Ordering Heuristics for k-clique Listing

Rong-Hua Li[†*], Sen Gao[†], Lu Qin[‡], Guoren Wang[†], Weihua Yang[§], Jeffrey Xu Yu[#]

[†]*Beijing Institute of Technology, Beijing, China;* [*]*National Engineering Laboratory for Big Data System Computing Technology*
[§]*Taiyuan University of Technology, China;* [‡]*University of Technology, Sydney, Australia*
[#]*The Chinese University of Hong Kong, Hong Kong, China*
lironghuabit@126.com; Gawssin@gmail.com; Lu.Qin@uts.edu.au;
wanggrbit@126.com; yangweihua@tyut.edu.cn; yu@se.cuhk.edu.hk

ABSTRACT

Listing all k-cliques in a graph is a fundamental graph mining problem that finds many important applications in community detection and social network analysis. Unfortunately, the problem of k-clique listing is often deemed infeasible for a large k, as the number of k-cliques in a graph is exponential in the size k. The state-of-the-art solutions for the problem are based on the ordering heuristics on nodes which can efficiently list all k-cliques in large real-world graphs for a small k (e.g., $k \leq 10$). Even though a variety of heuristic algorithms have been proposed, there still lacks a thorough comparison to cover all the state-of-the-art algorithms and evaluate their performance using diverse real-world graphs. This makes it difficult for a practitioner to select which algorithm should be used for a specific application. Furthermore, existing ordering based algorithms are far from optimal which might explore unpromising search paths in the k-clique listing procedure. To address these issues, we present a comprehensive comparison of all the state-of-the-art k-clique listing and counting algorithms. We also propose a new color ordering heuristics based on greedy graph coloring techniques which is able to significantly prune the unpromising search paths. We compare the performance of 14 various algorithms using 17 large real-world graphs with up to 3 million nodes and 100 million edges. The experimental results reveal the characteristics of different algorithms, based on which we provide useful guidance for selecting appropriate techniques for different applications.

PVLDB Reference Format:
Rong-Hua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, Jeffrey Xu Yu. Ordering Heuristics for k-clique Listing. *PVLDB*, 13(11): 2536-2548, 2020.
DOI: https://doi.org/10.14778/3407790.3407843

1. INTRODUCTION

Real-world graphs, such as social networks, biological networks, and communication networks often consist of cohesive subgraph structures. Mining cohesive subgraphs from a graph is a fundamental problem in network analysis which has attracted much attention in the database and data mining communities [2, 26, 37, 36, 25, 24]. Perhaps the most elementary cohesive substructure in a graph is the k-clique structure which has been widely used in a variety of network analysis applications [34, 17, 30, 6, 20, 11].

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407843

Given a graph G, a k-clique is a subgraph with k nodes such that each pair of nodes is connected with an edge. Listing all k-cliques in a graph is a fundamental graph mining operator which finds important applications in community detection and social network analysis. In particular, Palla et al. [34] proposed a k-clique percolation approach to detect overlapping communities in a network, in which a k-clique listing algorithm is used for computing all k-cliques. Mitzenmacher et al. [30] presented an algorithm to find large near cliques which also requires to list all k-cliques. Sariyüce et al. [37] developed a nucleus decomposition method to reveal the hierarchy of dense subgraphs, in which listing all k-cliques is an important building block. Tsourakakis [46] investigated a k-clique densest subgraph problem which also makes use of k-cliques as building blocks. In addition, algorithms for k-clique listing have also been used for story identification in social media [2] and detect the latent higher-order organization in real-world networks [6].

Motivated by the above applications, several practical algorithms have been developed for listing/counting all k-cliques in large real-world graphs [9, 29, 32, 20, 11]. Chiba and Nishizeki [9] developed the first practical algorithm (referred to as the Chiba-Nishizeki algorithm) to solve such a problem, which can handle many large real-world graphs. An appealing feature of the Chiba-Nishizeki algorithm is that its running time relies mainly on the arboricity [31] of the graph, which is typically very small for real-world graphs [27, 13]. To improve the Chiba-Nishizeki algorithm, Ortmann and Brandes [32] proposed a general ordering-based framework to list triangles in a graph which can also be applied to list k-cliques. Compared to the Chiba-Nishizeki algorithm, the striking feature of the ordering-based framework is that it can be easily parallelized. Danisch et al. [11] also developed a similar ordering-based framework to list all k-cliques, with a particular focus on a degeneracy-ordering based algorithm. They showed that the degeneracy-ordering based algorithm is significantly faster than the Chiba-Nishizeki algorithm. However, they did not compare their algorithm with the other ordering-based algorithms, such as the algorithm based on the degree ordering [32]. Another practical algorithm is MACE [29]. Although MACE was initially proposed to enumerate all maximal cliques, it can also be adapted to list k-cliques [29]. Except for exact k-clique counting, there exist two notable algorithms to approximate the number of k-cliques. Jain and Seshadhri [20] developed an elegant randomized algorithm called Turán-Shadow based on the classic Turán theorem [48]. The Turán-Shadow algorithm is able to quickly estimate the number of k-cliques, but it cannot output all k-cliques. ERS is also a randomized algorithm for estimating k-clique counts which was developed in the theory community [12]. ERS can achieve a sublinear time complexity in theory, but its practical performance is unknown.

Although the significance of the k-clique listing problem and the many efforts devoted to investigating it, a comprehensive experimental evaluation of various algorithms for the problem still appears elusive, with existing studies being incomplete by only

considering a subset of algorithms (e.g., [11]), or not applying to list general k-cliques (e.g., [32]). This renders it difficult for a practitioner to determine which algorithm should be used for a specific application. Furthermore, existing ordering based algorithms, such as [11] and [32], are far from optimal which may explore many unpromising search paths in the k-clique listing procedure. To address these issues, we carry out an extensive experimental comparison of various algorithms for the k-clique listing problem. We also propose a new color ordering heuristics based on greedy graph coloring techniques [18, 49] which can significantly prune unpromising search paths. Using a variety of large real-world graphs with up to 3 million nodes and 100 million edge, we systematically compare the performance of 14 different algorithms (see Table 1). Our experimental results reveal the characteristics of different algorithms, based on which we present useful guidance for the selection of appropriate methods for various scenarios (see Table 4 and Fig. 15). In summary, the contributions of this paper are:

- We present a thorough experimental study of the known algorithms for listing/counting k-cliques using a variety of large real-world graphs. To the best of our knowledge, this is the first work that compares all the practical k-clique listing/counting techniques from an empirical viewpoint.
- We propose a new color ordering heuristics, based on which we develop three color-ordering based algorithms for k-clique listing. An appealing feature of the color-ordering based algorithms is that they can significantly prune unpromising search paths in the k-clique listing procedure, which allows us to list k-cliques for a large k value. Moreover, our optimized color-ordering based algorithm can also achieve the same time and space complexity as those of the state-of-the-art algorithm. The experimental results indicate that the optimized color-ordering based algorithm outperforms all other algorithms for listing k-cliques.
- We also evaluate the parallel variants of all ordering-based algorithms, and the results show a high degree of parallelism of the ordering-based algorithms. The source codes of this paper are publicly accessible at Github [1].

2. PRELIMINARIES

Let $G = (V, E)$ be an undirected graph, where V ($|V| = n$) and E ($|E| = m$) denote the set of nodes and edges respectively. We denote with $N_u(G)$ the set of neighbor nodes of u in G, and $d_u(G) = |N_u(G)|$ denotes the degree of u in G. A subgraph $H = (V_H, E_H)$ is called an induced subgraph of G if $V_H \subseteq V$ and $E_H = \{(u, v)|(u, v) \in E, u \in V_H, v \in V_H\}$. Below, we will give three important concepts called degeneracy [13], arboricity [31], and h-index respectively which are often used to design and analyze k-clique listing algorithms [9, 27, 11].

Given a graph G and an integer k, a k-core, denoted by C_k, is a maximal induced subgraph of G such that every node in C_k has a degree no smaller than k, i.e., $d_u(C_k) \geq k$ for every $u \in C_k$ [40]. The core number of a node u, denoted by c_u, is the largest integer k such that there exists a k-core containing u [40]. The maximum core number of a graph G, denoted by δ, is the maximum value of core numbers among all nodes in G. The maximum core number δ is also referred to as the degeneracy of G [13].

The degeneracy of a graph is closely related to a classic concept called arboricity [31], which is frequently used to measure the sparsity of a graph. Specifically, the arboricity of a graph G, denoted by α, is defined as the minimum number of forests into which its edges can be partitioned. It is well known that the degeneracy of G is a 2-approximation of the arboricity α, i.e., $\alpha \leq \delta \leq 2\alpha - 1$. Note that the core numbers of nodes can be computed in linear time using the classic core-decomposition algorithm [4]. As a result, the degeneracy of a graph can be efficiently determined. The arboricity of a graph, however, is hard to compute [16], therefore practitioners often use the degeneracy to approximate the arboricity of a graph [3].

Another related concept is called h-index of a graph G. The h-index of G is defined as the maximum integer η such that the graph contains η nodes of degree at least η [14]. More formally, $\eta \triangleq \arg\max_k(|\{u|d_u(G) \geq k, u \in V\}| \geq k)$. As shown in [27], η is an upper bound of α, and it is also bounded by $\sqrt{2m}$, i.e., $\alpha \leq \eta \leq \sqrt{2m}$. Notice that the degeneracy, arboricity, and h-index are often very small in real-world graphs [13].

Given a graph G and a parameter k, the k-clique listing/counting problem is a problem of listing/counting all complete subgraphs with size k in G. The state-of-the-art k-clique listing/counting algorithms are based on a graph orientation framework [32, 15, 11]. Let $\pi : V \to \{1, \cdots, n\}$ be a fixed total ordering on nodes in G. Then, for any undirected graph $G = (V, E)$, we are able to obtain a directed graph, denoted by $\vec{G} = (V, \vec{E})$, by orienting each edge $e \in E$ from the lower numbered node to the higher numbered node. More specifically, for each undirected edge $(u, v) \in E$, we obtain a directed edge $(u, v) \in \vec{E}$ if $\pi(u) < \pi(v)$ based on the total order π. Clearly, the directed graph $\vec{G}$ obtained by such an edge-orientation procedure is a directed acyclic graph (DAG), i.e., $\vec{G}$ cannot contain a directed cycles. The algorithms on the basis of such a graph orientation framework are referred to as ordering-based algorithms. In the following section, we will describe all the practical algorithms for k-clique listing/counting.

3. ALGORITHMS

Existing algorithms for k-clique listing and counting can be classified in two categories: exact algorithms and approximation algorithms. For the exact algorithms, we can further classify them in two subcategories: non-ordering based algorithms and ordering-based algorithms. Two representative non-ordering based algorithms are the classic Chiba-Nishizeki algorithm [9] and the MACE algorithm [29]. The ordering-based algorithms include the degree-ordering based algorithm [32, 15] and the degeneracy-ordering based algorithm [11], which are the two state-of-the-art k-clique listing algorithms [11]. We find that existing ordering-based algorithms can be optimized by a coloring-based pruning technique, resulting in a set of color-ordering based algorithms. Approximation algorithms for k-clique counting are often based on smart sampling techniques. Notable examples include the Turán-Shadow algorithm [20] and the ERS algorithm [12]. Table 1 describes our classification of different algorithms and summarizes their detailed properties. Below, we first give a detailed description for each algorithm, and then present a horizontal comparison of different algorithms in terms of running time, memory usage, parallelizability, and accuracy.

