Subgraph Enumeration in Massive Graphs*

Francesco Silvestri

University of Padova, Dip. Ingegneria dell’Informazione, Padova, Italy
silvest1@dei.unipd.it

Abstract

We consider the problem of enumerating all instances of a given sample graph in a large data graph. Our focus is on determining the input/output (I/O) complexity of this problem. Let \( E \) be the number of edges in the data graph, \( k = O(1) \) be the number of vertexes in the sample graph, \( B \) be the block length, and \( M \) be the main memory size. The main result of the paper is a randomized algorithm that enumerates all instances of the sample graph in \( O\left(\frac{E^{k/2}}{BM^{k/2}-1}\right) \) expected I/Os if the maximum vertex degree of the data graph is \( \sqrt{EM} \). Under some assumptions, the same bound also applies with high probability. Our algorithm is I/O optimal: we show that any algorithm enumerating \( T \) instances must always use \( \Omega\left(\frac{T}{BM^{k/2}-1}\right) \) I/Os when the sample graph belongs to the Alon class, which includes cliques, cycles and every graph with a perfect matching.

1 Introduction

This paper targets the problem of enumerating all subgraphs of an input data graph that are isomorphic to a given sample graph. Subgraph enumeration is a tool for analyzing the structural and functional properties of networks. It has been used for analyzing the evolution of social networks [1], the dynamics of Internet at the level of autonomous systems [2], the significance of motifs in biological networks [3]. Typical sample graphs are cliques (in particular triangles), cycles and paths.

The aim of this paper is to assess the input/output (I/O) complexity of the enumeration problem when the data graph does not fit in the main memory. The main result of the paper is an external memory (EM) randomized algorithm for subgraph enumeration. Let \( E \) be the number of edges in the input data graph, \( k = O(1) \) be the number of vertexes in the sample graph, \( B \) be the block length, and \( M \) be the main memory size. Then, the randomized algorithm requires \( O\left(\frac{E^{k/2}}{BM^{k/2}-1}\right) \) I/Os in expectation as soon as the maximum vertex degree in the data graph is \( \sqrt{EM} \). We show that, with some adjustments, the same I/O complexity holds with high probability if the memory size is \( \Omega\left(\sqrt{E \log E}\right) \). The randomized algorithm relies on a deterministic algorithm, also presented in this paper and of independent interest, which exploits a suitable independent set of the sample graph.

We also propose a lower bound on the I/O complexity for the enumeration problem when the sample graph belongs to the Alon class [4]. This class includes important graphs like cliques, cycles and, more in general, every graph with a perfect matching. The bound shows that any algorithm enumerating \( T \) instances must always use \( \Omega\left(\frac{T}{BM^{k/2}-1} + \frac{T^{2/k}}{B}\right) \) I/Os.

The focus of the paper is on the enumeration of edge-induced subgraphs which are isomorphic to the sample graph. However, we claim that the randomized algorithm can be extended to the

---

*This work was done while the author was visiting the IT University of Copenhagen. It is supported by: University of Padova project CPDA121378, MIUR of Italy project AMANDA, Danish National Research Foundation Sapere Aude program.
enumeration of vertex-induced subgraphs (or simply, induced subgraphs) isomorphic to the sample graph.

1.0.1 Our results.

The analyses of our algorithms do not assume $k$ to be constant and clearly state the dependency of the I/O complexities on $k$. We define $s \geq 1$ to be the size of the largest independent set $S$ of the sample graph $H$ such that each vertex in $H$ is adjacent to at most one vertex in $S$. The results given in this paper are the followings.

1. In Section 3, we give a deterministic algorithm for subgraph enumeration that exploits the independent set $S$ of the sample graph. Its I/O complexity is $O((4k)^{k-s-1}E^{k-s}/(BM^{k-s-1}))$. The algorithm works even in a read-only external memory.

2. In Section 4, we propose a randomized algorithm for subgraph enumeration. It exploits the random coloring technique in [5] for decomposing the problem into smaller subproblems that are solved with the previous deterministic algorithm. Its expected I/O complexity is $O((k^{k-s-1}E^{k/2}/(BM^{k/2-1}))$ if the maximum vertex degree of the input graph is $\sqrt{EM}$. Moreover, we show that the claimed I/O complexity is achieved with high probability when $M = \Omega(\sqrt{E \log E})$ by applying some adjustments to the coloring process.

3. In Section 5, we prove that the enumeration of $T$ instances of a sample graph in the Alon class requires, even in the best case, $\Omega(T/(BM^{k/2-1}) + T^{2/k}/B)$ I/Os. When $k = \mathcal{O}(1)$, this result shows that the randomized algorithm is optimal since there exists a graph (i.e., a clique with $\sqrt{E}$ vertices) containing $T = \Theta(E^{k/2})$ instances of the sample graph.

4. In Section 6, we discuss some extensions: we explain how to enumerate induced subgraphs using our algorithms; we remove the assumption on the maximum degree for the randomized algorithm; we upper bound the running time of our algorithms. For the sake of simplicity, the last two results are derived assuming the sample graph to be a $k$-clique, but we conjecture that similar ideas apply to other sample graphs as well.

We do not require algorithms to list all instances of the sample graph, that is to store all instances on the external memory. We simply consider algorithms that enumerate instances: that is, for each instance, they call a function $\text{emit}(\cdot)$ with the edges of the instance as input parameters. We suppose that a call to $\text{emit}(\cdot)$ performs no I/Os and requires $O(1)$ operations. As argued in [5], this is a natural assumption in external memory since it reduces the I/O complexity and it is satisfied by many applications where instances are intermediate results pipelined to a subsequent computation and are not required to be permanently stored (e.g., consider an application searching for the instance that maximizes a given objective function). Nevertheless, our upper and lower bounds can be easily adapted to list all instances on the external memory by increasing the I/O complexity of an unavoidable additive $\Theta(T/B)$ factor, where $T$ is the number of instances in the graph.

