Theoretical Computer Science 770 (2019) 25–33

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

A new decomposition technique for maximal clique enumeration for sparse graphs [☆]

George Manoussakis [1]

Ben-Gurion University of the Negev, Israel

ARTICLE INFO

Article history:
Received 16 March 2018
Received in revised form 15 August 2018
Accepted 15 October 2018
Available online 17 October 2018
Communicated by A. Marchetti-Spaccamela

Keywords:
Enumeration
Counting
Algorithms
Maximal cliques
k-degenerate graphs

ABSTRACT

Given a graph, we are interested in the question of finding all its maximal cliques. This question models the community detection problem and has been extensively studied. Here, we approach it under the light of an important graph parameter. The *degeneracy* of a graph G is the smallest integer k such that every subgraph of G contains a vertex of degree at most k. Using a new decomposition technique, we present two output sensitive algorithms for the maximal clique enumeration problem. The first one has enumeration time depending only on the degeneracy of the graph. This is the first such result in the literature. This algorithm requires that the cliques are stored in memory. Thus, we propose a second one, which has enumeration time depending on the degeneracy and the maximum degree, and which only requires memory polynomial in the degeneracy of the graph (besides the space to store the graph itself). Then we show that this algorithm can be easily parallelized. As a by-product of our decomposition technique, we propose new algorithms for the maximum clique and p-clique problems as well as an approximation algorithm counting maximal cliques, with approximation ratio the maximum clique size.

1. Introduction

Degeneracy, introduced by Lick et al. [21], is a common measure of the sparseness of a graph and is closely related to other sparsity measures such as arboricity and thickness. Graphs with low degeneracy appear often in practice. For instance, the World Wide Web graph, citation networks, and collaboration graphs have low arboricity [15], which is a graph parameter within a constant factor of degeneracy. Moreover, planar graphs have degeneracy at most five [21] and the Barabàsi–Albert model of preferential attachment [1], frequently used as a model for social networks, produces graphs with bounded degeneracy. See Fig. 1 for some more examples.

It thus seems relevant, when designing algorithms for problems for which the input graph is generally sparse, to take into account its degeneracy. For example, when considering the community detection problem in real world graphs. A community is a densely connected subgraphs of the graph. This question has attracted a lot of research as it finds applications in many domains, such as biology, sociology [13], as well as physics [20], for instance. Modeling communities can be done using the notion of maximal cliques, that is, maximal subgraphs in which all vertices are pairwise connected.

Therefore, it seems interesting both from a practical and theoretical point of view to study the maximal clique enumeration problem in sparse graphs. The question has been extensively studied. We can essentially distinguish between two

☆ An extended abstract of this work has been presented in IPEC 2017 [23].
E-mail address: gomanous@gmail.com.

1 Current address: Université de Versailles Saint-Quentin-En-Yvelines, UFR des sciences, 45 avenue des États-Unis, 78035 Versailles cedex.

https://doi.org/10.1016/j.tcs.2018.10.014
0304-3975/© 2018 Elsevier B.V. All rights reserved.

Graph	n	m	k
citationCiteseer	268 495	1.1M	15
Flickr	1.72M	15.6M	568
uk-2002	18M	261M	943
Twitter	41.7M	1.20B	2490

Fig. 1. Real world graphs with their vertex and edge set sizes as well as their degeneracy. Taken from Shin et al. [26].

Table 1
Bounds for maximal clique enumeration where $q-1 \le k \le \Delta \le n-1$. $^+$: These are polynomial time delay algorithms. Their delay is equal to their enumeration time divided by the number of maximal cliques α. The space bounds do not include the space needed to store the graph. The results are presented chronologically.

