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, $k = \mathcal{O}(1)$ 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 $\mathcal{O}\left(E^{k/2}/\left(BM^{k/2-1}\right)\right)$ 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, in the worst-case, when the sample graph belongs to the Alon class, which includes cliques, cycles and every graph with a perfect matching: indeed, we show that any algorithm enumerating T instances must always use $\Omega\left(T/\left(BM^{k/2-1}\right)\right)$ I/Os and there are graphs for which $T = \Omega\left(E^{k/2}\right)$. Finally, we propose a parallelization of the randomized algorithm in the MapReduce framework.

*Part of this work was done while the author was visiting the IT University of Copenhagen. It is supported in part by University of Padova project CPDA121378, MIUR of Italy project AMANDA, Danish National Research Foundation Sapere Aude program.

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 [15], the dynamics of Internet at the level of autonomous systems [11], the significance of motifs in biological networks [12]. Typical sample graphs are cliques (e.g., 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, $k = \mathcal{O}(1)$ 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 $\mathcal{O}\left(E^{k/2}/\left(BM^{k/2-1}\right)\right)$ 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 strongly 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 [2]. 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 $\Omega\left(T/\left(BM^{k/2-1}\right) + T^{2/k}/B\right)$ I/Os, and then the randomized algorithm is optimal in the worst case since there exist graphs where $T = \Omega\left(E^{k/2}\right)$.

The randomized algorithm decomposes the enumeration problems into small independent subproblems and can hence be easily parallelized in shared and distributed-memory models. However, each input edge is required in a large number of subproblems and this fact may significantly affect the performance of distributed-memory machines, where concurrent read is not defined and each edge has to be actually replicated. A naive parallelization of subproblems among the available processing elements may significantly increase the amount of data that are sent over the network in a short time interval, potentially arising performance issues related to the congestion of the network. Since the MapReduce framework has been widely adopted for designing large scale computation in distributed-memory platforms, we target this problem by proposing a MapReduce algorithm which trades the round number R with the maximum size λ of a reducer and the total amount of communicated data Λ: for any $\lambda \geq 1$ and $\Lambda \geq E$, there exists a MapReduce algorithm requiring $\mathcal{O}\left(E^{k/2}/(\Lambda\lambda^{k/2-1})\right)$ rounds and where each edge is replicated $\mathcal{O}(\Lambda/E)$ times in each round.[1]

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 `emit`$(\cdot)$ with the edges of the instance as input parameters. 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 by increasing the I/O complexity of an unavoidable additive $\Theta(T/B)$ factor, where T is the number of instances in the graph.

Algorithms in this paper are presented for the enumeration of edge-induced subgraphs which

[1] A similar issue may arise even in shared-memory models with limited concurrent reads. An algorithm targeting this issue can be derived by adapting the MapReduce algorithm to a suitable shared-memory model like the parallel external memory model [3].

are isomorphic to the sample graph. However, we claim that our algorithms can be extended to the enumeration of vertex-induced subgraphs isomorphic to the sample graph.

Previous 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. [9] gave an optimal algorithm requiring $\mathcal{O}(\text{sort}(E))$ I/Os for graphs with constant arboricity; this algorithm however does not efficiently scale with larger arboricity. Chu et al. [6] and Hu et al. [13] proposed algorithms for a generic graph incurring $\mathcal{O}\left(E^2/(BM)\right)$ I/Os. Pagh and Silvestri [19] have recently improved this bound by proposing cache-aware and cache-oblivious algorithms, based on a random vertex coloring technique, that exhibit an optimal $\Theta\left(E^{3/2}/\left(B\sqrt{M}\right)\right)$ expected I/O complexity.

The enumeration of subgraphs has also been targeted in other models. A one-round algorithm for enumeration in the parallel MapReduce framework has been target by Afrati et al. [2] and by Suri and Vassilvitskii [23] for triangles, by Afrati et al. [1] for general sample graphs, and by Finocchi et al. [8] for small cliques. The last paper does not translate into an I/O efficient algorithm since subproblem size cannot be tuned to fit internal memory size (unless $M = E$). On the other hand, the first three papers are based on partitioning techniques similar to the one used by our randomized algorithm, however these works assume a random input and provide weak bounds in our worst-case scenario. Park et al. [20] has recently proposed a MapReduce algorithm for triangle enumeration that exhibits a tradeoff between round number and the communication in each round.