1.0.2 Related work.

To the best of our knowledge, this is the first paper to deal with the I/O complexity of the enumeration of a generic sample graph. Previous works have targeted the I/O complexity of triangle enumeration. Goodrich et al. [6] gave an optimal algorithm requiring $O(\text{sort}(E))$ I/Os for graphs with constant arboricity; this algorithm however does not efficiently scale with larger arboricity. Chu et al. [7] and Hu et al. [8] proposed algorithms for a generic graph incurring $O(E^2/(BM))$ I/Os. Pagh and Silvestri [5] have recently improved this bound by proposing cache-aware and cache-oblivious algorithms, based on a random vertex coloring technique, that exhibit an optimal $\Theta(E^{3/2}/(B\sqrt{M}))$ expected I/O complexity. In the special case where the sample graph is a triangle, our deterministic algorithm is similar to the one proposed in [8], but
it does not need to manage in a different way vertexes with degree smaller or equal than $M$ and vertexes with degree larger than $M$. Moreover, our randomized algorithm matches the cache-aware algorithm in [5], but we are able to prove that the $O \left( E^{3/2} / (B \sqrt{M}) \right)$ I/O complexity applies with high probability if the main memory is sufficiently large.

The enumeration of subgraphs has also been targeted in other models. Enumeration in the parallel MapReduce framework has been target by Afrati et al. [4] and by Suri and Vassilvitskii [9] for triangles, and by Afrati et al. [10] for general sample graphs. These papers are based on partitioning techniques similar to the one used by our randomized algorithm, however these works assume a random input and do not provide strong bounds in our settings. The enumeration of $k$-cliques, for a given $k \geq 3$, has been targeted by Chiba and Nishizeki [11] in the RAM model, but it requires $\Omega \left( E^{k/2} / B \right)$ I/Os in a memory hierarchy. Multiway-join is a problem from database theory similar to subgraph enumeration: however, the most relevant algorithms (see e.g. [12]) ignore the memory hierarchy and do not efficiently translate into our settings (a generous analysis would give $\Omega \left( E^{k/2} / B \right)$ I/Os).

A related research topic that has been extensively investigated is the design of algorithms for detecting the existence of a given sample graph and/or for counting the number of its instances (see, e.g., [13] and references therein). However, enumeration cannot be reduced to these results since they usually rely on fast matrix multiplication or sampling.

2 Preliminaries

We study our algorithms in the external memory model [13], which consists of an internal memory of $M$ words and an external memory of unbounded size. The processor can only use data stored in the internal memory and move data between the two memories in blocks of consecutive $B$ words. We suppose each vertex and edge to require one memory word. The I/O complexity of an algorithm is defined as the number of input/output blocks moved between the two memories by the algorithm.

We denote with $G = (V, E)$ the simple and undirected input data graph. For notational convenience and consistency with earlier papers, whenever the context is clear we use $E$ as a shorthand for the size of a set $E$ (and similarly for other sets). We denote with $\text{deg}(v)$ the degree of a vertex $v \in V$. We assume that the elements of $V$ are ordered according to degree, breaking ties among vertexes of the same degree arbitrarily but in a consistent way. We represent the edge set $E$ with adjacency lists stored in consecutive positions of the external memory.

We denote with $H = (V_H, E_H)$ the connected, simple and undirected sample graph that we are looking for in the input graph $G$. Let $k = |V_H|$ and $V_H = \{h_1, \ldots, h_k\}$. An instance of $H$ in $G$ is a tuple $(v_1, \ldots, v_k)$ of $k$ distinct vertexes of $G$ such that $(v_i, v_j) \in E$ for each edge $(h_i, h_j) \in E_H$. An instance is induced if $(v_i, v_j) \in E$ if and only if $(h_i, h_j) \in E_H$. For a given instance we say that vertex $h_i$ (resp., edge $(h_i, h_j)$) is mapped onto $v_i$ (resp., $(v_i, v_j)$). We denote with $s$ the size of the largest independent set $S$ of the sample graph $H$ such that each vertex in $H$ is adjacent to at most one vertex in $S$; clearly, $s \geq 1$. We also assume that $h_{k-s+1}, \ldots, h_k$ denote the vertexes in $S$.

For the sake of simplicity, in this paper we do not exploit automorphisms in the sample graph. An automorphism is a permutation $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$ such that for each edge $(h_i, h_j) \in E_H$ we have $(h_{\pi(i)}, h_{\pi(j)}) \in E_H$. This means that if $(v_1, \ldots, v_k)$ is an instance of $H$ in $G$, then $(v_{\pi(1)}, \ldots, v_{\pi(k)})$ is an instance as well. A technique for exploiting automorphisms in subgraph enumeration is proposed in [10] and uses suitable conjunctive queries on the instances. The same technique can be applied to our algorithms, improving the exponent of the $k^{O(k)}$ term in the I/O complexities. More details will be provided in the full version.

2 Namely, instances are edge-induced subgraphs of $G$, while induced instances are vertex-induced subgraphs of $G$. In both cases, they have to be isomorphic to $H$.  