Algorithm	Setup	Enumeration	Space
Bron–Kerbosch [3]	$\mathcal{O}(m)$	unbounded	$\mathcal{O}(n+q\Delta)$
Tomita et al. [27]	$\mathcal{O}(m)$	$\mathcal{O}(3^{n/3})$	$\mathcal{O}(n+q\Delta)$
Eppstein et al. [12]	$\mathcal{O}(m)$	$\mathcal{O}(k(n-k)3^{k/3})$	$\mathcal{O}(n+k\Delta)$
Johnson et al. [17]$^+$	$\mathcal{O}(mn)$	$\alpha\mathcal{O}(mn)$	$\mathcal{O}(\alpha n)$
Tsukiyama et al. [28]$^+$	$\mathcal{O}(n^2)$	$\alpha\mathcal{O}((n^2-m)n)$	$\mathcal{O}(n^2)$
Chiba et al. [7]$^+$	$\mathcal{O}(m)$	$\alpha\mathcal{O}(mk)$	$\mathcal{O}(m)$
Makino et al. [22]$^+$	$\mathcal{O}(mn)$	$\alpha\mathcal{O}(\Delta^4)$	$\mathcal{O}(m)$
Chang et al. [6]$^+$	$\mathcal{O}(m)$	$\alpha\mathcal{O}(\Delta h^3)$	$\mathcal{O}(m)$
Makino et al. [22]$^+$	$\mathcal{O}(n^2)$	$\alpha\mathcal{O}(n^{2.37})$	$\mathcal{O}(n^2)$
Comin et al. [8]$^+$	$\mathcal{O}(n^{5.37})$	$\alpha\mathcal{O}(n^{2.09})$	$\mathcal{O}(n^{4.27})$
Conte et al. [9]$^+$	$\mathcal{O}(m\log^{\mathcal{O}(1)}(m+n))$	$\alpha\mathcal{O}(qd(\Delta+qd)\log^{\mathcal{O}(1)}(m+n))$	$\mathcal{O}(q)$
Conte et al. [9]$^+$	$\mathcal{O}(m\log^{\mathcal{O}(1)}(m+n))$	$\alpha\mathcal{O}(\min\{mk,qk\Delta\}\log^{\mathcal{O}(1)}(m+n))$	$\mathcal{O}(k)$
This paper	$\mathcal{O}(m+poly(k)n)$	$\alpha\mathcal{O}(poly(k))$	$\mathcal{O}(\alpha q)$
This paper	$\mathcal{O}(m+poly(k)n)$	$\alpha\mathcal{O}(poly(k)qk\Delta)$	$\mathcal{O}(poly(k))$

Δ = max degree, k = degeneracy, q = maximum clique size, α = number of maximal cliques, h = smallest integer such that $|\{v \in V : |N(v)| \ge h\}|$.

families of algorithms. On one side, *worst-case output size* algorithms have been proposed. Their complexities match the maximal number of maximal cliques one can find in the considered graphs. For instance, Tomita et al. [27] propose an algorithm enumerating all maximal cliques of a general n-order graph in time $\mathcal{O}(3^{n/3})$. This is worst-case output size optimal in general graphs, as for instance the Moon–Moser graphs have $\Theta(3^{n/3})$ cliques [5,25]. Thus, even printing the cliques of these graphs would require at least $\Omega(3^{n/3})$ time. Similarly, for k-degenerate graphs, Eppstein et al. [12] prove a $\mathcal{O}((n-k)3^{k/3})$ bound on the maximal number of maximal cliques and then show an algorithm running in time $\mathcal{O}(k(n-k)3^{k/3})$. The two algorithms described above are based on the Bron–Kerbosch algorithm [3]. These results are summarized in the first three rows of Table 1.

In the worst case, the number of cliques can be exponential, therefore any algorithm for this problem has exponential complexity. On the other hand, if the specific input graph has few cliques, it may be relevant to achieve better time complexity than in the general case. Thus, this is why algorithms whose running time depends on the number of maximal cliques of the graph have also been considered. This family of algorithms is often referred to as *polynomial delay output sensitive* algorithms. Their time complexities can be divided into a preprocessing phase followed by an enumeration phase. During the enumeration phase, maximal cliques of the graph are outputted with polynomial delay: the wait between the output of two maximal cliques is bounded by some polynomial in the parameters of the graph. For example, the algorithm of Johnson et al. [17] has setup time $\mathcal{O}(mn)$ and polynomial time delay $\mathcal{O}(mn)$. Thus, after the setup phase, this algorithm requires $\alpha\mathcal{O}(mn)$ time, α being the number of maximal cliques, to output all the maximal cliques of the graph. It is *output sensitive* since the enumeration time depends on the number of maximal cliques of the graph. All the algorithms that fall into this category are listed in Table 1. For these specific algorithms, the time delay is equal to the enumeration time divided by α, the number of maximal cliques.

