Parallel Algorithms for Finding Large Cliques in Sparse Graphs

Lukas Gianinazzi[1], Maciej Besta[1], Yannick Schaffner[2], Torsten Hoefler[1]

[1]Department of Computer Science, ETH Zurich; [2]Department of Mathematics, ETH Zurich

ABSTRACT

We present a parallel k-clique listing algorithm with improved work bounds (for the same depth) in sparse graphs with low degeneracy or arboricity. We achieve this by introducing and analyzing a new pruning criterion for a backtracking search. Our algorithm has better asymptotic performance, especially for larger cliques (when k is not constant), where we avoid the straightforwardly exponential runtime growth with respect to the clique size. In particular, for cliques that are a constant factor smaller than the graph's degeneracy, the work improvement is an exponential factor in the clique size compared to previous results. Moreover, we present a low-depth approximation to the community degeneracy (which can be arbitrarily smaller than the degeneracy). This approximation enables a low depth clique listing algorithm whose runtime is parameterized by the community degeneracy.

CCS CONCEPTS

• **Theory of computation → Parallel algorithms**.

KEYWORDS

parallel graph algorithms; clique listing; arboricity; degeneracy;

1 INTRODUCTION

Finding large cliques has many applications in the social sciences, bioinformatics, computational chemistry, and others [1, 8, 12, 23, 33, 34, 43, 48, 51, 52]. As the problem is NP-hard and remains hard even when parameterized by the size of the clique k [28], it makes sense to consider special families of graphs for which the problem is tractable. If the work of an algorithm can be written as $O\left(f(P)N^c\right)$ (for some function f of the *parameters* P and some polynomial N^c of the input size N), the problem is *fixed parameter tractable* (FPT) with respect to the parameters P [27]. The clique problem is FPT with respect to certain *structural sparsity parameters* of graphs.

As a classic example family of sparse graphs, planar graphs do not contain cliques with more than 4 vertices, and all cliques in a planar graph can be listed in linear time in the size of the graph [42]. Planar graphs are members of a more general class of sparse graphs for which every subgraph has a low-degree vertex. This notion of sparsity is called *degeneracy*. A closely related family of sparse graphs is defined by being decomposable into a small number of forests. The *arboricity* measures the number of such forests required for a given graph. Many real-world graphs are sparse [5, 6, 9–11, 14] and have a low degeneracy and arboricity [4, 7, 25, 49, 50].

In this work, we focus on improving the work and depth of FPT algorithms to list all k-cliques in these sparse graphs, improving both work and depth significantly as a function of the clique size k. Our approach utilizes the notion of *edge communities* - vertices with whom an edge forms a triangle. We further introduce a notion of *relevant edges*, that effectively upper bounds the community's size. This enables us to reject edges that cannot be a part of a k-clique,

pruning unnecessary recursive calls. We show that this pruning criterion asymptotically reduces the work compared to previous approaches.

1.1 Preliminaries

We consider a (directed or undirected) graph $G = (V, E)$ with n vertices V and m edges E. We will generally assume the graph is connected and therefore $m = \Omega(n)$. To orient a graph *by a total order*, direct its edges from the endpoint lower in the total order to the endpoint higher in the total order. A graph that has been oriented by a total order is acyclic by construction. The subgraph of a graph G induced by a vertex set V' is $G[V']$. The *neighbors* of a vertex u in the graph G are the set of all vertices in V that are connected to u by edges in E. In a directed graph, the *out-neighbors* are $N_G^+(u) = \{v \in V \mid (u,v) \in E\}$, and the *in-neighbors* are $N_G^-(v) = \{u \in V \mid (u,v) \in E\}$. If the graph G is clear from the context, we write $N(u)$, $N^+(u)$, and $N^-(u)$ for short.

In an undirected graph, the *community* $C_G(u,v) = C_G(e)$ of an edge $e = \{u,v\}$ is the intersection of the neighbors of its endpoints. In a directed graph, the *community* $C_G(u,v) = C_G(e)$ of an edge $e = (u,v)$ is the intersection of the *out*-neighbors of u and the *in*-neighbors of v. If the graph G is clear from the context, we write $C(u,v)$ and $C(e)$ for short.

Sparse Graphs. A graph $G = (V,E)$ is s-*degenerate*, if in all induced subgraphs H of G, there is a vertex with degree at most s [37]. The *degeneracy* of a graph is the smallest s such that the graph is s-degenerate. Note that an s-degenerate graph can have an unbounded maximum degree. For example, the star graph is 1-degenerate but has a maximum degree of $n - 1$. An s-degenerate graph can be oriented such that every vertex has at most s out-neighbors (greedily remove a vertex with the smallest degree in the remaining subgraph). The order in which this greedy procedure removes vertices is a *degeneracy order*.

A graph $G = (V, E)$ is σ-*community degenerate*, if every (non-edgeless) subgraph G' has an edge e with $|C_{G'}(e)| \le \sigma$. The *community degeneracy* of a graph is the smallest value σ, such that the graph is σ-community degenerate [18]. The community degeneracy is strictly smaller than the degeneracy: $\sigma < s$ and there are families of graphs where the community degeneracy is asymptotically smaller. For example, the d-dimensional hypercube has degeneracy $s = d$, but community degeneracy $\sigma = 0$. Moreover, the community degeneracy can be 1 for graphs where the degeneracy is $\Theta(n)$ and there are $\Theta(n)$ triangles. This is the case for a graph which has the complete bipartite graph as a subgraph, with $n/2$ vertices in each part, and additionally one part of the graph induces a line graph on $n/2$ vertices. Buchanan et al. [18] also show empirically that the community degeneracy is significantly smaller (27%-80%) than the degeneracy for some real-world graphs (when the degeneracy is significantly larger than the clique number).

The smallest number of forests into which a graph can be decomposed (meaning that every edge is in exactly one of the forests) is the *arboricity* α of a graph. It is closely related to the degeneracy, in particular $\alpha \leq s < 2\alpha$ [41].

k-Cliques. An induced subgraph of G that is a complete graph with k-vertices is a k-clique. It follows from the definition that an s-degenerate graph does not contain any $(s + 2)$-clique and a graph with arboricity α does not contain any $(2\alpha + 1)$-clique (using $\alpha > s/2$). In this paper, we generally assume $k \geq 4$.

Deciding the size of a maximum clique is a classic NP-hard problem [31] and so is deciding if the graph has a k-clique. The problem of finding a clique of size k remains hard when parameterized by k: it is W[1]-hard [28]. However, the problem becomes FPT for graphs when parameterized by degeneracy or arboricity [21].

Model of Computation. Our execution model is a shared-memory parallel computer with concurrent reads and exclusive writes (CREW PRAM) [44]. We express the performance of our algorithms in the *work/depth model*. The *work* is the total number of elementary operations performed by all processors in any algorithm execution. The *depth* is the length of a critical path. An algorithm with work W and depth D can be scheduled on a CREW PRAM with p processors so that it takes $O(W/p + D)$ time steps [15, 44].

1.2 Related Work

Chiba and Nishizeki [21] presented a result on k-clique listing for low arboricity graphs. Their algorithm takes $O(m\alpha^{k-2})$ work. Danish et al. [25] gave improved bounds in terms of the degeneracy. Their algorithm does $O(m(\frac{s}{2})^{k-2})$ work. Since $\alpha \leq s < 2\alpha$, this is never worse than Chiba and Nishizeki's work bound, but can be faster by a term exponential in k for graphs where the degeneracy is close to the arboricity. The algorithm has a depth of $O(n)$, stemming from how the graph is directed with a degeneracy order. Recently, Shi et al. [49] gave a variant on the algorithm that uses a fast parallel approximation to the degeneracy order (in addition to providing some improvements in the data structure used to represent the graph during the recursive search). When k is constant (which they assume), this approximation does not impact the runtime. However, the work increases by a factor exponential in k because of this approximation. These k-clique listing algorithms are not well suited to finding very large cliques, as their bounds become *super-exponential* in s (and α) when the size of the cliques is $k = \Omega(s)$. This is far from optimal, as a graph with degeneracy s has at most $(n - s + 1)2^s$ cliques overall [55].

Previous k-clique listing algorithms [21, 25, 49] use a backtracking scheme combined with a clique-growing pattern, where a subgraph is expanded into larger and larger cliques, until a k-clique is found or determined to be unreachable. A common optimization [25, 49] is to *orient* the input graph. In k-clique counting, one usually orients a graph with a total order on the vertices [25, 49]. A popular choice of order is the degeneracy-order as it assigns low out degrees and can be computed in linear time [38]. Other works on k-cliques focus on reordering heuristics [36], using GPUs [2].

Listing *maximal* cliques is also a subject of numerous works [16, 19, 20, 40, 47]. For example, Eppstein [29] presents a variant of Bron-Kerbosch's algorithm [17, 53] to find *maximal* cliques. It takes
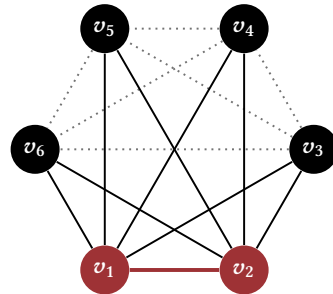

Figure 1: To support a k-clique, an edge must have $k - 2$ triangles that contain it. In other words, its community must have a size of $k - 2$. In the example, the community of the edge $\{v_1, v_2\}$ contains all the other vertices $\{v_3, v_4, v_5, v_6\}$. Hence, it could potentially support a 6-clique. Indeed, the edge $\{v_1, v_2\}$ does support a 6-clique.

$O\left(sn3^{s/3}\right)$ time, which is close to the $(n - s)3^{s/3}$ lower bound [29] on the number of maximal cliques in an s-degenerate graph. A *maximum* clique can be computed in $O\left(2^{0.276n}\right)$ time from the relationship to maximum independent sets [45].

Many other variants of clique listing exist, such as diversified top-k cliques [56], the k-clique densest subgraph problem [39, 54], and densest subgraph discovery [30].

Converting a graph to a directed, acyclic graph gives an effective way to assign cliques to certain vertices: A clique can e.g., be associated with its topologically smallest or largest vertex. This way, double counting can be avoided and the directions reduce the size of the search space.