3
3 Deterministic EM Algorithm

In this section, we present the deterministic algorithm that will be subsequently used for solving subproblems in the randomized algorithm. For the sake of simplicity, we assume that there exists at least one vertex in $V_H/S$ that is not adjacent to vertexes in $S$ or is adjacent to a vertex in $S$ with degree at least two; we denote this vertex with $h_{k-s}$. We will later see how to remove this assumption.

Partition the adjacency lists of $G$ into consecutive chunks $C_i$ of size $M/(4k)$, where $1 \leq i \leq 4kE/M$. A vertex whose adjacency list is completely contained in a chunk is said complete, and incomplete otherwise. Note that each chunk contains at most two incomplete vertexes. Let $\Gamma_i$ be the set containing all vertexes in $V$ whose adjacency lists are completely or partially in $C_i$. We say that an edge $(u, v)$ is in $C_i$ if it appears in the (part of) adjacency list of $u$ or $v$ contained in $C_i$. We let $V_i \subseteq V$ and $E_i \subseteq E$, for each $1 \leq i < k$, denote support vertex and edge sets used in the algorithm.

The algorithm works in $(4kE/M)^{k-s-1}$ rounds where all possible combinations of $k-s-1$ chunks are loaded in memory. Consider a generic round and let $C_{\ell_1}, \ldots, C_{\ell_{k-s-1}}$, for suitable values of $\ell_1, \ldots, \ell_{k-s-1}$ denote the $k-s-1$ chunks loaded in memory. The algorithm fills $V_i$ and $E_i$ for each $1 \leq i < k$ and $i \neq k-s$ with suitable values from the loaded chunks. Then, it enumerates all instances of $H$ in $G$ where vertex $h_i$ is mapped onto a vertex in $V_i$ and edges adjacent to $h_i$ are mapped onto edges in $E_i$, for each $1 \leq i \leq k$ and $i \neq k-s$. The enumeration of these instances advances in $V$ iterations where the “free” vertex $h_{k-s}$ is mapped onto a suitable vertex in $V$. More in details, each round performs the following operations:

1. Load in memory $C_{\ell_1}, \ldots, C_{\ell_{k-s-1}}$, set $V_i$ to $\Gamma_{\ell_i}$ and fill $E_i$ with edges in $C_{\ell_i}$, for each $1 \leq i \leq k-s-1$.

2. Set $V_i$ with $k-s+1 \leq i \leq k$ (i.e., the sets associated with vertexes in the independent set) as follows. Let $\Lambda_i$ be the neighbors of $h_i$ in $H$. Then for each $k-s+1 \leq i \leq k$ and for each $v \in V_i$, vertex $v$ is added to $V_i$ if for each $j$ with $h_j \in \Lambda_i \setminus \{v_{k-s}\}$ there exists a vertex $u_j \in V_j$ such that edge $(u_j, v) \in E_j$ (i.e., add in $\Lambda_i$ all vertexes that can appear as $h_i$ in an instance, according with sets $V_j$ and $E_j$ for each $1 \leq j \leq k-s-1$).

3. Scan the input edge set $E$ and add each edge $(v, v')$ to $E_i$ if $v \in V_i$, $v' \in V_j$. When an edge is added to a set, it is marked late if it was not already in the set. Note that an edge $(v, v')$ may be late in $E_i$ but not in $E_j$.

4. Enumerate all instances of $H$ in $G$ where vertex $h_i$ is mapped onto a vertex in $V_i$ and edges adjacent to $h_i$ are mapped onto edges in $E_i$, for each $1 \leq i \leq k$ and $i \neq k-s$. The enumeration advances in $V$ iterations, and each iteration is organized as follows:

(a) Set $V_{k-s}$ to a suitable vertex $v \in V$.

(b) Scan the adjacency list of $v$ and add each edge $(v, v')$, where $v' \in V_i$ to $E_{k-s}$ and $E_i$. Again, each added edge is marked late if it was not already in the set.

(c) Using a naive approach (see Section 6), enumerate in main memory all instances of $H$ in $G$ where $h_i$ is mapped onto a vertex in $V_i$ and edges adjacent to $h_i$ are mapped onto edges in $E_i$, for each $1 \leq i \leq k$.

(d) Empty sets $V_{k-s}$ and $E_{k-s}$.

5. Empty sets $V_i$ and $E_i$ for each $1 \leq i \leq k$.

An instance $(v_1, \ldots, v_k)$ may be enumerated many times if it contains an incomplete vertex. As example, consider an edge $(v_1, v_2)$, with $v_1 \in \Gamma_1$ incomplete and $v_2 \in \Gamma_2$ complete. This edge is inserted into $E_1$ in $c$ rounds, where $c$ is the number of chunks containing the adjacency list of $v_1$: indeed, it is added any time one of the $c$ chunks is used for filling $E_1$ and $v_2$ is in $V_2$. However, $(v_1, v_2)$ is not marked as late only when the chunk containing this edge is loaded into $E_1$. 


Therefore, we perform the following check in order to emit each instance once. For each vertex \( h_i \) with \( 1 \leq i \leq k - s - 1 \) we define the probe index \( p_i \), with \( 1 \leq p_i \leq k \): if \( h_i \) is adjacent to a vertex \( h_j \) in the independent set \( S \) then \( p_i = j \), otherwise \( p_i \) is set to an arbitrary value \( j \) such that \( h_i \) is adjacent to \( h_j \). Given an instance, the edge where \( (h_i, h_p) \) is mapped is said probe edge of \( h_i \). An instance \( (v_1, \ldots, v_k) \) is actually enumerated (i.e., function \texttt{emit} is called) only if the probe edge \( (v_i, v_p) \) is not late in \( E_i \) for each \( 1 \leq i \leq k - s - 1 \).