The enumeration of k-cliques, for a given $k \geq 3$, has been targeted by Chiba and Nishizeki [5] 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 (e.g., [18]) 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., [17] and references therein). However, enumeration cannot be reduced to these results since they usually rely on fast matrix multiplication or sampling.

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. (All proofs are available in [22].)

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 $\mathcal{O}\left((4k)^{k-s-1}E^{k-s}/(BM^{k-s-1})\right)$. As an example assume $k = \mathcal{O}(1)$: we get $\mathcal{O}\left(E^{k-1}/(BM^{k-2})\right)$ if the sample graph is a k-clique ($s = 1$), and $\mathcal{O}\left(E^{2k/3}/(BM^{2k/3-1})\right)$ if the sample graph is a path of even length k ($s = k/3$). 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 [19] for decomposing the problem into smaller subproblems that are solved with the previous deterministic algorithm. Its expected I/O complexity is $\mathcal{O}\left(k^{3(k-s-1)}E^{k/2}/\left(BM^{k/2-1}\right)\right)$ 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 $M = \Omega\left(\sqrt{E}\log E\right)$ 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, $\Omega\left(T/\left(BM^{k/2-1}\right)+T^{2/k}/B\right)$ 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 $\sqrt{E}$ vertexes) containing $T=\Theta\left(E^{k/2}\right)$ instances of the sample graph.

4. In Section 6, we propose a parallelization of the randomized algorithm in the MapReduce framework [7]. The MapReduce algorithm exhibits a tradeoff among round number, total amount of shuffled data (i.e., total communication) and reducer size. For $k=\mathcal{O}(1)$ and for any $\lambda \geq 1$ and $\Lambda \geq E$, the algorithm requires $\Theta\left(E^{k/2}/(\Lambda\lambda^{k/2-1})\right)$ rounds; in each round the total amount of shuffled data is $\mathcal{O}(\Lambda)$ and the expected reducer size is $\mathcal{O}(\lambda)$. For given values of Λ and λ, the round number is optimal if the subgraph is in the Alon class.

5. In Section 7, we discuss some extensions: we upper bound the running time of our algorithms; we explain how to enumerate induced subgraphs using our algorithms; we remove the assumption on the maximum degree for the randomized algorithm. The last result is derived assuming the sample graph to be a k-clique, but we conjecture that similar ideas apply to other sample graphs as well.

For the sake of simplicity, in this paper we do not exploit automorphisms in the sample graph. A technique for exploiting automorphisms in subgraph enumeration is proposed in [1] and can be applied to our algorithms, improving the exponent of the $k^{\mathcal{O}(k)}$ term in the I/O complexities.

Comparison with previous work. This paper extends to a generic sample graph the upper and lower bounds proposed by Pagh and Silvestri in [19] for triangle enumeration. Our randomized algorithm builds on their randomized coloring technique and includes their cache-aware algorithm when the sample graph is a triangle. However, this paper substantially differs from [19] since novel and non-trivial results are required for getting the generalization. Besides specific technicalities required for the generalization (e.g., Lemma 1, Theorem 4), we give a new deterministic algorithm for solving small subproblems generated by the coloring technique, and we show that the I/O complexity of the randomized algorithm holds even with high probability. In the special case where the sample graph is a triangle, our deterministic algorithm recalls the one proposed in [13], but it does not need to manage in a different way vertexes of the data graph with degree $\leq M$ and with degree $> M$. Finally, our MapReduce algorithm includes as special cases the algorithms in [20, 23, 1]. In particular, the starting point of our algorithm is the tradeoff between round number and shuffled data in [20] for enumerating triangles, which we extend to arbitrary subgraphs using a novel and general technique for distributing subproblems among rounds.

2 Preliminaries

Models. We study our algorithms in the external memory model, which has been widely adopted in the literature (see, e.g., the survey by Vitter [24]). The model 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.

The parallelization of the randomized algorithm is carried out in the MapReduce framework [7] using the computational model in [21]. A MapReduce algorithm proceeds in rounds and each round is defined as follows: the input of each round is a multiset of key-value pairs; the input pairs are grouped by key in the *shuffle* operation, and each group of pairs with the same key is given to a

reducer which performs a sequential computation and outputs a new multiset of pairs; the output of a round is the union of the output of all reducers and can be used as input in the subsequent round.[2] In each round, an algorithm should use $\mathcal{O}(\lambda)$ memory words per reducer and the total amount of data in the shuffle operation should be $\mathcal{O}(\Lambda)$, where $\lambda \geq 1$ and $\Lambda \geq E$ are given parameters. Since the shuffle in each round is an expensive operation, an efficient MapReduce algorithm should reduce the number of rounds under the constraints given by λ and Λ.

