Efficient k-Clique Count Estimation with Accuracy Guarantee

Lijun Chang
The University of Sydney
Sydney, Australia
Lijun.Chang@sydney.edu.au

Rashmika Gamage
The University of Sydney
Sydney, Australia
rashmika.gamage@sydney.edu.au

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hong Kong, China
yu@se.cuhk.edu.hk

ABSTRACT

Counting and enumerating all occurrences of k-cliques, i.e., complete subgraphs with k vertices, in a large graph G is a fundamental problem with many applications. However, exact solutions are often infeasible due to the exponential growth in the number of k-cliques when k increases. Thus, a more practical approach is approximately counting and uniformly sampling k-cliques. Turán-Shadow and DPColorPath are two state-of-the-art algorithms for approximately counting k-cliques. The general idea is first constructing a sample space that is a superset of all k-cliques in G, and then sampling t *elements* uniformly-at-random (u.a.r.) from the sample space for a pre-determined t; the k-clique count is estimated as the sample space size multiplied by the ratio of k-cliques among the t samples. Although techniques have been proposed in Turán-Shadow for setting t to ensure the estimation accuracy, the theoretically chosen t is often too large to be practical. As a result, both of the existing algorithms used a fixed t in their implementations and thus do not offer accuracy guarantee. In this paper, we propose the first randomized algorithm that achieves the theoretical estimation accuracy and the practical efficiency at the same time. Different from the existing algorithms, we pre-determine the number s of *k-clique samples* that are required to achieve the estimation accuracy. Consequently, we can estimate the running time of the sampling stage (i.e., time taken to sample s k-cliques), for a given sample space. Then, we propose to balance the time of constructing/refining the sample space and the time of the sampling stage, by stopping the refinement of the sample space once the elapsed time is comparable to the estimated time of the sampling stage. Extensive empirical studies on large real graphs show that our algorithm SR-kCCE provides an accurate k-clique count estimation and also runs efficiently. As a by-product, our algorithm can also be used for *efficiently sampling* a certain number of k-cliques u.a.r. from G.

PVLDB Reference Format:
Lijun Chang, Rashmika Gamage, and Jeffrey Xu Yu. Efficient k-Clique Count Estimation with Accuracy Guarantee. PVLDB, 17(11): 3707 - 3719, 2024.
doi:10.14778/3681954.3682032

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://lijunchang.github.io/kCliqueCountEstimation.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3682032

1 INTRODUCTION

Counting and enumerating all occurrences of a small subgraph pattern (also known as graphlets or motifs) in a large graph is a fundamental problem with applications in may domains such as social network analysis, bioinformatics, cybersecurity and physics (see [6, 22] and references therein). Among the subgraph patterns, of particular interest is k-clique: a complete subgraph with k vertices. Small cliques have been used as building blocks for detecting overlapping communities [20], identifying large near-cliques [12, 25, 26], and uncovering the hierarchical structure of dense subgraphs [23].

As a special case of k-clique for $k = 3$, the triangle enumeration problem has been extensively studied [4, 5, 19]. The state-of-the-art algorithms enumerate all triangles in a graph G with n vertices and m edges in $O(m \cdot \alpha(G))$ time, where $\alpha(G) \leq \lceil \sqrt{2m+n} \rceil$ is the degeneracy of G. As $\alpha(G)$ usually is small for real graphs [9], all triangles in a graph can be efficiently enumerated in practice. However, the success of triangle enumeration has not been translated to an efficient enumeration of all k-cliques for a slightly larger k, e.g., 20, due to an exponential growth in the number of k-cliques. For example, every k-vertex-subset of a maximum clique is a k-clique. Let $\omega(G)$ be the maximum clique size of G. The number of k-cliques in G is at least $\binom{\omega(G)}{k}$ which is huge when $\omega(G)$ is large; as shown in Table 2, $\omega(G)$ could be hundreds or even thousands.

To overcome the overwhelming quantity of k-cliques, the Pivoter algorithm [14] counts k-cliques in a graph without explicitly enumerating all of them, by using a combinatorial counting technique in some parts of the enumeration process. Algorithmically, it is a simple adaptation of the maximal clique enumeration algorithm proposed in [9] by ignoring the exclusion set X. Nevertheless, a prominent insight was proved in [14] which shows that all cliques (of different sizes) are uniquely represented without duplication in the enumeration tree. As a result, Pivoter is able to simultaneously count k-cliques in G for all possible k values. The main ingredient that makes Pivoter more efficient than the existing k-clique enumerating algorithms (such as [5, 8]) is the pivoting technique introduced in [3, 9] which can significantly reduce the search space. However, as pointed out in [15] (with the same first-author as [14]), Pivoter still takes an extremely long time for some graphs. Also, as shown in our experiments, Pivoter does not finish within 5 hours for several of our tested graph instances.

In view of the inefficiency of enumerating all k-cliques and exactly counting the number of k-cliques, a more practical approach would be *uniformly* sampling k-cliques and *approximately* counting k-cliques. In particular, SCTL*-Sample [12] samples a set of 10^7 k-cliques for computing the k-clique densest subgraph. It first materializes the enumeration tree of Pivoter, referred to as SCT (for Succinct Clique Tree), and then samples k-cliques from the SCT. However, in some cases, it is impossible to materialize the full SCT; for example, the computation may be too expensive (same as

Pivoter), or the full SCT may be too large to be stored. For these cases, SCTL*-Sample sets a size lower bound k' for the cliques that are kept in SCT, and prunes all branches that only contain cliques of size smaller than k'; k' is typically set as $\omega(G) - 1$ in [12]. We remark that (1) constructing the pruned SCT could still be expensive, (2) sampling k-cliques from the pruned SCT is not a uniform sampling when $k' > k$, since certain k-cliques are always excluded, and (3) for counting k-cliques, Pivoter implicitly traverses the SCT in a depth-first manner and thus does not need to explicitly store it.

Let $C_k(G)$ be the set of all k-cliques in G and $\mathsf{cnt}_k(G) = |C_k(G)|$ be the number of k-cliques in G; we refer to a k-clique simply by its set of vertices. There are two lines of work for estimating $\mathsf{cnt}_k(G)$: Turán-Shadow [13] and DPColorPath [29]. They both conduct the computation in two stages: Stage-I constructs a sample space $\mathcal{S}_k(G)$ that is a superset of $C_k(G)$, and Stage-II samples t elements uniformly-at-random (u.a.r.) from $\mathcal{S}_k(G)$ where an element of $\mathcal{S}_k(G)$ is a vertex set of cardinality k. Let s be the number of k-cliques among the t samples. They use $\hat{\mu} = \frac{s}{t}$ as an estimation of the *k-clique density* $\mu_{\mathcal{S}_k(G)} = \frac{\mathsf{cnt}_k(G)}{|\mathcal{S}_k(G)|}$, and $\widehat{\mathsf{cnt}}_k(G) = |\mathcal{S}_k(G)| \cdot \frac{s}{t}$ as an estimation of $\mathsf{cnt}_k(G)$, the number of k-cliques in G; note that, $|\mathcal{S}_k(G)|$ is the number of sample elements in $\mathcal{S}_k(G)$ and can be efficiently obtained. Turán-Shadow and DPColorPath differ in:

- How to construct the sample space $\mathcal{S}_k(G)$.
- How to set the number t of samples.

Following the Chernoff bound [18], if $t \geq \frac{3\ln(2/\delta)}{\mu_{\mathcal{S}_k(G)} \cdot \varepsilon^2}$ for a given relative error parameter $\varepsilon \in (0,1)$ and a failing probability $\delta \in (0,1)$, then the estimation $\widehat{\mathsf{cnt}}_k(G)$ has the accuracy guarantee that $\Pr\left(\left|\widehat{\mathsf{cnt}}_k(G) - \mathsf{cnt}_k(G)\right| > \varepsilon \cdot \mathsf{cnt}_k(G)\right) \leq \delta$. However, $\mu_{\mathcal{S}_k(G)}$ itself is the quantity that we are estimating. To circumvent this, Turán-Shadow computes a lower bound μ_{lb} of $\mu_{\mathcal{S}_k(G)}$ and sets t as $\frac{3\ln(2/\delta)}{\mu_{\mathrm{lb}} \cdot \varepsilon^2}$ which then guarantees the estimation accuracy. The main idea of computing μ_{lb} is utilizing the Erdös's theorem [10], a generalization of Turán's theorem, to obtain a lower bound of the number of k-cliques in a subgraph g when its edge density is at least $1 - \frac{1}{k-1}$. Thus, Turán-Shadow iteratively refines $\mathcal{S}_k(G)$ — by partitioning a sample subspace of $\mathcal{S}_k(G)$ into multiple subspaces while retaining all k-cliques — until Erdös's theorem can be applied to all subspaces of $\mathcal{S}_k(G)$; here, each sample subspace is a subgraph of G. However, μ_{lb} is a pessimistic lower bound and $\frac{3\ln(2/\delta)}{\mu_{\mathrm{lb}} \cdot \varepsilon^2}$ could be very large. In view of this, the implemented version of Turán-Shadow in [13] simply sets t as a fixed number (e.g., 5×10^4) and thus loses the accuracy guarantee. On the other hand, DPColorPath only refines the initial $\mathcal{S}_k(G)$ once and uses a fixed number of samples (e.g., $t = 5 \times 10^7$) in Stage-II; thus, it also has no accuracy guarantee on the estimation $\widehat{\mathsf{cnt}}_k(G)$.

In this paper, we propose the SR-kCCE algorithm to estimate the number $\mathsf{cnt}_k(G)$ of k-cliques in G, which not only guarantees the estimation accuracy but also runs efficiently in practice. Our algorithm follows the general two-stage framework of Turán-Shadow and DPColorPath, but differs from them in the following ways. Firstly, we use a new strategy for determining when to stop the sampling process of Stage-II. Instead of pre-determining the number t of samples, we pre-determine the number s of required *k-clique* samples. Following the stopping rule theorem of [7], the estimation accuracy is guaranteed when $s \geq \gamma = 1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}$ where e is the Euler's number. One advantage of our sampling stopping condition is that the running time of Stage-II can be estimated. Consequently, we can balance the running time of the two stages by stopping the refinement of the sample space $\mathcal{S}_k(G)$ (i.e., Stage-I) once the elapsed time is comparable to the estimated running time of Stage-II; note that, Stage-II is expected to run faster if we spend more time on refining $\mathcal{S}_k(G)$. Specifically, Stage-II's expected running time is $\frac{\gamma}{\mu_{\mathcal{S}_k(G)} \cdot T_{\mathrm{sample}}}$ where T_{sample} is the average time of sampling an element u.a.r. from $\mathcal{S}_k(G)$ and verifying whether the sampled element forms a clique in G. We propose techniques to compute an estimation $\tilde{T}_{\mathrm{sample}}$ of T_{sample} and an auxiliary estimation $\tilde{\mu}$ of $\mu_{\mathcal{S}_k(G)}$. Note that, we still use the estimation $\hat{\mu} = \frac{s}{t}$ (rather than $\tilde{\mu}$) in the ultimate estimation $\widehat{\mathsf{cnt}}_k(G)$ of $\mathsf{cnt}_k(G)$, and consequently the accuracy of the estimations $\tilde{T}_{\mathrm{sample}}$ and $\tilde{\mu}$ only affects the running time of our algorithm but not the accuracy of the ultimate estimation $\widehat{\mathsf{cnt}}_k(G)$. We propose strategies to refine $\mathcal{S}_k(G)$ in a way aiming to increase $\mu_{\mathcal{S}_k(G)}$ such that the overall running time of our algorithm is reduced. As a by-product, our algorithm can also be used for efficiently sampling a certain number of k-cliques u.a.r. from G, and thus can be incorporated into [12] for approximate k-clique densest subgraph computation which, as we discussed earlier, is computed based on a set of sampled k-cliques. Note that, our algorithm actually samples $\gamma \approx 1.5 \times 10^7$ k-cliques for estimating $\mathsf{cnt}_k(G)$ when $\varepsilon = 0.001$.

Contributions. Our main contributions are summarized as follows.

- We propose the first k-clique count estimation algorithm that achieves the theoretical estimation accuracy and the practical efficiency at the same time.
- We abstract a general framework that captures both the existing algorithms and our algorithm. We anticipate that more algorithms could be designed based on the framework.
- We use a new sampling stopping condition to guarantee the accuracy of the estimation.
- We propose techniques to refine the sample space in a way aiming to balance the running time of the two stages.