**Theorem 1.** The above algorithm correctly enumerates all instances of a given sample graph \( H \) and its I/O complexity is \( O((4k)^{k-s-1} \frac{E_M^{\sqrt{k}}}{M}) \).

**Proof.** We first prove the correctness of the algorithm. Consider an instance \( (v_1, \ldots, v_k) \), and recall that \( p_i \) is the probe index of \( h_i \). Let \( C_{\ell_i} \) be the chunk containing \( (v_i, v_p) \) with \( v_i \in \Gamma_i \), for each \( 1 \leq i \leq k - s - 1 \). Consider the unique round where chunks \( C_{\ell_1}, \ldots, C_{\ell_{k-s-1}} \) are loaded in memory in this order. Then, for each \( 1 \leq i \leq k - s - 1 \), we have that \( v_i \in V_i \) \((v_i, v_p) \in E_i \) and \((v_i, v_p) \) is not marked as late. Also we have \( v_p \in V_p \). Indeed for each \( j \) such that \( h_j \in \Delta_p / h_{k-s} \), we have that \( p_j = p_i \), \((v_j, v_p) \in E_j \) and hence, by construction, \( v_p \in V_p \). The given instance is then enumerated in the unique (within the given round) iteration where \( V_{k-s} \) is set to \( v_{k-s} \).

Note that all edges of the instance are in the right edge set since they are added during the scan of the whole edge list \( E \) (step 3) and of the adjacency list of \( v_{k-s} \) (step 4.b). The instance is enumerated once since edges \((v_i, v_p)\) for each \( 1 \leq i < k - s \) are all not marked as late only when chunks \( C_{\ell_1}, \ldots, C_{\ell_{k-s-1}} \) are loaded in memory in this order (a different order may enumerate an automorphism but not the same instance).

We now analyze the I/O complexity of the algorithm. The cost for enumerating instances in step 4.c is negligible since the problem fits in memory and all operations are performed in main memory. Indeed, the space required by the \( k \) sets \( E_i \) is at most \( M \), including late vertexes (this is a consequence of the constant \( 4 \) in the chunk size \( M/(4k) \)). (Note that sets \( V_i \) are not actually created, and chunks \( C_{\ell_i} \) can be removed from the internal memory after step 1). Then the I/O complexity of each round is asymptotically upper bounded by a constant number of scans of the whole edge list. Since there are \((4kE/M)^{k-s-1}\) rounds, the claimed I/O complexity follows.

We note that the deterministic algorithm performs no writes on the external memory. It can thus run on a read-only disk with the same I/O complexity.

At the beginning we made the assumption that \( h_{k-s} \) is not adjacent to edges in the independent set \( S \), or it is adjacent to a vertex in \( S \) with degree two. This assumption is required so that the values of vertexes in \( S \) is given by the values in \( E_1, \ldots, E_{k-s-1} \). Suppose now that vertex \( h_{k-s} \) is adjacent to a vertex \( h_j \in S \) with degree one. This case can be solved as follows. Consider a generic iteration where \( h_{k-s} \) is set to \( v \). The algorithm first enumerates all instances without \( h_j \) and then sets \( h_j \) to each vertex adjacent to \( v \) which is not in the instance. This requires an additional scan of the adjacency list of \( v \) and does not asymptotically increase the I/O complexity.

### 4 Randomized EM Algorithm

We are now ready to introduce the randomized algorithm. The algorithm, by making use of the random coloring technique in [3], decomposes the problem into small subproblems of expected size \( O(M) \), which are then solved with the previous deterministic algorithm. We first prove the expected I/O complexity and then show how to get the high probability.

Let \( \xi : V \rightarrow \{1, \ldots, c\} \), with \( c = \sqrt{E/M} \), be a vertex coloring chosen uniformly at random from a family of \( 2k \)-wise independent family of functions. The coloring \( \xi \) partitions the edge set \( E \) into \( c^2 \) sets of expected size \( M \). For each pair of colors \( \tau_1, \tau_2 \in \{1, \ldots, c\} \) and \( \tau_1 \leq \tau_2 \), we denote with \( E_{\tau_1, \tau_2} \) the set containing edges colored with \( \tau_1 \) and \( \tau_2 \), that is \( E_{\tau_1, \tau_2} = \{(u, v) \in E \mid \min\{\xi(u), \xi(v)\} = \tau_1, \max\{\xi(u), \xi(v)\} = \tau_2\} \). Each instance \( (v_1, \ldots, v_k) \) of the sample graph can be colored by \( \xi \) in \( c^k \) ways, and it said to be \((\tau_1, \ldots, \tau_k)\)-colored if \( \xi(v_i) = \tau_i \) for each \( 1 \leq i \leq k \).
The randomized algorithm enumerates all instances by decomposing the problem into \( c^k \) subproblems. Each subproblem finds all \((\tau_1, \ldots, \tau_k)\)-colored instances according with a given \( k \)-tuple of colors using the previous deterministic algorithm on the edge set \( \bigcup_{r_i \leq t} E_{\tau_i, \tau_i} \). The algorithm is organized as follows:

1. Random select a coloring \( \xi \) from a \( 2k \)-wise independent family of functions.
2. Using sorting, store edges in \( E_{\tau_1, \tau_2} \) in consecutive positions, for each color pair \((\tau_1, \tau_2)\).
3. For each \( k \)-tuple of colors \((\tau_1, \ldots, \tau_k)\), enumerate all \((\tau_1, \ldots, \tau_k)\)-colored instances using the algorithm in Section 3 on the sets \( E_{\tau_i, \tau_i} \), for each \( \tau_i \leq \tau_j \).