1.3 Our Contributions

We present an algorithm that provides, for the same depth, improved work bounds for k-clique counting in s-degenerate graphs. In particular, when $k = \Omega(s)$, the improvement grows exponentially with k. Considering the case where k is not constant is important because that is when the problem becomes hard. See Table 1 for a comparison of our bounds compared to previous k-clique listing results in sparse graphs. Our algorithm differs in two relevant aspects from previous approaches [21, 25, 49].

First, existing approaches use the degree of a vertex to decide whether the neighborhood of a vertex *can* contain a smaller clique. In contrast, our algorithm looks at edges and their *triangles* to grow the clique: Each edge of a k-clique participates in (at least) $k - 2$ triangles. See Figure 1 for an example. Our algorithm preprocesses the graph such that each triangle (a, b, c) is *supported* by exactly one of its edges. If edge (a, c) supports triangle (a, b, c), we say that vertex b is in the *community* of edge (a, c). To find a k-clique, we only need to consider edges that support at least $k - 2$ triangles. We add this edge to the clique and recursively search for a $(k-2)$-clique in the subgraph induced by the community of that edge.

Second, we observe that we can use the number of vertices ordered between the endpoints of an edge as a proxy for the number of triangles it supports, as illustrated in Figure 2. This observation allows us to exclude edges from the search space because they cannot support a k-clique, without actually looking at their triangles in every step. Moreover, this simplification enables us to upper
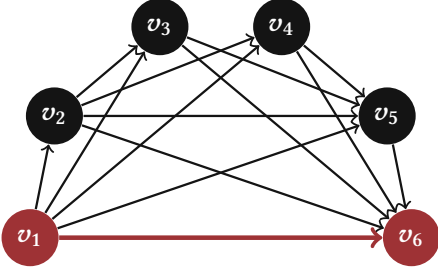
Figure 2: If the graph is oriented by a total order, we can use the number of vertices ordered between the endpoints of an edge as a proxy to determine if it can support a k-clique. If there are less than $k - 2$ vertices ordered between the endpoints, the edge cannot support a k-clique. In the example, only the edge (v_1, v_6) could support a 6-clique using this pruning rule. Therefore, we call the edge *relevant*. However, the graph only contains two 5-cliques and no 6-clique because there is no edge (v_3, v_4).

bound our algorithm's work, improving on the state-of-the-art for k-clique counting in s-degenerate graphs.

Pruning edges by excluding those which do not have enough vertices ordered between their endpoints is what is responsible for the $\Theta\left(\left(\frac{1}{1-k/s}\right)^k\right)$ factor improvement in the work.

We also discuss different ways to orient and preprocess the graph. These graph orientations leads to three variants with different work/depth tradeoffs, as seen in Table 1.

Finally, we show clique counting results parameterized by the community degeneracy. Our algorithm uses a total order *on the edges* in addition to a total order on the vertices to get better results for graphs where $\sigma < s - 1$. We again get three variants with different work/depth tradeoffs. To achieve the sublinear-depth variants, we present a novel low-depth approximation algorithm for the community degeneracy.

2 COMMUNITY-CENTRIC CLIQUE LISTING

We begin with an informal description of the ideas behind our *community-centric clique listing* algorithm. Afterward, we state the algorithm and bound its work and depth. Let us start with a simple observation about k-cliques, on which we will gradually expand. *Any induced subgraph on a clique is a clique as well.* A slight reformulation of this observation is the primary building block of the algorithm by Chiba and Nishizeki [21], and subsequently, of Danisch et al. [25] and Shi et al. [49]: If a vertex's neighborhood contains a $(k-1)$-clique, then we have found a k-clique [21]. This insight allows a simple backtracking formulation, where one recursively searches for smaller cliques in the local neighborhood of vertices. In the practical implementations [21, 25, 49], this formulation gives rise to a form of candidate growing. In each recursive call, the current candidate motif grows by one vertex to a larger clique. During the backtracking, the candidate motif shrinks again.

Observe that one is not limited to grow a motif by a single vertex in each step, but can extend the motif by any kind of clique in its neighborhood, such as an edge (i.e., a 2-clique). Whereas each

	Work	Depth
Chiba/Nishizeki [21]	$O\left(m\alpha^{k-2}\right)$	$O\left(m\alpha^{k-2}\right)$
Danish et al. [25]	$O\left(km\left(\frac{s}{2}\right)^{k-2}\right)$	$O\left(n + \log^2 n\right)$
Shi et al. [49]	$O\left(m(s(1+\varepsilon))^{k-2}\right)^{\dagger}$	$O\left(k \log n + \log^2 n\right)^{\star}$
Our Results for Degeneracy §2:		
Best Work §4.1	$O\left(km\left(\frac{s+3-k}{2}\right)^{k-2}\right)$	$O\left(n + k \log n\right)$
Hybrid §4.2	$O\left(kns\left(\frac{s+3-k}{2}\right)^{k-2}\right)$	$O\left(s + k \log n + \log^2 n\right)$
Best Depth §4.1	$O\left(km\left(\frac{s(2+\varepsilon)+3-k}{2}\right)^{k-2}\right)$	$O\left(k \log n + \log^2 n\right)$
Our Results for Community Degeneracy §2:		
Best Work §4.3	$O\left(ms + km\left(\frac{\sigma+4-k}{2}\right)^{k-2}\right)$	$O\left(n + k \log n\right)$
Hybrid §t4.3	$O\left(ms + kn\sigma\left(\frac{\sigma+4-k}{2}\right)^{k-2}\right)$	$O\left(\sigma + k \log n + \log^2 n\right)$
Best Depth §4.3	$O\left(ms + km\left(\frac{(3+\varepsilon)\sigma+4-k}{2}\right)^{k-2}\right)$	$O\left(k \log n + \log^2 n\right)$

Table 1: Bounds for listing all k-cliques in a graph ($k \geq 4$) with degeneracy s, community degeneracy σ, and arboricity α. The parameter ε is some positive real *constant*. Recall that $\alpha \leq s < 2\alpha$ and $k \leq \sigma + 2 \leq s + 1$. Bounds marked with † hold in expectation and those marked with * hold with high probability in n.

vertex has a fixed number of neighbors, each edge belongs to a fixed number of triangles. In a k-clique, each edge participates in $k - 2$ triangles. If the clique is oriented by a total order, we can refine the statement. The *supporting edge* of a k-clique in a graph oriented by a total order is the edge connecting the first and last vertex in the total order. It holds that:

OBSERVATION 1. *The supporting edge e of a k-clique oriented by a total order, has a community $C(e)$ of size at least $k - 2$. All other edges have a smaller community.*

In Section 2.2, we show the following bounds for our algorithm:

THEOREM 2.1. *Let $G = (V, E)$ be a directed acyclic graph oriented by a total order, such that its largest community has size γ and its largest out-degree is $\tilde{s}$. Then, the work performed by Algorithm 1 is*

$$O\left(m(\tilde{s} + \gamma^2) + km\left(\frac{\gamma + 4 - k}{2}\right)^{k-2}\right),$$

and its depth is

$$O\left(\log^2 n + k \log \gamma\right).$$

In Section 4, we discuss different approaches to to orient a graph, leading to the work/depth tradeoffs in Table 1.

2.1 Algorithm Construction

Algorithms 1 and 2 summarize our clique listing algorithm. Input to Algorithm 1 are a graph DAG oriented by a total order and an integer

$k > 3$. First, construct the edge communities and sort them (this speeds up the later intersection operations). Second, loop over all edges and recurse on their communities (line 3) with the recursive clique counting Algorithm 2.

Algorithm 1 Listing all k-cliques in the oriented graph DAG

1: Build the communities and sort them
2: **for all** edges e with at least $k - 2$ triangles **do in parallel**
3:     RECURSIVECOUNT(DAG, $C(e), k - 2$)
4: **end for**

The recursive procedure is shown in Algorithm 2. As input, it takes a directed graph DAG, a set of candidate vertices I and a constant $c > 0$, stating the number of vertices still required to complete a k-clique. The recursion has two base cases: for $c = 1$ (line 2), where each vertex in the candidate set completes the partial motif to a k-clique and for $c = 2$ (line 4), where each connected pair of vertices completes the k-clique.

Given a set of vertices I with a total order, the *distance function* $\delta_I : I \times I \mapsto \{0, \dots, |I - 2|\}$ maps each pair of vertices (u, v) to the number of elements in V that are ordered between u and v in the total order. We keep the set I in a sorted array. Then, for any two candidates u and v, we compute $\delta_I(u, v)$ from their indices in I.

In the recursive case, iterate over all vertex pairs (u, v), whose distance $\delta_I(u, v)$ to each other, given the total order and the candidate set I, allows them to support a k-clique (line 6). For each such pair, probe the existence of the corresponding edge (line 7) and, if successful, intersect the candidate set with the community of the edge (line 8) to create a new candidate set I'. Then, make the next recursive call with the new candidate set I', requiring $c - 2$ vertices to complete a k-clique (line 9).

See Figure 3 for an example of an execution of Algorithm 1.

**Algorithm 2** Recursively search for cliques of size c in DAG$[I]$.

1: **if** $c = 1$ **then**
2:     For every vertex v in I, **return** a clique.
3: **else if** $c = 2$ **then**
4:     For every edge (u, v) in $G[I]$, **return** a clique.
5: **end if**
6: **for all** pairs $e = (u, v) \in I \times I$ s.t. $\delta_I(u, v) \geq c - 2$ **do**
7:     **if** e is an edge in DAG **then**
8:         $I' \leftarrow I \cap C(e)$
9:         RECURSIVECOUNT(DAG, $I', c - 2$)
10:     **end if**
11: **end for**

2.2 Work/Depth Analysis

Let us first bound the cost of the preprocessing in Algorithm 1. We can compute the triangles of a graph in $O(m\tilde{s})$ work and $O(\log^2 n)$ depth [49], which gives us the communities. Sorting the communities takes $O(m\gamma \log \gamma)$ work and $O(\log \gamma)$ depth [22]. We build for each edge e an adjacency matrix of $G[C(e)]$ and rename the vertices to be consecutive integers. Next, for each edge e' in such a subgraph $G[C(e)]$, we build a boolean indicator table for the vertices in its
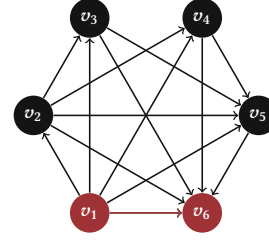
(a) Loop over all edges that can support a 4-clique, i.e., 4 triangles, and call Algorithm 2. Only edge (v_1, v_6) has a community of size at least 4. For the call to Algorithm 2, $I = \{v_2, v_3, v_4, v_5\}$ and $c = 4$.
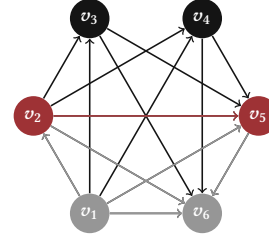
(b) Iterate over all pairs in the candidate set with a distance of 2 that form an edge. Only the edge (v_2, v_5) has a distance of 2.
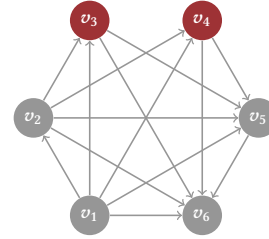
(c) Among the remaining candidates, there is only one pair: (v_3, v_4). The edge between the pair does not exist. Hence algorithm 2 aborts and reports that there is no 6-clique.