Extensive empirical studies on large real graphs demonstrated the efficiency and effectiveness of our algorithm SR-kCCE.

Organization. The remainder of the paper is organized as follows. Preliminaries and the problem statement are given in Section 2. We abstract a general framework for k-clique count estimation and present our new algorithm SR-kCCE in Section 3. Experimental results are discussed in Section 4, and related works are reviewed in Section 5. Finally, Section 6 concludes the paper.

2 PRELIMINARIES

In this paper, we focus on a large unweighted and undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of undirected edges. We denote the number of vertices and the number of edges in G by $n = |V|$ and $m = |E|$, respectively. The set of neighbors of a vertex u in G is $N_G(u) = \{v \in V \mid (u,v) \in E\}$, and the degree of u in G is $d_G(u) = |N_G(u)|$. Given a vertex subset $S \subseteq V$, we use $E(S)$ to denote the subset of E whose both end-points are in S, i.e., $E(S) = \{(u,v) \in E \mid u \in S \wedge v \in S\}$, and use $G[S]$ to denote

the subgraph of G induced by S, i.e., $G[S] = (S, E(S))$. For ease of presentation, we simply refer to an unweighted and undirected graph as a graph, omit the subscript G from the notations when the context is clear, and abbreviate $N_{G[S]}(u)$ as $N_S(u)$. For an arbitrary graph g, we refer to its set of vertices and its set of edges by $V(g)$ and $E(g)$, respectively.

Definition 2.1 (Clique). A graph g is a clique if it has an edge between every pair of distinct vertices, or equivalently, $d_g(u) = |V(g)| - 1, \forall u \in V(g)$.

Apparently, any clique in G must be a vertex-induced subgraph of G. Thus, we simply use a vertex set to denote a clique in G. A clique C is called a k-clique if it contains exactly k vertices, i.e., $|C| = k$. We use $C_k(G)$ to denote the set of all k-cliques in G, and $\mathsf{cnt}_k(G) = |C_k(G)|$ the number of k-cliques in G.

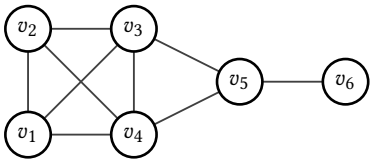


Figure 1: An example graph

Consider the graph in Figure 1, there are five 3-cliques $C_3(G) = \{\{v_1, v_2, v_3\}, \{v_1, v_2, v_4\}, \{v_1, v_3, v_4\}, \{v_2, v_3, v_4\}, \{v_3, v_4, v_5\}\}$ and one 4-clique $C_4(G) = \{\{v_1, v_2, v_3, v_4\}\}$ with $\omega(G) = 4$.

Problem Statement. Given a large graph G, an integer k, a relative error parameter $\varepsilon \in (0, 1)$ and a failing probability $\delta \in (0, 1)$, we in this paper study the problem of approximately counting the number of k-cliques in G, i.e., aim to design a randomized algorithm that returns an estimated value $\widehat{\mathsf{cnt}}_k(G)$ satisfying

$$\Pr\left(\left|\widehat{\mathsf{cnt}}_k(G) - \mathsf{cnt}_k(G)\right| > \varepsilon \cdot \mathsf{cnt}_k(G)\right) \leq \delta$$

That is, with probability at least $1 - \delta$, the relative error is at most ε.

Frequently used notations are summarized in Table 1.

Table 1: Frequently used notations

Notation	Meaning
$C_k(G)$	the set of all k-cliques in G
$\mathsf{cnt}_k(G)$	the number of k-cliques in G, i.e., $\lvert C_k(G)\rvert$
$\widehat{\mathsf{cnt}}_k(G)$	an estimation of $\mathsf{cnt}_k(G)$
$\mathcal{S}_k(G)$	a sample space of $C_k(G)$ satisfying $\mathcal{S}_k(G) \supseteq C_k(G)$
$\mathbb{S}_k(G)$	a shadow that compactly represents $\mathcal{S}_k(G)$
$\mu_{\mathcal{S}_k(G)}$	the k-clique density of $\mathcal{S}_k(G)$, i.e., $\mathsf{cnt}_k(G)/\lvert\mathcal{S}_k(G)\rvert$
$\hat{\mu}, \tilde{\mu}$	estimations of $\mu_{\mathcal{S}_k(G)}$
μ_{lb}	a lower bound of $\mu_{\mathcal{S}_k(G)}$
(R, S)	a sample subspace of $\mathbb{S}_k(G)$
$\mathcal{P}_\ell(S)$	a sample space for the set $C_\ell(S)$ of ℓ-cliques in $G[S]$
t	the total number of samples taken from $\mathcal{S}_k(G)$
s	the number of samples that are k-cliques
γ	$1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}$

2.1 Oriented Graph and Color-Path

Orienting the input undirected graph G into a directed acyclic graph (DAG) is a technique often used in clique-related problems, which not only removes duplicates but also improves the running time. Specifically, given a total ordering $\prec$ of V, an undirected edge (u, v) is transformed into a directed edge $\langle u, v\rangle$ if $u \prec v$ and into $\langle v, u\rangle$ otherwise. Denote the oriented graph by $\vec{G}$. The set of out-neighbors of u in $\vec{G}$ is $N^+_{\vec{G}}(u) = \{v \in V \mid \langle u, v\rangle \in E(\vec{G})\}$. For presentation simplicity, we omit the subscript $\vec{G}$ when the context is clear. Note that, every clique in G is also a (directed) clique in $\vec{G}$, and vice versa. Thus, the number of k-cliques in G is the same as the number of k-cliques in $\vec{G}$. For example, Figure 2 shows the oriented version of the graph in Figure 1 according to the total ordering $v_6 \prec v_5 \prec v_4 \prec v_3 \prec v_2 \prec v_1$. The 4-clique $\{v_1, v_2, v_3, v_4\}$ of G corresponds to the directed 4-clique (v_4, v_3, v_2, v_1) in $\vec{G}$.

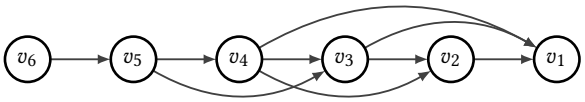


Figure 2: The oriented version $\vec{G}$ of the graph in Figure 1 based on degeneracy ordering $v_6 \prec v_5 \prec v_4 \prec v_3 \prec v_2 \prec v_1$

The most popular ordering $\prec$ used in the literature is the degeneracy ordering. A total ordering $v_1 \prec v_2 \prec \cdots \prec v_n$ is a *degeneracy ordering* if for each $1 \leq i \leq n$, v_i has the smallest degree in the subgraph $G[\{v_i, \ldots, v_n\}]$ [17]. The degeneracy ordering can be computed in $O(m)$ time by iteratively removing the smallest-degree vertex [17]. A related concept is the *degeneracy* of G, denoted $\alpha(G)$, which is the maximum out-degree in the DAG $\vec{G}$ obtained by orienting G according to the degeneracy ordering. The total ordering $v_6 \prec v_5 \prec \cdots \prec v_1$ used in the above example is a degeneracy ordering; for example, v_6 has the smallest degree in the input graph, and v_5 becomes the smallest-degree vertex after removing v_6. From Figure 2, we can see that the maximum out-degree is $|N^+(v_4)| = 3$, and thus the degeneracy is $\alpha(G) = 3$.

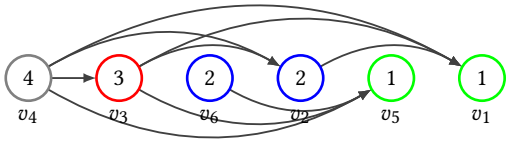


Figure 3: The oriented version $\vec{G}$ of the graph in Figure 1 based on color ordering $v_4 \prec v_3 \prec v_6 \prec v_2 \prec v_5 \prec v_1$

DPColorPath [29] proposed the concept of *color-path* to sample k-cliques. The general idea is that given an oriented graph $\vec{G}$, every k-clique in it contains a unique directed path consisting of k vertices. For simplicity, we call a directed path consisting of k vertices as a k-path. DPColorPath orients the graph based on a color ordering of the graph, for defining color-paths. In particular, a color ordering is obtained by first computing a coloring for the graph (i.e., a color number for each vertex such that there is no edge between any two vertices of the same color), and then ordering the vertices in decreasing order regarding their colors. As a result, all vertices in a k-path have different colors. For the graph in Figure 1, a coloring of the vertices is shown in Figure 3, and the coloring-based total ordering is $v_4 \prec v_3 \prec v_6 \prec v_2 \prec v_5 \prec v_1$. Consequently, the set of all 3-paths in $\vec{G}$ is $\{(v_4, v_3, v_2), (v_4, v_3, v_1), (v_4, v_3, v_5), (v_4, v_2, v_1), (v_3, v_2, v_1)\}$; note

that, in this particular example, all 3-paths are 3-cliques in G, but this may not always be the case. A dynamic programming is proposed in [29] to efficiently count the number of k-paths and uniformly sample a k-path. Specifically, given the oriented graph $\vec{G}$, it computes $H(v,i)$, the number of i-paths starting from v, for each v and each $1 \le i \le k$ via the following recurrence.

$$H(v,i) = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{if } i > 1 \text{ and } N^+(v) = \emptyset \\ \sum_{u \in N^+(v)} H(u, i-1) & \text{if } i > 1 \text{ and } N^+(v) \neq \emptyset \end{cases}$$

Filling the dynamic programming table $H(\cdot,\cdot)$ takes $O(|E(G)| \cdot k)$ time. Then, the total number of k-paths in $\vec{G}$ is $\sum_{v \in V(G)} H(v,k)$. A random k-path can be obtained by randomly sampling vertices one-by-one for the path. For example, to get the first vertex, each vertex $v \in S$ is sampled with probability $\frac{H(v,k)}{\sum_{v' \in S} H(v',k)}$ where $S = V(G)$. Let v_1 be the first vertex sampled. Then, to get the second vertex, we replace S with $N^+(v_1)$ and decrease k by 1; so on so forth.

3 OUR APPROACH

In this section, we propose a new randomized algorithm SR-kCCE for efficiently estimating the k-clique count with accuracy guarantee. Before that, we first abstract a general framework for k-clique count estimation in Section 3.1; the main purposes are three-fold.

- The framework enables us to show the similar ideas between Turán-Shadow [13] and DPColorPath [29], despite that they are proposed as entirely different algorithms.
- As our algorithm SR-kCCE also follows the framework, this enables us to concentrate on SR-kCCE's new ideas in Sections 3.2–3.4.
- We anticipate that the general framework will facilitate further development of algorithms for efficient k-clique count estimation with accuracy guarantee.

3.1 A General Framework

Our general framework for k-clique count estimation is shown in Algorithm 1, which conducts the computation in two stages. Stage-I (Lines 1–6) constructs a *sample space* $\mathcal{S}_k(G)$ that satisfies the following three conditions:

- $\mathcal{S}_k(G)$ is a superset of $C_k(G)$, i.e., $C_k(G) \subseteq \mathcal{S}_k(G)$,
- $|\mathcal{S}_k(G)|$, the number of sample elements in $\mathcal{S}_k(G)$, can be efficiently obtained, and
- Sampling an element uniformly-at-random (u.a.r.) from $\mathcal{S}_k(G)$ can be efficiently conducted.