3.1 The Chiba-Nishizeki Algorithm

We start by describing the classic Chiba-Nishizeki algorithm [9], denoted by Arbo, which is the first practical algorithm for listing k-cliques. The algorithm first sorts the nodes in a non-increasing order of degree (line 7 of Algorithm 1). Then, the algorithm processes the nodes following this order (line 8 of Algorithm 1). For each node v, the algorithm creates a subgraph G_v induced by v's neighbors, and then recursively executes the same procedure on such an induced subgraph (lines 9-10 of Algorithm 1). It should be noted that when handling an induced subgraph G_v, the algorithm needs to reorder the nodes in G_v based on their degrees. After processing a node v, the algorithm removes v from the current graph to avoid that any k-clique containing v is repeatedly listed (line 11 of Algorithm 1). Algorithm 1 shows the pseudocode of the Chiba-Nishizeki algorithm. The striking feature of Algorithm 1 is that its time complexity is closely related to the arboricity of the graph. More specifically, Algorithm 1 lists all k-cliques in $O(km\alpha^{k-2})$ time using linear space, where α is the arboricity of the graph. For many real-world graphs, the arboricity is typically very small, thus Arbo is efficient in practice.

Table 1: Summary of different algorithms (" × " : no or not applicable; "✓" : yes; "?" : no existing implementation; ω : the maximum clique size; α : arboricity; η : h-index; Δ : the maximum degree; δ : degeneracy; C_k : the number of k-clique)

Problem	Category	Algorithms	Ordering Heuristics	Time Complexity	Space Complexity	Parallel
k-clique listing	exact	Arbo [9]	×	$O(km\alpha^{k-2})$	$O(m+n)$	×
		MACE [29]	×	$O(knm\alpha^{k-2})$	$O(m+n)$	×
		Degree [32, 15]	✓(degree ordering)	$O(km(\eta/2)^{k-2})$	$O(m+n)$	✓
		Degen [11]	✓(degeneracy ordering)	$O(km(\delta/2)^{k-2})$	$O(m+n)$	✓
		DegCol (improved Degree)	✓(color ordering)	$O(km(\Delta/2)^{k-2})$	$O(m+n)$	✓
		DegenCol (improved Degen)	✓(color ordering)	$O(km(\Delta/2)^{k-2})$	$O(m+n)$	✓
		DDegCol (optimized DegCol)	✓(optimized color ordering)	$O(km(\delta/2)^{k-2})$	$O(m+n)$	✓
		DDegree (optimized Degree)	✓(optimized degree ordering)	$O(km(\delta/2)^{k-2})$	$O(m+n)$	✓
	approx.	TuranSD [20]	×	$O(n\alpha^{k-1})$	$O(n\alpha^{k-2}+m)$	×
		ERS [12]	×	$\tilde{O}(n/C_k^{1/k}+m^{k/2}/C_k)$	$O(m+n)$	?
triangle listing ($k=3$)	exact	LDegree [32]	✓(degree ordering)	$O(\alpha m)$	$O(m+n)$	?
		LDegen [32]	✓(degeneracy ordering)	$O(\alpha m)$	$O(m+n)$	?
maximum clique search ($k=\omega$)	exact	RDS [33, 35]	×	$O(n2^n)$	$O(m+n)$	×
		MC-BRB [8]	×	$O(n2^n)$	$O(m+n)$	×

Algorithm 1: The Chiba-Nishizeki Algorithm (Arbo)

Input: An graph G and an integer k
Output: All k-cliques

Arbo $(G, \emptyset, k)$;
Procedure Arbo (G, R, l);
if $l = 2$ **then**
  for *each edge* (u, v) *in* G **do**
    output a k-clique $R \cup \{(u, v)\}$;
else
  Sort the nodes in G such that $d_{v_1}(G) \geq \cdots \geq d_{v_{|V_G|}}(G)$;
  for $i = 1$ **to** $|V_G|$ **do**
    Let G_{v_i} be the subgraph of G induced by the set of neighbors of v_i;
    Arbo $(G_{v_i}, R \cup \{v_i\}, l-1)$;
    Delete v_i from G;

Algorithm 2: The Ordering Based Framework

Input: An graph G and an integer k
Output: All k-cliques

Let π be a total ordering on nodes; /* degree ordering or degeneracy ordering */;
Let $\vec{G}$ be a DAG generated by π;
ListClique$(\vec{G}, \emptyset, k)$;
Procedure ListClique$(\vec{G}, R, l)$;
if $l = 2$ **then**
  for *each edge* (u, v) *in* $\vec{G}$ **do**
    output a k-clique $R \cup \{(u, v)\}$;
else
  for *each node* $v \in \vec{G}$ **do**
    Let $\vec{G}_v$ be the subgraph of $\vec{G}$ induced by all v's out-going neighbors;
    ListClique$(\vec{G}_v, R \cup \{v\}, l-1)$;

3.2 The MACE Algorithm

The MACE algorithm [29] was originally proposed to enumerate maximal cliques in a graph, but it can also be adapted to k-clique listing and counting. The main idea of the algorithm is very simple. Since any subset of a k-clique is also a clique, we can obtain a k-clique by recursively adding nodes to a sub-clique. Based on this, a depth-first backtracking algorithm can be easily devised. Specifically, in each recursion of the algorithm, for each node v that is not in the current sub-clique C, if $C \cup v$ is a valid clique and $1 + |C| < k$, then the algorithm recursively invokes the same procedure to list the k-cliques including $C \cup v$. To avoid duplications, the algorithm can process the nodes following a predefined node ordering (e.g., lexicographic order of node IDs). More specifically, in each recursion, the algorithm only considers the nodes that are with ranks larger than those of the nodes in the sub-clique C. The algorithm is shown to be polynomial delay which takes at most $O(n)$ time to list a k-clique [29]. Since the number of k-clique can be bounded by $O(km\alpha^{k-2})$[9], the worst-case time complexity of the algorithm is $O(knm\alpha^{k-2})$. As shown in [29], the space complexity of MACE is $O(m+n)$. Similar to the Arbo algorithm, MACE is also not easy to be parallelized, because it involves a depth-first backtracking procedure.

3.3 Existing Ordering Based Algorithms

In this subsection, we describe two ordering based algorithms which turn out to be the most efficient algorithms for the k-clique listing problem [32, 11]. The ordering-based algorithms was originally designed to list all triangles in an undirected graph [32]. Recently, it is successfully extended to list all k-cliques in an undirected graph [15, 11]. These algorithms start by computing a total ordering on nodes, and then construct a DAG based on the total ordering. After that, the algorithms list all k-cliques on the DAG which can prevent that a same k-clique is listed more than once. Specifically, by the ordering based algorithms, each k-clique R is only listed once when the algorithm processes the node in R with the smallest value in the total order. Algorithm 2 shows the framework of the ordering based algorithms.

As indicated in [11], the running time of the ordering based algorithms depends mainly on the orderings on nodes. Unfortunately, finding the best ordering for the k-clique listing algorithms is a NP-hard problem [11], thus we resort to seek good heuristic ordering approaches in practice. Below, we discuss two existing ordering heuristics which are the degree ordering [32] and the degeneracy ordering [11].

The degree-ordering based algorithm. Given a graph G, we can construct a total ordering by sorting the nodes in an non-decreasing of degree (break ties by node identities). We refer to such a total ordering as a degree ordering. Let π_d be a degree ordering. Clearly, for two nodes with identities u, v, and $u < v$, we have $\pi_d(u) \prec \pi_d(v)$ if $d_u(G) \leq d_v(G)$.

Let $\vec{G}_d$ be a DAG generated by the degree ordering π_d. Specifically, for each undirected edge $(u, v) \in G$ with $\pi_d(u) \prec \pi_d(v)$, we creates a directed edge (u, v) in $\vec{G}_d$. Let $N_u^+(\vec{G}_d)$ and $d_u^+(\vec{G}_d)$ be the set of outgoing neighbors of u in $\vec{G}_d$ and the out-degree of u, respectively. An appealing feature of such a DAG is that the out-degree of a node in $\vec{G}_d$ is bounded by the h-index of G.

LEMMA 1. *For any $u \in \vec{G}_d$, $d_u^+(\vec{G}_d) \leq \eta$, where η is the h-index of G.*

Algorithm 2 equipped with a degree ordering heuristics is referred to as a degree-ordering based algorithm, denoted by Degree in Table 1. Since the out-degree of the nodes in $\vec{G}_d$ is bounded by the h-index of the original undirected graph G (Lemma 1), we are capable of deriving the time complexity of the degree-ordering based algorithm as follows.

THEOREM 1. *Given a graph G and an integer k, the degree-ordering based algorithm lists all k-cliques in $O(km(\eta/2)^{k-2})$ time using $O(m+n)$ space, where η is the h-index of G.*

The degeneracy-ordering based algorithm. Given a graph G with degeneracy δ, a node ordering is called a degeneracy ordering if every node in G has δ or fewer neighbors that come later in the ordering [13]. It is easy to derive that the classic core-decomposition algorithm [4] that repeatedly deletes a node with minimum degree can generate a degeneracy ordering. We refer to such a ordering obtained by the core-decomposition algorithm [4] as a core-based degeneracy ordering, and it will be abbreviated as a degeneracy ordering in the rest of this paper. Obviously, the degeneracy ordering π_δ derived by the core-decomposition algorithm is a total ordering, thus the directed graph induced by π_δ is a DAG. We denote such a DAG by $\vec{G}_\delta$. It is easy to show that $d^+_u(\vec{G}_\delta) \leq \delta$ for any node $u \in \vec{G}_\delta$.

Algorithm 2 equipped with a degeneracy ordering (line 1 of Algorithm 2) is referred to as a degeneracy-ordering based algorithm, denoted by Degen in Table 1. Let δ be the degeneracy of a graph. It was shown that Degen has $O(km(\delta/2)^{k-2})$ time complexity using $O(m+n)$ space [11], which is slightly lower than that of the Arbo algorithm (because $\delta \leq 2\alpha - 1$).

3.4 The Color Ordering Based Algorithms

The main defects of existing ordering based algorithms are twofold: (1) the algorithms are very costly for listing k-cliques with a large k, especially when k is close to the size of a maximum clique (denoted by ω); (2) the algorithms are unable to prune unpromising search paths, in which no k-clique can be found. To overcome these shortcomings, we propose a new ordering heuristics, called color ordering, based on the greedy graph coloring technique [18, 49].

Assume that the graph G can be colored using χ colors. Then, we assign an integer color value taking from $[1, \cdots, \chi]$ to each node in G using any greedy coloring algorithm [18, 49] so that no two adjacent nodes have the same color value. After that, we sort the nodes in a non-increasing order based on their color values (break ties by node identities). We refer to such a total ordering as a color ordering, denoted by π_χ. Let $\vec{G}_\chi$ be the DAG induced by π_χ. Clearly, for each node $v \in \vec{G}_\chi$, the color value of v is no less than the color values of its outgoing neighbors. A striking feature of the color ordering is that the color values of the nodes can be used to prune unpromising search paths in the k-clique listing procedure. Algorithm 3 shows the pseudocode of the color-ordering based algorithm.