Figure 3: Example of using Algorithm 1 to search (in vain) for a 6-clique in the above graph. Black vertices belong to the candidate set for the next call to Algorithm 2, red vertices and edges are being considered for extending the clique, and grey vertices and edges have been already visited.
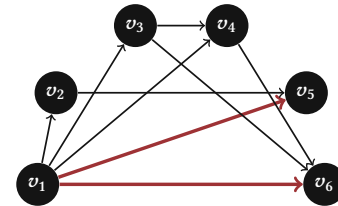

Figure 4: In the example graph, the edges relevant with respect to 3 are $\mathcal{R}_3^E(G) = \{(v_1, v_5), (v_1, v_6)\}$ (marked red). These are the edges that could potentially support a 5-clique. The pairs $\mathcal{R}_3^P(G)$ relevant with respect to 3 also include (v_2, v_6).

"local" community $C_{G[C(e)]}(e')$. This preprocessing speeds up the intersections and edge probing. It costs $O(m\gamma^2)$ work.

Before we bound the work and depth of the recursive Algorithm 2, we need some additional notation. Given a set of vertices V, a pair (u, v) of vertices is *relevant* with respect to $c \in \mathbb{N}_0$, if and only if $\delta_V(u, v) \geq c$, meaning that there are at least c vertices ordered between u and v. The set of *relevant pairs* of V with respect to c and the total order of V is $\mathcal{R}_c^P(V)$. An edge $e = (u, v)$ is relevant with respect to c, if it forms a relevant pair with respect to c. Note that an edge that is not relevant with respect to $k - 2$ cannot support a k-clique and that Algorithm 2 only recurses on the relevant edges (with respect to $c - 2$). The set of relevant edges with respect to c in the graph $G = (V, E)$ is $\mathcal{R}_c^E(G)$. See Figure 4 for an illustration of the relevant pairs and edges.

Next, let us bound the work and depth of Algorithm 2. For each relevant edge e, computing the intersection of I with $C(e)$ takes $O(\log \gamma)$ depth and $O(|C(e)| + |I|)$ work: For each element in I use the indicator table for $C(e)$ to test if it is in $C(e)$. Perform a parallel prefix sum [44] to gather the elements in the intersection. The work for edge probing is $O\left(\mathcal{R}_{c-2}^P(I)\right)$ and the depth is $O(1)$. At the base cases, the work is $O(k)$ per listed clique, and the depth is $O(1)$.

The recursion in Algorithm 2 has depth $\lfloor \frac{k-2}{2} \rfloor$. Hence, the critical path has length $O(\log n + k + k \log(\gamma + 1))$. As long as $\gamma \neq 0$, the additive k term is negligible. Next, we express the work of Algorithm 2 recursively.

Let $W(c, I)$ be the work of Algorithm 2 for candidate set I and parameter c. We express the work recursively as follows:

$$W(1, I) \leq O(k \ |I|)$$

$$W(2, I) \leq O\left(|\mathcal{R}_0^P(I)| + k \ |\mathcal{R}_0^E(G[I])|\right)$$

$$W(c, I) \leq O\left(|\mathcal{R}_{c-2}^P(I)| + \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} |C(e)| + |I| + W(c - 2, I \cap C(e))\right)$$

By telescoping this formula, we can see that the key to tackling this recursion is a bound on the following sum:

$$\sum_{e \in \mathcal{R}_c^E(G)} |\mathcal{R}_{c-2}^E(G[C(e)])| \ .$$

The sum corresponds to the loop in the algorithm that iterates over the relevant edges. For each edge e, the term $|\mathcal{R}_{c-2}^E(G[C(e)])|$ is a bound on the number of relevant edges in the subgraph that the algorithm recurses on. In Section 3, we establish the following bounds on this quantity.

Lemma 2.2. *For a graph $G = (V, E)$ oriented by a total order and any $c \geq 2$, we have that:*

$$\sum_{e \in \mathcal{R}_c^E(G)} |\mathcal{R}_{c-2}^E(G[C(e)])| \ \leq \ \sum_{e \in \mathcal{R}_c^E(G)} |\mathcal{R}_{c-2}^P(C(e))|$$

$$\leq \left(\frac{n - c}{2}\right)^2 |\mathcal{R}_c^E(G)|.$$

We focus on bounding the total cost incurred for listing the cliques at the leaves of the recursion for $c \geq 2$. The proofs bounding the remaining cost (associated with the computation of intersections and edge testing) are similar and can be found in Appendix A.1.

Observation 2. *The number of non-trivial recursive calls of Algorithm 2 (i.e., excluding the base case $c = 1$) is given by the function $r(c) = \lfloor \frac{c}{2} \rfloor$. Moreover, note that $r(c) + 1 = r(c + 2)$.*

For some constant a, we define the following function to help unifying the analysis when c is even or odd:

$$b(c, i) \leq \begin{cases} a \cdot k & \text{if } c \text{ is even} \\ a \cdot k \cdot i & \text{else.} \end{cases}$$

The constant a is chosen such that the work to list a k-clique is at most ak and the cost of listing $i = |I|$ k-cliques is at most $a \cdot k \cdot i$.

Lemma 2.3. *Let $L(c, I)$ be the work of the algorithm incurred at the leaves of the recursion in Algorithm 2. Then, we have for any $c \geq 2$,*

$$L(c, I) \leq \left(\frac{|I| - c + 2}{2}\right)^{2r(c-2)} |\mathcal{R}_{c-2}^E(G[I])| \ b(c, |I| - 2r(c)) \ .$$

Proof. The proof is by induction on c. The base cases $c = 2$ and $c = 3$ are quickly verified. Assume now, that the inequality on $L(c', I)$ holds for all $c' \leq c$. We show that it also holds for $c + 2$.

$$L(c, I)$$

$$\leq \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} L(c - 2, I \cap C(e))$$

$$\leq \sum_{e \in \mathcal{R}_{c-4}^E(G[I])} \left(\frac{|I \cap C(e)| - c + 4}{2}\right)^{2r(c-4)} |\mathcal{R}_{c-4}^E(G[I \cap C(e)])|$$
$$\cdot b(c - 2, |I \cap C(e)| - 2r(c - 2))$$

$$\leq \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} \left(\frac{|I| - c + 2}{2}\right)^{2r(c-4)} |\mathcal{R}_{c-4}^E(G[I \cap C(e)])|$$
$$\cdot b(c - 2, |I| - 2 - 2r(c - 2)).$$

Now, we apply Lemma 2.2 (on the subgraph $G[I]$):

$$\leq \left(\frac{|I| - c + 2}{2}\right)^{2r(c-4)+2} |\mathcal{R}_{c-2}^E(G[I])| \ b(c - 2, |I| - 2 - 2r(c - 2))$$

Now, we use Observation 2 for both occurrences of the function r. Furthermore, since c and $c - 2$ are either both odd or both even, we can replace the $c - 2$ in the call to b with c:

$$L(c, I) \leq \left(\frac{|I| - c + 2}{2}\right)^{2r(c-2)} |\mathcal{R}_{c-2}^E(G[I])| \ b(c, |I| - 2r(c)),$$

which is of the desired form and concludes the proof. □

See Appendix A.2 for a proof of the overall work incurred by Algorithm 1, finishing the proof of Theorem 2.1.

3 COMBINATORICS

In this section, we prove Lemma 2.3, which is the key ingredient in the work proof. We begin by introducing the notation that we need for the proof. See also Figure 5 and Figure 6 for an illustration of the relevant concepts.

3.1 Additional Notation

A vertex $u \in V$ is a *relevant out-vertex* with respect to c, if there exists a vertex $v \in V$, such that (u, v) forms a relevant pair with respect to c. The set of relevant out-vertices with respect to c and the total order in V is denoted $\mathbb{P}_c^+(V)$. Similarly, a vertex $v \in V$ is a *relevant in-vertex* with respect to c, if there exists a vertex $u \in V$, such that (u, v) forms a relevant pair with respect to c. The set of
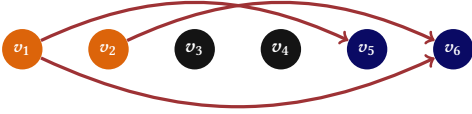
Figure 5: Ordered between the pairs (v_1, v_5), (v_1, v_6) and (v_2, v_6) are at least 3 other vertices. Hence, they are all relevant pairs with respect to 3. That is, they constitute the set $\mathcal{R}_3^P(G)$. Each of the relevant pairs starts with one of the vertices in $\mathbb{P}_3^+(V) = \{v_1, v_2\}$ (marked orange) and ends with one of the vertices in $\mathbb{P}_3^-(V) = \{v_5, v_6\}$ (marked blue).
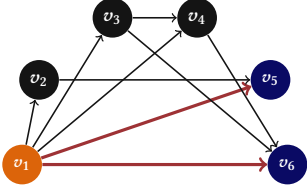

Figure 6: In the example graph $\mathcal{R}_3^E(G) = \{(v_1, v_5), (v_1, v_6)\}$. Hence, the relevant out-vertices with respect to 3 are $\mathbb{E}_3^+(G) = \{v_1\}$ (marked orange). Each such vertex is the head of an edge that has distance 3. The vertices $\mathbb{E}_3^-(G, v_1) = \{v_5, v_6\}$ are the tails of those edges (marked blue).

relevant in-vertices with respect to c and the total order in V is denoted $\mathbb{P}_c^-(V)$. See Figure 5 for an example.