We first prove a technical lemma that upper bounds the expected number \( X_t \) of possible tuples of \( t \) edges in \( E \) that are colored in the same way by \( \xi \). It is easy to see that a closed form of this quantity is \( X_t = \sum_{\tau_1 \leq \tau_2, E_{\tau_1, \tau_2} \geq t} E_{\tau_1, \tau_1}^{-1} \).

**Lemma 1.** Let \( \xi : V \to \{1, \ldots, c\} \) be chosen uniformly at random from a \( 2k \)-wise independent family of hash functions, where \( c = \sqrt{E/M} \). If \( M = \Omega(t^2) \) and the maximum vertex degree in \( G \) is \( \sqrt{EM} \), then \( \mathbb{E}[X_t] \leq (2t)^{-1} EM^{t-1} \).

**Proof.** We prove the claim by induction on \( t \). The claim is verified for \( t = 1 \) since \( \mathbb{E}[X_1] = E \). For each tuple \( e = (e_1, \ldots, e_t) \) of \( t \) distinct edges in \( E \), and for each \( 2 \leq i \leq t \), let \( Y^e_t = 1 \) if \( e_i \) is in the same set \( E_{\tau_i, \tau_i} \), for some colors \( \tau_1, \tau_2 \), of edges \( e_1, \ldots, e_{i-1} \), and 0 otherwise. Set \( Y^e_1 = 1 \). We get \( X_t = \sum_{e} Y^e_t \). By the \( 2k \)-wiseness of \( \xi \), we get

\[
\Pr(Y^e_t = 1) \leq \begin{cases} 
\Pr(Y^e_{t-1} = 1)/c^2 & \text{if } e_t \text{ is not adjacent to } e_1 \ldots e_{t-1} \\
\Pr(Y^e_{t-1} = 1)/c & \text{if } e_t \text{ is adjacent to } e_1 \ldots e_{t-1} \text{ on one vertex} \\
\Pr(Y^e_{t-1} = 1) & \text{if } e_t \text{ is adjacent to } e_1 \ldots e_{t-1} \text{ on two vertexes}
\end{cases}
\]

Each \((t-1)\)-tuple \( e' \) can be extended by at most \( E \) edges that are not connected with \( e' \), or by \( 2(t-1)\sqrt{EM} \) edges that are connected to \( e' \) on just one vertex (recall that the maximum degree of a vertex is \( \sqrt{EM} \)), or by \((t-1)(2t-3)\) edges that are connected to \( e' \) on two vertexes. Therefore, we get

\[
\mathbb{E}[X_t] = \sum_{e} \Pr(Y^e_t = 1) \leq \mathbb{E}[X_{t-1}] \left( \frac{E}{c^2} + 2(t-1)(2t-3) \right).
\]

Since the right term is upper bounded by \( 2tM\mathbb{E}[X_{t-1}] \), the lemma follows.

By summing the I/O costs of the \( c^k \) subproblems and by exploiting Lemma 1 we get the following theorem.

**Theorem 2.** The above randomized algorithm enumerates all instances of a given sample graph \( H \). If the maximum vertex degree of \( G \) is \( \sqrt{EM} \), then the expected I/O complexity of the algorithm is \( O\left( k^{3(k-s-1)} \frac{k^{k/2}}{2tM^{t-1}}\right) \).

**Proof.** The correctness easily follows since each instance is colored with a suitable color tuple \((\tau_1, \ldots, \tau_k)\) and is enumerated only in the subproblem associated with this color tuple. The cost of each subproblem is given by Theorem 1. The I/O complexity \( Q(E) \) of the algorithm is upper
bounded by the sum of the costs of all \( c^k \) subproblems. Then,

\[
Q(E) = \mathcal{O}\left(\frac{(4k)^{k-s-1}}{BM^{k-s-1}} \sum_{(\tau_1, ..., \tau_k) \cap \tau_i \leq \tau_j} \left( \sum_{\tau_i \leq \tau_j} E_{\tau_i, \tau_j} \right)^{k-s} \right)
\]

\[
\leq \mathcal{O}\left(\frac{(4k)^{2(k-s-1)}}{BM^{k-s-1}} \sum_{(\tau_1, ..., \tau_k) \cap \tau_i \leq \tau_j} E_{\tau_i, \tau_j}^{k-s} \right)
\]

\[
\leq \mathcal{O}\left(\frac{e^{k-2}(4k)^{2(k-s-1)}}{BM^{k-s-1}} \sum_{\tau_1 \leq \tau_2} E_{\tau_1, \tau_2}^{k-s} \right)
\]

\[
\leq \mathcal{O}\left(\frac{e^{k-2}k^{3(k-s-1)}}{BM^{k-s-1}} \sum_{\tau_1 \leq \tau_2, \tau_1, \tau_2 \geq t} \frac{E_{\tau_1, \tau_2}!}{(E_{\tau_1, \tau_2} - k + s)!} \right).
\]

By the linearity of expectation and by Lemma [1], we get the claimed result. □ □

We remark that our deterministic algorithm is crucial for getting the claimed I/Os complexity. Indeed, the algorithm used in the subproblems should require \( \mathcal{O}(M/B) \) I/Os for solving subproblems of size \( \Theta(M) \) (note that subproblems may not perfectly fit the memory size). Using existing enumeration algorithms, which require \( \Omega(M^{k/2}/B) \) I/Os for solving subproblems of size \( \Theta(M) \), would increase the total I/O complexity by a multiplicative factor \( \Omega(M^{k/2-1}) \).