In Algorithm 3, the algorithm first invokes a greedy coloring algorithm [18, 49] to obtain a valid coloring for nodes (line 1). Based on the color values, the algorithm constructs a total ordering (line 2) and generates a DAG (line 3). After that, the algorithm recursively lists all k-cliques using a similar procedure as used in Algorithm 2. However, unlike Algorithm 2, Algorithm 3 is able to use the color values to prune unpromising search paths. Specifically, in line 11, when the algorithm explores a node v with a color value (denoted by $\mathsf{color}(v)$) smaller than l, the algorithm can safely prune the search paths rooted at v. The reason is as follows. By the color ordering heuristics, each out-going neighbor of v has a color value strictly smaller than $\mathsf{color}(v)$ (because v's out-going neighbors cannot have the same color value as that of v). Since $\mathsf{color}(v) < l$, all v's out-going neighbors have color values strictly smaller than $l-1$. As a result, v does not have $l-1$ out-going neighbors with different colors, indicating that v cannot be contained in any l-clique. Note that such a pruning strategy not only improves the performance of the algorithm, but it also enables the algorithm to list large k-cliques if the value of k is near to the maximum clique size. The traditional ordering-based algorithms introduced in Section 3.3, however, cannot be used to list large k-cliques.

Note that in Algorithm 3, the greedy coloring procedure colors the nodes following a fixed node ordering. When processing a node v, the greedy coloring procedure always selects the minimum color value that has not been used by v's neighbors to color v (lines 17-21). Clearly, various node orderings used in the greedy coloring procedure will generate different color orderings (line 15). Below, we introduce two color orderings based on an *inverse degree ordering* [49] and an *inverse degeneracy ordering* [18]. The algorithms based on these two color orderings are denoted by DegCol and DegenCol in Table 1 respectively.

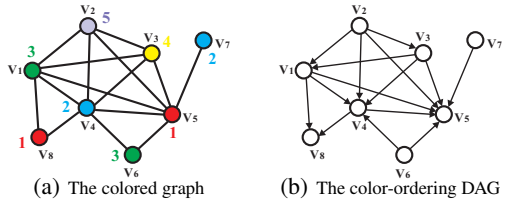
Algorithm 3: The Color-ordering Based Algorithm

```
   Input: An graph G and an integer k
   Output: All k-cliques
1  color [1, ··· , n] ←GreedyColoring(G);
2  Let π be the total ordering on nodes generated by color values;
3  Let G⃗ be a DAG generated by π;
4  ColorListClique(G⃗, ∅, k);
5  Procedure ColorListClique(G⃗, R, l);
6  if l = 2 then
7      for each edge (u, v) in G⃗ do
8          output a k-clique R ∪ {(u, v)};
9  else
10     for each node v ∈ G⃗ do
11         if color(v) < l then continue;
12         Let G⃗_v be the subgraph of G⃗ induced by all v's out-going neighbors;
13         ColorListClique(G⃗_v, R ∪ {v}, l − 1);
14 Procedure GreedyColoring(G);
15 Let π′ be a total ordering on nodes; /* π′ is an inverse degree ordering or an
     inverse degeneracy ordering */;
16 flag(i) ← −1 for i = 1, ··· , χ;
17 for each node v ∈ π′ in order do
18     for u ∈ N_v(G) do
19         flag(color(u)) ← v;
20     k ← min{i|i > 0, flag(i) ≠ v};
21     color(v) ← k;
22 return color(v) for all v ∈ G;
```

The degree-based color ordering. This color ordering is generated by the following steps. First, we invoke the greedy coloring procedure to color the nodes following a non-increasing ordering of degree (break ties by node ID). Then, we sorts the nodes in a non-increasing ordering of color value (break ties by node ID). The following example illustrates the detailed procedure for generating such a degree-based color ordering.

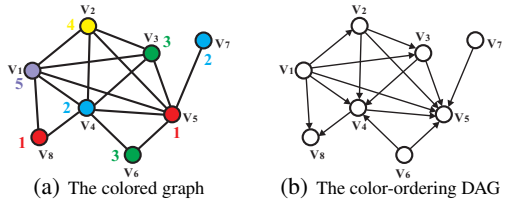
EXAMPLE 1. *Consider the graph shown in Fig. 1(a). It is easy to see that* $(v_5, v_4, v_1, v_3, v_2, v_8, v_6, v_7)$ *is an inverse degree ordering. Following this ordering, the algorithm first colors* v_5 *with the smallest color* 1*, and then colors* v_4 *with a color 2, and the other nodes are iteratively colored in a similar way. Fig.1(a) depicts the results of this coloring approach. Based on the color values, we can easily obtain the color ordering* $(v_2, v_3, v_6, v_1, v_7, v_4, v_8, v_5)$ *(break ties by node ID). The DAG generated by this color ordering is shown in Fig. 1(b).*

Unlike the traditional ordering heuristics shown in Section 3.3, the out-degree of a node in the DAG generated by the degree-based color ordering cannot be bounded by the h-index or the degeneracy of the graph. In this case, we can only obtain a trivial upper bound for the out-degree, which is the maximum degree of the graph Δ. As a result, the worst-case time complexity of Algorithm 3 is $O(km(\Delta/2)^{k-2})$ based on a similar analysis in Theorem 1. However, in practice, the running time of such a color-ordering based algorithm can be much lower than the worst-case time complexity as confirmed in our experiments. The reason is as follows. By the *inverse degree ordering*, the node with a high degree might be colored with a small color value, thus the high-degree nodes tend to be having a low rank in the degree-based color ordering. As a result, the high-degree nodes might have a relatively small out-degree in the DAG generated by the degree-based color ordering. Based on this, the *real* time consumption



(a) The colored graph (b) The color-ordering DAG

Figure 1: Illustration of the degree-based color ordering



(a) The colored graph (b) The color-ordering DAG

Figure 2: Illustration of the degeneracy-based color ordering

of the algorithm can be much lower than the worst-case time complexity $O(km(\Delta/2)^{k-2})$.

The degeneracy-based color ordering. Except the *inverse degree ordering*, we can also use an *inverse degeneracy ordering* [18] to color the nodes in Algorithm 3. Note that such an inverse degeneracy ordering can be easily obtained by reversing the node-deletion ordering of the core-decomposition algorithm [4]. The color ordering generated by this approach is referred to as a degeneracy-based color ordering. We use the following example to illustrate such a greedy coloring procedure, as well as the degeneracy-based color ordering.

EXAMPLE 2. *Consider the graph shown in Fig. 2(a). The ordering* $(v_5, v_4, v_3, v_2, v_1, v_8, v_6, v_7)$ *is an inverse degeneracy ordering. Following this ordering, we can obtain a colored graph shown in Fig. 2(a) by the greedy coloring procedure. Based on the color values, it is easy to derive that* $(v_1, v_2, v_3, v_6, v_4, v_7, v_8, v_5)$ *is a color ordering. Fig. 2(b) shows the DAG generated by this color ordering.*

Similar to the degree-based color ordering, the maximum out-degree of the DAG generated by the degeneracy-based color ordering could be equal to Δ in the worst case. Thus, the worst-case time complexity of Algorithm 3 with a degeneracy-based color ordering is $O(km(\Delta/2)^{k-2})$. However, in practice, the running time of our algorithm is much lower than the worst-case time complexity as verified in our experiments. The reasons are as follows. By the *inverse degeneracy ordering*, a node with a large core number may be assigned by a small color value, thus such a node might have a low rank in the degeneracy-based color ordering. As a result, the DAG generated by the degeneracy-based color ordering might be *similar* to the DAG induced by the degeneracy ordering, indicating that the *real* time cost of Algorithm 3 can be much lower than $O(km(\Delta/2)^{k-2})$. In addition, Algorithm 3 also applies the color values to prune unpromising search paths, thus it can be very efficient in practice.

The optimized color-ordering based algorithm. Here we present a simple but effective strategy to reduce the worst-case time complexity of our color-ordering based algorithms. In particular, we can first generate a DAG $\vec{G}$ based on the degeneracy ordering. Let $N_v^+(\vec{G})$ be the set of out-going neighbors of v in $\vec{G}$. Then, for each v, we construct a subgraph $G_v = (V_v, E_v)$ of the original undirected G that is induced by the nodes in $N_v^+(\vec{G})$ ($V_v = N_v^+(\vec{G})$). After that, for each v with $|V_v| \geq k-1$, we iteratively list all $k-1$ cliques in G_v using Algorithm 3. The pseudocode of such an optimized algorithm is shown in Algorithm 4. The optimized degree-based color ordering algorithm is denoted by DDegCol in Table 1. An important feature of this algorithm is that its worst-case time complexity is $O(km\delta^{k-2})$, which is the same as that of the degeneracy-ordering based algorithm.

Algorithm 4: The Optimized Color-ordering Based Algorithm

Input: An graph G and an integer k
Output: All k-cliques
1 Let π be a degeneracy ordering;
2 Let $\vec{G}$ be a DAG generated by π;
3 Let $N_v^+(\vec{G})$ be the set of out-going neighbors of v;
4 Let $G_v = (V_v, E_v)$ be the subgraph of G induced by the nodes in $N_v^+(\vec{G})$;
5 **for** *each* $v \in G$ **do**
6 **if** $|V_v| \geq k-1$ **then**
7 Invoke Algorithm 3 on the subgraph G_v with parameter $k-1$;

THEOREM 2. *Given a graph G and an integer k, Algorithm 4 lists all k-cliques in $O(km(\delta/2)^{k-2})$ time using linear space, where δ is the degeneracy of G.*

Optimized degree-ordering based algorithm. The above optimization strategy can also be used to reduce the worst-case time complexity of the degree-ordering based algorithm. Specifically, instead of using Algorithm 3, we make use of the degree-ordering based algorithm to list all $(k-1)$-cliques on G_v in the line 7 of Algorithm 4. Such an optimized degree-ordering based algorithm is denoted by DDegree in Table 1. By a similar analysis in Theorem 2, the time complexity of DDegree is $O(km(\delta/2)^{k-2})$.

3.5 Parallel Ordering Based Algorithms

In [11], Danisch et al. proposed two general parallel strategies for ordering-based algorithms, namely, NodeParallel and EdgeParallel respectively. Note that all the above ordering-based algorithms can be parallelized using the NodeParallel or the EdgeParallel strategy. Let $\vec{G}_v$ be the subgraph of $\vec{G}$ induced by the outgoing neighbors of v. Recall that by the ordering-based algorithm, the subgraph $\vec{G}_v$ for each node v can be processed independently (see line 9 of Algorithm 2, line 10 of Algorithm 3, and line 5 of Algorithm 4). The NodeParallel strategy processes all of such G_v's in parallel. However, since the k-cliques may not be distributed uniformly in all G_v's, which gives rise to unbalanced workloads on different CPUs. This shortcoming can be alleviated by the EdgeParallel strategy. Let (u, v) be an edge in G, and $\vec{G}_{uv}$ be the subgraph of $\vec{G}$ induced by the common outgoing neighbors of u and v (the subgraph induced by $N_u^+(\vec{G}) \cap N_v^+(\vec{G})$). The EdgeParallel strategy processes all $\vec{G}_{uv}$'s in parallel. Specifically, EdgeParallel invokes the ListClique$(\vec{G}_{uv}, \{u, v\}, k-2)$ procedure (or the ColorListClique$(\vec{G}_{uv}, \{u, v\}, k-2)$ procedure for color-ordering based algorithm) for all $\vec{G}_{uv}$'s in parallel. Since the $\vec{G}_{uv}$'s are generally smaller than the $\vec{G}_v$'s, thus EdgeParallel can achieve a higher degree of parallelism, which is also confirmed in our experiments.

Remark. It is worth remarking that the EdgeParallel strategy can also be used for parallelizing Algorithm 4. In particular, for each directed edge (u, v) in the degeneracy-ordering DAG $\vec{G}$, we can obtain a set of common out-going neighbors $N_{uv}^+ = N_u^+(\vec{G}) \cap N_v^+(\vec{G})$. Then, we construct a subgraph G_{uv} of G that is induced by the nodes in N_{uv}^+. After that, the algorithm can process all G_{uv}'s in parallel.