Our contributions. We propose a new decomposition technique which, used together with existing algorithms of Table 1, yields output sensitive algorithms for maximal clique enumeration and other problems. In the original conference paper [23], we provided the following output sensitive algorithm. It has setup time $\mathcal{O}(m+s(k+1)n)$ and enumeration time $\alpha\mathcal{O}(f(k+1)q)$. Here, $s(k+1)$ (resp. $f(k+1)$) is the preprocessing time (resp. enumeration time) for maximal clique enumeration in a $(k+1)$-order graph. For example, using the algorithm of Makino et al. [22] which has setup time $\mathcal{O}((k+1)^2)$ and enumeration time $\mathcal{O}((k+1)^{2.37})$ for a graph of order $(k+1)$, our algorithm has setup time $\mathcal{O}(m+n(k+1)^2)$ and enumeration time $\alpha\mathcal{O}(q(k+1)^{2.37})$. Note that $q \le k+1$. Since in a $(k+1)$-order graph all the graph parameters (number of edges and vertices, the maximum degree, the maximum clique size, etc.) are bounded by some function of k, our algorithm will always have enumeration time depending only on the degeneracy of the graph, whatever output sensitive algorithm of Table 1 we use. This is the first such algorithm. On the downside, the algorithm does not have polynomial time delay. It also requires that the maximum cliques are stored in memory. Thus, our algorithm needs $\mathcal{O}(\alpha q)$ space, besides the space needed to store the graph (in our case, the graph can be stored using adjacency lists). This result can be found in the before last row of Table 1.

In a second part, we provide new results which were not found in the conference paper [23]. First, we propose a second algorithm for maximal clique enumeration. It has similar time complexity and is given in the last row of Table 1. The $\mathcal{O}(poly(n))$ functions for the preprocessing and the enumeration are the same as for the previous algorithm. That is, the time needed to preprocess and enumerate cliques in a $(k+1)$-order graph. But this algorithm also requires an additional $\mathcal{O}(k\Delta)$ factor in the enumeration complexity. On the other hand, the space complexity is better than the first algorithm. Similarly as for the other complexities, it is the space complexity required for maximal clique enumeration in a $(k+1)$-order graph. Taking the same example as before, using the algorithm of Makino et al. [22], our algorithm would have space complexity at most $\mathcal{O}(k^2)$. This fact is interesting on its own. Conte et al. [9] proposed the first polynomial time delay algorithm requiring only sublinear space (besides the space to store the graph), given a degeneracy ordering of the graph. Thus, using their algorithm, (or any algorithm of Table 1 with linear space requirement) we also get a sublinear space algorithm, as well as an improved enumeration time (which does not depend on n and m, contrary to theirs). However, we cannot guarantee a bounded delay, contrary to their result. Then, we show that this algorithm can be very easily parallelized which may be of practical interest.

Finally, using our decomposition technique, we provide two new algorithms for other clique problems. The first one returns a maximum clique, is parameterized by degeneracy and has complexity $\mathcal{O}(m+(n-k)T_k)$ where T_k is the complexity of solving the maximum clique problem in a graph of order k (see Xiao and Nagamochi [31] for the current fastest such algorithm). This result improves on the algorithm of Buchanan et al. [4] which has complexity $\mathcal{O}(nm+nT_k)$.

Secondly, we consider another graph parameter, called the inductive independence number, which generalizes degeneracy. This parameter has first been introduced by Akcoglu et al. [10] and naturally generalizes chordal graphs. It has then been studied more extensively by Borodin and Ye [32]. In these two papers, the authors studied algorithmic and structural properties of this new family of graphs and showed that the general recognition problem is NP-complete. However, several natural classes of graphs are inductive k-independent for a small constant k. For instance, chordal graphs are inductive 1-independent, planar graphs are inductive 3-independent and claw-free graphs are inductive 2-independent. We prove that the p-CLIQUE problem (the problem of deciding whether a graph has a clique of size p) can be solved in time $\mathcal{O}(p^2(p+r-1)^{rp}n)$, with r the inductive independence number of the input graph.

