrey Scott Vitter (Eds.). DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 50. American Mathematical Society, Providence, RI, 119–130.

[2] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. 2007. Tuning Bandit Algorithms in Stochastic Environments. In *Proceedings of International Conference on Algorithmic Learning Theory*. 150–165.

[3] Austin R Benson, David F Gleich, and Jure Leskovec. 2016. Higher-order organization of complex networks. *Science* 353, 6295 (2016), 163–166.

[4] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. 2022. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45, 1 (2022), 657–668.

[5] Lijun Chang, Rashmiak Gamage, and Jeffrey Xu Yu. 2024. Efficient k-Clique Count Estimation with Accuracy Guarantee. In *Proceedings of the VLDB Endowment*. 3707–3719.

[6] Lijun Chang, Mouyi Xu, and Darren Strash. 2022. Efficient Maximum k-Plex Computation over Large Sparse Graphs. In *Proceedings of the VLDB Endowment*. 127–139.

[7] Jie Chen and Yousef Saad. 2010. Dense subgraph extraction with application to community detection. *IEEE Transactions on Knowledge and Data Engineering* 24, 7 (2010), 1216–1230.

[8] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing* 14, 1 (1985), 210–223.

[9] Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. 2000. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on Computing* 29, 5 (2000), 1484–1496.

[10] Maximilien Danisch, Oana Balalau, and Mauro Sozio. 2018. Listing k-Cliques in Sparse Real-World Graphs. In *Proceedings of the ACM Web Conference*. 589–598.

[11] David Eppstein, Maarten Löffler, and Darren Strash. 2010. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *International Symposium on Algorithms and Computation*. 403–414.

[12] Eleanor J Gardiner, Peter J Artymiuk, and Peter Willett. 1997. Clique-detection algorithms for matching three-dimensional molecular structures. *Journal of Molecular Graphics and Modelling* 15, 4 (1997), 245–253.

[13] Jiong Guo, Iyad A Kanj, Christian Komusiewicz, and Johannes Uhlmann. 2011. Editing graphs into disjoint unions of dense clusters. *Algorithmica* 61 (2011), 949–970.

[14] Peter L. Hammer and Bruno Simeone. 1981. The splittance of a graph. *Combinatorica* 1, 3 (1981), 275–284.

[15] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique counting. In *Proceedings of the ACM International Conference on Web Search and Data Mining*. 268–276.

[16] Shweta Jain and C. Seshadhri. 2022. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *Proceedings of the ACM Web Conference*. 1191–1202.

[17] Donald E Knuth. 1997. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (3rd ed.). Addison-Wesley Professional. Section 3.4.2.

[18] Ankam Ravi Kumar. 2005. Discovering large dense subgraphs in massive graphs. In *Proceedings of the VLDB Endowment*. 721–732.

[19] Rong-Hua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-Clique Listing. In *Proceedings of the VLDB Endowment*. 2536–2548.

[20] R. Duncan Luce. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15, 2 (1950), 169–190.

[21] R. Duncan Luce and Albert D. Perry. 1949. A method of matrix analysis of group structure. *Psychometrika* 14, 2 (1949), 95–116.

[22] Sergei Maslov and Kim Sneppen. 2002. Specificity and stability in topology of protein networks. *Science* 296, 5569 (2002), 910–913.

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

[24] Yehyun Nam, Jihoon Jang, Kunsoo Park, Jianye Yang, and Cheng Long. 2026. DIST: Efficient k-clique listing via induced subgraph trie. *The VLDB Journal* 35, 3, Article 19 (2026).

[25] Evelien Otte and Ronald Rousseau. 2002. Social network analysis: A powerful strategy, also for the information sciences. *Journal of Information Science* 28, 6 (2002), 441–453.

[26] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (2005), 814–818.

[27] John W. Raymond and Peter Willett. 2002. Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design* 16, 7 (2002), 521–533.

[28] Ahmet E. Sariyüce, Comandur Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. 2017. Nucleus decompositions for identifying hierarchy of dense subgraphs. *ACM Transactions on the Web* 11, 3 (2017), 1–27.

[29] Stephen B Seidman and Brian L Foster. 1978. A graph-theoretic generalization of the clique concept. *Journal of Mathematical Sociology* 6, 1 (1978), 139–154.

[30] Bintao Sun, Maximilien Danisch, TH Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. In *Proceedings of the VLDB Endowment*. 1628–1640.

[31] Brian K. Tanner, Gary Warner, Henry Stern, and Scott Olechowski. 2010. Koobface: The evolution of the social botnet. In *Proceedings of IEEE eCrime Researchers Summit*. 1–10.

[32] Charalampos Tsourakakis. 2015. The k-Clique Densest Subgraph Problem. In *Proceedings of the ACM Web Conference*. 1122–1132.

[33] Charalampos Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria Tsiarli. 2013. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 104–112.

[34] Charalampos E. Tsourakakis, Jakub Pachocki, and Michael Mitzenmacher. 2017. Scalable Motif-Aware Graph Clustering. In *Proceedings of the ACM Web Conference*. 1451–1460.

[35] Peter Uetz, Loic Giot, Gerard Cagney, Traci A. Mansfield, Richard S. Judson, James R. Knight, Daniel Lockshon, Vaibhav Narayan, Maithreyan Srinivasan, Pascale Pochart, et al. 2000. A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae. *Nature* 403, 6770 (2000), 623–627.

[36] Michael D. Vose. 1991. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering* 17, 9 (1991), 972–975.

[37] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2024. Efficient k-clique listing: An edge-oriented branching strategy. *Proceedings of the ACM on Management of Data* 2, 1, Article 7 (2024).

[38] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2022. Lightning Fast and Space Efficient k-Clique Counting. In *Proceedings of the ACM Web Conference*. 1191–1202.

[39] Xiaowei Ye, Miao Qiao, Rong-Hua Li, Qi Zhang, and Guoren Wang. 2024. Scalable k-clique Densest Subgraph Search. *arXiv preprint arXiv:2403.05775* (2024).

[40] Hao Yin, Austin R. Benson, and Jure Leskovec. 2018. Higher-Order Clustering in Networks. *Physical Review E* 97, 5 (2018), 052306.

[41] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. 2006. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics* 22, 7 (2006), 823–829.

[42] Zhirong Yuan, You Peng, Peng Cheng, Li Han, Xuemin Lin, Lei Chen, and Wenjie Zhang. 2022. Efficient k-Clique Listing with Set Intersection Speedup. In *Proceedings of IEEE International Conference on Data Engineering*. 1955–1968.

[43] Yingli Zhou, Qingshuo Guo, Yixiang Fang, and Chenhao Ma. 2024. A counting-based approach for efficient k-clique densest subgraph discovery. *Proceedings of the ACM on Management of Data* 2, 3, Article 119 (2024).