$\mathbb{E}_c^+(G)$ are all relevant out-vertices with respect to c, that are part of a *relevant edge*, not only a relevant pair. Similarly, $\mathbb{E}_c^-(G, u)$ are all relevant in-vertices with respect to c, that form a relevant edge together with u, see Figure 6. Formally, we have:

$$\mathbb{E}_c^+(G) = \{u \mid \exists w : (u, w) \in \mathcal{R}_c^E(G)\} ,$$

$$\mathbb{E}_c^-(G, u) = \{v \mid (u, v) \in \mathcal{R}_c^E(G)\} .$$

For a set S and a *predicate* P, we write $S[P]$ to select the subset of S that satisfies the predicate P, e.g., $V[< u]$ contains all vertices in V smaller than u in the order $<$.

3.2 Bounds on Relevant Edges and Pairs

Let us begin with two simple observations on relevant pairs:

OBSERVATION 3. *In a set of vertices V with a total order, the number of relevant out- and in-vertices with respect to $c \in \mathbb{N}_0$ is:*

$$|\mathbb{P}_c^+(V)| = |\mathbb{P}_c^-(V)| = |V| - (c + 1) .$$

PROOF. The last (respectively first) $(c + 1)$ vertices in V cannot be relevant with respect to c. □

OBSERVATION 4. *In a set of vertices V with a total order, the number of relevant pairs in V with respect to $c \in \mathbb{N}_0$ is:*

$$|\mathcal{R}_c^P(V)| = |\mathbb{P}_c^+(V)|\frac{|V| - c}{2} = |\mathbb{P}_c^-(V)|\frac{|V| - c}{2} = \binom{|V| - c}{2} .$$

PROOF. The first relevant out-vertex forms relevant pairs with exactly $|V| - (c + 1)$ other vertices. The second out-vertex with $|V| - (c + 2)$ vertices, and so on, until the very last relevant out-vertex only participates in one relevant pair. The count of relevant pairs is thus a sum over a descending sequence from $|V| - (c + 1)$ down to 1. Since there are $|\mathbb{P}_c^+(V)| = |V| - (c + 1)$ terms, the sum

can be computed with Gauss' sum formula. A symmetric argument relates the relevant in-vertices to the relevant pairs. □

We continue with a simpler (weaker) Lemma related to Lemma 2.2:

LEMMA 3.1. *For a graph $G = (V, E)$ oriented by a total order where the largest community has size γ and any $c \geq 2$, we have that:*

$$\sum_{e \in \mathcal{R}_c^E(G)} |\mathcal{R}_{c-2}^E(G[N(e)])| \leq \binom{\gamma - c + 2}{2}|\mathcal{R}_c^E(G)| .$$

PROOF. Every relevant edge is also a relevant pair, so we bound the number of relevant pairs $|\mathcal{R}_{c-2}^P(C(e))|$ for each edge e. For any edge $e \in E$, by Observation 4 and $|C(e)| \leq \gamma$,

$$|\mathcal{R}_{c-2}^E(C(e))| \leq |\mathcal{R}_{c-2}^P(C(e))| = \binom{|C(e)| - (c - 2)}{2}.$$

□

When applied to a graph with at most $\gamma + 2$ vertices, Lemma 2.3 provides a factor 2 better bound than Lemma 3.1. This factor is crucial for the work bound as it will be amplified exponentially with k by the recursion. Now, let us turn to the proof of our main result on counting relevant edges.

PROOF OF LEMMA 2.2. For each term of the sum, we rewrite $|\mathcal{R}_{c-2}^P(C(e))|$ using Observation 4. Note that $|C(e)| \leq n - 2$ to upper bound the term:

$$\sum_{e \in \mathcal{R}_c^E(G)} |\mathcal{R}_{c-2}^P(C(e))| \tag{1}$$

$$\leq \frac{n - c}{2} \sum_{e \in \mathcal{R}_c^E(G)} |\mathbb{P}_{c-2}^+(C(e))|, \tag{2}$$

$$\leq \frac{n - c}{2} \sum_{u \in \mathbb{E}_c^+(G)} \sum_{v \in \mathbb{E}_c^-(G, u)} |\mathbb{P}_{c-2}^+(C(u, v))|. \tag{3}$$

Now, we relate the set of relevant in-vertices that are part of a relevant edge to a set of vertices that are part of a relevant pair:

$$\mathbb{E}_c^-(G, u) \supseteq \mathbb{P}_c^-(\{u\} \cup N_G^+(u)) = \mathbb{P}_{c-1}^-(N_G^+(u)) \tag{4}$$

Moreover, for every vertex v that is in $\mathbb{E}_c^-(G, u)$ and is not in $\mathbb{P}_c^-(\{u\} \cup N^+(u))$, the set $\mathcal{R}_{c-2}^P(C(u, v))$ is empty: Consider a vertex in $\mathbb{E}_c^-(G, u)$ that is not in $\mathbb{P}_c^-(\{u\} \cup N_G^+(u))$. Then, there are less than c vertices ordered between u and v that are inside $N_G^+(u)$ (using that u is smaller than its out-neighbors). Hence, the set $N_G^+(u)[< v]$ has size less than c. If the set $\mathcal{R}_{c-2}^P(C(u, v))$ is not empty, then there is a pair (u', v') in it with distance $\delta_{C(u,v)}(u', v') \geq c - 2$. But this implies the set $C(u, v)$ has size at least c. Because $C(u, v)$ is a subset of $N_G^+(u)[< v]$ (all vertices in $N^-(v)$ come before v in the total order) this contradicts the fact that $N_G^+(u)[< v]$ has size less than c.

This means that the terms in the inner sum corresponding to vertices v that are in $\mathbb{E}_c^-(G, u)$ but not in $\mathbb{P}_c^-(\{u\} \cup N^+(u))$ can be dropped without changing the sum. Applying this observation and

$C(u, v) \subseteq N^+(u)[< v]$, we continue:

$$\frac{n-c}{2} \sum_{u \in \mathbb{E}_c^+(G)} \sum_{v \in \mathbb{E}_c^-(G,u)} |\mathbb{P}_{c-2}^+ (C(u,v))| \qquad (5)$$

$$= \frac{n-c}{2} \sum_{u \in \mathbb{E}_c^+(G)} \sum_{v \in \mathbb{P}_{c-1}^-(N^+(u))} |\mathbb{P}_{c-2}^+ (C(u,v))| \qquad (6)$$

$$\leq \frac{n-c}{2} \sum_{u \in \mathbb{E}_c^+(G)} \sum_{v \in \mathbb{P}_{c-1}^-(N^+(u))} |\mathbb{P}_{c-2}^+ (N^+(u)[< v])| \qquad (7)$$

Continue using $\mathbb{P}_{c-2}^+ (N^+(u)[< v]) = \mathbb{P}_{c-1}^+ (N^+(u)[\leq v])$:

$$= \frac{n-c}{2} \sum_{u \in \mathbb{E}_c^+(G)} \sum_{v \in \mathbb{P}_{c-1}^-(N^+(u))} |\mathbb{P}_{c-1}^+ (N^+(u)[\leq v])| \ . \qquad (8)$$

Above, the inner sum now counts all pairs in $N^+(u)$, that are relevant with respect to $c - 1$. We rewrite the sum using Observation 4 applied to the subgraph of G induced by $N^+(u)$ (which has at most $n - 1$ vertices):

$$\leq \frac{n-c}{2} \sum_{u \in \mathbb{E}_c^+(G)} |\mathbb{P}_{c-1}^- (N^+(u))| \frac{n-1-(c-1)}{2} \qquad (9)$$

$$\leq \left(\frac{n-c}{2}\right)^2 \sum_{u \in \mathbb{E}_c^+(G)} |\mathbb{E}_c^- (G,u)|, \qquad (10)$$

$$\leq \left(\frac{n-c}{2}\right)^2 |\mathcal{R}_c^E (G)|. \qquad (11)$$

In the second step, we upper bounded $|\mathbb{P}_{c-1}^- (N^+(u))|$ using Equation (4). The sum now counts again over the edges, relevant with respect to c, which concludes the proof. □

4 GRAPH ORIENTATION

Our clique listing algorithm requires a vertex ordering such that when we orient the graph, its communities have a small size γ. We explore three different approaches. The first two approaches rely on existing results on degeneracy orders. The third hybrid approach uses a different outer loop to achieve a better work/depth tradeoff for large cliques k.

We also present how to get algorithms with bounds that are parametric in the community degeneracy. These algorithms require an order *on the edges* that reduces the size of the subgraphs used in the recursive calls over the edges' communities.

4.1 Computing a Degeneracy Order

We can use previous results on degeneracy order to compute such an order, with different work/depth tradeoffs. A greedy algorithm computes the degeneracy order in linear time, but has linear depth.

LEMMA 4.1. *Computing a degeneracy order takes $O(m)$ work and $O(n)$ depth [38].*

Although no low depth algorithm is known to compute the degeneracy order, it can be approximated effectively in parallel. A total order on the vertices such that the graph oriented by it has maximum out-degree βs is a β-approximate degeneracy order.

LEMMA 4.2 (BESTA ET AL.[3], SHI ET AL. [49]). *Computing a $(2+\varepsilon)$-approximate degeneracy order takes $O(m)$ work and $O(\log n \log_{1+\varepsilon} n)$ depth.*

Note that the size of the largest community is at most the maximum out-degree minus 1. Hence, using these vertex orders to orient the graph, we obtain two of the results in Table 1 from Theorem 2.1, namely the best work and the best depth results for degeneracy.

4.2 The Hybrid Approach

While computing the exact degeneracy order provides the best work-bound in terms of m, k, and s, its depth is linear in n. Using an approximate degeneracy order yields a clique counting algorithm that is work-inefficient by a term that is exponential in k. This inefficiency persists even for very small ε because the approximation leads to a factor $2 + \varepsilon$ larger maximum out-degree in the worst case. Hence the factor $\Theta((2 + \varepsilon)^k)$ work-inefficiency arises.