Essentially, each sample element of $\mathcal{S}_k(G)$ is a vertex set of cardinality k (which may possibly need to satisfy some other conditions). Given the constructed sample space $\mathcal{S}_k(G)$, Stage-II (Lines 7–13) draws random samples (i.e., elements) from it and then estimates $\mathsf{cnt}_k(G)$ as $|\mathcal{S}_k(G)|$ multiplied by the proportion of samples that are k-cliques. Specifically, suppose we sample t elements from $\mathcal{S}_k(G)$, and let X_i be 1 if the i-th sampled element forms a clique in G and $X_i = 0$ otherwise. Then, $\mathsf{cnt}_k(G)$ is estimated as

$$\widehat{\mathsf{cnt}}_k(G) = |\mathcal{S}_k(G)| \cdot \frac{1}{t} \sum_{i=1}^{t} X_i \tag{1}$$

Algorithm 1: kCliqueCountEstimation$(G, k, \varepsilon, \delta)$

Input: A graph G, an integer k, and accuracy parameters ε and δ
Output: An estimation $\widehat{\mathsf{cnt}}_k(G)$ of $\mathsf{cnt}_k(G)$

/* Stage-I: construct a sample space $\mathcal{S}_k(G)$, represented by a compact structure $\mathbb{S}$ */
1 $\mathbb{S} \leftarrow \{(\emptyset, V)\}$;
2 **while** *the construction stopping condition is not satisfied* **do**
3 Choose a sample subspace (R,S) and remove it from $\mathbb{S}$;
  /* Refine the subspace (R,S) by partitioning it */
4 **for each** $v \in S$ **do**
5 Add $(R \cup v, N_S(v))$ to $\mathbb{S}$;
6 $S \leftarrow S \setminus v$;
/* Stage-II: sample $\mathcal{S}_k(G)$ to estimate $\mathsf{cnt}_k(G)$ */
7 $|\mathcal{S}_k(G)| \leftarrow \sum_{(R,S) \in \mathbb{S}} |\mathcal{P}_{k-|R|}(S)|$;
8 $s \leftarrow 0$; $t \leftarrow 0$;
9 **while** *the sampling stopping condition is not satisfied* **do**
10 Sample a subspace (R,S) from $\mathbb{S}$ with probability $\frac{|\mathcal{P}_{k-|R|}(S)|}{|\mathcal{S}_k(G)|}$;
11 Sample an element P u.a.r. from $\mathcal{P}_{k-|R|}(S)$;
12 **if** *P forms a clique in G* **then** $s \leftarrow s+1$;
13 $t \leftarrow t+1$;
14 **return** $\widehat{\mathsf{cnt}}_k(G) \leftarrow |\mathcal{S}_k(G)| \cdot \frac{s}{t}$;

The general idea is defining the *k-clique density* of a sample space $\mathcal{S}_k(G)$ as

$$\mu_{\mathcal{S}_k(G)} = \frac{\mathsf{cnt}_k(G)}{|\mathcal{S}_k(G)|} \tag{2}$$

which is between 0 and 1. Note that, $\mathsf{cnt}_k(G)$ is fixed, and thus the k-clique density only depends on $\mathcal{S}_k(G)$; that is, different sample spaces will result in different $\mu_{\mathcal{S}_k(G)}$ values. Then, $\mathsf{cnt}_k(G) = |\mathcal{S}_k(G)| \cdot \mu_{\mathcal{S}_k(G)}$, and the problem becomes estimating $\mu_{\mathcal{S}_k(G)}$ since we assumed that $|\mathcal{S}_k(G)|$ can be efficiently obtained. Algorithm 1 estimates $\mu_{\mathcal{S}_k(G)}$ by the empirical mean $\hat{\mu}$ of t samples taken based on $\mu_{\mathcal{S}_k(G)}$, i.e.,

$$\hat{\mu} = \frac{1}{t} \sum_{i=1}^{t} X_i \tag{3}$$

and thus $\widehat{\mathsf{cnt}}_k(G) = |\mathcal{S}_k(G)| \cdot \hat{\mu}$ is an unbiased estimation of $\mathsf{cnt}_k(G)$.

3.1.1 Shadow. The most naive sample space $\mathcal{S}_k(G)$ could be the set of all *k-subsets* (i.e., subsets of cardinality k) of V. However, the size of this naive sample space is enormous (i.e., $\binom{n}{k}$) which would require a lot of samples to successfully sample a k-clique from it. Thus, $\mathcal{S}_k(G)$ needs to be shrunk by removing non-cliques from it. At the same time, we also need a data structure to compactly represent $\mathcal{S}_k(G)$ by noting that explicitly representing every element of $\mathcal{S}_k(G)$ would take at least $|C_k(G)|$ space which could be too large to be stored. To achieve this, we borrow the term *shadow* from [13] to compactly represent $\mathcal{S}_k(G)$, and give a precise definition for it.

Definition 3.1 (Shadow). Given a graph G and an integer k, a shadow $\mathbb{S}_k(G)$ is a set of vertex pairs (R,S), representing sample subspaces, such that

- $R \subseteq V$ and R forms a clique in G, where R could be $\emptyset$;
- $S \subseteq V \setminus R$ and every vertex of S is fully adjacent to R in G;
- For every k-clique C in G, there is a unique subspace $(R,S) \in \mathbb{S}_k(G)$ such that $R \subseteq C$ and $C \setminus R \subseteq S$.

We remark that, for estimating $\mathsf{cnt}_k(G)$, it is sufficient to store $|R|$, instead of R, in $\mathbb{S}_k(G)$. But if the purpose is to sample and report k-cliques from G, then it is necessary to explicitly store R. Following from the definition of shadow, it holds that

$$\mathsf{cnt}_k(G) = \sum_{(R,S)\in\mathbb{S}_k(G)} \mathsf{cnt}_{k-|R|}(S) \tag{4}$$

That is, the k-cliques of G are uniquely distributed into the subspaces $(R,S) \in \mathbb{S}_k(G)$. Given a vertex subset $S \subseteq V$ and an integer ℓ, let $\mathcal{P}_\ell(S)$ be the ***elementary sample structure*** which is a superset of $C_\ell(S)$, the set of all ℓ-cliques of $G[S]$; for example, $\mathcal{P}_\ell(S)$ could be $\{P \subseteq S \mid |P| = \ell\}$. Given a sample subspace $(R,S) \in \mathbb{S}_k(G)$, let $\mathcal{S}_k(R,S)$ be the Cartesian product of $\{R\}$ and $\mathcal{P}_{k-|R|}(S)$, i.e.,

$$\mathcal{S}_k(R,S) = \{R\} \times \mathcal{P}_{k-|R|}(S) = \{R \cup P \mid P \in \mathcal{P}_{k-|R|}(S)\} \tag{5}$$

Then, the sample subspace (R,S) together with $\mathcal{P}_{k-|R|}(S)$ form a compact representation of $\mathcal{S}_k(R,S)$, and a shadow $\mathbb{S}_k(G)$ is a compact representation of the sample space $\mathcal{S}_k(G)$ where

$$\mathcal{S}_k(G) = \bigcup_{(R,S)\in\mathbb{S}_k(G)} \mathcal{S}_k(R,S) = \bigcup_{(R,S)\in\mathbb{S}_k(G)} \{R\} \times \mathcal{P}_{k-|R|}(S) \tag{6}$$

and

$$C_k(G) \subseteq \mathcal{S}_k(G) \text{ if and only if } C_{k-|R|}(S) \subseteq \mathcal{P}_{k-|R|}(S)$$

A random element of $\mathcal{S}_k(G)$ can be sampled from $\mathbb{S}_k(G)$ in two steps. Firstly, a random subspace (R,S) is sampled from $\mathbb{S}_k(G)$ with probability proportional to $|\mathcal{P}_{k-|R|}(S)| = |\mathcal{S}_k(R,S)|$ (Line 10). Then, an element P is sampled u.a.r from $\mathcal{P}_{k-|R|}(S)$ (Line 11). That is, the element sampled for $\mathcal{S}_k(G)$ is $R \cup P$.

In Algorithm 1, the shadow $\mathbb{S}$ is initialized as $\{(\emptyset, V)\}$, and it is iteratively refined (by Lines 3–6) to shrink the corresponding sample space $\mathcal{S}_k(G)$. For example, Figure 5 illustrates the shadow refinement process for the graph in Figure 4 and $k = 4$. Each node in the tree represents a sample subspace (R,S), where nodes with $|R| + |S| < k$ are pruned from the tree. When refining a subspace (R_i, S_i), it is replaced by all its children in the tree. That is, the shadow consists of all leaf nodes in the tree. When walking from the root to a leaf node in the tree, R grows while S shrinks.

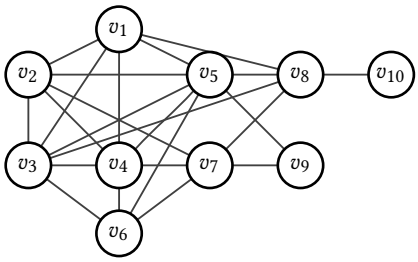


Figure 4: Another example graph

Lemma 3.2. *Lines 1–6 of Algorithm 1 correctly construct a valid shadow according to Definition 3.1.*

Proof. We prove the lemma by induction. Firstly, the initial $\mathbb{S}_0 = \{(\emptyset, V)\}$ trivially satisfies all conditions of Definition 3.1. Now, suppose that $\mathbb{S}$ satisfies all conditions of Definition 3.1, and let $\mathbb{S}'$ be the result of one refinement (i.e., running Lines 3–6 of Algorithm 1 once). We prove that $\mathbb{S}'$ also satisfies all conditions of Definition 3.1.

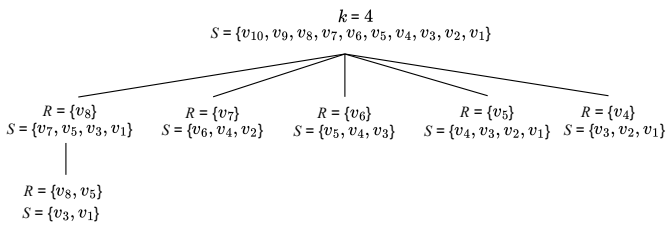


Figure 5: Shadow refinement process for the graph in Figure 4

Let (R,S) be the sample subspace selected at Line 3 to refine $\mathbb{S}$, and $\{(R_i, S_i)\}_{i=1}^{l}$ be the subspaces added at Line 5. It is easy to see that (R_i, S_i) satisfies the first two conditions of Definition 3.1 for each $1 \le i \le l$. To prove the last condition, let's consider an arbitray k-clique C such that $R \subseteq C \subseteq R \cup S$, and let v_j be the first vertex of $S \cap C$ according to the order they are processed at Line 4. Then, the subspace $(R_j, S_j) = (R \cup v_j, N_S(v_j))$ must satisfy $R_j \subseteq C \subseteq R_j \cup S_j$. Moreover, we can see that for each $i' < j$, $R_{i'} \not\subseteq C$ according to the definition of v_j, and for each $i'' > j$, $C \not\subseteq R_{i''} \cup S_{i''}$ since $v_j \notin R_{i''} \cup S_{i''}$. Hence, the lemma holds. □

We remark from the above proof that Lines 1–6 of Algorithm 1 correctly construct a valid shadow no matter what is the stopping condition at Line 2 and which subspace (R,S) is selected at Line 3.

3.1.2 Instantiations of the Framework. Both of the existing algorithms, Turán-Shadow [13] and DPColorPath [29], can be considered as instantiations of the framework in Algorithm 1. For Turán-Shadow [13], the elementary sample structure $\mathcal{P}_\ell(S)$ (used in Equation (6)) is defined as $\mathcal{P}_\ell(S) = \{P \subseteq S \mid |P| = \ell\}$. The construction stopping condition at Line 2 is: $\forall (R,S) \in \mathbb{S}_k(G)$, the *edge density* of $G[S]$ defined as $|E(S)|/\binom{|S|}{2}$ is higher than $1 - \frac{1}{k-|R|-1}$. If the stopping condition is not satisfied, then a sample subspace $(R,S) \in \mathbb{S}$ with $|E(S)|/\binom{|S|}{2} \le 1 - \frac{1}{k-|R|-1}$ is chosen at Line 3 to be refined. The sampling stopping condition at Line 9 is $t \ge \frac{3\ln(2/\delta)}{\mu_{\mathrm{lb}} \cdot \varepsilon^2}$, where μ_{lb} is a lower bound of the k-clique density $\mu_{\mathcal{S}_k(G)}$ and

$$\mu_{\mathrm{lb}} = \min_{(R,S)\in\mathbb{S}} \frac{(k-|R|)!}{(k-|R|-1)^{k-|R|-2} \cdot |S|^2} \tag{7}$$

However, the lower bound μ_{lb} computed by Equation (7) could be very small in practice as shown in Figure 9 in our experiments, and thus an extremely large and impractical number t of samples would be needed to achieve the accuracy guarantee. In view of this, the implemented version of Turán-Shadow in [13] sets t as a fixed value 5×10^4, and thus loses the accuracy guarantee on the estimated value $\widehat{\mathsf{cnt}}_k(G)$.