4.0.3 Getting the high probability.

If \( M = \Omega(\sqrt{E} \log E) \) and the maximum degree is \( \sqrt{E}M \), the randomized coloring process can be slightly modified to get the claimed I/O complexity with probability \( 1 - 1/E \). A vertex \( v \in V \) has high degree if \( \sqrt{E} \leq \deg(v) \leq \sqrt{E}M \), and has low degree if \( \deg(v) < \sqrt{E} \). The coloring process is modified as follows. The colors of low degree vertexes are assigned independently and uniformly at random. The colors of high degree vertexes are set by partitioning vertexes into \( c \) groups so that the sum of degrees within each group is in the range \([\sqrt{E}M, 2\sqrt{EM}]\), and then high degree vertexes within the \( i \)-th group get color \( i \) (this operation can be performed in \( \mathcal{O}(1) \) sorts).

Our argument relies on the technique by Janson [15, Theorem 2.3] for obtaining a strong deviation bound for sums of dependent random variables, which we recall here for completeness. Let \( X = \sum_{i=1}^{p} Y_i \) where each \( Y_i \) is a random variable with \( Y_i - E[Y_i] \leq 1 \), and let \( R = \sum_{i=1}^{p} \text{Var}(Y_i) \). Denote with \( \Delta \) the maximum degree of the dependency graph of \( Y_1, ..., Y_p \): this is a graph with vertex set \( Y = \{1, ..., p\} \) such that if \( B \subset Y \) and \( i \notin \text{connected to a vertex in } B \), then \( Y_i \) is independent of \( \{Y_j\}_{j \in B} \). Then, for any \( d > 0 \), we have \( \Pr(X \geq (1 + d)E[X]) \leq e^{-\frac{R}{2\sigma^2(1 + 2d)^2}} \).

It deserve to be noticed that it is possible to color low degree vertexes with a coloring from a \( 2k \)-wise independent family and still get the claimed I/O complexity with probability \( 1 - 1/E^4 \), for \( 0 \leq \epsilon \leq 1/4 \), as soon as \( M \geq E^{3/4+\epsilon} \). It suffices to use a technique by Gradwoh and Yehudayo [16, Corollary 3.2] in our argument instead of the aforementioned result by Janson [15, Theorem 2.3].

Theorem 3. Let \( M = \Omega(\sqrt{E} \log E) \) and let the maximum vertex degree of \( G \) be \( \sqrt{EM} \). Then, with probability at least \( 1 - 1/E \), the I/O complexity of the above algorithm is \( \mathcal{O}\left(k^{3(k-s-1)} \frac{E^{k/2}}{BM^{k-s-1}} \right) \).

Proof. Let \( E^L \) be the set of edges in \( E \) connecting two low degree vertexes. We also define \( E^M = E/E^L, E^L_{\tau_1, \tau_2} = E_{\tau_1, \tau_2} \cap E^L, E^H_{\tau_1, \tau_2} = E_{\tau_1, \tau_2} \cap E^H \). We first show that the size of \( E^L_{\tau_1, \tau_2} \) for any color pair \( \tau_1, \tau_2 \) is smaller than \( 2M \) with probability at least \( 1 - 1/(2E) \). Assume for simplicity that \( |E^L| = |E| \). For each edge \( e \in E^L \), define the random variable \( Y_e \) to be 1 if edge \( e \) is in \( E^L_{\tau_1, \tau_2} \), and 0 otherwise. We thus have \( E^L_{\tau_1, \tau_2} = \sum_{e \in E} Y_e \). Each random variable \( Y_e \) depends on the at most \( 2\sqrt{E} \) variables associated with edges adjacent to \( e \), while it is independent
of the remaining ones. Since $Y_e - \mathbb{E}[Y_e] < 1$, we use the aforementioned result by Janson by setting $p = E$, $\mathbb{E}[E^L_{r_1,r_2}] = M$, $R = E(1/e^2 - 1/c^4) < M$, $d = 1$, $\Delta = 2\sqrt{E}$. Then we get $\Pr \left( E^L_{r_1,r_2} \geq 2M \right) \leq e^{-\frac{3M}{8\sqrt{E}}}$. The probability that $E^L_{r_1,r_2}$ is smaller than $16M$ for every color pair is at least $1 - c^2e^{-\frac{25M}{32\sqrt{E}}} \geq 1 - 1/(2E)$ when $M = \Omega \left( \sqrt{E \log E} \right)$.

We now show that the set $E^H_{r_1,r_2}$ has size $8M$ with probability at least $1 - 1/(2E)$. There are at most $2\sqrt{M}$ high degree vertexes colored with a given color. Then, there cannot be more than $4M$ edges connecting two high degree vertexes in $E^H_{r_1,r_2}$. Consider now the set $E^{H*}$ of edges connecting high degree vertexes of colors $r_1$ or $r_2$ to low degree vertexes. We have $E^{H*} \leq 4\sqrt{EM}$. For each $e \in E^{H*}$, define the random variable $Y_e$ to be 1 if the low degree vertex gets color $r_1$ or $r_2$, and 0 otherwise. We have $E^{H*} \leq \sum_{e \in E^{H*}} Y_e$. Since random variables may be dependent, we apply again the result by Janson with $p = 4\sqrt{EM}$, $\mathbb{E}[E^{H*}] = 8M$, $R = \sum_{e \in E^{H*}} Var(Y_e) \leq 8M$, $d = 1$, $\Delta = 2\sqrt{E}$ (since only low degree vertexes are randomly colored). Then, $\Pr \left( E^{H*}_{r_1,r_2} \geq 16M \right) \leq e^{-\frac{25M}{32\sqrt{E}}}$. Then, the probability that $E^{H*}_{r_1,r_2}$ is smaller than $16M$ for every color pair is at least $1 - c^2e^{-\frac{25M}{32\sqrt{E}}} \geq 1 - 1/(2E)$ when $M = \Omega \left( \sqrt{E \log E} \right)$.