3.6 Approximation Algorithms

The Turán-Shadow algorithm. The Turán-Shadow algorithm [20], denoted by TuranSD, is a randomized algorithm that is designed to estimate the number of k-cliques in an undirected graph. The algorithm involves two sub-procedures: ShadowConstruction and Sampling (Algorithm 5). In the ShadowConstruction procedure, the algorithm constructs a data structure called Turán Shadow based on the classic Turán theorem [48]. Specifically, the Turán theorem states that for any graph G, if the density of G, denoted by

Algorithm 5: The Turán-Shadow Algorithm (TuranSD)

Input: An graph G and an integer k
Output: An estimation of the number k-cliques

1 **Procedure** ShadowConstruction(G, k);
2 $\mathcal{T} \leftarrow \{(V, k)\}, \mathcal{S} \leftarrow \emptyset$;
3 **while** $\exists (H, l) \in \mathcal{T}$ *s.t.* $\rho(H) \leq 1 - 1/(l-1)$ **do**
4 Let G_H be the subgraph of G induced by H;
5 $\vec{G}_H \leftarrow$ construct a DAG by the degeneracy ordering on G_H;
6 Let $N_v^+(\vec{G}_H)$ be the set of out-going neighbors of v in $\vec{G}_H$;
7 **for** *each* $u \in H$ **do**
8 **if** $l \leq 2$ *or* $\rho(N_v^+(G_H)) > 1 - 1/(l-2)$ **then**
9 $\mathcal{S} \leftarrow \mathcal{S} \cup \{(N_v^+(G_H), l-1)\}$;
10 **else**
11 $\mathcal{T} \leftarrow \mathcal{T} \cup \{(N_v^+(G_H), l-1)\}$;
12 $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(H, l)\}$;

13 **Procedure** Sampling$(\mathcal{S})$;
14 $w(H) \leftarrow \binom{|H|}{l}$ for each $(H, l) \in \mathcal{S}$;
15 $p(H) \leftarrow w_H / \sum_{(H,l)\in\mathcal{S}} w(H)$;
16 **for** $r = 1$ **to** t **do**
17 Independently sample (H, l) from $\mathcal{S}$ based on the probability $p(H)$;
18 $R \leftarrow$ randomly picking l nodes from H;
19 **if** R forms a l-clique **then** $X_r \leftarrow 1$;
20 **else** $X_r \leftarrow 0$;
21 **return** $\frac{\sum_r X_r}{t} \sum_{(H,l)\in\mathcal{S}} w(H)$;

$\rho(G) = m/\binom{|n|}{2}$, is larger than $1 - 1/(k-1)$, then G contains a k-clique [48].

For a given integer k, the Turán Shadow $\mathcal{S}$ consists of a set of pairs (H, l), where $H \subseteq V$ is a subset of nodes and $l \leq k$ is an integer. For each pair $(H, l) \in \mathcal{S}$, the density of the subgraph induced by H, denoted by $\rho(G_H)$, is larger than the so-called Turán threshold $1 - 1/(l-1)$. Therefore, for a pair (H, l), the subgraph G_H must contain a l-clique by Turán theorem.

Jain and Seshadhri [20] proposed an elegant refinement procedure to construct such a Turán Shadow. The pseudocode of the refinement procedure is shown in Algorithm 5 (lines 1-12). Initially, the algorithm sets $\mathcal{T} = \{(V, k)\}$ and the Turán Shadow $\mathcal{S} = \emptyset$ (line 2). Then, the algorithm iteratively picks a pair (H, l) from $\mathcal{T}$ that does not satisfy the Turán threshold (line 3). With such a pair (H, l), the algorithm constructs a DAG $\vec{G}_H$ for H based on the degeneracy ordering (lines 4-5). For each node $v \in H$, the algorithm creates an outgoing neighborhood $N_v^+(\vec{G}_H)$ in $\vec{G}_H$ (line 6). Subsequently, the algorithm constructs a set of $|H|$ pairs $\{(N_v^+(\vec{G}_H), l-1) | v \in H\}$. For any pair $(N_v^+(\vec{G}_H), l-1)$ that meets the Turán threshold will go to the Turán Shadow $\mathcal{S}$, otherwise it goes to $\mathcal{T}$ (lines 7-11). After that, the algorithm deletes the pair (H, l) from $\mathcal{T}$ (line 12), and recurses on the updated $\mathcal{T}$ (line 3). The idea behind the ShadowConstruction procedure is that it iteratively refines the pairs in $\mathcal{T}$ until all pairs satisfies the Turán threshold.

Based on the Turán Shadow $\mathcal{S}$, Jain and Seshadhri [20] proved that there exists a one-to-one mapping between a k-clique in G and a l-clique in G_H for a pair $(H, l) \in \mathcal{S}$, where G_H is a subgraph induced by H. As a result, counting the number of k-cliques in G is equivalent to compute the total number of l-cliques in G_H for each pair (H, l). To do this, a simple weighted sampling procedure is sufficient to estimate the l-clique counts in $\mathcal{S}$ [20]. The pseudocode of such a sampling procedure is detailed in Algorithm 5 (lines 13-21). As shown in [20], Algorithm 5 can obtain a $1 + \epsilon$ approximation with high probability using $O(n\alpha^{k-1} + m)$ time and $O(n\alpha^{k-2} + m)$ space.

The ERS Algorithm. The ERS algorithm, proposed by Eden, Ron, and Seshadhri [12], is a randomized algorithm for approximating the number of k-cliques. Unlike the TuranSD algorithm, ERS is based on a query model where a query algorithm can randomly perform three queries on the graph: (1) degree queries (i.e., query a node's degree), neighbor queries (i.e., query a node's neighbors), and pair queries (i.e., query two nodes to determine whether they form an edge). The general idea of the ERS algorithm is as follows. First, ERS randomly samples a set of vertices S from the graph. Let C_k be the number of k-cliques in the graph, and $c_k(v)$ be the number of k-cliques that are incident to v. Then, the algorithm estimates C_k by $n/(k|S|) \times \sum_{v\in S} c_k(v)$. Note that $c_k(v)$ can be computed by using the query algorithm. However, for a random v, $c_k(v)$ can have very large variance, resulting in the estimator is inaccurate. The ERS algorithm makes use of a complicated *clique assignment* technique to reduce the variance. In theory, the authors prove that ERS can achieve a $1+\epsilon$ approximation of the number of k-cliques with high probability. The total time complexity of ERS is $\tilde{O}(n/C_k^{1/k} + m^{k/2}/C_k)$ where the notion $\tilde{O}$ hides a $poly(\log n)$ term [12], and the space complexity of the algorithm is $O(m+n)$. When $C_k \geq m^{k/2-1}$, the time complexity of the ERS algorithm is sublinear with respect to the graph size.

3.7 Algorithms for Special k Values

In this subsection, we describe 4 state-of-the-art k-clique listing algorithms for two special k values: $k = 3$ and $k = \omega$ (ω is the maximum clique size).

Triangle listing and counting algorithms. Clearly, when $k = 3$, a k-clique is a triangle. The state-of-the-art triangle listing algorithm is an ordering-based algorithm [32], which is similar to Algorithm 2. Specifically, the algorithm first orients the edges of the graph from the lower-ranked node to the higher-ranked node based on a pre-defined node ordering (e.g., degree ordering and degeneracy ordering). Then, for each directed edge (u, v) with $u \prec v$, the algorithm identifies every node w in the set $N^+(u) \cap N^-(v)$, where $N^+(u)$ denotes the set of outgoing neighbors of u and $N^-(v)$ is the set of incoming neighbors of v. Obviously, nodes u, v, and w form a triangle. Such an algorithm lists each triangle only once and it can achieve $O(\alpha m)$ time complexity using $O(m+n)$ space. We implement and evaluate this state-of-the-art algorithm using the degree ordering and the degeneracy ordering respectively (denoted by LDegree and LDegen in Table 1).

Maximum clique search algorithms. When $k = \omega$, existing maximum clique search algorithms can also be used for k-clique listing. In this paper, we evaluate two state-of-the-art maximum clique search algorithms: RDS [33, 35] and MC-BRB [8]. The RDS algorithm is a branch-and-bound algorithm which is based on a Russian Doll Search paradigm [33]. Specifically, RDS first orders the nodes as $(v_1, v_2, \cdots, v_n)$. Let $S_i = \{v_i, v_{i+1}, \cdots, v_n\}$ be the subset of V, and ω_i be the maximum clique size of the subgraph $G[S_i]$ induced by S_i. Clearly, if $\omega_i = \omega_{i+1} + 1$, the maximum clique in $G[S_i]$ contains v_i, otherwise $\omega_i = \omega_{i+1}$. RDS recursively computes $\omega_n, \omega_{n-1}, \cdots, \omega_1$ using backtracking. An interesting pruning rule of RDS is that if we find a clique with size greater than t, then we can prune the search when we consider v_i to become the $(j+1)$-th candidate node and $j + \omega_i \leq t$ [33, 35]. It is easy to see that RDS uses $O(m+n)$ space and takes at most $O(n2^n)$ time in the worst case, because the search space of the backtracking procedure is bounded by $O(2^n)$. Note that although RDS has an exponential time complexity, it is typically very efficient on real-world sparse graphs as shown in [35].

A more recent maximum clique search algorithm MC-BRB proposed by Chang [8] can be considered as another state-of-the-art algorithm. The key idea of MC-BRB is that it first transforms the maximum clique problem to a set of k-clique finding problems, each working on an ego-network. Then, MC-BRB applies several smart reduction techniques to reduce the size of the ego-network while preserving the existence of a k-clique. With the reduction techniques, a branch-reduce-and-bound framework is developed to identify the k-clique [8] efficiently. Similar to RDS, MC-BRB takes at most $O(n2^n)$ time in the worst case and uses $O(m+n)$ space. The practical performance of MC-BRB was shown to be much better than the other algorithms on sparse graphs [8].

3.8 Horizontal Comparison of Algorithms

Running time. As shown in Table 1, MACE has the highest worst-case time complexity among all the exact k-clique listing algorithms, followed by two color-ordering based algorithms (DegCol and DegenCol), the degree-ordering based algorithm (Degree), the Arbo algorithm, the degeneracy-ordering based algorithm (Degen), and two optimized algorithms (DDegCol and DDegree). Note that although Arbo has a lower time complexity than DegCol, DegenCol, and Degree, all these ordering-based algorithms are expected to be faster than Arbo in practice, because Arbo needs to sort the nodes by their degrees in each recursion which is often very costly. Compared to the other ordering-based algorithms, Degree takes less time to compute the ordering, but it may consumes more time in the recursive k-clique listing procedure due to the higher time complexity. Compared to Degen, the two color-ordering based algorithms DegCol and DegenCol have a slightly higher time complexity, but it can significantly reduce the search space by using color-based pruning technique. Although the two optimized ordering-based algorithms DDegCol and DDegree achieve the best time complexity, both of them use a size-based constraint for pruning in the first recursion which is typically less effective than the color-based pruning rule as used in DegCol and DegenCol. With all these, there does not exist a clear winner among all the exact algorithms, thus our experimental analysis is critical. Additionally, for the approximation algorithms, the worst-case time complexity of TuranSD is typically higher than that of ERS, but its time overhead is still much lower than all the exact algorithms.