For DPColorPath [29], the elementary sample structure $\mathcal{P}_\ell(S)$ is defined as the set of ℓ-paths in $G[S]$ as introduced in Section 2.1. The construction stopping condition (Line 2 of Algorithm 1) is that for every $(R,S) \in \mathbb{S}_k(G)$, $|R| \ge 1$. Equivalently, DPColorPath only refines $\mathbb{S}$ (i.e., runs Lines 3–6) once. That is, the shadow constructed by DPColorPath for the graph in Figure 4 is the top part of Figure 5, i.e., by removing the node with $R = \{v_8, v_5\}$. The sampling stopping condition (Line 9) is that the number t of samples is above a fixed threshold (e.g., 5×10^7). Consequently, DPColorPath does not have any accuracy guarantee on the estimated value $\widehat{\mathsf{cnt}}_k(G)$.

Our new algorithm SR-kCCE is also an instantiation of the framework. SR-kCCE uses the same definition of $\mathcal{P}_\ell(S)$ as DPColorPath, but differs from the existing algorithms in the following aspects.

- We use a new sampling stopping condition to guarantee the accuracy for the estimated value $\widehat{\mathsf{cnt}}_k(G)$. (Section 3.2)
- We refine the shadow $\mathbb{S}$ in such a way to balance the running time of the two Stages, by proposing a new construction stopping condition and a new strategy of choosing which sample subspace (R, S) in $\mathbb{S}$ to refine. (Section 3.3)
- We adopt the alias method to speed up the sampling process, and propose techniques to achieve efficiently sampling without storing all the alias structures. (Section 3.4)

3.2 A New Sampling Stopping Condition

In this subsection, we focus on the accuracy of our estimation, which is determined by Stage-II of our algorithm.

Given a sample space $\mathcal{S}_k(G)$ (in its compact form $\mathbb{S}_k(G)$), the existing algorithms Turán-Shadow [13] and DPColorPath [29] first determine the number t of samples and then draw t samples u.a.r. from $\mathcal{S}_k(G)$. One strategy of ensuring the estimation accuracy, as proposed in Turán-Shadow, is to compute a lower bound μ_{lb} of $\mu_{\mathcal{S}_k(G)} = \frac{\mathsf{cnt}_k(G)}{|\mathcal{S}_k(G)|}$ and then set t as $\frac{3\ln(2/\delta)}{\mu_{\text{lb}} \cdot \varepsilon^2}$. However, the computed μ_{lb} is a pessimistic lower bound, and as a result, the obtained t is often too large to be practically used. Consequently, Turán-Shadow uses a fixed value (e.g., 5×10^4) for t in their implementation and thus loses the accuracy guarantee.

Algorithm 2: SR-Estimator($\mathbb{S}_k(G), k, \varepsilon, \delta$)

Input: A shadow $\mathbb{S}_k(G)$, an integer k, accuracy parameters ε and δ
Output: An estimation $\widehat{\mathsf{cnt}}_k(G)$ of $\mathsf{cnt}_k(G)$

1 $s \leftarrow 0$; $t \leftarrow 0$;
2 **while** $s < 1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}$ **do**
3 Sample an element A u.a.r. from $\mathcal{S}_k(G)$;
4 **if** *A forms a clique in G* **then** $s \leftarrow s + 1$;
5 $t \leftarrow t + 1$;
6 **return** $\widehat{\mathsf{cnt}}_k(G) \leftarrow |\mathcal{S}_k(G)| \cdot \frac{s}{t}$;

In this paper, we use a different strategy for determining when to stop the sampling process. Specifically, we first determine the number s of required successful samples (i.e., samples that are k-cliques), and then keep taking samples from $\mathcal{S}_k(G)$ until s successful samples have been taken. We still use $\frac{s}{t}$ to estimate $\mu_{\mathcal{S}_k(G)}$. The pseudocode is shown in Algorithm 2, where the details of how to efficiently sample an element from $\mathcal{S}_k(G)$ at Line 3 will be discussed in Section 3.4. Based on the *stopping rule theorem* of [7], the estimation accuracy is guaranteed when $s \geq 1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}$ where e is the Euler's number. Thus, we call our estimator as SR-Estimator where SR stands for stopping rule. We refer to the quantity $1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}$ by γ in the remainder of the paper.

Theorem 3.3. *The $\widehat{\mathsf{cnt}}_k(G)$ outputted by Algorithm 2 satisfies $\Pr\big(|\widehat{\mathsf{cnt}}_k(G) - \mathsf{cnt}_k(G)| > \varepsilon \cdot \mathsf{cnt}_k(G)\big) \leq \delta$. When Algorithm 2 stops, the expected value of t satisfies $\frac{\gamma \cdot |\mathcal{S}_k(G)|}{\mathsf{cnt}_k(G)|} \leq \mathrm{E}[t] < \frac{(\gamma+1) \cdot |\mathcal{S}_k(G)|}{\mathsf{cnt}_k(G)}$.*

Proof. Note that $\frac{s}{t}$ is an estimation of $\mu_{\mathcal{S}_k(G)} = \frac{\mathsf{cnt}_k(G)}{|\mathcal{S}_k(G)|}$. Thus, the theorem follows from [7]. □

We remark that (1) the $\widehat{\mathsf{cnt}}_k(G)$ outputted by Algorithm 2 is an unbiased estimation of the number $\mathsf{cnt}_k(G)$ of k-cliques in G, and (2) the accuracy guarantee of Algorithm 2 (i.e., $\Pr\big(|\widehat{\mathsf{cnt}}_k(G) - \mathsf{cnt}_k(G)| > \varepsilon \cdot \mathsf{cnt}_k(G)\big) \leq \delta$) holds for any shadow $\mathbb{S}_k(G)$ that satisfies Definition 3.1. But different shadows will result in different running time. We will propose techniques in the next subsection to construct the shadow in such a way to improve the overall performance, i.e., the total running time of Stage-I and Stage-II.

3.3 Balance the Running Time of the Two Stages

In this subsection, we focus on the efficiency aspect of our algorithm. Our high-level idea is to balance the running time of the two stages. Note that, the existing algorithms conduct the two stages independently, and either stage could take significantly more time than the other stage; this impedes their overall performance.

Construction Stopping Condition. Following from Theorem 3.3, the expected number t of samples taken when Algorithm 2 terminates is approximately $\frac{\gamma}{\mu_{\mathcal{S}_k(G)}}$ where $\mu_{\mathcal{S}_k(G)} = \frac{\mathsf{cnt}_k(G)}{|\mathcal{S}_k(G)|}$. Let T_{sample} be the average time of running Lines 3–4 of Algorithm 2, i.e., sampling an element from $\mathcal{S}_k(G)$ and verifying whether it forms a clique in G. Then, the running time (note that, not the time complexity) of Algorithm 2 (i.e., Stage-II of SR-kCCE) is

$$\text{Running time of Algorithm 2} \approx \frac{\gamma}{\mu_{\mathcal{S}_k(G)}} \cdot T_{\text{sample}} \tag{8}$$

We prove in the lemma below that after each refinement of (a sample subspace of) the shadow $\mathbb{S}_k(G)$, the corresponding $\mathcal{S}_k(G)$ shrinks and consequently $\mu_{\mathcal{S}_k(G)}$ increases.

Lemma 3.4. *Assume that the same vertex ordering is used in defining $\mathcal{P}_{k-|R|}(S)$ for different subspaces of $\mathbb{S}_k(G)$ and that vertices are processed (at Line 4 of Algorithm 1) in the same order as in obtaining $\mathcal{P}_{k-|R|}(S)$. Then, each refinement of (a sample subspace of) the shadow $\mathbb{S}_k(G)$ makes the corresponding $\mathcal{S}_k(G)$ smaller.*

Proof. Suppose $(R, S) \in \mathcal{S}_k(G)$ is refined into $\{(R_i, S_i)\}_{i=1}^{l}$ by running Lines 3–6 of Algorithm 1 once. We prove in the following that $\bigcup_{i=1}^{l} \mathcal{S}_k(R_i, S_i) \subseteq \mathcal{S}_k(R, S)$; see Equation (5) for the definition of $\mathcal{S}_k(R, S)$. Firstly, by the same argument as in the proof of Lemma 3.2, we have that $\mathcal{S}_k(R_{i'}, S_{i'}) \cap \mathcal{S}_k(R_{i''}, S_{i''}) = \emptyset$ for any $1 \leq i' \neq i'' \leq l$. Secondly, let $\ell = k - |R|$ and $v_i = R_i \setminus R$, and let's consider any $(\ell-1)$-path P in the oriented version of $G[S_i]$. Then, $v_i \cup P$ must be an ℓ-path in the oriented version of $G[S]$, since $P \subseteq S_i = N_S^+(v_i)$. Hence, the lemma holds. □

If we stop Stage-I immediately after initializing $\mathbb{S}_k(G)$ as $\{(\emptyset, V)\}$, then Stage-I will be very efficient since the while loop (Lines 2–6 of Algorithm 2) is not executed; however, Stage-II (i.e., Algorithm 2) will take a very long time as the k-clique density is expected to be very low due to the large size of $\mathcal{P}_k(V)$. When we spend more time on refining $\mathbb{S}_k(G)$, the running time of Stage-I increases, but the running time of Stage-II (i.e., Equation (8)) will decrease since $\mu_{\mathcal{S}_k(G)}$ increases as discussed above; Figure 6 illustrates this phenomenon. Thus, intuitively *a sweet spot would be that the two stages take similar running time*; it can be shown that the overall time

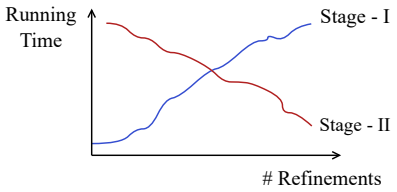


Figure 6: Running time of the two stages w.r.t. the number of refinement operations

would be at most twice the minimum running time if the optimal number of refinements is conducted.

When determining whether we should stop Stage-I and go to Stage-II, it is straightforward to get the elapsed time of Stage-I. However, it is non-trivial to get a good estimation on the running time of Stage-II without actually executing it. Considering Equation (8), γ is a fixed quantity as ε and δ are given in the input, and T_{sample} can be easily estimated by taking a few samples from $\mathcal{S}_k(G)$ and measuring the running time. What remains is estimating the k-clique density $\mu_{\mathcal{S}_k(G)}$, which however is the quantity that we are trying to estimate; that is, we fall into a loop. To resolve this issue, we propose to compute an auxiliary estimation for $\mu_{\mathcal{S}_k(G)}$, and denote it by $\tilde{\mu}$, to distinguish it from our ultimate estimation $\hat{\mu} = \frac{s}{t}$. Note that, the estimation $\tilde{\mu}$ does not need to have any theoretical accuracy guarantee, and the accuracy of the estimation $\tilde{\mu}$ only affects the running time of our algorithm but not the accuracy of the ultimate estimation $\hat{\mu}$. Thus, our construction stopping condition is

$$\text{Elapsed time} \;\geq\; \frac{\gamma}{\tilde{\mu}} \cdot \tilde{T}_{\text{sample}}$$

Compute an Auxiliary Estimation $\tilde{\mu}$. The estimation $\tilde{\mu}$ is used for determining whether we should switch from Stage-I to Stage-II. That is, we need to continuously compute the estimation $\tilde{\mu}$ for $\mu_{\mathcal{S}_k(G)} = \frac{\text{cnt}_k(G)}{|\mathcal{S}_k(G)|}$ when $\mathcal{S}_k(G)$ is dynamically changing. This is in contrast to Algorithm 2 where $\mathcal{S}_k(G)$ is assumed to be fixed. As a result, Algorithm 2 as well as the existing algorithms cannot be used to compute the auxiliary estimation $\tilde{\mu}$.