Therefore, we present a hybrid approach that is only work-inefficient by a factor $\frac{ns}{m} \leq s$ and has a depth that reduces the term linear in n to linear in s. The idea is that an approximate degeneracy order already guarantees that the subgraphs induced by the out-vertices have $O(s)$ vertices. It then remains to solve these subgraphs with the clique listing algorithm that uses an exact degeneracy order. The algorithm goes as follows:

Compute a (2.5)-approximate degeneracy order in parallel and orient the graph according to it. Then, for each vertex v, orient the graph $G[N^+(v)]$ using an (exact) degeneracy order, then run Algorithm 2 on $G[N^+(v)]$.

This hybrid approach has $O\left(kns\left(\frac{s+3-k}{2}\right)^{k-2}\right)$ work and depth $O\left(s + k \log s + \log^2 n\right)$. For each vertex v, the induced subgraph $G[N^+(v)]$ has $O(s|N^+(v)|)$ edges and $O(s)$ vertices. The cost of building these subgraphs and their communities is $O\left(sN^+(v)\right)$ per vertex v and $O(ms)$ overall. For $k \geq 4$, the preprocessing cost does not dominate. Finally, apply the bounds from Theorem 2.1 on each subgraph for $\gamma = s - 1$.

4.3 Parameterizing by Community Degeneracy

Next, we present a preprocessing approach that uses a total order of the edges in addition to an order on the vertices. This preprocessing naturally leads to algorithms whose runtime is parameterized by the community degeneracy of the graph.

To reduce the runtime for the recursive clique listing algorithm, we want to reduce the size of the subgraphs induced by the communities of an edge. We observe that it suffices to recurse only on a subset of the community, namely we can introduce an order on the edges of the graph. Then, we only consider the community of an edge e in the subgraph induced by edges ordered higher than e. In other words, these are the endpoints of the triangles supported by edges whose other edges come after the supporting edge. The approach is summarized in Algorithm 3 and goes as follows:

In addition to computing a total order on the vertices, first, compute *a total order $\leq$ on the edges*. Then, traverse the edges in this order and for each edge $e = (u, v)$ construct the candidate set $C_{(V,E[e\leq])}(e)$, that is, the community of the edge e in the subgraph of G consisting of the edges ordered after e in the total order. For each candidate set, run the recursive clique counting Algorithm 2 with parameter $c = k - 2$.

To reduce the work of this algorithm, the chosen edge order is crucial. We minimize the maximum size of the candidate sets with

a greedy algorithm: repeatedly choose and remove an edge that support the smallest number of triangles in the remaining subgraph. By definition of the community degeneracy σ, the largest subgraph constructed in this way has σ vertices. Note that the order of the vertices used for the recursive clique listing does not influence the runtime and can be arbitrary (say by vertex id).

Algorithm 3 Listing all k-cliques in graph G

1: DAG← orient G by a total vertex order
2: Compute a total order $\preceq$ of the edges of G
3: **for all** edges $e = \{u, v\}$ **do in parallel**
4:     $V' \leftarrow C_{(V, E[e \preceq])}(e)$
5:     RECURSIVECOUNT(DAG, V', $k - 2$)
6: **end for**

Observe that the set of candidates V' constructed for edge e is a subset of the community $C_G(e)$ of e in the input graph. Moreover, each k-clique is contained in at least one of the constructed subgraphs, namely the one which corresponds to the edge in the clique lowest in the community degeneracy order. The total order on the vertices ensures that each clique is reported exactly once.

THEOREM 4.3. *Let γ be the largest size of the set V' takes in Algorithm 3. Then, the cost of Algorithm 3, excluding the cost of preprocessing, is $O\left(km\left(\frac{\gamma+4-k}{2}\right)^{k-2}\right)$ work and $O(k \log(\gamma + 1))$ depth to list all k-cliques.*

The analysis is basically the same as in Appendix A.2, except that we have a better bound for the induced subgraphs given by the maximum size of $|V'| = \gamma$.

We again propose three different variants on implementing the preprocessing in Algorithm 3. The simplest way is the greedy approach outlined above, which leads to the best work.

The problem with this approach is that it has linear depth in n to compute the edge order. Instead of computing the order of the edges greedily, we can compute an approximation of the order, summarized in Algorithm 4 and described in the following:

Compute the triangles T in G. For each edge, keep track of the triangles that contain it (and vice versa). Moreover, for each edge e, keep a count of the remaining number of triangles $|C(e)|$ that contain e. Then, repeat the following until the graph has no edges left: Select all the edges with $|C(e)| \leq (3 + \varepsilon)T/m$ triangles. The constant ε must be positive. Remove those edges from the graph and add them to the total order, breaking ties arbitrarily. For each triangle that contained an edge in the removed set of edges, its edges e need to update their count $|C(e)|$. Then, update the overall count of remaining triangles T.

Next, we make a few observations to help us bounding the work, depth, and quality of the approximate community degeneracy order.

OBSERVATION 5. *A graph with community degeneracy σ has at most σm triangles.*

PROOF. Remove the edges of the graph in a greedy manner, picking the edge with the smallest remaining number of triangles that contain it next. By definition of the community degeneracy, each subgraph encountered this way has an edge with σ triangles

---

Algorithm 4 Computing a $(3 + \varepsilon)$-approximate community degeneracy order of the graph G

1: Compute the triangles T in G and link them with their edges.
2: Count for each edge the triangles $|C(e)|$ that contain it.
3: **while** $E \neq \emptyset$ **do**
4:     $E' \leftarrow$ the edges with at most $|C(e)| \leq (3 + \varepsilon)T/m$ triangles.
5:     Add the edges in E' to the total order, break ties arbitrarily.
6:     Remove E' from the graph
7:     Update each $C(e)$ to reflect the removal of E'.
8:     Update the number of triangles T.
9: **end while**

---

that contain it. The process terminates after m steps. Hence, the process removes at most $m\sigma$ triangles until no triangle is left. □

OBSERVATION 6. *Algorithm 4 terminates after $O\left(\log_{1+\varepsilon} m\right)$ iterations*

PROOF. From Observation 5, it follows that the average number of triangles per edge is at most $3\sigma T/m$ in each iteration (each triangle is counted once for each of its edges). From any set of numbers, at most a $\frac{1}{1+\varepsilon}$ fraction of numbers are larger than $(1 + \varepsilon)$ times the average. Hence, each iteration reduces the number of edges by a factor at least $(1 + \varepsilon)$. □

LEMMA 4.4. *Algorithm 4 computes an order $\preceq$ on the edges such that for each edge e, the set $C_{(V, E[e \preceq])}(e)$ has size at most $(3 + \varepsilon)\sigma$. It takes $O\left(ms + m\sigma\right)$ work and $O\left(\log n \log_{1+\varepsilon} n\right)$ depth.*

PROOF. Listing the triangles takes $O\left(ms\right)$ work and $O\left(\log^2 n\right)$ depth [21]. Each iteration of the loop takes work proportional to the number of removed edges and $O\left(\log n\right)$ depth. By Observation 6, the algorithm terminates after $O\left(\log_{1+\varepsilon} n\right)$ iterations.

For the bound on the size of the communities, note that by Observation 5, $T/m \leq \sigma$. Hence, each of the removed edges satisfy $|C(e)| \leq (3 + \varepsilon)T/m \leq (3 + \varepsilon)\sigma$. □

The results for the community degeneracy in Table 1 follow from Theorem 4.3 by using either the sequential greedy preprocessing, Lemma 4.4, or by using a hybrid approach similar to Section 4.2.

5 CONCLUSION AND FUTURE WORK

We presented work improvements over previous algorithms on k-clique listing in s-degenerate and σ-community-degenerate graphs. The improvements are exponential in the clique size k, in particular when $k = \Theta(s)$.

Many interesting open questions remain. There remains a gap of $O\left(ns/m\right)$ between the work of our best poly-logarithmic depth algorithm and our lowest work algorithm. Can we do better? Moreover, it would be interesting if the work can be further reduced when k is not constant. It might be interesting to consider generalizations that extend the cliques by larger motifs such as triangles. Finally, are there other interesting classes of graphs where the clique problem is tractable?

REFERENCES

[1] Charu C Aggarwal, Haixun Wang, et al. *Managing and mining graph data*, volume 40. Springer, 2010.

[2] Mohammad Almasri, Izzat El Hajj, Rakesh Nagi, Jinjun Xiong, and Wen-mei Hwu. K-clique counting on gpus. *arXiv preprint arXiv:2104.13209*, 2021.

[3] M. Besta, A. Carigiet, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, and T. Hoefler. High-performance parallel graph coloring with strong guarantees on work, depth and quality. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1401–1417, Los Alamitos, CA, USA, nov 2020. IEEE Computer Society.

[4] Maciej Besta, Armon Carigiet, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, and Torsten Hoefler. High-performance parallel graph coloring with strong guarantees on work, depth, and quality. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17. IEEE, 2020.

[5] Maciej Besta, Marc Fischer, Vasiliki Kalavri, Michael Kapralov, and Torsten Hoefler. Practice of streaming processing of dynamic graphs: Concepts, models, and systems. *arXiv preprint arXiv:1912.12740*, 2019.

[6] Maciej Besta and Torsten Hoefler. Accelerating irregular computations with hardware transactional memory and active messages. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pages 161–172, 2015.

[7] Maciej Besta and Torsten Hoefler. Survey and taxonomy of lossless graph compression and space-efficient graph representations. *arXiv preprint arXiv:1806.01799*, 2018.

[8] Maciej Besta, Raghavendra Kanakagiri, Grzegorz Kwasniewski, Rachata Ausavarungnirun, Jakub Beránek, Konstantinos Kanellopoulos, Kacper Janda, Zur Vonarburg-Shmaria, Lukas Gianinazzi, Ioana Stefan, et al. Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems. *arXiv preprint arXiv:2104.07582*, 2021.

[9] Maciej Besta, Emanuel Peter, Robert Gerstenberger, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *arXiv preprint arXiv:1910.09017*, 2019.

[10] Maciej Besta, Michał Podstawski, Linus Groner, Edgar Solomonik, and Torsten Hoefler. To push or to pull: On reducing communication and synchronization in graph computations. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, pages 93–104, 2017.

[11] Maciej Besta, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, and Torsten Hoefler. Log (graph) a near-optimal high-performance graph representation. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, pages 1–13, 2018.