Notation. 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 set E (and similarly for other sets). We denote with $\deg(v)$ the degree of a vertex $v \in V$. We assume that the sizes of V and E are known, that all vertexes in V are labeled with an unique identifier, and that the edge set E is represented with adjacency lists that are stored in consecutive positions of the external memory and sorted by identifier. We observe that these assumptions can be guaranteed by suitably sorting and scanning the input edges without asymptotically affecting the I/O complexity of our algorithms.

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 $(h_i, h_j) \in E_H$.[3] 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)). An instance is enumerated by calling a function `emit`$(\cdot)$ with the edges of the instance as input parameters; a call to `emit`$(\cdot)$ performs no I/Os and requires $\mathcal{O}(1)$ operations. 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 let $h_{k-s+1}, \ldots, h_k$ denote the vertexes of H in S.

3 Deterministic EM Algorithm

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.

The high-level description of the algorithm is the following. The adjacency lists of E are partitioned into $\Theta(kE/M)$ chunks containing $\Theta(M/k)$ edges. The algorithm works in rounds: in each round a combination of $k - s - 1$ chunks, for any possible combination, are loaded in internal memory and the algorithm enumerates all instances where vertex h_i, for each $1 \leq i \leq k - s - 1$, is mapped onto vertexes of G whose adjacency lists are in the i-th chunk. The enumeration within each round progresses in iterations: in each iteration, the algorithm enumerates all the instances required by the round assuming that h_{k-s} is set to a vertex $v \in V$, for any possible value of V. Within each iteration, the algorithm exploits the fact that the possible values onto which the vertexes $h_{k-s+1} \ldots, h_k$ (i.e., in the independent set S) can be mapped, given the values of $h_0, \ldots, h_{k-s}$, are significantly reduced and a scan of the adjacency list of v (and the chunks in memory) suffices for the enumeration. A more detailed explanation follows.

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 called *complete*, and

[2] The original MapReduce framework [7] also uses *mappers* for mapping each input pair into a new set of pairs. However, mappers are ignored in the computational model since they can be replaced by reducers without asymptotically affecting the round number [16].

[3] Namely, *instances* are edge-induced subgraphs of G, while *induced instances* are vertex-induced subgraphs of G.

incomplete otherwise. Note that each chunk contains at most two incomplete vertexes. Let Γ_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 and E_i, for each $1 \leq i < k$, denote suitable vertex and edge sets that will be defined within each round: V_i (resp. E_i) will contain the vertexes (resp., edges) of the data graph where vertex h_i of the sample graph (resp., edges adjacent to h_i) will be mapped onto in the round.

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 $\ell_1, \ldots, \ell_{k-s-1}$, be the $k-s-1$ chunks in memory. Each round performs the following operations:

1. Load in memory $C_{\ell_1}, \ldots, C_{\ell_{k-s-1}}$, set V_i to Γ_{ℓ_i} and fill E_i with edges in C_{ℓ_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 Π_i be the set containing the vertexes in $\{h_1, \ldots, h_{k-s-1}\}$ which are adjacent to h_i in H. Then for each $k-s+1 \leq i \leq k$ and for each $v \in V$, vertex v is added to V_i if for each j with $h_j \in \Pi_i$ there exists a vertex $u_j \in V_j$ such that edge $(u_j, v) \in E_j$ (note that V_{k-s} is still empty at this point). No I/Os are needed in this step.

3. Scan the input edge set E and add each edge (v, v') to E_i and E_j 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') to E_{k-s} and E_i if $v' \in V_i$. Again, each added edge is marked late if it was not already in the set.

   (c) Using a naive approach (see Section 7), 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 an example suppose that $v_1 \in \Gamma_1$ is incomplete, $v_i \in \Gamma_i$ is complete for any $i \geq 2$, and edge (v_1, v_2) is in the instance. The edge (v_1, v_2) 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_i is in V_i for any $i \geq 2$. Then, the instance is enumerated c times. However, we note that (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_i}) is mapped is called *probe edge* of h_i. An instance $(v_1, \ldots, v_k)$ is actually enumerated (i.e., function `emit` is called) only if the probe edge (v_i, v_{p_i}) is not late in E_i for each $1 \leq i \leq k-s-1$.