Memory usage. All the exact k-clique listing algorithms have $O(m + n)$ space complexity, thus their space overheads are comparable. For a more subtle comparison, the degree-ordering based algorithm (Degree) uses less space than the others. This is because Degree only needs to maintain a degree array and the input graph while the other algorithms require to store several other additional information. Specifically, both the degeneracy-based ordering algorithm (Degen) and the optimized degree ordering algorithm (DDegree) have to store an additional $O(n)$ array to compute the degeneracy ordering. The color-based ordering algorithms (DegCol, DegenCol,DDegCol) need an $O(n)$ array to maintain the colors of the nodes. Arbo uses $O(n)$ additional space to sort the degree array in linear time in each recursion, and MACE also needs $O(n)$ space to store several additional information for pruning the backtracking procedure. For the approximation algorithms, ERS is space-efficient which consumes $O(m + n)$ space, while TuranSD is more memory intensive as it takes $O(n\alpha^{k-2} + m)$ space to maintain the Turán-Shadow.

Parallelizability. All the ordering-based k-clique listing algorithms in Table 1 can be easily parallelized, because all these algorithms independently list k-cliques on each node's out-going neighborhood. Arbo, however, is very hard to be parallelized. This is because Arbo includes a sequential step in line 11 (see Algorithm 1), which leads to that the current iteration on the subgraph induced by v_i's neighbors (line 9 of Algorithm 1) depends on the graph $G \setminus \{v_{i-1}\}$ obtained in the previous iteration. Such a sequential step makes an efficient parallelization of the algorithm non-trivial. MACE and TuranSD are also very difficult to be parallelized, because both of them involve a procedure of depth-first search, resulting in an efficient parallelization of the algorithm not easy. For the ERS algorithm, the node sampling procedure can be easily parallelized, but it is unclear whether the variance reduction technique used in ERS is parallelizable.

Accuracy. Since all exact algorithms shown in Table 1 can obtain the exact k-cliques count, we mainly compare the accuracy of two approximation algorithms. Recall that TuranSD performs sampling on the pre-computed Turán-Shadow structure $\mathcal{S}$ rather than on the input graph. For each element $(H, l) \in S$, since the node set H is often very dense (denser than the Turán threshold), the successful rate of sampling a l-clique on H can be very high, and therefore TuranSD is very accurate. ERS performs node sampling on the input graph and then applies a complicated variance reduction technique to cut the variance of the estimator. Although ERS can achieve a $1 + \epsilon$ approximation in theory, its practical accuracy is generally much lower than that of TuranSD as shown in our experiments.

3.9 Other Related Algorithms

k-clique listing and counting. Concurrently with our work, Jain and Seshadhri [21] developed an elegant k-clique counting algorithm, called PIVOTER, based on a classic *pivoting* technique, which was originally proposed for reducing the recursion tree of the maximal clique enumeration algorithm [7]. In particular, the PIVOTER algorithm first constructs a *succinct clique tree* based on the pivoting technique. Such an elegant tree structure maintains a compressed unique representation of all cliques in the graph. Then, PIVOTER can extract exact clique counts for all values of k based on the succinct clique tree. In [21], PIVOTER was shown to be faster than Degen [11]. An interesting question is whether the color-ordering based heuristics developed in this paper can be used to further optimize the PIVOTER algorithm. Again, concurrently with our work, a recent technical report [41] proposed a work-efficient parallel k-clique listing algorithm based on a newly-developed parallel low-degree orientation technique. In [41], such a parallel algorithm is shown to be faster than the parallel version of Degen when using 60 threads. It is also interesting to study whether the color-based pruning technique proposed in this paper can be applied to further speed up the work-efficient parallel algorithm [41]. We leave all the above mentioned issues for future work. Additionally, the k-clique listing problem was also studied in the MapReduce setting [15]. In particular, Finocchi et al. [15] proposed a MapReduce algorithm for k-clique listing based on a degree ordering heuristics, where the degree ordering heuristics has already been evaluated in this paper.

Triangle listing and counting. Except the triangle listing and counting algorithm described in Section 3.7, there exist several other algorithms that do not evaluate in this paper, since those algorithms are either less efficient than the algorithms evaluated or cannot obtain the exact solution. Two notable exact algorithms are Schank's *edge iterator* algorithm [38] and Latapy's compact-forward algorithm [23]. The time complexity of these two exact algorithms are $O(\alpha m)$[38, 23]. Tsourakakis et al. [47] developed an edge sampling algorithm to estimate the number of triangles in a graph. Recently, several algorithms have been proposed to handle the case when the graph does not fit into the main memory. Becchetti et al. [5] devised a triangle counting algorithm in the semi-streaming model. Chu and Cheng [10], and Hu et al. [19] proposed I/O-efficient algorithms for triangle listing. Suri and Vassilvitskii [42], and Kolda et al. [22] developed triangle listing algorithms in the MapReduce setting.

Maximum clique search. There also exist several other maximum clique search algorithms not evaluated in this paper. For example, Lu et al. [28] proposed an efficient randomized algorithm which can find a near maximum clique efficiently. Segundo et al. [39] developed an efficient branch-and-bound algorithm based on bit-parallel technique. Tomita [44, 45, 43] proposed a series of maximum clique search algorithms based on the well-known Bron-Kerbosch algorithm [7] for enumerating maximal cliques. As shown in [8], all the above mentioned algorithms are either less efficient than MC-BRB or cannot work on large sparse graphs.

4. EXPERIMENTS

4.1 Experimental Setup

Datasets. We collect 17 large real-world graphs obtained from (`http://networkrepository.com/`). We divide the datasets into two groups based on the size of a maximum clique (ω): small-ω graphs (the first 9 graphs in Table 2), for which we are capable

Table 3: Runtime of different exact algorithms for listing all k-cliques (for all $k \geq 3$, in second)

Dataset	#Cliques	Arbo	MACE	Degree	Degen	DegCol	DegenCol	DDegCol	DDegree
Nasasrb	50,915,452,049	INF	INF	9,872	9,965	1,346	1,464	**1,307**	1,950
FBWosn	87,432,996,809	INF	INF	INF	INF	3,171	2,751	**2,119**	3,408
WikiTrust	12,652,027,321	11,568	INF	2,962	2,710	421	430	**326**	503
Youtube	44,272,612	65	320	20	19	**12**	**12**	17	14
Pokec	3,229,825,345	3,252	5,740	1,107	1,086	**236**	241	434	392
WikiCN	17,495,574,003	INF	INF	4,636	4,566	**519**	526	598	790
Shipsec5	12,961,780,899	7,347	17,130	2,734	2,435	354	**337**	352	515
BaiduBK	7,968,788,787	6,559	19,920	2,167	2,107	**390**	398	492	543
SocFBa	13,238,147,662	13,899	INF	3,522	3,544	786	773	**695**	809

Table 2: Datasets ($N_{\max}$: #. maximum cliques, 1K=1,000)

Dataset	$n = \|V\|$	$m = \|E\|$	ω	$N_{\max}$	δ	Δ
Nasasrb	54,870	1,311,227	24	1,939	36	275
FBWosn	63,731	817,090	30	2	85	2K
WikiTrust	138,587	715,883	25	54	65	12K
Youtube	1,157,828	2,987,624	17	2	50	28.8K
Pokec	1,632,803	22,301,964	29	6	48	15K
WikiCN	1,930,270	8,956,902	33	2	128	30K
Shipsec5	179,104	2,200,076	24	744	30	75
BaiduBK	2,140,198	17,014,946	31	4	83	98K
SocFBa	3,097,165	23,667,394	25	35	75	5K
WebSK	121,422	334,419	82	3	82	590
Citeseer	227,320	814,134	87	1	87	1K
WebStan	281,904	1,992,636	61	10	87	39K
DBLP	317,080	1,049,866	114	1	114	343
Digg	770,799	5,907,132	50	192	237	17.6K
Orkut	2,997,166	106,349,209	47	7	254	27.5K
Skitter	1,696,415	11,095,298	67	4	112	35K
DielFilter	420,408	16,232,900	45	15,446	57	302

of listing all k-cliques (for all k), and large-ω graphs (the last 8 graphs in Table 2), for which we can only list k-cliques for small k values (or large k-cliques with k values near to ω). This is because if ω is large, the number of k-cliques in the maximum clique is exponential large for a relatively large k, and thus any exact k-clique listing algorithm is doomed to failure. For example, if $\omega = 60$ and $k = 20$, then a 60-clique contains 4.2×10^{15} 20-cliques. For each graph dataset, we report its maximum clique size ω, the number of maximum cliques $N_{\max}$, the maximum k-core number δ, and the maximum degree Δ, which could affect the running time of the k-clique listing/counting algorithms. Table 2 summarizes the statistics of our datasets.

Algorithms. In our experiments, we evaluate all 14 algorithms shown in Table 1. For Arbo, MACE, Degen, TuranSD, and MC-BRB, we use the C++ implementation of these algorithms provided by the original authors. We implement all the other 9 algorithms in C++. In addition, we also implement the parallel variants of 6 ordering-based algorithms in C++ and OpenMP.

Experimental settings. We conduct our experiments on a Linux machine equipped with 2 Intel Xeon 2.40GHz CPUs with 12 cores (a total of 24 threads) and with 128 GB RAM. Unless otherwise specified, we evaluate all algorithms with a varying k from 3 to 9. We also evaluate three color-ordering based algorithms and the DDegree algorithm by varying k from $\omega - 8$ to ω, where ω is the size of a maximum clique. Note that except the color-ordering based algorithms and DDegree, all the other exact algorithms presented in Section 3 are intractable for listing large k-cliques when k is varied from $\omega - 8$ to ω. In addition, we also evaluate the parallel implementations of all the ordering-based algorithms by varying the number of threads from 1 to 24. In all our experiments, we set the time limit to 8 hours for each algorithm. The running time of any algorithm that exceeds 8 hours is recorded with a special symbol "INF".

4.2 Exact Algorithms on Small-ω Graphs

Runtime for listing all k-cliques. Table 3 reports the running time of various exact algorithms for listing all k-cliques (k is varied from 3 to ω) on 9 small-ω graphs. On these datasets, we observe that the three color-ordering based algorithms (DegCol, DegenCol, and DDegCol) achieve similar performance, and all of them significantly outperform the other algorithms. For example,

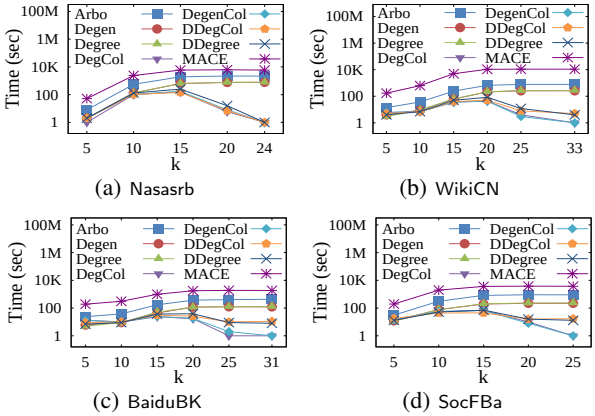


Figure 3: Runtime of exact algorithms on small-ω graphs

in the Nasasrb dataset, the running time of DegCol, DegenCol, and DDegCol are 1,346, 1,464, and 1,307 seconds respectively. However, in the same dataset, the running time of DDegree, Degree and Degen are 1,950, 9,872, and 9,965 seconds respectively. Arbo and MACE perform even worse and they are intractable for listing all k-cliques in this dataset (i.e., they cannot finish in 8 hours). This is because all our color-ordering based algorithms are equipped with a color-value based pruning strategy, which can largely prune the search paths when listing large k-cliques. In general, the traditional ordering-based algorithms (Degree and Degen) are significantly faster than MACE and Arbo, which are consistent with the results shown in [32, 11]. In addition, we can see that the optimized degree-ordering based algorithm (DDegree) performs very well. The running time of DDegree is slightly higher than those of the color-ordering based algorithms, but it is considerably lower than that of the traditional ordering based algorithms (Degree and Degen). The reason could be that DDegree applies the out-degrees to prune the nodes that are definitely not contained in any k-clique (see line 6 of Algorithm 4); such a pruning rule might be very effective when listing large k-cliques.