[12] Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, et al. Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra. *VLDB*, 2021.

[13] Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, Peter Tatkowski, Esref Ozdemir, Adrian Balla, Marcin Copik, Marek Konieeczny, OnurMutlu, and Torsten Hoefler. Graphminesuite: Enabling high-performance and programmable graph mining algorithms [bench. & analysis. unpublished, N.D.

[14] Maciej Besta, Simon Weber, Lukas Gianinazzi, Robert Gerstenberger, Andrey Ivanov, Yishai Oltchik, and Torsten Hoefler. Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–25, 2019.

[15] Guy E. Blelloch. Programming parallel algorithms. *Commun. ACM*, 39(3):85–97, March 1996.

[16] Vincent Bouchitté and Ioan Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1-2):17–32, 2002.

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

[18] Austin Buchanan, Jose L. Walteros, Sergiy Butenko, and Panos M. Pardalos. Solving maximum clique in sparse graphs: an o(nm+n2$^{d/4}$ algorithm for d-degenerate graphs. *Optim. Lett.*, 8(5):1611–1617, 2014.

[19] James Cheng, Yiping Ke, Ada Wai-Chee Fu, Jeffrey Xu Yu, and Linhong Zhu. Finding maximal cliques in massive networks. *ACM Transactions on Database Systems (TODS)*, 36(4):1–34, 2011.

[20] James Cheng, Linhong Zhu, Yiping Ke, and Shumo Chu. Fast algorithms for maximal clique enumeration with limited memory. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1240–1248, 2012.

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

[22] Richard Cole. Parallel merge sort. *SIAM J. Comput.*, 17(4):770–785, 1988.

[23] Diane J Cook and Lawrence B Holder. *Mining graph data*. John Wiley & Sons, 2006.

[24] Intel® Corporation. Intel®product specifications. https://ark.intel.com/content/www/us/en/ark.html#@Processors. Accessed: 2021-02-23.

[25] Maximilien Danisch, Oana Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In Pierre-Antoine Champin, Fabien L. Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 589–598. ACM, 2018.

[26] Centro Svizzero di Calcolo Scientifico. Cscs home page. https://www.cscs.ch. Acessed: 2021-02-23.

[27] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[28] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[29] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation - 21st International Symposium, ISAAC 2010, Jeju Island, Korea, December 15-17, 2010, Proceedings, Part I*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414. Springer, 2010.

[30] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. Top-k overlapping densest subgraphs. *Data Mining and Knowledge Discovery*, 30(5):1134–1165, 2016.

[31] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1990.

[32] Inc. GCC Team, Free Software Foundation.

[33] Chuntao Jiang, Frans Coenen, and Michele Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1):75–105, 2013.

[34] Victor E Lee, Ning Ruan, Ruoming Jin, and Charu Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer, 2010.

[35] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[36] Rong-Hua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. Ordering heuristics for k-clique listing. *Proceedings of the VLDB Endowment*, 13(12):2536–2548, 2020.

[37] Don R. Lick and Arthur T. White. k-degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970.

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

[39] Michael Mitzenmacher, Jakub Pachocki, Richard Peng, Charalampos Tsourakakis, and Shen Chen Xu. Scalable large near-clique detection in large-scale networks via sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 815–824, 2015.

[40] Natwar Modani and Kuntal Dey. Large maximal cliques enumeration in sparse graphs. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1377–1378, 2008.

[41] C. St.J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, s1-36(1):445–450, 1961.

[42] Christos H. Papadimitriou and Mihalis Yannakakis. The clique problem for planar graphs. *Information Processing Letters*, 13(4):131–133, 1981.

[43] Saif Ur Rehman, Asmat Ullah Khan, and Simon Fong. Graph mining: A survey of graph mining techniques. In *Seventh International Conference on Digital Information Management (ICDIM 2012)*, pages 88–92. IEEE, 2012.

[44] John H. Reif. *Synthesis of Parallel Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1993.

[45] J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7(3):425–440, 1986.

[46] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[47] Matthew C Schmidt, Nagiza F Samatova, Kevin Thomas, and Byung-Hoon Park. A scalable, parallel algorithm for maximal clique enumeration. *Journal of Parallel and Distributed Computing*, 69(4):417–428, 2009.

[48] Bin Shao, Haixun Wang, and Yanghua Xiao. Managing and mining large graphs: systems and implementations. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 589–592, 2012.

[49] Jessica Shi, Laxman Dhulipala, and Julian Shun. Parallel clique counting and peeling algorithms. *CoRR*, abs/2002.10047, 2020.

[50] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowl. Inf. Syst.*, 54(3):677–710, 2018.

[51] Lei Tang and Huan Liu. Graph mining applications to social network analysis. In *Managing and Mining Graph Data*, pages 487–513. Springer, 2010.

[52] Etsuji Tomita, Tatsuya Akutsu, and Tsutomu Matsunaga. Efficient algorithms for finding maximum and maximal cliques: Effective tools for bioinformatics. In *Biomedical engineering, trends in electronics, communications and software*. IntechOpen, 2011.

[53] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments.

Theor. Comput. Sci., 363(1):28–42, 2006.

[54] Charalampos E. Tsourakakis. The k-clique densest subgraph problem. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 1122–1132, 2015.

[55] David R. Wood. On the maximum number of cliques in a graph. *Graphs Comb.*, 23(3):337–352, 2007.

[56] Long Yuan, Lu Qin, Xuemin Lin, Lijun Chang, and Wenjie Zhang. Diversified top-k clique search. *The VLDB Journal*, 25(2):171–196, 2016.

A WORK BOUNDS CONTINUED

For completeness, we include the remaining derivation of the work of our algorithm. We prove the contribution of the intersections and the edge existence probing separately in Appendix A.1 and combine them with the clique listing term from Section 2.2 in Appendix A.2.

A.1 Recursive Work Cost

We bound the work incurred by Algorithm 2 for intersecting the edge neighborhoods. Note that because every community (and candidate set I) has size at most γ by assumption and the sets are sorted in advance, the cost of any particular intersection is at most $d\gamma$ for some constant d.

LEMMA A.1. *Let $Q(c, I)$ be the work of Algorithm 2 incurred for intersection of sets, for a graph where the largest community has size γ. Then, we have for $c \geq 3$,*

$$Q(c, I) \leq d\gamma \left(\sum_{p=0}^{r(c-3)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) |\mathcal{R}_{c-2}^E (G[I])|.$$

PROOF. The proof is again by induction on c. For the base case $c = 2$ there are trivially no intersections required. For the cases $c = 3$ and $c = 4$, the algorithm only needs to intersect the community of each edge relevant with respect to 1 (resp., 2) once with I and the bound holds. Assume that the bound holds for all $c' \leq c$. We show that it also holds in the case $c + 2$. For that, we start with a recursive expression for the work and apply the induction hypothesis.

$$Q(c, I) \tag{12}$$

$$\leq \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} d\gamma + Q(c - 2, I \cap C(e)), \tag{13}$$

$$\leq \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} d\gamma +$$

$$d\gamma \left(\sum_{p=0}^{r(c-5)} \left(\frac{|I \cap C(e)| - c + 4}{2} \right)^{2p} \right) |\mathcal{R}_{c-4}^E (G[I \cap C(e)])| \tag{14}$$

We rearrange the terms and use that $|I \cap C(e)| \leq |I| - 2$:

$$\leq d\gamma \ |\mathcal{R}_{c-2}^E (G[I])| +$$

$$d\gamma \left(\sum_{p=0}^{r(c-5)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} |\mathcal{R}_{c-4}^E (G[I \cap C(e)])| \ . \tag{15}$$

Now, we apply Lemma 2.2:

$$\leq d\gamma \ |\mathcal{R}_{c-2}^E (G[I])| +$$

$$d\gamma \left(\sum_{p=0}^{r(c-5)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) \left(\frac{|I| - c + 2}{2} \right)^2 |\mathcal{R}_{c-2}^E (G[I])| \tag{16}$$

$$\leq d\gamma \ |\mathcal{R}_{c-2}^E (G[I])| +$$

$$d\gamma \left(\sum_{p=1}^{r(c-5)+1} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) \cdot |\mathcal{R}_{c-2}^E (G[I])| \ . \tag{17}$$

Now, we use Observation 2 and use pull the first term into the sum as well. This results in the desired form, which concludes the proof:

$$Q(c, I) \leq d\gamma \cdot \left(\sum_{p=0}^{r(c-3)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) |\mathcal{R}_{c-2}^E (G[I])| \ . \tag{18}$$

□

Next, we bound the cost incurred for probing for edge existence. By building a perfect hash table (or a adjacency matrix for each subgraph induced by $C(e)$ for every edges e), the cost to probe for the existence of an edge is at most some constant f.

LEMMA A.2. *Let $S(c, I)$ be the work of Algorithm 2 incurred for probing for edge existence. Then, we have for $c \geq 2$,*

$$S(c, I) \leq f \cdot \left(\sum_{p=0}^{r(c-2)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) |\mathcal{R}_{c-2}^P (I)|$$

PROOF. The proof is by induction. If $c = 2$ or $c = 3$, then the algorithm only needs to test all pairs relevant with respect to 0 or 1 in I once and the bound holds. Assume that the bound holds for all $c' \leq c$. We show that it also holds in the case $c + 2$. We write:

$$S(c, I) \tag{19}$$

$$\leq f \ |\mathcal{R}_{c-2}^P (I)| + \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} S(c - 2, I \cap C(e)) \tag{20}$$

$$\leq f \ |\mathcal{R}_{c-2}^P (I)| +$$

$$\sum_{e \in \mathcal{R}_{c-2}^E(G[I])} \left(\sum_{p=0}^{r(c-4)} \left(\frac{|I \cap C(e)| - c + 4}{2} \right)^{2p} \right) |\mathcal{R}_{c-4}^P (I \cap C(e))| \tag{21}$$

Use that $|I \cap C(e)| \leq |I| - 2$, to pull two factors in front of the sum:

$$\leq f \ |\mathcal{R}_{c-2}^P (I)| + \tag{22}$$

$$f \left(\sum_{p=0}^{r(c-4)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) \sum_{e \in \mathcal{R}_{c-2}^E(G[I])} |\mathcal{R}_{c-4}^P (I \cap C(e))| \ . \tag{23}$$