In view of this, we propose a different method to compute the auxiliary estimation $\tilde{\mu}$. As $|\mathcal{S}_k(G)|$ can be incrementally obtained when $\mathbb{S}_k(G)$ changes, we only need to estimate $\text{cnt}_k(G)$. Recall that $\text{cnt}_k(G) = \sum_{(R,S)\in\mathbb{S}_k(G)} \text{cnt}_{k-|R|}(S)$. Thus, we propose to compute an estimation $\ddot{\mu}$ of the $(k-|R|)$-clique density of $G[S]$, i.e., $\frac{\text{cnt}_{k-|R|}(S)}{|\mathcal{P}_{k-|R|}(S)|}$, for every $(R,S) \in \mathbb{S}$, and store $\ddot{\mu}$ together with (R,S). That is, we revise each sample subspace of $\mathbb{S}_k(G)$ to $(R,S,\ddot{\mu})$, where $\ddot{\mu}$ is a (rough) estimation of $\frac{\text{cnt}_{k-|R|}(S)}{|\mathcal{P}_{k-|R|}(S)|}$. Consequently, we estimate $\text{cnt}_{k-|R|}(S)$ as $|\mathcal{P}_{k-|R|}(S)| \cdot \ddot{\mu}$, and compute an auxiliary estimation of $\text{cnt}_k(G)$ as

$$\widetilde{\text{cnt}}_k(G) = \sum_{(R,S,\ddot{\mu})\in\mathbb{S}} |\mathcal{P}_{k-|R|}(S)| \cdot \ddot{\mu}$$

and an auxiliary estimation of $\mu_{\mathcal{S}_k(G)}$ as

$$\tilde{\mu} = \frac{\widetilde{\text{cnt}}_k(G)}{\sum_{(R,S,\ddot{\mu})\in\mathbb{S}} |\mathcal{P}_{k-|R|}(S)|}$$

We remark that, $\widetilde{\text{cnt}}_k(G)$ is a biased estimation of $\text{cnt}_k(G)$ and $\tilde{\mu}$ is a biased estimation of $\mu_{\mathcal{S}_k(G)}$. Nevertheless, this dose not affect the accuracy of our final estimation $\hat{\mu}$ and $\widetilde{\text{cnt}}_k(G)$, as discussed above.

Algorithm 3: ShadowConstruction$(G, k, \varepsilon, \delta)$

Input: A graph G, an integer k, and accuracy parameters ε and δ
Output: A shadow $\mathbb{S}$

1 $\widetilde{\text{cnt}}_k(G) \leftarrow 1$; $|\mathcal{S}_k(G)| \leftarrow |\mathcal{P}_k(V)|$; $\tilde{T}_{\text{sample}} \leftarrow \infty$;
2 $\mathbb{S} \leftarrow \{(\emptyset, V, \widetilde{\text{cnt}}_k(G)/|\mathcal{S}_k(G)|)\}$;
3 **while** ElapsedTime() $< (1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}) \frac{|\mathcal{S}_k(G)|}{\widetilde{\text{cnt}}_k(G)} \tilde{T}_{\text{sample}}$ **do**
4 $(R, S, \ddot{\mu}) \leftarrow \arg\min_{(R',S',\ddot{\mu}')\in\mathbb{S}} \ddot{\mu}'$;
5 Remove $(R, S, \ddot{\mu})$ from $\mathbb{S}$;
6 $\widetilde{\text{cnt}}_k(G) \leftarrow \widetilde{\text{cnt}}_k(G) - |\mathcal{P}_{k-|R|}(S)| \cdot \ddot{\mu}$;
7 $|\mathcal{S}_k(G)| \leftarrow |\mathcal{S}_k(G)| - |\mathcal{P}_{k-|R|}(S)|$;
8 **if** $R = \emptyset$ **then** $n_{\text{sample}} \leftarrow 0$; $T_{\text{total}} \leftarrow 0$;
9 **for each** $v \in S$ **do**
10 $(R', S') \leftarrow (R \cup v, N_S(v))$;
11 Compute $|\mathcal{P}_{k-|R'|}(S')|$, and other auxiliary information that are needed for sampling elements from $\mathcal{P}_{k-|R'|}(S')$;
12 $\ddot{\mu}' \leftarrow$ the proportion of k-cliques among $\frac{|S'|^2}{k-|R'|}$ elements sampled u.a.r. from $\mathcal{P}_{k-|R'|}(S')$;
13 Add $(R', S', \ddot{\mu}')$ to $\mathbb{S}$;
14 $\widetilde{\text{cnt}}_k(G) \leftarrow \widetilde{\text{cnt}}_k(G) + |\mathcal{P}_{k-|R'|}(S')| \cdot \ddot{\mu}'$;
15 $|\mathcal{S}_k(G)| \leftarrow |\mathcal{S}_k(G)| + |\mathcal{P}_{k-|R'|}(S')|$;
16 $S \leftarrow S \setminus v$;
17 **if** $R = \emptyset$ **then**
18 $n_{\text{sample}} \leftarrow n_{\text{sample}} + \frac{|S'|^2}{k-|R'|}$;
19 $T_{\text{total}} \leftarrow T_{\text{total}}$+ the running time of Line 12;
20 **if** $R = \emptyset$ **then** $\tilde{T}_{\text{sample}} \leftarrow \frac{T_{\text{total}}}{n_{\text{sample}}}$;
21 **return** $\mathbb{S}$;

The Pseudocode of Shadow Construction. Based on the above discussions, the pseudocode of our shadow construction algorithm is shown in Algorithm 3. We first initialize $\widetilde{\text{cnt}}_k(G)$ as 1, $|\mathcal{S}_k(G)|$ as $|\mathcal{P}_k(V)|$, $\tilde{T}_{\text{sample}}$ as ∞, and $\mathbb{S}$ as $\{(\emptyset, V, \widetilde{\text{cnt}}_k(G)/|\mathcal{S}_k(G)|)\}$ (Lines 1–2). We estimate the running time of invoking Algorithm 2 with the shadow $\mathbb{S}$ as $\gamma \cdot \frac{|\mathcal{S}_k(G)|}{\widetilde{\text{cnt}}_k(G)} \cdot \tilde{T}_{\text{sample}}$ (Line 3). As long as the current elapsed time is less than the estimated running time of Algorithm 2 with the current $\mathbb{S}$, we choose a subspace from $\mathbb{S}$ to refine (Lines 4–20). Specifically, we choose the subspace $(R, S, \ddot{\mu})$ that has the smallest $\ddot{\mu}$ to refine (Line 4), in the hope of increasing the k-clique density $\mu_{\mathcal{S}_k(G)}$. To refine $(R, S, \ddot{\mu})$, we first update $\widetilde{\text{cnt}}_k(G)$ and $|\mathcal{S}_k(G)|$ as we are removing $(R, S, \ddot{\mu})$ from $\mathbb{S}$ (Lines 5–7), and then for each $v \in S$ (Line 9), we add $(R \cup v, N_S(v))$ into $\mathbb{S}$ (Line 13) and remove v from S (Line 16). Let (R', S') be $(R \cup v, N_S(v))$ (Line 10). We compute $|\mathcal{P}_{k-|R'|}(S')|$ by conducting a dynamic programming (Line 11) as discussed in Section 2.1. At the same time, we also compute and store some auxiliary information that will be used to speedup the sampling process of Line 12; details will be given in Section 3.4. After that, we sample $\frac{|S'|^2}{k-|R'|}$ elements u.a.r. from $\mathcal{P}_{k-|R'|}(S')$, and compute $\ddot{\mu}'$ as the proportion of samples that are cliques in G (Line 12). When adding $(R', S', \ddot{\mu}')$ to $\mathbb{S}$, we also update $\widetilde{\text{cnt}}_k(G)$ and $|\mathcal{S}_k(G)|$ (Lines 14–15). For the first refinement (i.e., when $R = \emptyset$), we record the total number n_{sample} of samples drawn and the total time T_{total} of sampling (Lines 17–19), and then compute $\tilde{T}_{\text{sample}}$ as $\frac{T_{\text{total}}}{n_{\text{sample}}}$ (Line 20).

3.4 Efficient Sampling

In this subsection, we discuss how to efficiently sample elements u.a.r. from $\mathcal{S}_k(G)$, which is needed at Line 3 of Algorithm 2 and Line 12 of Algorithm 3. Abstractly, we need to efficiently sample values from a discrete probability distribution. For example, given a sample subspace $(R,S) \in \mathbb{S}_k(G)$, as discussed in Section 2.1 the first vertex of an $(k-|R|)$-path in $\mathcal{P}_{k-|R|}(S)$ is sampled from S with each vertex $v \in S$ being sampled with probability $\frac{H(v,k-|R|)}{\sum_{v' \in S} H(v',k-|R|)}$. A naive method would take $O(|S|)$ time to sample a vertex from S with non-equal probability; this is slow. We adopt the alias method [27] to speed up the sampling operation. Specifically, given a discrete probability distribution of r values, after constructing the alias structure in $O(r)$ time and space, a random value can be sampled from it in $O(1)$ time [27].

However, it is non-trivial to incorporate the alias method into our algorithm. For example, if the alias structures are not pre-computed and stored, then constructing the alias structures online would dominate the sampling time and thus the alias method will not be helpful. On the other hand, if we store the alias structures for all sample subspaces of $\mathbb{S}_k(G)$, it would take an excessive space and thus is infeasible. Specifically, the total size of the alias structures for one subspace (R,S) is $|S|^2 \cdot (k-|R|)$, since (1) it has $|S| \cdot (k-|R|)$ distributions (one for each $v \in S$ and $1 \le i \le k-|R|$), and (2) each distribution is of size $|S|$. Consequently, the total alias structure size for all subspaces of $\mathbb{S}_k(G)$ would be $\sum_{(R,S) \in \mathbb{S}_k(G)} |S|^2 \cdot (k-|R|)$, which is much larger than the shadow size.

In view of the above issues, we propose to only store the alias structures in the memory for one subspace at a time. That is, we group together all tasks that are sampling from the same subspace, such that we can construct the alias structure online, conduct multiple samplings using the same alias structure, and then discard the constructed alias structure. It is easy to see that the sampling tasks at Line 12 of Algorithm 3 are naturally grouped based on the sample subspaces. Then, based on the alias structures constructed at Line 11, sampling $\frac{|S'|^2}{k-|R'|}$ elements u.a.r. from $\mathcal{P}_{k-|R'|}(S')$ takes $\frac{|S'|^2}{k-|R'|} \cdot (k-|R'|)$ time in total. Verifying whether they form cliques in G takes $\frac{|S'|^2}{k-|R'|} \cdot (k-|R'|)^2 = |S'|^2 \cdot (k-|R'|)$ time in total, the same running time as Line 11; this is how we determined the number of samples at Line 12. To implement this idea for Line 3 of Algorithm 2, we propose to sample elements from $\mathcal{S}_k(G)$ in a batch manner; the details are shown in Algorithm 4.

Recall from Lines 10–11 of Algorithm 1 that a random element of $\mathcal{S}_k(G)$ is sampled in two steps: firstly, a random subspace (R,S) is sampled from $\mathbb{S}_k(G)$ with probability $\frac{|\mathcal{P}_{k-|R|}(S)|}{|\mathcal{S}_k(G)|}$, and then a random $(k-|R|)$-path is sampled u.a.r. from $\mathcal{P}_{k-|R|}(S)$. Thus, to sample t_b elements u.a.r. from $\mathcal{S}_k(G)$, we first determine the number of samples that will be taken from each subspace by sampling the subspaces (Lines 5–8), and then sample the corresponding number of elements u.a.r. from each subspace (Lines 9–14). Note that, samples taken in this way are still u.a.r. from $\mathcal{S}_k(G)$. As the expected number of samples required to get γ k-cliques (to satisfy the stopping condition at Line 4) is $\frac{\gamma}{\mu_{\mathcal{S}_k(G)}}$, we set the number of samples to be taken in each batch as $t_b = \frac{\gamma}{\tilde{\mu}}$. We observe in our experiments that

Algorithm 4: SR-Estimator2($\mathbb{S}_k(G), k, \varepsilon, \delta, \tilde{\mu}$)

Input: A shadow $\mathbb{S}_k(G)$, an integer k, accuracy parameters ε and δ, and a rough estimation $\tilde{\mu}$ of the k-clique density $\mu_{\mathcal{S}_k(G)}$

Output: An estimation $\widehat{\mathsf{cnt}}_k(G)$ of $\mathsf{cnt}_k(G)$

1 $s \leftarrow 0$; $t \leftarrow 0$;
2 $\gamma \leftarrow 1 + \frac{4(1+\varepsilon)(e-2)\ln(2/\delta)}{\varepsilon^2}$; $t_b \leftarrow \frac{\gamma}{\tilde{\mu}}$;
3 Construct the alias structure for sample subspaces from $\mathbb{S}_k(G)$;
4 **while** $s < \gamma$ **do**
5 **for each** *subspace* $(R,S) \in \mathbb{S}_k(G)$ **do** Initialize $c_{R,S} \leftarrow 0$;
6 **for** $i \leftarrow 1$ ***to*** t_b **do**
7 Sample a subspace $(R,S) \in \mathbb{S}_k(G)$ with prob. $\frac{|\mathcal{P}_{k-|R|}(S)|}{|\mathcal{S}_k(G)|}$;
8 $c_{R,S} \leftarrow c_{R,S} + 1$;
9 **for each** *subspace* $(R,S) \in \mathbb{S}_k(G)$ *with* $c_{R,S} > 0$ **do**
10 Construct the alias structure for sampling $\mathcal{P}_{k-|R|}(S)$;
11 **for each** $j \leftarrow 1$ ***to*** $c_{R,S}$ **do**
12 Sample an element P u.a.r. from $\mathcal{P}_{k-|R|}(S)$;
13 **if** *P forms a clique in G* **then** $s \leftarrow s + 1$;
14 $t \leftarrow t + 1$;
 /* Discard the constructed alias structure */
15 **return** $\widehat{\mathsf{cnt}}_k(G) \leftarrow |\mathcal{S}_k(G)| \cdot \frac{s}{t}$;

Algorithm 4 typically terminates after one or two iterations of the while loop at Line 4.