Runtime of various algorithms with varying k. We plot the running time achieved by each algorithm as a function of k on Nasasrb, WikiCN, BaiduBK, and SocFBa in Fig. 3. The results on the other small-ω graphs are consistent. As can be seen, the running time of Arbo, MACE, Degree, and Degen increases as k increases. However, for the three color-ordering based algorithms and the DDegree algorithm, the running time first increases as k increases to $\omega/2$, and then drops when k increases to ω. The reason could be that the pruning performance of these four algorithms becomes more effective for a larger k. The results also suggest that the performance of the color-ordering based algorithms and DDegree seems to match the number of outputs of the problem, since the number of k-cliques of a graph also exhibits a similar function of k. In addition, we observe that the three color-ordering based algorithms are slightly faster than DDegree, and all of them outperform the other competitors. We also note that both Degree and Degen significantly outperform MACE and Arbo, which are consistent with the previous results. In summary, for small-ω graphs (e.g., $\omega \leq 30$), the color-ordering based algorithms are clearly the winners, thus they are recommended to use in this case.

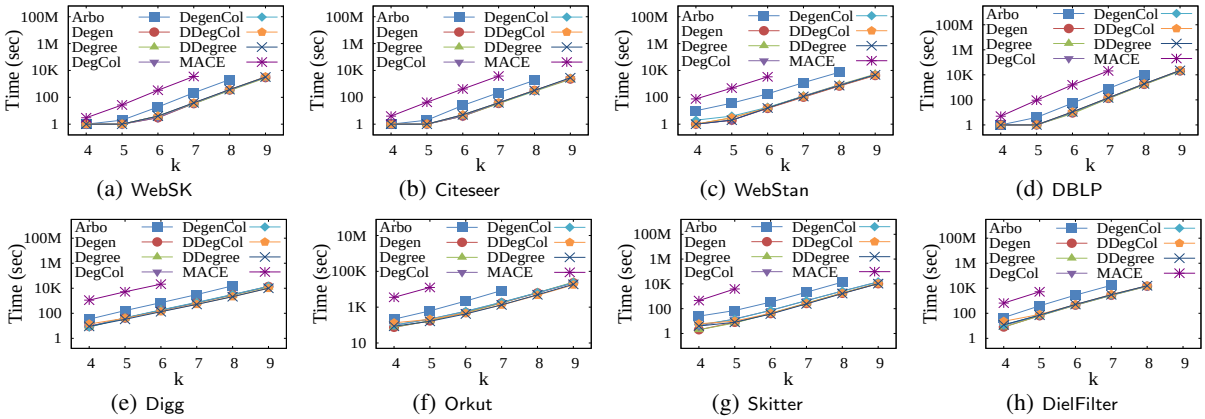


Figure 4: Runtime of different exact algorithms on large-ω graphs with varying k from 4 to 9

4.3 Exact Algorithms on Large-ω Graphs

Fig. 4 shows the running time of various exact algorithms on 8 large-ω graphs, with varying k from 3 to 9. As desired, the running time of each algorithm increases with an increasing k. All the ordering-based algorithms achieve similar performance, and they are significantly faster than Arbo and MACE. In addition, we can also observe that both the optimized color-ordering based algorithm (DDegCol) and the optimized degree-ordering based algorithm (DDegree) seem to be slightly faster than the other ordering-based algorithms (especially for a large k). Taking the Orkut dataset as an example (Fig. 4(f)), DDegCol and DDegree consumes $17,595$ and $18,902$ seconds respectively when $k = 9$. However, under the same parameter setting, Degree, Degen, DegCol, and DegenCol takes $23,200$, $24,827$, $22,610$, $23,467$ seconds respectively. The DDegCol algorithm, for example, improves the running time over Degree, Degen, DegCol, and DegenCol by 24%, 29%, 25%, and 22%, respectively. The reason could be the pruning rule equipped within the optimized algorithms (Algorithm 4) is effective for a relatively large value of k.

We also plot the running time of DegCol, DegenCol, DDegCol, and DDegree on large-ω graphs in Fig. 5, with varying k from $\omega - 8$ to ω. Note that Arbo, MACE, and the traditional ordering-based algorithms (Degree and Degen) are intractable for listing k-cliques if k is near to the maximum clique size. We observe that the performance of the three color-ordering based algorithms are significantly better than DDegree on most datasets, due to the powerful color-value based pruning technique. As two exceptions, on Digg and Orkut, the DDegree algorithm is faster than DegCol and DegenCol, but it is considerably worse than DDegCol. The reason could be that both the degeneracy and the maximum degree of these two graphs are very large, thus the number of colors obtained by the greedy coloring procedure in both DegCol and DegenCol can also be very large, which reduces the color-value based pruning performance in Algorithm 3.

In summary, all ordering-based algorithms perform very well for listing small k-cliques on large-ω graphs. Arbo, MACE, and the traditional ordering-based algorithms cannot be used to list k-cliques when k is near to the maximum clique size. For both small and large values of k, the overall performance of the optimized color-ordering based algorithm (DDegCol) seems to be better than the other algorithms. Hence, such an optimized color-ordering based algorithm is recommended to use for large-ω graphs.

4.4 Memory Usage of Exact Algorithms

We evaluate the memory overheads of various exact algorithms on Digg and Orkut for $k = 4$ and $k = 8$ respectively. The results are shown in Fig. 6. Similar results can also be observed on the other datasets and for the other values of k. As expected, the space usage of each algorithm is only a few times larger than the graph size, since all the exact algorithms presented in Section 3 have linear space complexity. These results indicate that all the exact algorithms are space-efficient for listing k-cliques in large real-world graphs. In addition, for a more subtle comparison, we can observe that Degree consumes less memory than the other exact algorithms, which are consistent with our analysis in Section 3.8.

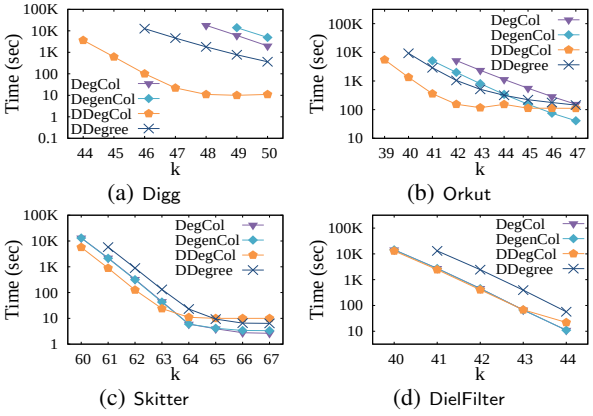


Figure 5: Runtime of different exact algorithms on large-ω graphs with varying k from $\omega - 8$ to $\omega - 1$

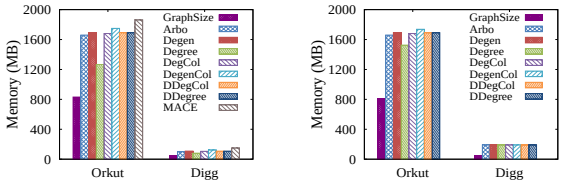
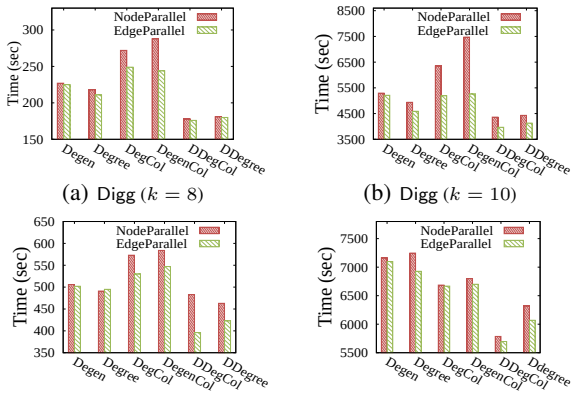


Figure 6: Memory usage of exact algorithms

4.5 Evaluation of Parallel Algorithms

In this subsection, we carry out a set of experiments to evaluate the performance of six ordering-based parallel algorithms. Note that both Arbo and MACE cannot be parallelized, thus we preclude it in these experiments.

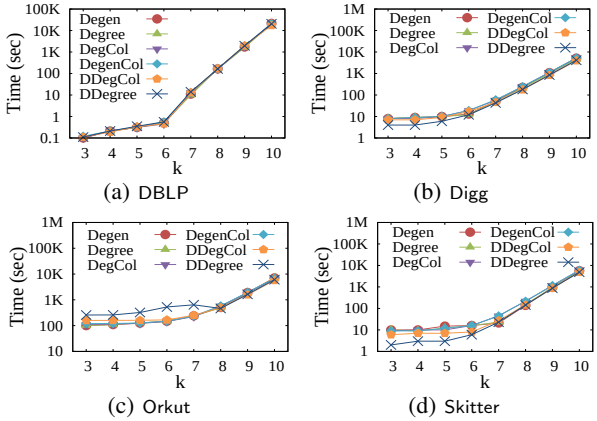
NodeParallel vs. EdgeParallel strategies. We start by comparing the performance of the NodeParallel algorithms and the EdgeParallel algorithms. Fig. 7 shows the results on Digg and Orkut for $k = 8$ and $k = 10$ respectively. Note that for all parallel algorithms, we are able to list all 10-cliques on all datasets in 8 hours. Again, the results on the other datasets and for the other values of k are consistent. From Fig. 7, we can see that the performance of all algorithms with the NodeParallel strategy is worse than that with the EdgeParallel strategy. This is because for all algorithms, the EdgeParallel strategy achieves a relatively larger degree of parallelism than the NodeParallel strategy. Additionally, we observe that DDegCol with the EdgeParallel strategy significantly outperforms the other parallel algorithms, which further demonstrates the superiority of the DDegCol algorithm.



(a) Digg ($k = 8$) (b) Digg ($k = 10$)

(c) Orkut ($k = 8$) (d) Orkut ($k = 10$)

Figure 7: NodeParallel **vs.** EdgeParallel **strategies**



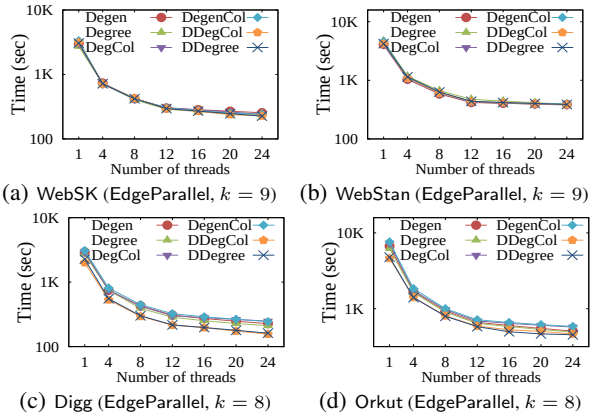
(a) DBLP (b) Digg

(c) Orkut (d) Skitter

Figure 8: Running time of various algorithms with the EdgeParallel **strategy (vary k)**

Evaluation of EdgeParallel **algorithms with varying k.** Fig. 8 plots the running time of different EdgeParallel algorithms as a function of k on the DBLP, Digg, Orkut, and Skitter datasets. Similar results can be obtained on the other datasets. As desired, the running time of each parallel algorithm increases as k increases. All algorithms achieve similar running times. For a large value of k, both the parallel DDegCol and DDegree algorithms slightly outperform the other competitors. For example, in Skitter, the running time of the parallel DDegCol and DDegree algorithms are 4,834 and 4,873 seconds respectively. In contrast, the running time achieved by the parallel Degree, Degen, DegCol, and DegenCol algorithms are slightly higher, which are 5,672, 5,155, 5,989, and 5,983 seconds, respectively. The results are consistent with what we observe in Fig. 7.