We note that the deterministic algorithm performs no writes on the external memory, and hence it can run on a read-only disk with the same I/O complexity. The I/O complexity of the algorithm is bounded by the following theorem.

Theorem 1. *The above algorithm correctly enumerates all instances of a given sample graph H and its I/O complexity is* $\mathcal{O}\left((4k)^{k-s-1}\frac{E^{k-s}}{BM^{k-s-1}}\right)$.

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. Under this assumption, the mapping of vertexes in S is given by edges in $E_1, \ldots, E_{k-s-1}$ (step 2) and, in each iteration within step 4, it suffices to load in memory a vertex v and the edges from v to vertexes already in V_i, for any $i \neq k-s$. However, this is not verified if h_{k-s} is adjacent to a vertex $h_j \in S$ with degree one since the mapping of h_j depends only on the mapping of h_{k-s}. In this case, when h_{k-s} is mapped onto a vertex v of G (step 4.a), the whole adjacency list of v has to be loaded in memory since each edge in the list can be mapped onto edge (h_{k-s}, h_j). If the edge list of v is smaller than $M/(4k)$ (i.e., it fits in memory) then the list is loaded and the instances enumerated; otherwise, it suffices to load a chunk of $M/(4k)$ edges of the adjacency list of the vertex, enumerate all instances where edge (h_{k-s}, h_j) is mapped onto edges in the chunk, and then iterate with the next chunk.

We observe that the deterministic algorithm requires a particular independent set S of the sample graph (i.e., each vertex not in S is adjacent to at most one vertex in S) in order to correctly enumerate instances with incomplete vertexes. As an example consider the following case. Let h_0 be adjacent to vertexes h_{k-1} and h_k in the independent set and suppose that there exists an instance where vertexes v, v', v'' be mapped onto h_0, h_{k-1}, h_k respectively. If v is incomplete and the edges (v, v') and (v, v') are in distinct chunks, then the two edges may not be at the same time in the internal memory and then the instance cannot be emitted. This problem disappears if the maximum degree of the input data graph is $\mathcal{O}(M/k)$ since there are no incomplete vertexes and then all edges connected to a vertex are available within a single chunk. In this case, it can be proved that the I/O complexity reduces to $\mathcal{O}\left((4k)^{k-s'-1}E^{k-s'}/(BM^{k-s'-1})\right)$ I/Os, where s' is the size of a traditional independent set S' of the sample graph. This implies that enumerating sample graphs with an independent set of size $s = k/2$ (e.g., cycles of even lengths, paths of odd length, meshes) requires $\mathcal{O}\left(E^{k/2}/(BM^{k/2-1})\right)$ I/Os, which is optimal if the subgraph is in the Alon class (see Section 5).

4 Randomized EM Algorithm

We are now ready to introduce the randomized algorithm. The algorithm, by making use of the random coloring technique in [19], decomposes the problem into small subproblems of expected size $\mathcal{O}(M)$, 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 $\xi : V \to \{1, \ldots, c\}$, with $c = \sqrt{E/M}$, be a vertex coloring chosen uniformly at random from a family of $2(k-s)$-wise independent family of functions. The coloring ξ 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_{τ_1,τ_2} the set containing edges colored with τ_1 and τ_2, that is $E_{\tau_1,\tau_2} = \{(u,v) \in E | \min\{\xi(u), \xi(v)\} = \tau_1, \max\{\xi(u), \xi(v)\} = \tau_2\}$. Each instance $(v_1, \ldots, v_k)$ of the sample graph can be colored by ξ in c^k ways, and it is 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 to a given k-tuple of

colors using the previous deterministic algorithm on the edge set $\cup_{\tau_i \leq \tau_j} E_{\tau_i,\tau_j}$. The algorithm is organized as follows:

1. Randomly select a coloring ξ from a $2(k-s)$-wise independent family of functions.
2. Using sorting, store edges in E_{τ_1,τ_2} in consecutive positions, for each color pair (τ_1, τ_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_{τ_i,τ_j}, for each $\tau_i \leq \tau_j$.

We have the following 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 ξ. It is easy to see that a closed form of this quantity is $X_t = \sum_{\tau_1 \leq \tau_2, E_{\tau_1,\tau_2} \geq t} \frac{E_{\tau_1,\tau_2}!}{(E_{\tau_1,\tau_2}-t)!}$ (note that sets E_{τ_1,τ_2} with less than t edges do not contribute).