3.5 Implementation Details

In this subsection, we give some implementation details. Firstly, given a subspace (R,S) retrieved from $\mathbb{S}$ at Line 4 of Algorithm 3, we compute the exact ℓ-clique count in $G[S]$ by invoking Pivoter if $|S|$ is sufficiently small compared to ℓ, and refine it according to Lines 9–16 otherwise. Specifically, we compute the exact ℓ-clique count in $G[S]$ if $|S| \le \ell + 10$ and $|S| \le 2 \cdot \ell$. The main motivation is that when $|S|$ is relatively small compared to ℓ, then Pivoter can compute the exact ℓ-clique count efficiently. We remark that DPColorPath [29] uses a similar strategy, but it computes the exact count when the average degree of $G[S]$ is at most $k-1$; recall that, DPColorPath only refines the sample space $\mathbb{S}$ once. Consequently, the k-clique count $\mathsf{cnt}_k(G)$ is divided into two parts: the partial k-clique count obtained by Pivoter, denoted pcnt, and the number of k-cliques in $\mathcal{S}_k(G)$, denoted ecnt. That is, $\mathsf{cnt}_k(G) = \mathsf{pcnt} + \mathsf{ecnt}$, and our task is to compute an estimation $\widehat{\mathsf{ecnt}}$ of ecnt. Note that

$$\begin{aligned}
&\Pr\Big(\big|\mathsf{pcnt} + \widehat{\mathsf{ecnt}} - \mathsf{cnt}_k(G)\big| \ge \varepsilon \cdot \mathsf{cnt}_k(G)\Big) \\
=&\Pr\Big(\big|\widehat{\mathsf{ecnt}} - \mathsf{ecnt}\big| \ge \varepsilon \cdot \big(\tfrac{\mathsf{pcnt}}{\mathsf{ecnt}} \cdot \mathsf{ecnt} + \mathsf{ecnt}\big)\Big) \\
\le&\Pr\Big(\big|\widehat{\mathsf{ecnt}} - \mathsf{ecnt}\big| \ge \varepsilon \cdot \big(\tfrac{\mathsf{pcnt}}{|\mathcal{S}_k(G)|} + 1\big) \cdot \mathsf{ecnt}\Big)
\end{aligned}$$

Thus, we use $\varepsilon' = \varepsilon \cdot \big(\frac{\mathsf{pcnt}}{|\mathcal{S}_k(G)|} + 1\big)$ when sampling from $\mathcal{S}_k(G)$ to estimate ecnt. Moreover, when $\varepsilon' \ge 1$ which is equivalent to $\frac{\mathsf{pcnt}}{\mathsf{pcnt}+|\mathcal{S}_k(G)|} \ge 1-\varepsilon$, we directly return pcnt as the estimation of $\mathsf{cnt}_k(G)$ which has a relative error at most ε (i.e., bypass Stage-II).

Secondly, we process vertices at Line 9 of Algorithm 3 according to the degeneracy ordering in $G[S]$. Consequently, $N_S(v)$ obtained at Line 10 is the set of v's out-neighbors in the graph oriented according to the degeneracy ordering, and thus each sample subspace

Table 2: Statistics of the graphs and k-clique counts ($\alpha(G)$ is degeneracy; $\omega(G)$ is maximum clique size; * indicates that Pivoter does not finish within 5 hours and the k-clique count $\mathsf{cnt}_k(G)$ is estimated by SR-kCCE with $\varepsilon = 0.001$ and thus is inexact)

Graph	n	m	$\alpha(G)$	$\omega(G)$	$k=6$	$k=9$	$k=12$	$k=15$	$k=20$
gowalla	196591	950327	51	29	28928240	86198580	160683420	201454150	24649947
Stanford	281903	1992636	71	61	4859571082	1.2027E+12	9.6277E+13	2.9464E+15	1.5271E+17
youtube	1134890	2987624	51	17	8443803	3721987	219257	1068	0
as-skitter	1696415	11095298	111	67	9759000981	2.7817E+12	2.6823E+14	1.1122E+16	1.2836E+18
com-lj	3997962	34681189	360	327	1.0991E+13	5.8783E+17*	1.5869E+22*	2.2706E+26*	5.4862E+32*
soc-lj	4846609	42851237	372	321	2.0703E+13	1.1787E+18*	3.4728E+22*	5.2356E+26*	1.3111E+33*
com-orkut	3072441	117185083	253	51	7.5249E+10	7.2481E+12	4.1504E+14	1.0468E+16	3.3041E+17
uk-2002	18459128	261556721	943	944	5.3175E+14	2.5176E+20*	5.8005E+25*	6.7597E+30*	5.5535E+38*
it-2004	41290577	1027474895	3224	3222	4.7890E+18*	4.0417E+26*	1.2625E+34*	1.8311E+41*	4.2466E+52*
friendster	65608366	1806067135	304	129	5.9927E+10	4.0033E+13	5.4619E+16	3.3864E+19	3.5762E+23

$(R, S) \in \mathbb{S}$ satisfies $|S| \leq \alpha(G)$. As a result, the alias structures constructed at Line 11 of Algorithm 3 and Line 10 of Algorithm 4 take $O(\alpha(G)^2 \cdot k)$ space. Thirdly, to compute $|\mathcal{P}_\ell(S)|$ and to sample from $\mathcal{P}_\ell(S)$, we compute a coloring for the vertices of S based on the subgraph $G[S]$; note that, DPColorPath only computes the graph coloring once for the entire graph G, and then reuses the coloring for all subgraphs of G.

4 EXPERIMENTS

In this section, we evaluate the estimation accuracy and efficiency of our proposed algorithm SR-kCCE for k-clique count estimation, by comparing it with the following competitors.

- Pivoter: the state-of-the-art exact algorithm proposed in [14] for counting k-cliques.
- Turán-Shadow [1]: the existing k-clique count estimation algorithm proposed in [13].
- DPColorPath: the most-recent k-clique count estimation algorithm proposed in [29].

We implemented our own version of Pivoter, because our algorithm invokes Pivoter to get the exact k-clique count for sample subspaces that are small as discussed in Section 3.5. We also implemented our own version of DPColorPath, as it is straightforward to implement based on our code. We compared our results with those in [29], and found that our implementations are generally faster than the ones used in [29]. Also note that, we modified DPColorPath and Turán-Shadow to incorporate our sampling stopping condition discussed in Section 3.2 to ensure the accuracy guarantee.

All algorithms are implemented in C++ and ran in a single-thread mode. We conduct the experiments on a machine with an Intel Core i7-8700 CPU and 64GB main memory and running Ubuntu 18.04. We set a time limit of 5 hours for each testing.

Datasets. We evaluate the algorithms on 10 large real graphs that are downloaded from http://networkrepository.com/networks.php. Statistics as well as k-clique counts of the graphs are shown in Table 2. The number $\mathsf{cnt}_k(G)$ of k-cliques is computed by Pivoter; if Pivoter does not finish within the time limit, we use the estimation outputted by our algorithm SR-kCCE with the relative error parameter $\varepsilon = 0.001$ as $\mathsf{cnt}_k(G)$. The fourth column shows the degeneracy $\alpha(G)$ while the fifth column shows the maximum clique size $\omega(G)$.

[1] https://bitbucket.org/sjain12/cliquecounting

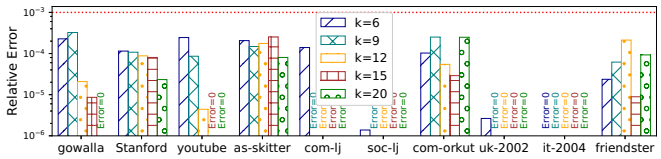
Parameters. For the testings, we choose the clique size k from $\{6, 9, 12, 15, 20\}$ and the relative error parameter ε from $\{0.001, 0.005, 0.01, 0.05\}$. We fix the failing probability $\delta = 0.01$ since it does not affect the running time much; recall that, the time of the sampling stage only depends logarithmically on $\frac{1}{\delta}$.

Metrics. For each testing, we record both the processing time and the memory usage. The recorded processing time excludes only the time of loading a graph from disk to memory. The recorded memory usage is the Maximum Resident Set Size reported by /usr/bin/time. In addition, we also calculate the actual relative error of an algorithm's estimation, which is $\frac{|\widehat{\mathsf{cnt}}_k(G) - \mathsf{cnt}_k(G)|}{\mathsf{cnt}_k(G)}$.

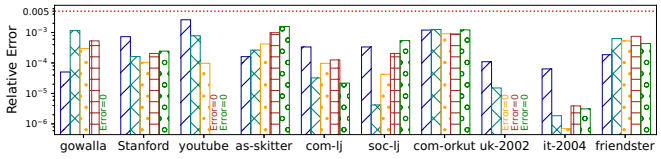
4.1 Actual Accuracy of the Algorithms

In this testing, we first evaluate the practical accuracy of the k-clique count estimation computed by our algorithm SR-kCCE. Specifically, we report the relative error $\frac{|\widehat{\mathsf{cnt}}_k(G) - \mathsf{cnt}_k(G)|}{\mathsf{cnt}_k(G)}$ in Figure 7, where $\widehat{\mathsf{cnt}}_k(G)$ is the estimation computed by SR-kCCE and $\mathsf{cnt}_k(G)$ is obtained from Table 2. The omitted bars have relative error 0; note that for some of the cases (e.g., gowalla and youtube) this is because SR-kCCE computes the exact k-clique count with $\mathsf{pcnt} = \mathsf{cnt}_k(G)$ and $\mathcal{S}_k(G) = \emptyset$ (see Section 3.5 for details of pcnt), while for other cases (e.g., com-lj, soc-lj, uk-2002 and it-2004 in Figure 7(a)) this is because the relative error is computed based on the estimation of SR-kCCE with $\varepsilon = 0.001$. We can see from Figure 7 that our algorithm always produces an accurate estimation with the actual relative error being much lower than the given error parameter ε. As a result, we can use the estimation $\widehat{\mathsf{cnt}}_k(G)$ computed by our algorithm SR-kCCE with $\varepsilon = 0.001$ to replace $\mathsf{cnt}_k(G)$ in the cases that computing the exact $\mathsf{cnt}_k(G)$ is time-consuming.