Therefore, we have that each $E_{r_1,r_2}$ has size at most $16M$ with probability at least $1 - 1/E$. Since each subproblem receives at most $k^2$ edge sets, the I/O complexity of a subproblem is $O \left( \frac{2^{6(k-s)}}{k^{2(k-s)-1}} M/B \right)$. Since there are $c^k$ subproblems and $2^{6(k-s)} = O \left( k^{2(k-s)-1} \right)$, the claimed I/O complexity follows.

5 Lower Bound on I/O Complexity

In this section we provide a lower bound to the I/O complexity of any algorithm that enumerates $T$ instances. We suppose that the sample graph belongs to a particular class of graphs, which has been named Alon class by Afrati et al. [4]. A graph in the Alon class has the property that vertexes can be partitioned into disjoint sets such that the subgraph induced by each partition is either a single edge, or contains an odd-length Hamiltonian cycle. As in previous works [8,5] on triangle enumeration, we assume that the information on an edge is atomic and requires at least one memory word. This assumption is similar to the indivisibility assumption which is common in lower bounds on the I/O complexity.

Theorem 4. For any input graph, an algorithm that enumerates $T$ distinct instances of a sample graph $H$ in the Alon class requires, even in the best case, $\Omega \left( T/(BM^{k/2-1}) + T^{2/k}/B \right)$.

Proof. The proof mimics the argument in [5] for triangle enumeration, but exploits the fact that there cannot be more than $\Theta \left( m^{k/2} \right)$ instances of a subgraph in the Alon class in a graph of $m$ edges [17]. The execution of an algorithm on a memory of size $M$ can be simulated, without increasing the I/O complexity, in a memory of size $2M$ so that the computation proceeds in rounds. In each round (with the possible exception of the last round) there are $\Theta \left( M/B \right)$ I/Os, and memory blocks are read from (resp., written on) the external memory only at the begin (resp., end) of a round. (We refer to [5] for more details on the simulation.) By the aforementioned result on the Alon class, $\Theta \left( M^{k/2} \right)$ instances can be enumerated in a round since there are at most $2M$ edges in memory. Then, there must be at least $\lceil T/\Theta \left( M^{k/2} \right) \rceil$ rounds. Since each round needs $\Theta \left( M/B \right)$ I/Os, we get the first part of the claim. The second term follows since $\Omega \left( T^{2/k} \right)$ input edges must be read to enumerate $T$ distinct instances.

The above lower bound shows that our randomized algorithm is optimal when $k = O \left( 1 \right)$. Indeed, if the data graph is a complete graph with $\sqrt{E}$ vertexes, there exist $T = \Theta \left( E^{k/2} \right)$ instances of any sample graph with $k$ vertexes.

Note that this is not the case if low degree vertexes were colored with $2k$-wise independent hash functions.
6 Extensions

Enumeration of Induced Subgraphs.

The deterministic and randomized algorithms can be easily adapted to enumerate all induced instances of a given subgraph. The I/O complexity of the deterministic algorithm does increase asymptotically, while the I/O complexity of the randomized algorithm shows only a small increase in the exponent of the term $k^{\mathcal{O}(k)}$. It suffices to run the deterministic algorithm as the subgraph was a $k$-clique (then $s = 1$). In each iteration, the algorithm contains all edges in $E$ between any pair of vertexes in $\bigcup_{i=1}^{s-1} V_i$. Then, all instances of $H$ are found, but only induced instances are enumerated. This is possible since all edges between vertexes in the instance are available in memory. The I/O complexity of the algorithm then becomes $\mathcal{O}\left((4k)^{k-2}E^{k-1}/(BM)^{k-2}\right)$. By using this algorithm for solving subproblems in the randomized algorithm, we get a new algorithm for induced subgraphs requiring $\mathcal{O}\left(k^{3(k-2)}E^{k/2}/(BM^{k/2-1})\right)$ I/Os, assuming that the maximum vertex degree is $\sqrt{EM}$. The high probability result applies as well.

Enumerating $k$-cliques in graphs with large degree.

Although the assumption in the randomized algorithm that the maximum degree is at most $\sqrt{EM}$ is reasonable for real datasets, it can be easily removed when the subgraph is a $k$-clique using the following recursive approach.

1. (Base case) If $k = 3$ enumerate all triangles using the cache-aware algorithm in [5] and return.

2. Enumerate all $k$-cliques involving only low degree vertexes using the randomized algorithm in Section 4.

3. Let $V_H = \{v_1, \ldots, v_r\}$, with $r \leq 2\sqrt{E/M}$, denote the set of vertexes with degree larger than $\sqrt{EM}$. For each $v \in V_H$ set $h_1 = v$, and perform the following operations

   (a) Enumerate recursively all $(k-1)$-cliques on the subgraph of $G$ obtained by removing $v$ and all vertexes not adjacent to $v$.

   (b) For each $(k-1)$-clique $(v_1, \ldots, v_{k-1})$ emitted in the previous step[3] emit the $k$ instances obtained by the $k$ ways to insert $v$ into $(v_1, \ldots, v_{k-1})$.

The correctness of the algorithm can be easily derived (note that vertexes in the graph used in the recursive call are connected to $h_1$). The I/O complexity of this algorithm is a factor $\mathcal{O}(k^2)$ larger than the one of the randomized algorithm.