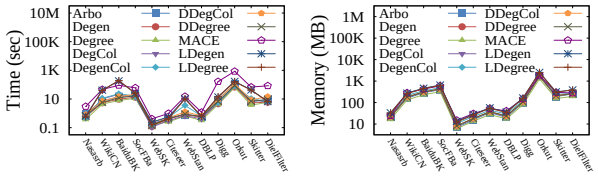
Speedup of different parallel algorithms. We evaluate the speedup of a variety of parallel algorithms, in which the speedup is defined as the running time of the sequential algorithm divided by the running time of the parallel algorithms when using t threads (t is varied from 1 to 24). Figs. 9(a-d) and Figs. 9(e-h) show the speedup of different NodeParallel and EdgeParallel algorithms, respectively. We can see that the running time of the EdgeParallel algorithms drops more quickly than those of the NodeParallel algorithms on the same dataset, indicating that the EdgeParallel strategy is better to balance the computational loads across the threads than the NodeParallel strategy. We note that the speedup of the NodeParallel algorithms becomes worse when the number of threads is larger than 12, which suggests that the NodeParallel strategy might be inappropriate for listing k-cliques in a massively parallel setting. In addition, we can also observe that both the parallel DDegCol and DDegree algorithms outperform the other parallel algorithms, which are consistent with our previous results.



(a) WebSK (EdgeParallel, $k = 9$) (b) WebStan (EdgeParallel, $k = 9$)

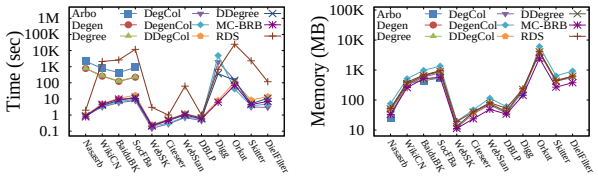
(c) Digg (EdgeParallel, $k = 8$) (d) Orkut (EdgeParallel, $k = 8$)

Figure 9: Speedup of different parallel algorithms



(a) Runtime (b) Memory usage

Figure 10: Comparison of triangle listing algorithms



(a) Runtime (b) Memory usage

Figure 11: Comparison of maximum clique search algorithms

In summary, the EdgeParallel strategy is significantly better than the NodeParallel strategy for all ordering-based algorithms. Generally, the former can achieve a good speedup on most datasets, while the latter is inappropriate for listing k-cliques in the massively parallel setting. The parallel variants of DDegCol and DDegree are slightly faster than the other algorithms, thus we recommend to use these parallel k-clique listing algorithms in practice.

4.6 Exact Algorithms for Special k

Comparison of triangle listing algorithms. We compare the performance of 10 different algorithms for triangle listing on 12 datasets (4 small-ω graphs and 8 large-ω graphs). The results are shown in Fig. 10. From Fig. 10(a), we can see that MACE performs significantly worse than all the other algorithms. In general, all the ordering-based algorithms have comparable running time. More subtly, we can observe that (1) LDegen and DegenCol seem to be slightly less efficient than the other ordering based algorithms; and (2) Degree is slightly faster than all the other algorithms on all datasets due to its simplicity and high efficiency. As shown in Fig. 10(b), the space overhead of all algorithms are comparable because all algorithms have the same space complexity. Again, we can see that Degree uses slightly less space than all the other competitors. As a result, we can conclude that Degree is the best algorithm for triangle listing in practice.

Comparison of maximum clique search algorithms. Fig. 11 shows the results of 9 maximum clique search algorithms on 12 datasets (4 small-ω graphs and 8 large-ω graphs). As shown in Fig. 11(a), Arbo, Degree, and Degen cannot work on all large-ω graphs, because they cannot list k-cliques for large k values. Generally, DegCol, DegenCol, DDegCol, DDegree, and MC-BRB achieve similar performance; and all of them perform much better than RDS on most datasets. More subtly, we can see that the performance of both MC-BRB and DDegCol is more robust than

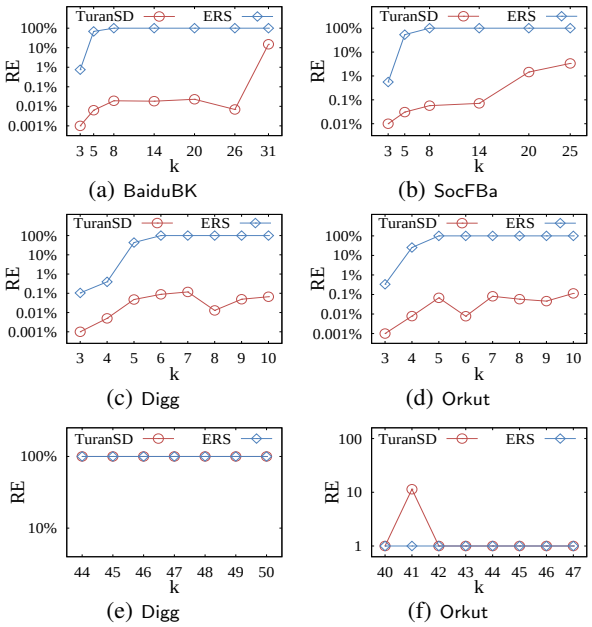


Figure 12: Relative error (RE) of approximation algorithms

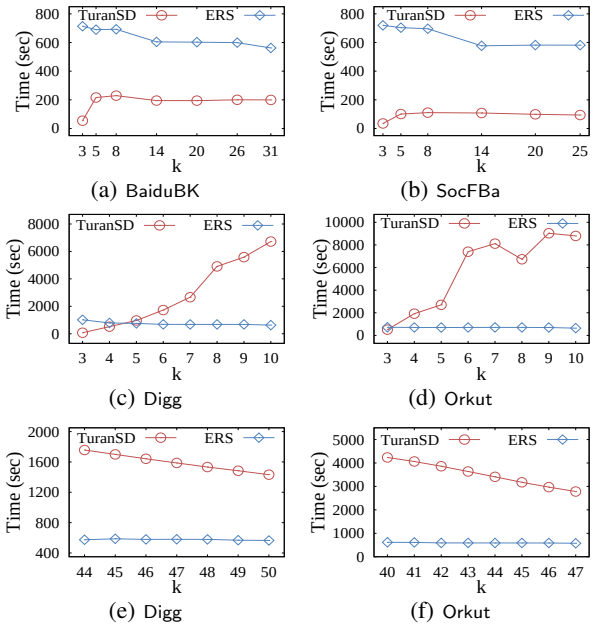


Figure 13: Runtime of approximation algorithms

those of the others over all datasets. For example, on Digg, both MC-BRB and DDegCol are at least one order of magnitude faster than DegCol, DegenCol, and DDegree. The memory usage of all algorithms are comparable as depicted in Fig. 11(b). We can also observe that MC-BRB consumes slightly less space than the other algorithms on most datasets. Based on these results, we can conclude that the overall performance of MC-BRB is the best among all the evaluated algorithms.

4.7 Evaluation of Approximation Algorithms

We evaluate two approximation algorithms (TuranSD and ERS) in terms of the relative errors (RE), running time, and space usage. The relative error is defined as $|N_k - \hat{N}_k|/N_k$, where N_k is the *true* k-clique count and $\hat{N}_k$ is an estimating count. To compute the relative error, we run each approximation algorithm 100 times and then take the average count over the 100 runs as the final estimating count (in each run, we set the sample size for each algorithm as suggested in its original paper).

Relative error. Fig. 12 shows the relative error of TuranSD and ERS on four datasets (2 small-ω and 2 large-ω graphs). Similar results can also be observed on the other datasets. From Fig. 12(a-d), we can see that TuranSD is pretty accurate in most cases, which has relative error lower than 1% for almost all k values on all datasets. ERS, however, performs very bad for $k \geq 4$ on all datasets. More interestingly, we observe that the relative error of TuranSD seems to increase as k increases on most datasets. Note that this observation was not explicitly revealed in the previous studies [20, 11]. The reason could be that the success probability of the sampling procedure in TuranSD (see line 19 of Algorithm 5) decreases when k increases, thus reducing the estimating precision of the TuranSD algorithm. Additionally, as shown in Fig. 12(e-f), both TuranSD and ERS perform extremely bad on all datasets when k is near to ω. Specifically, the relative errors of both TuranSD and ERS are no less than 1 which is clearly meaningless. These results indicate that (1) TuranSD is very accurate to estimate the k-clique count for small k values, but it might be unreliable for estimating the count of large k-cliques; (2) ERS only works well for $k = 3$ and it is unreliable for estimating k-clique counts when $k \geq 4$.

Running time. Fig. 13 shows the running time of TuranSD and ERS with varying k on 4 datasets. Note that previous studies in [20, 12, 11] did not evaluate how the parameter k affects the running time of TuranSD and ERS. As shown in Figs. 13, TuranSD is more efficient than ERS on small-ω graphs. On large-ω graphs, however, ERS is much faster than TuranSD, because the Turán-Shadow construction procedure is often very costly under this scenario. Additionally, from Figs. 13(c-f), we can see that the running time of TuranSD on large-ω graphs generally increases as k increases for small k, but for large k, the running time decreases when k increases. This is because for a small k (e.g., $k \leq 10$), the size of the Turán-Shadow on large-ω graphs generally increases as k increases, thus the time for constructing the Turán-Shadow increases. When k is near to ω, the Turán-Shadow size drops when k increases. It should be noted that the running time of TuranSD is much faster than all the exact k-clique listing algorithms on large-ω graphs. For example, on Digg, TuranSD takes 5,583 seconds when $k = 9$, while the best k-clique listing algorithm DDegCol consumes 10,031 seconds. These results indicate that TuranSD can quickly obtain good estimations of k-clique counts for small k values on large-ω graphs. However, for a relatively large k (e.g., $k \geq 10$), TuranSD is also very costly on large-ω graphs.