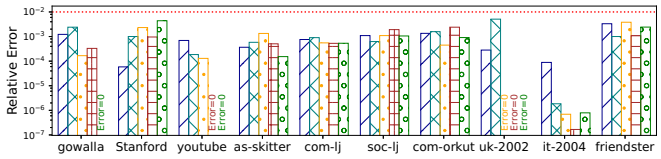
We then compare the practical accuracy of our algorithm SR-kCCE against the existing algorithms DPColorPath and Turán-Shadow. In addition, we also implemented a version of DPColorPath that takes a fixed 5×10^7 number of samples, denoted DPColorPath5e7. The results for $k \in \{6, 12, 20\}$ and $\varepsilon = 0.001$ are shown in Figure 8. We can see that our algorithm achieves the lowest error in most of the cases. Recall that, DPColorPath and Turán-Shadow in principle should also guarantee the relative error to be at most ε, as we modified their sampling stopping condition to be the one described in Section 3.2. However, Turán-Shadow has a much higher relative error than ε on uk-2002 for $k = 6$; we suspect that there is a bug in their implementation of Turán-Shadow. On the other



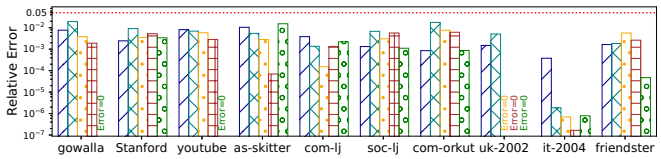
(a) $\varepsilon = 0.001$



(b) $\varepsilon = 0.005$

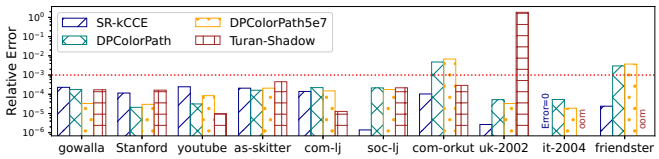


(c) $\varepsilon = 0.01$

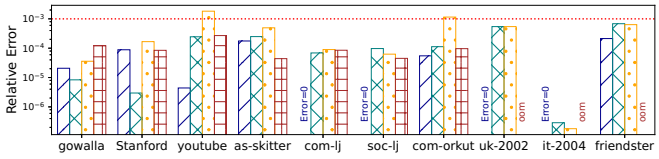


(d) $\varepsilon = 0.05$

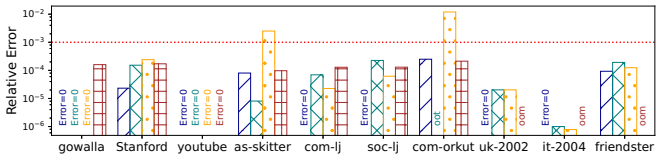
Figure 7: Actual relative error of our algorithm



(a) $k = 6$ and $\varepsilon = 0.001$



(b) $k = 12$ and $\varepsilon = 0.001$



(c) $k = 20$ and $\varepsilon = 0.001$

Figure 8: Actual relative error of different algorithms (oom: out-of-memory, oot: out-of-time)

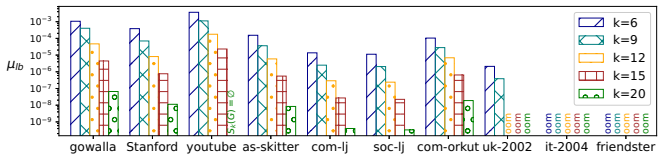


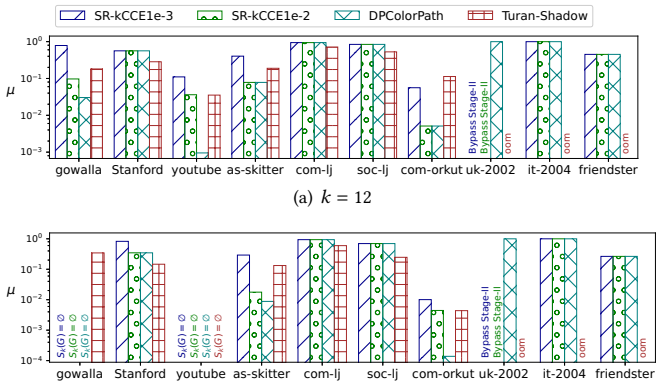
Figure 9: μ_{lb} of Turán-Shadow

hand, DPColorPath has relative error $> \varepsilon$ on com-orkut and friendster for $k = 6$ because its sampling is not uniform; specifically, to sample a batch of t_b elements from $\mathcal{S}_k(G)$, DPColorPath sets the number of elements to be sampled from a subspace $(R, S) \in \mathcal{S}_k(G)$ as $\lfloor t_b \cdot \frac{|\mathcal{P}_{k-|R|}(S)|}{|\mathcal{S}_k(G)|} \rfloor$. DPColorPath5e7 further has relative error $> \varepsilon$ on youtube (for $k = 12$), as-skitter (for $k = 20$), and com-orkut (for $k \geq 12$); this is because taking a fixed 5×10^7 number of samples in these cases is insufficient to achieve the accuracy guarantee, due to a low k-clique density $\mu_{\mathcal{S}_k(G)}$.

4.2 k-Clique Density of the Sampling Spaces

In this testing, we first evaluate the k-clique density lower bound μ_{lb} computed by Turán-Shadow as defined in Equation (7). The results are shown in Figure 9. Note that, (1) for $k = 20$ on youtube, Turán-Shadow prunes the entire graph as the maximum clique size is 17, and (2) Turán-Shadow runs out-of-memory for all other cases where the corresponding bars are missing from Figure 9. We can see that the lower bound μ_{lb} could be very small, e.g., 10^{-9} for $k = 20$ on com-lj and soc-lj. Thus, it is impractical to set the number of samples in the sampling stage to be $t = \frac{3\ln(2/\delta)}{\mu_{\text{lb}} \cdot \varepsilon^2}$ for achieving the accuracy guarantee, which would mean $t \approx 1.5 \times 10^{16}$ samples for $\varepsilon = 0.001$. This demonstrates that the theoretical version of Turán-Shadow that guarantees the estimation accuracy is impractical.

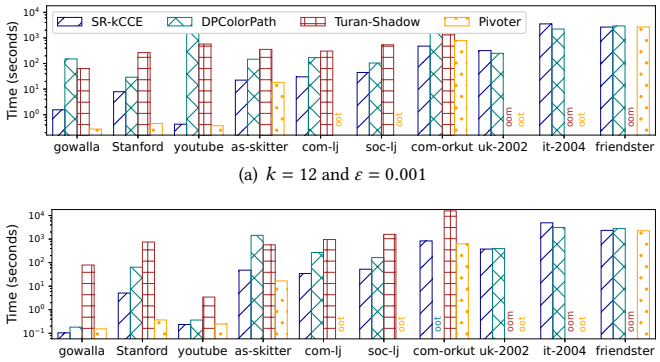
We then evaluate the k-clique density $\mu_{\mathcal{S}_k(G)}$ for the sample spaces that are constructed by DPColorPath, Turán-Shadow, and our algorithm SR-kCCE for different k values. Note that, (1) the higher the value of $\mu_{\mathcal{S}_k(G)}$, the better the performance of Stage-II, and (2) the partial count pcnt is not considered in $\mathcal{S}_k(G)$ (see Section 3.5). The sample spaces constructed by DPColorPath and Turán-Shadow are independent of the error parameter ε, but our algorithm SR-kCCE computes different sample spaces for different ε values. Thus, we report the results of SR-kCCE for $\varepsilon = 10^{-2}$ and $\varepsilon = 10^{-3}$, denoted as SR-kCCE1e-2 and SR-kCCE1e-3, respectively. The experimental results of the algorithms for $k = 12$ and 20 are illustrated in Figure 10. Firstly, we observe that $\mu_{\mathcal{S}_k(G)}$ of DPColorPath can be low (i.e., close to 10^{-3}) in some cases, e.g., $k = 12$ on youtube, and $k = 20$ on com-orkut. For these cases, DPColorPath runs slow when we want to achieve a high accuracy (e.g., $\varepsilon = 0.001$), see Figure 11 which will be discussed shortly. Secondly, we observe that $\mu_{\mathcal{S}_k(G)}$ of our algorithm SR-kCCE with $\varepsilon = 10^{-2}$ (i.e., SR-kCCE1e-2) is generally higher than that of DPColorPath, and SR-kCCE1e-3 further increases the k-clique density. This is because the expected number of samples needed in the sampling stage is inversely related to ε^2 and $\mu_{\mathcal{S}_k(G)}$ (see Theorem 3.3); when ε decreases, the number of required samples increases, and thus SR-kCCE refines the sample space more to increase $\mu_{\mathcal{S}_k(G)}$ and counterbalance this effect. We remark that for $k \in \{12, 20\}$ on uk-2002, our algorithm bypasses Stage-II since pcnt is close to pcnt + $|\mathcal{S}_k(G)|$ (see Section 3.5). Lastly, $\mu_{\mathcal{S}_k(G)}$ of Turán-Shadow



(a) $k = 12$

(b) $k = 20$

Figure 10: $\mu_{\mathcal{S}_k(G)}$ of the sample spaces constructed by the algorithms (SR-kCCE1e-3 and SR-kCCE1e-2 denote SR-kCCE with $\varepsilon = 10^{-3}$ and 10^{-2}, respectively)



(a) $k = 12$ and $\varepsilon = 0.001$

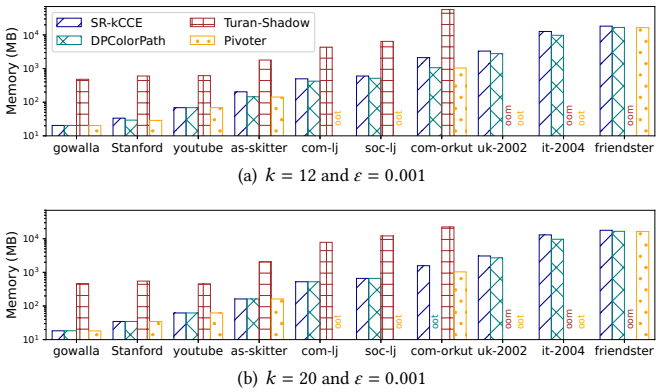
(b) $k = 20$ and $\varepsilon = 0.001$

Figure 11: Running time of the algorithms ($\varepsilon = 0.001$)

is generally high, at the expense of a significant time in the sample space refinement. Note that, as the elementary sample structure (i.e., $\mathcal{P}_\ell(S)$) used in Turán-Shadow is different from and inferior to that of DPColorPath (see Section 3.1.2), $\mu_{\mathcal{S}_k(G)}$ of Turán-Shadow could be lower than that of DPColorPath in some cases.

4.3 Time and Memory Usage of the Algorithms

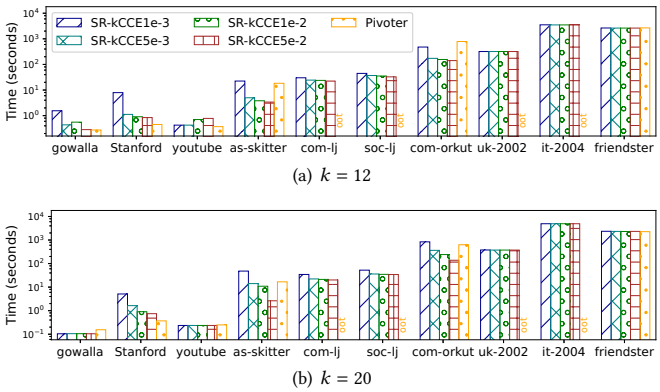
The running time of the algorithms for $k \in \{12, 20\}$ and $\varepsilon = 0.001$ are shown in Figure 11. Our algorithm SR-kCCE consistently outperforms Turán-Shadow due to the latter's significant time in refining the sample space. Our algorithm generally runs faster than DPColorPath, but they perform similarly on uk-2002 and it-2004. This is because the k-clique density $\mu_{\mathcal{S}_k(G)}$ of DPColorPath on these two graphs is already extremely high (i.e., > 0.96) as shown in Figure 10, while our algorithm takes more time in Stage-I. Note that, although DPColorPath also has a high $\mu_{\mathcal{S}_k(G)}$ on com-lj and soc-lj, our algorithm runs faster because of our more efficient sampling technique as discussed in Section 3.4. Lastly, the exact algorithm Pivoter performs quite well on some of the graphs, by running comparably or even occasionally faster than our algorithm. However,



(a) $k = 12$ and $\varepsilon = 0.001$

(b) $k = 20$ and $\varepsilon = 0.001$

Figure 12: Memory usage of the algorithms ($\varepsilon = 0.001$)



(a) $k = 12$

(b) $k = 20$

Figure 13: Running time of our algorithm for different ε

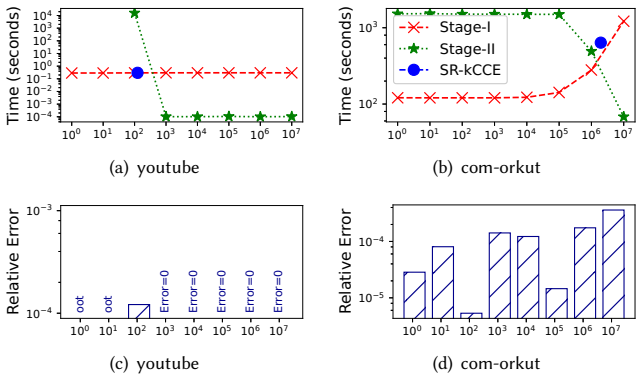
Pivoter's performance is unstable and unpredictable, e.g., it does not finish within 5 hours on four of the graphs.