Work Complexity.

Assume that $k = \mathcal{O}(1)$. The work of the deterministic algorithm is $\tilde{\mathcal{O}}\left(E^{k-s}\right)$ if the enumeration of instances with vertexes in $V_i$, for each $1 \leq i \leq k$, in Step 4.c is performed in $\tilde{\mathcal{O}}\left(M^{k-s-1}\right)$ work. Step 4.c can be implemented using a brute force approach that considers all the ways to select the $k - s - 1$ probe edges using edges in $E_1, \ldots, E_{k-s-1}$ (recall that $v_{k-s}$ is fixed within an iteration and that probe edges fix the remaining $k - 1$ vertexes) and then checks if the required edges do exist. Using this implementation, the expected work of the randomized algorithm is $\tilde{\mathcal{O}}\left(E^{k/2}M^{k/2-s}\right)$.

The work can be notable improved for $k$-cliques (then, $s = 1$) by exploiting the enumeration algorithm in [11]. This algorithm can be easily adapted to solve Step 4.c in $\tilde{\mathcal{O}}\left(M^{k/2}\right)$ work. The total work of the deterministic algorithm then becomes $\tilde{\mathcal{O}}\left(E^{k-1}/M^{k/2-1}\right)$. As a consequence, the work complexity of the randomized algorithm reduces to $\tilde{\mathcal{O}}\left(E^{k/2}\right)$. This result is just a polylog factor from the optimum since a clique of $\sqrt{E}$ vertexes contains $\Theta\left(E^{k/2}\right)$ $k$-cliques.

[3] We observe that the step can be performed without storing the $(k-1)$-cliques.
7 Conclusion

In this paper we have seen upper and lower bounds to the I/O complexity for the enumeration of all instances of a given sample graph with \( k \) vertexes. In particular, we have given a randomized algorithm requiring \( O \left( \frac{E^{k/2}}{(BM^{k/2} - 1)} \right) \) expected I/Os for \( k = O(1) \). A nice property of this algorithm is that it decomposes the problem into a large number of independent subproblems. It is then easy to derive an optimal parallel algorithm for multicore with a memory hierarchy without increasing the total I/O complexity. Moreover, the algorithm can be extended to the MapReduce framework, allowing to tradeoff replication and space requirements as done for matrix multiplication in [IS 4].

It deserves further investigations the analysis in high probability of the randomized algorithm in order to reduce the amount of required randomness and of internal memory size. Moreover, an interesting open problem is to design memory-efficient algorithms for subgraph enumeration that are output sensitive and are tight with respect to the best-case lower bound provided in this paper.

Acknowledgments. The author would like to thank Rasmus Pagh for useful discussions.

References

[1] Kairam, S.R., Wang, D.J., Leskovec, J.: The life and death of online groups: Predicting group growth and longevity. In: Proc. 5th WSDM. (2012) 673–682

[2] Gregori, E., Lenzini, L., Mainardi, S.: Parallel \( k \)-clique community detection on large-scale networks. IEEE Trans. Paral. Dist. Systems 24(8) (2013) 1651–1660

[3] Grochow, J.A., Kellis, M.: Network motif discovery using subgraph enumeration and symmetry-breaking. In: Proc. 11 RECOMB. (2007) 92–106

[4] Afrati, F.N., Sarma, A.D., Salihoglu, S., Ullman, J.D.: Upper and lower bounds on the cost of a Map-Reduce computation. Proc. VLDB Endow. 6(4) (2013) 277–288

[5] Pagh, R., Silvestri, F.: The input/output complexity of triangle enumeration. arXiv:1312.0723 (2013)

[6] Goodrich, M., Pszona, P.: External-memory network analysis algorithms for naturally sparse graphs. In: Proc. 19th ESA. Volume 6942 of LNCS. (2011) 664–676

[7] Chu, S., Cheng, J.: Triangle listing in massive networks. ACM Trans. Knowl. Discov. Data 6(4) (2012) 17:1–17:32

[8] Hu, X., Tao, Y., Chung, C.W.: Massive graph triangulation. In: Proc. ACM SIGMOD. (2013) 325–336

[9] Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: Proc. 20th ACM International Conference on World. (2011) 607–614

[10] Afrati, F.N., Fotakis, D., Ullman, J.D.: Enumerating subgraph instances using Map-Reduce. In: Proc. 29th ICDE. (2013) 62–73

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

[12] Ngo, H.Q., Porat, E., Ré, C., Rudra, A.: Worst-case optimal join algorithms. In: Proc. 31st PODS. (2012) 37–48

[13] Kolountzakis, M.N., Miller, G.L., Peng, R., Tsourakakis, C.E.: Efficient triangle counting in large graphs via degree-based vertex partitioning. Internet Mathematics 8(1-2) (2012) 161–185
[14] Vitter, J.S.: Algorithms and Data Structures for External Memory. Now Publishers Inc., Hanover, MA, USA (2008)

[15] Janson, S.: Large deviations for sums of partly dependent random variables. Random Structures & Algorithms 24(3) (2004) 234–248

[16] Gradwohl, R., Yehudayoff, A.: t-wise independence with local dependencies. Information Processing Letters 106(5) (2008) 208 – 212

[17] Alon, N.: On the number of subgraphs of prescribed type of graphs with a given number of edges. Israel Journal of Mathematics 38(1-2) (1981) 116–130

[18] Pietracaprina, A., Pucci, G., Riondato, M., Silvestri, F., Upfal, E.: Space-round tradeoffs for mapreduce computations. In: Proc. 26th ICS. (2012) 235–244