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

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 $\mathcal{O}\left(k^{3(k-s-1)} \frac{E^{k/2}}{BM^{k/2-1}}\right)$.*

We remark that our deterministic algorithm is crucial for getting the claimed I/O 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 $\Omega\left(M^{k/2}/B\right)$ I/Os for solving subproblems of size $\Theta(M)$, would increase the total I/O complexity by a multiplicative factor $\Omega\left(M^{k/2-1}\right)$.

Getting the high probability. If $M = \Omega\left(\sqrt{E} \log E\right)$ and the maximum degree is $\sqrt{EM}$, 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 $\sqrt{E} \leq \deg(v) \leq \sqrt{EM}$, 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 $[\sqrt{EM}, 2\sqrt{EM})$, 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 [14, 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 $Y_i - \mathbb{E}[Y_i] \leq 1$, and let $\psi = \sum_{i=i}^{p} \mathrm{Var}(Y_i)$. Denote with Δ the maximum degree of the dependency graph of $Y_1, \ldots, Y_p$: this is a graph with vertex set $Y = \{1, \ldots, p\}$ such that if $B \subset Y$ and $i \in Y$ is not connected to a vertex in B, then Y_i is independent of $\{Y_j\}_{j \in B}$. Then, for any $d > 0$, we have $\Pr(X \geq (1+d)\mathbb{E}[X]) \leq e^{-\frac{8d^2 \mathbb{E}[X]^2}{25\Delta(\psi + d\mathbb{E}[X]/3)}}$.

Theorem 3. *Let $M = \Omega\left(\sqrt{E} \log E\right)$ 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 $\mathcal{O}\left(k^{3(k-s-1)} \frac{E^{k/2}}{BM^{k/2-1}}\right)$.*

It deserves to be noticed that it is possible to color low degree vertexes with a coloring from a $2(k-s)$-wise independent family and still get the claimed I/O complexity with probability $1 - 1/E^{\epsilon}$, for $0 \leq \epsilon \leq 1/4$, as soon as $M \geq E^{3/4+\epsilon}$. It suffices to use a technique by Gradwohl and Yehudayoff [10, Corollary 3.2] in our argument instead of the aforementioned result by Janson [14, Theorem 2.3].

5 Lower Bound on I/O Complexity

In this section we provide a lower bound on 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. [2]. 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 [13, 19] on triangle enumeration, we assume that each edge or vertex requires at least one memory word. That is, at any point in time there can be at most M edges/vertexes in memory, and an I/O can move at most B edges/vertexes to or from memory. 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, $\Omega\left(T/(BM^{k/2-1}) + T^{2/k}/B\right)$ I/Os.*

The above lower bound shows that our randomized algorithm is optimal when $k = \mathcal{O}(1)$. 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.

6 Parallelization in MapReduce

A simple two-round MapReduce algorithm for subgraph enumeration follows by exploiting the decomposition into independent subproblems of size $\mathcal{O}(\lambda)$ of the randomized algorithm when $c = \sqrt{E/\lambda}$ colors are used, where $\lambda \geq 1$ denotes the size of each reducer. We assume that each edge e of the input data graph G is stored as a pair (e, ϕ), where e denotes the key and ϕ a dummy value. All subproblems are solved concurrently in the second round by assigning each subproblem to a reducer.[4] The first round replicates $\mathcal{O}\left((E/\lambda)^{k/2-1}\right)$ times each input pair (e, ϕ) since each edge is used in $\mathcal{O}\left((E/\lambda)^{k/2-1}\right)$ subproblems. Although this approach is quite simple, the amount of shuffled data in the second round is $\Lambda = \Theta\left(E(E/\lambda)^{k/2-1}\right)$. Since this overwhelming amount of data is generated within a single round and travels over the network, there might be performance issues related to the performance of the network and to system failure. Then, in some cases, it would be preferable to reduce the amount of data even at the cost of increasing the round number [21]. On the other hand, a naive distribution of subproblems among rounds for reducing the shuffled data may generate skewness in edge replication: for instance if all subproblems associated with a color k-tuple $(\tau_1, \tau_2, 1, \ldots, 1)$, for any τ_1 and τ_2, are solved in a single round, then the amount of shuffled data is $\mathcal{O}(E)$ and all edges in each edge set E_{τ_1,τ_2}, with $(\tau_1, \tau_2) \neq (1, 1)$ are replicated a constant number of times; however each edge in $E_{1,1}$ is replicated $\Theta(E/\lambda)$ times. The skewness in edge replication might arise load balancing issues, negatively impacting the efficiency of the algorithm.