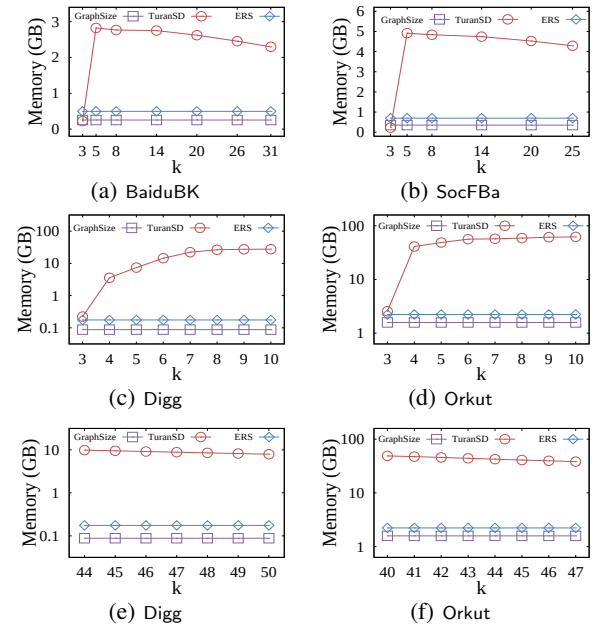


Figure 14: Memory usage of approximation algorithms

Table 4: Summary and recommendation (☆ stands for a half star; × means not applicable)

Methods	k-clique listing and counting				Triangle listing		Maximum clique search	
	Runtime	Memory	Parallelizability	Accuracy	Runtime	Memory	Runtime	Memory
Arbo	★★	★★★★	★	★★★★★	★★★★☆	★★★★☆	★	★★★★
MACE	★	★★★	★	★★★★★	★★★☆	★★★☆	★	★★★★
Degree	★★★	★★★★★	★★★★	★★★★★	★★★★★	★★★★★	★	★★★★
Degen	★★★	★★★★	★★★★	★★★★★	★★★★☆	★★★★☆	★	★★★★
DegCol	★★★☆	★★★★	★★★★	★★★★★	★★★★☆	★★★★☆	★★★★	★★★★
DegenCol	★★★☆	★★★★	★★★★	★★★★★	★★★★	★★★★☆	★★★★	★★★★
DDegCol	★★★★	★★★★	★★★★★	★★★★★	★★★★	★★★★☆	★★★★	★★★★
DDegree	★★★☆	★★★★	★★★★★	★★★★★	★★★★☆	★★★★☆	★★★★	★★★★
TuranSD	★★★★★	★★	★	★★★★	×	×	×	×
ERS	★★★★★	★★★★★	★★★	★	×	×	×	×
LDegree	×				★★★★☆	★★★★	×	×
LDegen	×				★★★★	★★★★	×	×
RDS	×				×	×	★★★	★★★★
MC-BRB	×				×	×	★★★★★	★★★★★

Memory usage. Fig. 14 shows the memory overheads of TuranSD and ERS as a function of k on 4 datasets. Note that the effect of the parameter k for the memory overheads of TuranSD and ERS was also not systematically studied in [20, 12, 11]. As can be observed in Figs. 14, the space usage of TuranSD is significantly larger than that of ERS for almost all k values on all datasets. More specifically, on the small-ω graphs, TuranSD consumes around $2-5$ times more space than ERS. On the large-ω graphs, however, the space usage of TuranSD is around two orders of magnitude larger than that of ERS. In addition, we can see that on large-ω graphs, the space overhead of TuranSD increases as k increases from 3 to 10. These results indicate that for a relatively large k, the space consumption of TuranSD is very high on large-ω graphs, which may prevent it to handle real-world large-ω graphs.

5. SUMMARY AND RECOMMENDATION

Summary. In this paper, we studied 14 sate-of-the-art algorithms for k-clique listing and counting (10 algorithms for general k values and 4 algorithms for special k values), developed new color ordering heuristics to further improve the performance of the existing ordering-based algorithms, and conducted an extensive experimental evaluation.

For efficiency, DDegCol is the fastest algorithm among all exact k-clique listing algorithms for general k values. Overall, the running times of all the other ordering-based algorithms (Degree, Degen, DegCol, DegenCol, and DDegree) are comparable. Both Arbo and MACE are significantly less efficient than the ordering-based algorithms. The two approximation algorithms (TuranSD and ERS) are generally much faster than all the exact algorithms, but both of them cannot obtain the exact number of k-cliques. For triangle listing, Degree outperforms all the other competitors due to its simplicity and high efficiency. Arbo and all the ordering-based algorithms exhibit similar efficiency. Again, MACE is slower than all the other algorithms for triangle listing. For maximum clique search, MC-BRB is slightly faster than all the other methods. In general, DegCol, DegenCol, DDegCol, DDegree and RDS achieve comparable running time, and Arbo, Degree, and Degen perform very bad for maximum clique search.

For memory overheads, ERS and Degree consume the least amount of memory among all k-clique listing algorithms for general k values. All the other exact algorithms uses comparable space, as they have the same space complexity. TuranSD, however, consumes much more space than all the other approaches, since it needs to store a Turán-Shadow structure which is significantly larger than the graph size. For triangle counting, the memory usage of Degree is also slightly less than all the other competitors. Again, all the other algorithms use similar space. For maximum clique search, the space usage of MC-BRB is the lowest compared to all the other algorithms; and for all the other algorithms, they consume similar space.

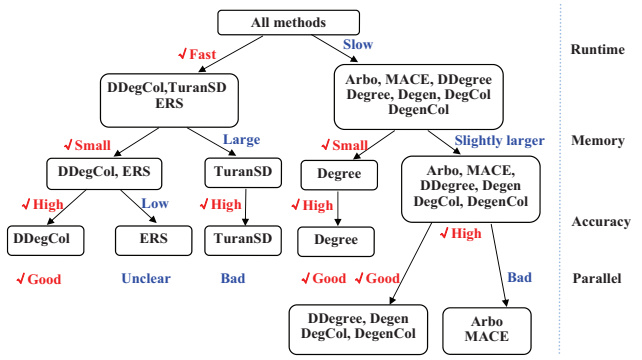


Figure 15: The decision tree for choosing a appropriate k-clique listing algorithm under different scenarios

The parallelizability ranking for k-listing algorithms is: DDegCol $\approx$ DDegree $>$ Degree $\approx$ Degen $\approx$ DegCol $\approx$ DegenCol $>$ ERS $>$ Arbo $\approx$ MACE $\approx$ TuranSD.

For accuracy, ERS performs very bad for estimating the k-clique count given that $k \geq 3$. TuranSD is pretty accurate for estimating the number of k-cliques with a small k, but this comes at the cost of considerable space overhead.

Recommendation. In Table 4, we summarize our recommendation levels for all algorithms based on 4 different metrics. The scale is from 1 to 5 stars, and the larger star number means higher ranking. For general k-clique listing algorithms, there does not exist a single winner. By considering different trad-offs, we recommend DDegCol for k-clique listing, since it exhibits good performance in running time, memory usage, parallelizability and accuracy. For triangle listing, we recommend Degree, as it is a clear winner in both running time and memory usage. Similarly, for maximum clique search, MC-BRB is the best algorithm compared to all the other methods in both running time and memory overhead.

The decision tree shown in Fig. 15 illustrates our suggested approach to select a proper algorithm for general k-clique listing under different constraints. Following the branch with red tick on the decision tree, the better algorithm(s) under the current constraint can be obtained. Clearly, we can see that the path from the root to the leaf of DDegCol contains all red ticks which indicates that DDegCol achieves the best trade-off capacity by considering various factors among all algorithms.

Acknowledgement. This work was partially supported by (i) NSFC Grants 61772346, U1809206, 61732003, 11671296; (ii) National Key R&D Program of China 2018YFB1004402; (iii) Beijing Institute of Technology Research Fund Program for Young Scholars; (iv) Research Grants Council of the Hong Kong SAR, China No. 14202919 and 14203618; (v) ARC FT200100787. Guoren Wang is the corresponding author of this paper.

6. REFERENCES

[1] https://github.com/gawssin/kcliquelisting.

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

[3] L. Barenboim and M. Elkin. Sublogarithmic distributed MIS algorithm for sparse graphs using nash-williams decomposition. In *PODC*, pages 25–34, 2008.

[4] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[5] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*, 2008.

[6] A. R. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295), 2016.

[7] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[8] L. Chang. Efficient maximum clique computation over large sparse graphs. In *KDD*, 2019.

[9] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[10] S. Chu and J. Cheng. Triangle listing in massive networks and its applications. In *KDD*, 2011.

[11] M. Danisch, O. D. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *WWW*, 2018.

[12] T. Eden, D. Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *STOC*, 2018.

[13] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013.

[14] D. Eppstein and E. S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *J. Graph Algorithms Appl.*, 16(2):543–567, 2012.

[15] I. Finocchi, M. Finocchi, and E. G. Fusco. Clique counting in mapreduce: Algorithms and experiments. *ACM Journal of Experimental Algorithmics*, 20:1.7:1–1.7:20, 2015.

[16] H. N. Gabow and H. H. Westermann. Forests, frames, and games: Algorithms for matroid sums and applications. *Algorithmica*, 7(5&6):465–497, 1992.

[17] E. Gregori, L. Lenzini, and S. Mainardi. Parallel (k)-clique community detection on large-scale networks. *IEEE Trans. Parallel Distrib. Syst.*, 24(8):1651–1660, 2013.

[18] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson. Ordering heuristics for parallel graph coloring. In *SPAA*, 2014.

[19] X. Hu, Y. Tao, and C. Chung. I/O-efficient algorithms on triangle listing and counting. *ACM Trans. Database Syst.*, 39(4):27:1–27:30, 2014.

[20] S. Jain and C. Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *WWW*, 2017.

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

[22] T. G. Kolda, A. Pinar, T. D. Plantenga, C. Seshadhri, and C. Task. Counting triangles in massive graphs with mapreduce. *SIAM J. Scientific Computing*, 36(5), 2014.

[23] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theor. Comput. Sci.*, 407(1-3):458–473, 2008.

[24] R. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In *SIGMOD*, 2018.

[25] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5):509–520, 2015.

[26] R.-H. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2453–2465, 2014.

[27] M. C. Lin, F. J. Soulignac, and J. L. Szwarcfiter. Arboricity, h-index, and dynamic algorithms. *Theor. Comput. Sci.*, 426:75–90, 2012.

[28] C. Lu, J. X. Yu, H. Wei, and Y. Zhang. Finding the maximum clique in massive graphs. *PVLDB*, 10(11):1538–1549, 2017.

[29] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. In *9th Scandinavian Workshop on Algorithm Theory*, 2004.

[30] M. Mitzenmacher, J. Pachocki, R. Peng, C. E. Tsourakakis, and S. C. Xu. Scalable large near-clique detection in large-scale networks via sampling. In *KDD*, 2015.

[31] C. S. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39(1):12–12, 1964.

[32] M. Ortmann and U. Brandes. Triangle listing algorithms: Back from the diversion. In *ALENEX*, 2014.

[33] P. R. J. Östergård. A fast algorithm for the maximum clique problem. *Discret. Appl. Math.*, 120(1-3):197–207, 2002.

[34] G. Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043), 2005.

[35] B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W. Liao, and A. N. Choudhary. Fast algorithms for the maximum clique problem on massive graphs with applications to overlapping community detection. *Internet Math.*, 11(4-5):421–448, 2015.

[36] L. Qin, R. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In *KDD*, 2015.

[37] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Çatalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, 2015.

[38] T. Schank. *Algorithmic Aspects of Triangle-Based Network Analysis*. PhD thesis, Universitat Karlsruhe (TH), 2007.

[39] P. S. Segundo, A. Lopez, and P. M. Pardalos. A new exact maximum clique algorithm for large and massive sparse graphs. *Comput. Oper. Res.*, 66:81–94, 2016.

[40] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983.

[41] J. Shi, L. Dhulipala, and J. Shun. Parallel clique counting and peeling algorithms. *CoRR*, abs/2002.10047, 2020.

[42] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, 2011.

[43] E. Tomita. Efficient algorithms for finding maximum and maximal cliques and their applications. In *WALCOM*, 2017.

[44] E. Tomita and T. Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *J. Global Optimization*, 44(2):311, 2009.

[45] E. Tomita, Y. Sutani, T. Higashi, S. Takahashi, and M. Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *WALCOM*, 2010.

[46] C. E. Tsourakakis. The k-clique densest subgraph problem. In *WWW*, 2015.

[47] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *KDD*, 2009.

[48] P. Turan. On an extremal problem in graph theory. *Mat. Fiz. Lapok*, 48(137):436–452, 1941.

[49] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Effective and efficient dynamic graph coloring. *PVLDB*, 11(3):338–351, 2017.