We use Lemma 2.2 on the graph $G[I]$ and use that every relevant edge is also a relevant pair:

$$\leq f \ |\mathcal{R}_{c-2}^P (I)| \ +$$

$$f \left(\sum_{p=0}^{r(c-4)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) \left(\frac{|I| - c + 2}{2} \right)^2 |\mathcal{R}_{c-2}^P (I)| \qquad (24)$$

$$= f \ |\mathcal{R}_{c-2}^P (I)| \ + f \left(\sum_{p=1}^{r(c-4)+1} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) |\mathcal{R}_{c-2}^P (I)| \quad (25)$$

Now, we apply observation 2 and pull the first term into the sum. We arrive at the desired form, concluding the proof:

$$S(c, I) \leq f \left(\sum_{p=0}^{r(c-2)} \left(\frac{|I| - c + 2}{2} \right)^{2p} \right) |\mathcal{R}_{c-2}^P (I)| \qquad (26)$$

$\square$

A.2 Overall Work Bound

So far, we derived the work of Algorithm 2. Now, we will apply the outer most loop in Algorithm 1: The loop itself simply iterates over all relevant edges and calls Algorithm 2 with $c = k - 2$. Note, that we can simplify the call to the base case function b with Observation 2 and the fact that $k - 2$ and k are either both even or both odd.

$$W \leq \sum_{e \in \mathcal{R}_{k-2}^E (G)} W(k - 2, C(e)), \qquad (27)$$

$$\leq f \sum_{e \in \mathcal{R}_{k-2}^E (G)} \left(\sum_{p=0}^{r(k-4)} \left(\frac{|C(e)| - k + 4}{2} \right)^{2p} \right) |\mathcal{R}_{k-4}^P (C(e))|$$

$$+ d\gamma \sum_{e \in \mathcal{R}_{k-2}^E (G)} \left(\sum_{p=0}^{r(k-5)} \left(\frac{|C(e)| - k + 4}{2} \right)^{2p} \right) |\mathcal{R}_{k-4}^E (G[C(e)])|$$

$$+ \sum_{e \in \mathcal{R}_{k-2}^E (G)} \left(\frac{|C(e)| - k + 4}{2} \right)^{2r(k-4)} |\mathcal{R}_{k-4}^E (G[C(e)])|$$
$$\cdot \ b(k, |C(e)| - 2r(k - 2)) \ . \qquad (28)$$

Use that $|C(e)| \leq \gamma$ by assumption:

$$W \leq f \left(\sum_{p=0}^{r(k-4)} \left(\frac{\gamma - k + 4}{2} \right)^{2p} \right) \sum_{e \in \mathcal{R}_{k-2}^E (G)} |\mathcal{R}_{k-4}^P (C(e))|$$

$$+ d\gamma \left(\sum_{p=0}^{r(k-5)} \left(\frac{\gamma - k + 4}{2} \right)^{2p} \right) \sum_{e \in \mathcal{R}_{k-2}^E (G)} |\mathcal{R}_{k-4}^E (G[C(e)])|$$

$$+ \left(\frac{\gamma - k + 4}{2} \right)^{2r(k-4)} b(k, \gamma - 2r(k - 2)) \sum_{e \in \mathcal{R}_{k-2}^E (G)} |\mathcal{R}_{k-4}^E (G[C(e)])|.$$
$$(29)$$

Now, apply Lemma 3.1 to bound the number of relevant edges and pairs in the subgraphs induced by all relevant edges in G:

$$W \leq 2f \cdot \left(\sum_{p=1}^{r(k-4)+1} \left(\frac{\gamma - k + 4}{2} \right)^{2p} \right) |\mathcal{R}_{k-2}^E (G)|$$

$$+ 2d\gamma \cdot \left(\sum_{p=1}^{r(k-5)+1} \left(\frac{\gamma - k + 4}{2} \right)^{2p} \right) |\mathcal{R}_{k-2}^E (G)|$$

$$+ 2 \left(\frac{\gamma - k + 4}{2} \right)^{2r(k-4)+2} b(k, \gamma - 2r(k)) \ |\mathcal{R}_{k-2}^E (G)|. \quad (30)$$

$$= 2f \cdot \left(\sum_{p=1}^{r(k-2)} \left(\frac{\gamma - k + 4}{2} \right)^{2p} \right) |\mathcal{R}_{k-2}^E (G)|$$

$$+ 2d\gamma \cdot \left(\sum_{p=1}^{r(k-3)} \left(\frac{\gamma - k + 4}{2} \right)^{2p} \right) |\mathcal{R}_{k-2}^E (G)|$$

$$+ 2 \left(\frac{\gamma - k + 4}{2} \right)^{2r(k-2)} b(k, \gamma - 2r(k - 2)) \ |\mathcal{R}_{k-2}^E (G)|$$

$$\leq 2f \cdot r(k-2) \left(\frac{\gamma - k + 4}{2} \right)^{2r(k-2)} |\mathcal{R}_{k-2}^E (G)|$$

$$+ 2\gamma \cdot r(k-2) \left(\frac{\gamma - k + 4}{2} \right)^{2r(k-3)} |\mathcal{R}_{k-2}^E (G)|$$

$$+ 2 \left(\frac{\gamma - k + 4}{2} \right)^{2r(k-2)} \cdot b(k, \gamma - 2r(k - 2)) \ |\mathcal{R}_{k-2}^E (G)|. \quad (31)$$

To further simplify the above expression, we make use of a simple observation:

OBSERVATION 7. *For $k \geq 1, k \leq \gamma + 2$, it holds that*

$$2\gamma \leq 2k(\gamma + 4 - k).$$

Now recall the definition of the level function:

$$r(k) := \left\lfloor \frac{k}{2} \right\rfloor.$$

Due to the floor operator, it evaluates to the same value for k even and $k' = k + 1$ odd, but the base case incurs different costs for k even and k' odd. Let's take a look at the listing part of the cost:

$$W(k \text{ even}) \leq 2 \left(\frac{\gamma - k + 4}{2} \right)^{k-2} \cdot k, \qquad (32)$$

$$W(k' \text{ odd}) \leq 2 \left(\frac{\gamma - k' + 4}{2} \right)^{k'-3} \cdot k'(\gamma - (k' - 3)), \quad (33)$$

$$\leq 2 \left(\frac{\gamma - k' + 4}{2} \right)^{k'-3} \cdot k' 2 \frac{\gamma - k' + 4}{2}, \qquad (34)$$

$$\leq 4 \left(\frac{\gamma - k' + 4}{2} \right)^{k'-2} \cdot k' \qquad (35)$$

Together with Observation 7, we see that if k is even, then the test and listing terms are dominant. In the odd case, the listing term is the dominant one, closely followed by the testing term. This concludes the wo

Graph	$|V|$	$|E|$	$|T|$	s	$\frac{|E|}{|V|}$	$\frac{|T|}{|V|}$	$\frac{|T|}{|E|}$
Orkut	3.1M	117.2M	627.6M	253	38.1	204.6	5.4
Ca-DBLP-2012	317K	1M	2.2M	113	3.3	7	2.1
Tech-As-Skitter	1.7M	11.1M	28.8M	111	6.5	17	2.6
Gearbox	153.7K	4.5M	4.6M	44	29	30	1
Chebyshev4	68K	1.9M	28.9M	68	28.9	424.2	14.7
Jester2	50.1K	1.7M	35.6M	128	34.1	703.3	20.6
Bio-SC-HT	2084	63K	1.4M	100	30.2	670.7	22.2

Table 2: Overview over the selected graphs. s is the degeneracy; T denotes the triangles in the graph. All graphs are publicly available [35, 46] and have been symmetrized.

B EXPERIMENTAL EVALUATION

In this section, we present an empirical evaluation of Algorithm 1. For all experiments, we use the (exact) degeneracy order from Section 4.1. We compare our implementation with the state-of-the-art implementations by Danisch et al. [25] and Shi et al. [49].

B.1 Data Set

We assembled a set of graphs to test the different implementations. These graphs represent different fields of applications and exhibit different structural properties [13]. Table 2 provides a brief overview of the graphs. The graphs 'Tech-as-skitter', 'Ca-DBLP-2012' and 'Orkut' have already been used previously for benchmarking by Danisch et al. [25] and Shi et al. [49]. Orkut is a relatively large online social network, Ca-DBLP-2012 is the collaboration network of the DBLP 2012, and Tech-as-Skitter is an internet topology graph, generated in 2005 by daily traceroutes [35].

In addition, we added some non-standard graphs to the set: 'Gearbox', 'Chebyshev4', 'Bio-SC-HT' and 'Jester2'. Gearbox is a structural network from an aircraft flap actuator. Chebyshev4 is a structural network as well. It is derived from a 4th order Chebyshev scheme for numerical integration. Jester2 is a joke-rating network, and Bio-SC-HT is a network derived from functional gene associations [46].

B.2 Experimental Setup

The bulk of our experiments was performed on the Piz Daint Computing platform by CSCS. We used the XC40 Multicore Compute nodes, equipped with two Intel® Xeon® E5-2695 v4 2.10GHz, 18 cores/36 threads, 45MB of Intel® Smart Cache and each with 64/128 GB RAM [24, 26].

The code was compiled in both instances with GCC 8.3 with the optimization flags `-O3` and `-march=native`. Piz Daint supports AVX2. GCC 8.3 supports OpenMP 4.5 [32], which was chosen for parallel programming.