We propose a MapReduce algorithm that exhibits a tradeoff among round number R, amount of shuffled data Λ, and reducer size λ; the algorithm also guarantees an even edge replication.

[4] We do not specify which algorithm is used within each reducer. However, our deterministic or randomized algorithms can be used by setting M to the cache/memory of the physical machine that will execute each reducer.

For any $\lambda \geq 1$ and $\Lambda \geq E$, we show that there exists a MapReduce algorithm that requires $\Theta\left(E^{k/2}/(\Lambda\lambda^{k/2-1})\right)$ rounds and each edge is replicated $\Theta\left(\Lambda/E\right)$ times in each round. For the sake of simplicity, we suppose that all quantities are positive integers, that $\sqrt{E/\lambda}$ is not divisible by $\{2, 3, \ldots, k\}$, and that the coloring function ξ is known at any time by any reducer. All assumptions can be relaxed with minor changes and without asymptotically affecting the results (there might be in each round a negligible number of edges which are replicated $o(\Lambda/E)$ times).

The algorithm works in $\hat{R} = E^{k/2}/(\Lambda\lambda^{k/2-1})$ steps. The input of each step is the input data graph and there is not output since instances are not stored. In the r-th step, with $r = 0, \ldots, \hat{R}-1$, the algorithm performs the following operations. Let $r_1, \ldots, r_{k-2}$ be values in $\{0, \ldots, \hat{R}^{1/(k-2)}\}$ obtained by partitioning into $k-2$ segments of equal size the $\log \hat{R}$-bit binary representation of r, that is $r = \sum_{i=0}^{k-2} r_i \hat{R}^{1/(k-2)}$. Then, the subproblems solved in the r-th round are all the subproblems associated with a coloring tuple $(\tau_1, \ldots, \tau_k)$ satisfying the following properties: colors τ_1 and τ_2 are in $\{0, \ldots, \sqrt{E/\lambda} - 1\}$, while τ_i, for any $i \in \{3, \ldots, k\}$, is equal to $\tau_i = i\tau_1 + \tau_2 + \ell_i + r_i(\Lambda/E)^{1/(k-2)} \mod \sqrt{E/\lambda}$ for some $\ell_i \in \{0, \ldots, (\Lambda/E)^{1/(k-2)} - 1\}$. We note that each step can be performed in two MapReduce rounds (one for edge replication and one for solving subproblems). We get the following claim.

Theorem 5. *Let $\lambda \geq 1$ and $\Lambda \geq E$ be arbitrary values. Then, the above MapReduce algorithm correctly enumerates all instances of a given sample graphs H and requires $\Theta\left(E^{k/2}/(\Lambda\lambda^{k/2-1})\right)$ rounds. In each round, the expected amount of shuffled data is $\mathcal{O}\left(\Lambda\right)$, each reducers uses $\mathcal{O}\left(\lambda\right)$ expected words, and each edge is replicated $\Theta\left(\Lambda/E\right)$ times. For given values of λ and Λ, the round number is optimal if the subgraph is in the Alon class.*

We observe that we get a MapReduce algorithm requiring $\Theta\left((E/\lambda)^{k/2-1}\right)$ rounds if $\Lambda = E$, while we get the aforementioned two-round MapReduce algorithm if $\Lambda = E(E/\lambda)^{k/2-1}$.

7 Further Extensions

Work Complexity. We analyze the work complexity when the sample graph is in the Alon class and $k = \mathcal{O}\left(1\right)$. By using the ideas in [1, Theorem 6.2], the enumeration (in internal memory) of instances with vertexes in V_i, for each $1 \leq i \leq k$, in Step 4.c of the deterministic algorithm can be performed in $\tilde{\mathcal{O}}\left(M^{k/2-1}\right)$ work (since vertex h_{k-s} is fixed in each iteration). Then the total work of the deterministic algorithm is $\tilde{\mathcal{O}}\left(E^{k-s}/M^{k/2-s}\right)$. As a consequence the expected work of the randomized algorithm becomes $\tilde{\mathcal{O}}\left(E^{k/2}\right)$, which is just a polylog factor from the optimum since instances in the Alon class (e.g., cliques) can appear $\Theta\left(E^{k/2}\right)$ times in the worst case.