The memory usage of the algorithms are shown in Figure 12. Our algorithm does not consume much more memory than Pivoter or DPColorPath which are memory efficient. On the other hand, Turán-Shadow runs out-of-memory on the three largest graphs.

The running time of our algorithm for different ε values are shown in Figure 13. We can see that our algorithm generally runs faster when we tolerate a higher relative error ε. But the running time may remain the same when varying ε on some of the graphs (e.g., the last three); this is because in these cases, Stage-II's running time which strongly depends on ε is negligible while Stage-I's running time does not change too much. We can also see that by setting $\varepsilon = 0.005$, our algorithm could outperform Pivoter in some of the cases that Pivoter can finish within the time limit.

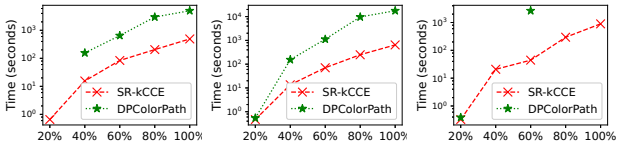
4.4 Ablation and Scalability Studies

The results of evaluating the performance of our algorithm by manually setting the number of refinements (#refinements) in Stage-I are illustrated in Figure 14, where #refinements varies from 1 to 10^7, $k = 15$ and $\varepsilon = 0.001$. Figures 14(a) and 14(b) show the running time (of Stage-I and Stage-II) on youtube and com-orkut, respectively. We can see that this conforms with our theoretical analysis in Section 3.3, i.e., when #refinements increases, the running time



(a) youtube (b) com-orkut (c) youtube (d) com-orkut

Figure 14: Vary number of refinements ($k = 15$ and $\varepsilon = 0.001$)



(a) $k = 12$ (b) $k = 15$ (c) $k = 20$

Figure 15: Scalability testing on subgraphs of com-orkut (vary percentage of vertices, $\varepsilon = 0.001$)

of Stage-I increases while that of Stage-II decreases. We remark that on youtube, the algorithm does not finish within 5 hours when #refinements ≤ 10, and $\mathcal{S}_k(G) = \emptyset$ when #refinements $\geq 10^3$. We also observe that the optimal #refinements (i.e., for achieving the fastest total running time) differ across the graphs. In Figures 14(a) and 14(b), we further report the total running time of our algorithm SR-kCCE where #refinements is automatically determined by Algorithm 3. We can see that although the automatically determined #refinements sometimes is not optimal, it is close to the optimal number. The relative errors achieved by the algorithm when varying #refinements are shown in Figures 14(c) and 14(d). We can see that they are all below the given ε; this is because our Stage-II algorithm guarantees the accuracy given any sample space $\mathcal{S}_k(G)$.

To test the scalability of our algorithm and DPColorPath, we run them on subgraphs that are randomly extracted from com-orkut. Specifically, we randomly sample x% vertices of com-orkut, and then take the subgraph induced by these vertices; x is chosen from $\{20, 40, 60, 80, 100\}$. The results for $k \in \{12, 15, 20\}$ are illustrated in Figure 15, where $\varepsilon = 0.001$. We can see that the running time of our algorithm SR-kCCE increases smoothly when the number of vertices increases. However, the performance of DPColorPath fluctuates a lot; it does not finish within 5 hours for $k = 12$ with $x = 20$, and for $k = 20$ with $x \in \{40, 80, 100\}$. This is because DPColorPath only refines the sample space once, and has no control on the k-clique density $\mu_{\mathcal{S}_k(G)}$ which affects Stage-II's running time; for example, $\mu_{\mathcal{S}_k(G)} \approx 7 \times 10^{-5}$ for $k = 12$ and $x = 20$. This demonstrates that our algorithm scales better than DPColorPath.

5 FURTHER RELATED WORK

Besides the related works discussed in the Introduction, we briefly summarize other related works in this section.

k-Clique Enumeration and Counting. Enumerating all k-cliques in a graph G has also been studied in the literature. The state-of-the-art algorithms run in $O\left(k \cdot m \cdot \left(\frac{\alpha(G)}{2}\right)^{k-2}\right)$ time [5, 8], and $O\left(m \cdot \alpha(G) + k \cdot m \cdot \left(\frac{\tau(G)}{2}\right)^{k-2}\right)$ time [28], where $\tau(G)$ is a parameter similar to the degeneracy $\alpha(G)$ but is defined based on edges and is obtained by iteratively removing the *edge* that participates in the minimum number of *triangles*; note that $\tau(G) \leq \alpha(G)$. These k-clique enumeration algorithms can be used for counting k-cliques. However, they are extremely slow due to explicitly enumerating all k-cliques. As demonstrated in [14], even the version of [8] that is optimized for counting k-cliques performs worse than Pivoter. Thus, we do not include them in our experiments.

Parallel algorithms and GPU algorithms have also been designed and implemented for exactly counting k-cliques [1, 11, 24]. They are designed based on either the k-clique enumeration algorithms of [5, 8] or Pivoter [14]. These parallel and GPU algorithms are shown to improve efficiency over single-core algorithms, due to exploiting the parallelism of multiple CPU cores or massive GPU cores. The state of the art is GPU-Pivoter proposed in [1]. However, GPU-Pivoter is still inefficient for moderate k values, due to the intrinsic complexity of exactly counting k-cliques. For example, the largest k tested in [1] is 11, and GPU-Pivoter still does not finish within 5 hours on com-lj for $k \geq 9$. In this paper, we focus on single-core algorithms, while our algorithm can be made faster by using parallelism which will be investigated in our future study.

Motif/Subgraph Counting. k-clique counting is a special case of motif/subgraph counting that counts the number of occurrences of a given query subgraph in a large graph. Algorithms have been proposed for exact, approximate and parallel subgraph counting, see [22] for a recent survey. Although these algorithms can be used for counting k-cliques, they will not be efficient since they do not utilize the special property of clique. Example techniques used in approximate subgraph counting include enumerate-generalize [21], path sampling [16], color coding [2] and etc. In particular, path sampling is similar to Algorithm 1 but without graph coloring, graph orientation and sample space refinement; this would perform even worse than DPColorPath that refines the sample space once. Furthermore, it is shown in [13] that color coding [2] and Graft [21] are outperformed by Turán-Shadow. Thus, we do not include these subgraph counting algorithms in our experiments.

6 CONCLUSION

In this paper, we proposed the first algorithm that achieves the theoretical estimation accuracy and the practical efficiency at the same time. We abstracted a general framework that captures both the existing algorithms and our algorithms. We anticipate that more (and possibly faster) algorithms could be designed based on our framework. As a by-product, our algorithm is also able to efficiently sample a certain number of k-cliques u.a.r. from a graph.

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council Fundings of DP220103731, and the Research Grants Council of Hong Kong, China under No. 14205520.

REFERENCES

[1] Mohammad Almasri, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-Mei Hwu. 2022. Parallel K-clique counting on GPUs. In *Proc. of ICS'22*. ACM, 21:1–21:14.

[2] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Trans. Knowl. Discovery Data* 12, 4 (April 2018), 1–25.

[3] Coenraad Bron and Joep Kerbosch. 1973. Finding All Cliques of an Undirected Graph (Algorithm 457). *Commun. ACM* 16, 9 (1973), 575–576.

[4] Lijun Chang and Lu Qin. 2018. *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer Series in the Data Sciences.

[5] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and Subgraph Listing Algorithms. *SIAM J. Comput.* 14, 1 (1985), 210–223.

[6] Seshadhri Comandur and Srikanta Tirthapura. 2019. Scalable Subgraph Counting: The Methods Behind The Madness. In *Proc. of WWW'19 (Tutorial)*. 1317–1318.

[7] 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.

[8] Maximilien Danisch, Oana Denisa Balalau, and Mauro Sozio. 2018. Listing k-cliques in Sparse Real-World Graphs. In *Proc. of WWW'18*. 589–598.

[9] David Eppstein, Maarten Löffler, and Darren Strash. 2013. Listing All Maximal Cliques in Large Sparse Real-World Graphs. *ACM Journal of Experimental Algorithmics* 18 (2013).

[10] Pál Erdös. 1969. On the number of complete subgraphs and circuits contained in graphs. *Časopis pro pěstování matematiky* 094, 3 (1969), 290–296.

[11] Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. 2015. Clique Counting in MapReduce: Algorithms and Experiments. *ACM J. Exp. Algorithmics* 20 (Oct. 2015), 1–20. https://doi.org/10.1145/2794080

[12] Yizhang He, Kai Wang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2023. Scaling Up k-Clique Densest Subgraph Detection. *Proc. ACM Manag. Data* 1, 1 (2023), 69:1–69:26.

[13] Shweta Jain and C Seshadhri. 2017. A fast and provable method for estimating clique counts using turan's theorem. In *Proc. of WWW'17*. 441–449.

[14] Shweta Jain and C. Seshadhri. 2020. The Power of Pivoting for Exact Clique Counting. In *Proc. WSDM'20*. ACM, 268–276.

[15] Shweta Jain and Hanghang Tong. 2022. YACC: A Framework Generalizing TuránShadow for Counting Large Cliques. In *Proc. of SDM'22*. 684–692.

[16] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *Proc. of WWW'15*. 495–505.

[17] David W. Matula and Leland L. Beck. 1983. Smallest-Last Ordering and clustering and Graph Coloring Algorithms. *J. ACM* 30, 3 (1983), 417–427.

[18] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[19] Mark Ortmann and Ulrik Brandes. 2014. Triangle Listing Algorithms: Back from the Diversion. In *Proc. of ALENEX'14*. 1–8.

[20] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature* 435, 7043 (June 2005), 814–818.

[21] Mahmudur Rahman, Mansurul Alam Bhuiyan, and Mohammad Al Hasan. 2014. Graft: An Efficient Graphlet Counting Method for Large Graph Analysis. *IEEE Trans. Knowl. Data Eng.* 26, 10 (Jan. 2014), 2466–2478.

[22] Pedro Ribeiro, Pedro Paredes, Miguel E. P. Silva, David Aparício, and Fernando M. A. Silva. 2022. A Survey on Subgraph Counting: Concepts, Algorithms, and Applications to Network Motifs and Graphlets. *ACM Comput. Surv.* 54, 2 (2022), 28:1–28:36.

[23] Ahmet Erdem Sariyüce, C. Seshadhri, Ali Pinar, and Ümit V. Çatalyürek. 2015. Finding the Hierarchy of Dense Subgraphs using Nucleus Decompositions. In *Proc. of WWW'15*. 927–937.

[24] Jessica Shi, Laxman Dhulipala, and Julian Shun. 2021. Parallel Clique Counting and Peeling Algorithms. In *Proc. of ACDA'21*. 135–146.

[25] Bintao Sun, Maximilien Danisch, T.-H. Hubert Chan, and Mauro Sozio. 2020. KClist++: A Simple Algorithm for Finding k-Clique Densest Subgraphs in Large Graphs. *Proc. VLDB Endow.* 13, 10 (2020), 1628–1640.

[26] Charalampos E. Tsourakakis. 2015. The K-clique Densest Subgraph Problem. In *Proc. of WWW'15*. 1122–1132.

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

[28] Kaixin Wang, Kaiqiang Yu, and Cheng Long. 2023. Efficient k-Clique Listing: An Edge-Oriented Branching Strategy. *CoRR* (2023).

[29] Xiaowei Ye, Rong-Hua Li, Qiangqiang Dai, Hongzhi Chen, and Guoren Wang. 2022. Lightning Fast and Space Efficient k-clique Counting. In *Proc. of WWW'22*. 1191–1202.