All measurements were repeated at least 10 times, except for the runs for Orkut and Jester with $k \geq 7$, which were repeated 5 times
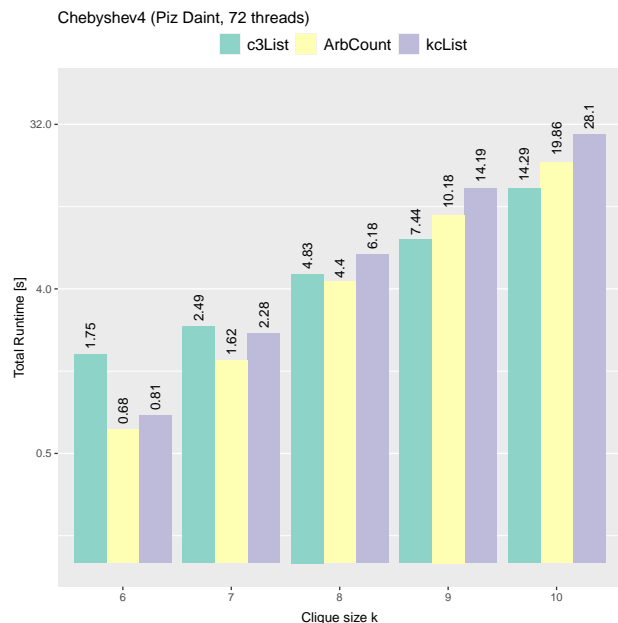

Figure 7: Results on the Chebyshev4 graph.

due to the long running times. The reported times are arithmetic averages over all measurements.
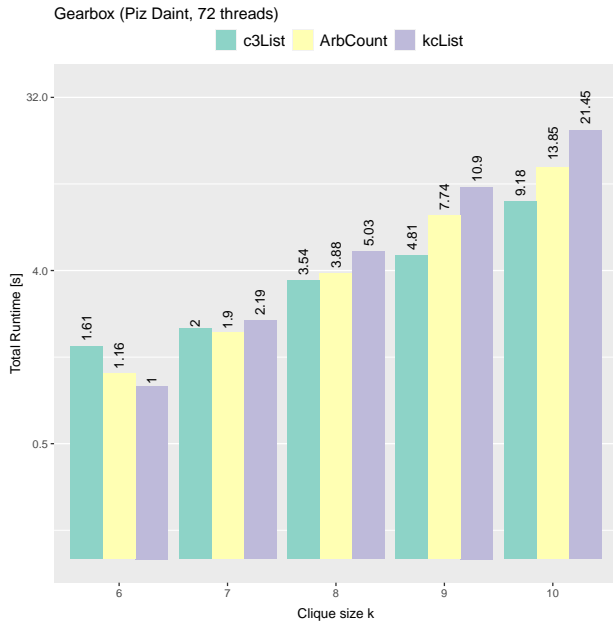
B.3 Results

See Figure 7, Figure 8 and Figure 9 for the comparison of our algorithm (c3List) with Shi et al. (ArbCount [49]) and Danisch et al. (kcList [25]). We report the runtime for 72 threads and vary the clique size k from 6 to 10.

The empirical standard deviation of the runtimes is less than 5.2% for clique sizes $k \geq 8$ for all algorithms and all graphs except the Gearbox graphs (which can be solved quickly). For Gearbox graphs with $k = 10$, the standard deviation is less than 6.4%. Other results need to be interpreted with care, as the standard deviation is larger than 10% for at least one graph and algorithm.

For small clique sizes ($k < 8$), it depends on the graph which algorithm is fastest. These results also have a higher variance.

For larger clique sizes ($k \geq 8$), Shi et al. [49] generally outperform Danisch et al. [25]. Our community-centric algorithm is faster than both on a majority of the instances starting at $k = 8$. The advantage of our implementation over the others generally grows with the clique size. This trend might be because of the non-constant base to the exponent in the work bounds of our algorithm.

Overall, for cliques of size $k \geq 9$, our algorithm is slower in one instance (Orkut) and outperforms the others by $3.4 - 37.9\%$. Our algorithm appears to be relatively better when there are few triangles per vertex. For those graphs (Tech-As-Skitter, Gearbox, CA-DBLP-2012), our algorithm is $16.5 - 37.9\%$ and $13.8 - 33.7\%$ faster than the next fastest implementation for cliques of size 9 and 10, respectively. The good performance on those graphs could be because our pruning technique is particularly effective when there are fewer triangles.
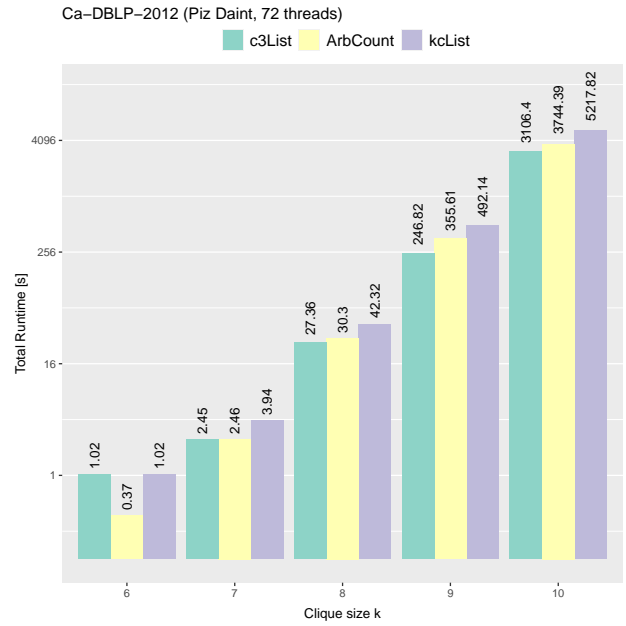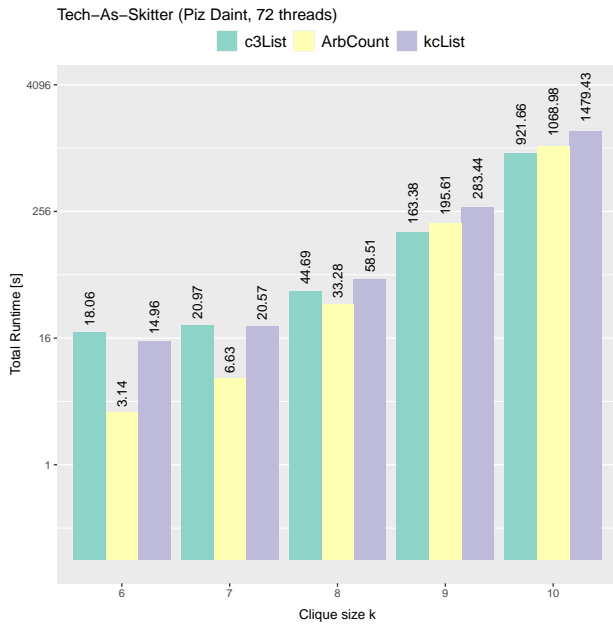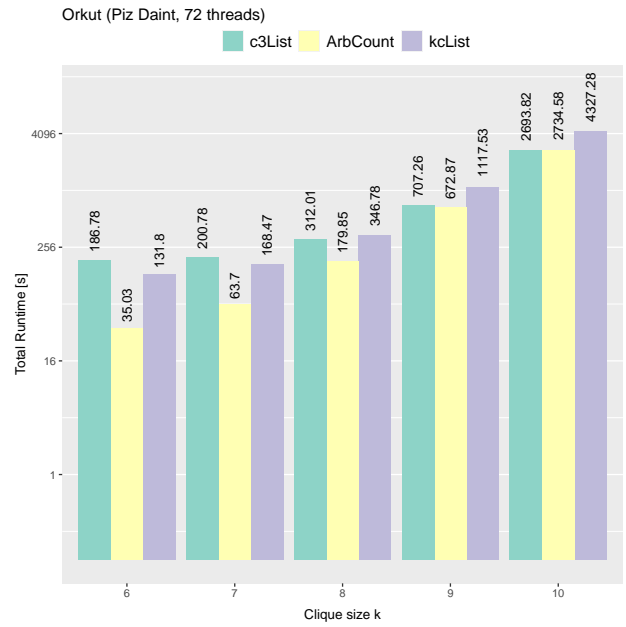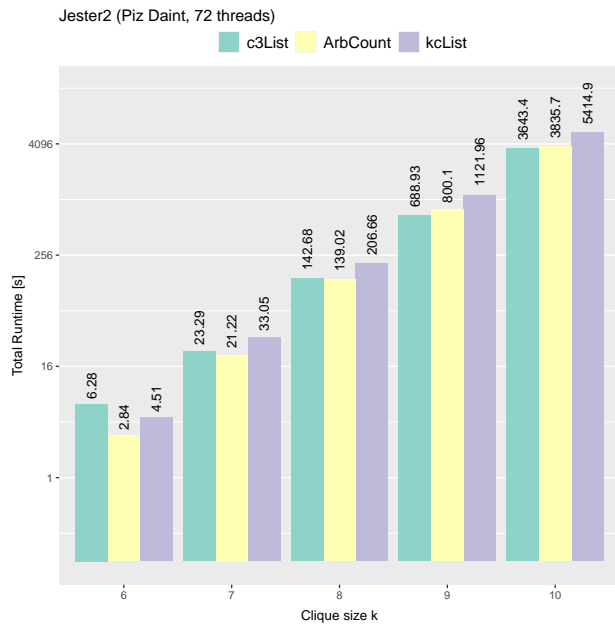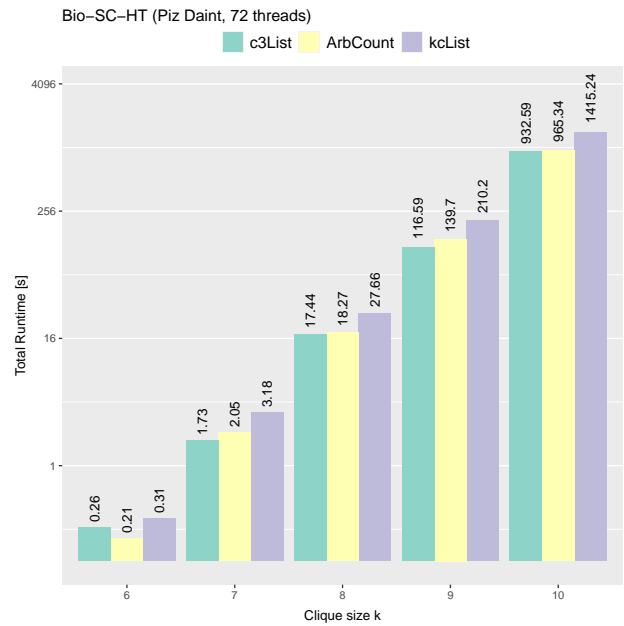
Figure 8: Runtime Results for 72 threads for varying clique sizes. Our algorithm is c3List, ArbCount is by Shi et al. [49], and kcList is by Danisch et al. [25]

Figure 9: Runtime Results for 72 threads for varying clique sizes. Our algorithm is c3List, ArbCount is by Shi et al. [49], and kcList is by Danisch et al. [25]