To the best of our knowledge, the only algorithm for enumerating a generic sample graph which does not belong to the Alon class is a brute-force approach. In this case, the deterministic algorithm requires $\tilde{\mathcal{O}}\left(E^{k-s}\right)$ work since Step 4.c can be performed in $\tilde{\mathcal{O}}\left(M^{k-s-1}\right)$ work using the brute-force approach; the expected work of the randomized algorithm then becomes $\tilde{\mathcal{O}}\left(E^{k/2}M^{k/2-s}\right)$. In this case the work may become the main bottleneck in a practical implementation.

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 $\cup_{i=1}^{k} V_i$. 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 an enumeration 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. (We conjecture that a similar algorithm can be extended to other sample graphs.)

1. (Base case) If $k = 3$ enumerate all triangles using the algorithm in [19] and return.
2. Enumerate all k-cliques with 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[5], 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 $\mathcal{O}\left(k^2\right)$ larger than the one of the randomized algorithm.

8 Conclusion

In this paper we have proposed 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 $\mathcal{O}\left(E^{k/2}/(BM^{k/2-1})\right)$ expected I/Os when $k = \mathcal{O}(1)$. A nice property of this algorithm is that it decomposes the problem into a large number of independent subproblems that can be solved concurrently. We have then proposed a MapReduce implementation of the randomized algorithm, although a similar algorithm can be designed for other parallel models such as multicores. This is the first paper to deal with the I/O complexity of the enumeration of a generic sample graph, and we would like to call for further investigations. Indeed, the enumeration of an arbitrary subgraph is a crucial tool for analyzing the structural and functional properties of networks, however previous papers have been limited to the I/O complexity of triangle enumeration.

The complexities of our algorithms have an exponential dependency on the vertex number k of the sample graph, and are thus mainly of theoretical interest. The lower bound shows that this is the best result in the worst case under standard assumptions. However, some experiments [20] on related MapReduce algorithms for triangle enumeration shows interesting performance and seems to suggest that the analysis of our algorithms can be improved by expressing the complexities as function of some properties of the input graph (e.g., arboricity) or of the output. An output sensitive algorithm for triangle enumeration has recently been proposed by Björklund et al. [4] in the RAM model, however the problem remains open in the external memory for the enumeration of an arbitrary subgraph as well as for triangle enumeration.

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

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

References

[1] F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using Map-Reduce. In *Proc. 29th ICDE*, pages 62–73, 2013.

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

[3] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *Proc. 20th SPAA*, pages 197–206, 2008.

[4] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick. Listing triangles. In *Proc. of 41st ICALP*, volume 8572 of *LNCS*, pages 223–234, 2014.

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

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

[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[8] I. Finocchi, M. Finocchi, and E. G. Fusco. Counting small cliques in mapreduce. arXiv:1403.0734, 2014.

[9] M. Goodrich and P. Pszona. External-memory network analysis algorithms for naturally sparse graphs. In *Proc. 19th ESA*, volume 6942 of *LNCS*, pages 664–676, 2011.

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

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

[12] J. A. Grochow and M. Kellis. Network motif discovery using subgraph enumeration and symmetry-breaking. In *Proc. 11 RECOMB*, pages 92–106, 2007.

[13] X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In *Proc. ACM SIGMOD*, pages 325–336, 2013.

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

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

[16] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Proc. 21 SODA*, pages 938–948, 2010.

[17] M. N. Kolountzakis, G. L. Miller, R. Peng, and C. E. Tsourakakis. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics*, 8(1-2):161–185, 2012.

[18] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *Proc. 31st PODS*, pages 37–48, 2012.

[19] R. Pagh and F. Silvestri. The input/output complexity of triangle enumeration. In *Proc. of 33rd PODS*, pages 224–233, 2014.

[20] H. Park, F. Silvestri, U. Kang, and R. Pagh. Mapreduce triangle enumeration with guarantees. 2014.

[21] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round tradeoffs for mapreduce computations. In *Proc. 26th ICS*, pages 235–244, 2012.

[22] F. Silvestri. Subgraph enumeration in massive graphs. arXiv:1402.3444, 2014.

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

[24] J. S. Vitter. *Algorithms and Data Structures for External Memory*. Now Publishers Inc., Hanover, MA, USA, 2008.