The organization of the document is as follows. In Section 2, we introduce some notations and definitions. In Section 3, we prove basic results. These results are used in Section 4 for the proof of the main contributions of the paper. The correctness and time complexity of the first algorithm are presented in Theorem 1 and Theorem 2. The second algorithm is proved in Theorem 3. In Section 5 we present our results for the maximum clique and p-clique problems. The approximation algorithm for counting maximal cliques can be found in the concluding remarks.

2. Definitions

We consider graphs of the form G=(V,E) which are simple, undirected, connected, with n vertices and m edges. We assume that they are stored in memory using adjacency lists. If $X \subset V$, the subgraph of G induced by X is denoted by $G[X]$. When not clear from the context, the vertex set of G will be denoted by $V(G)$. The set $N(x)$ is called the *open neighborhood* of the vertex x and consists of the vertex adjacent to x in G. *The closed neighborhood* of x, denoted by $N[x]$, is defined as the set $\{N(x) \cup x\}$. Given an ordering $v_1, \ldots, v_n$ of the vertices of G, V_i is the set of vertices following v_i including itself in this ordering, that is, the set $\{v_i, v_{i+1}, \ldots, v_n\}$. We now define a family of induced subgraphs of G, defined with respect of some ordering of its vertices. By $[n]$ we denote the set of integers $\{1, 2, \ldots, n\}$.

Definition 1. Let G be a n-order graph and $v_1, \ldots, v_n$ an ordering of its vertices. Graph G_i, $i \in [n]$, is the induced graph $G[N[v_i] \cap V_i]$.

The degeneracy of a graph can be defined in a few different ways. The one relevant to this paper is the following. A graph has degeneracy k, or is *k-degenerate*, if k is the smallest integer such that there is an ordering $v_1, \ldots, v_n$ of its vertices such that for all $i \in [n]$, we have $|N(v_i) \cap V_i| \leq k$. The degeneracy ordering can be computed in $\mathcal{O}(m)$ time [2]. The idea is essentially to remove iteratively vertices of minimum degree until the graph is empty. The order in which the vertices are removed yields the degeneracy ordering. Given a graph G, we will denote by σ_G its degeneracy ordering and, if $x \in V(G)$ then $\sigma_G(x)$ will be the rank of x (the index of its position) in σ_G. Similarly, a graph is inductive k-independent if k is the smallest integer such that there is an ordering $v_1, \ldots, v_n$ of its vertices such that for all $i \in [n]$, we have that the independence number (the size of the largest independent set) of graph G_i is less than k. As proved by Borodin and Ye, it is NP-complete to determine the inductive independence number of a graph. They also showed that it is W[1]-hard with respect to k.

In our algorithms, we will need to consider the vertices of some cliques as strings. We introduce some of the vocabulary that we use in the paper. Let Σ be an alphabet, that is, a non-empty finite set of symbols. Let a string s be any finite sequence of symbols from Σ; s will be a substring of a string t if there exists strings u and v such that $t = usv$. If u or v is not empty then s is a proper substring of t. It will be a *suffix* of t if there exists a string u such that $t = us$. If u is not empty, s is called a *proper suffix* of t. We also use a data structure to store all the proper suffixes of a given string. It is called suffix tree in the literature. Given a word of size n, we can construct a suffix tree containing all its suffixes in space and time $\mathcal{O}(n)$, see [24,29,30]. For a set of words $X = \{x_1, x_2, \ldots, x_r\}$, it is possible to construct a generalized suffix tree containing all the suffixes of the words in X, in an online fashion, in space and time $\mathcal{O}(\sum_{i=1}^{r}|x_i|)$, see [16, chapter 6] and [29] for instance.

3. Basic results

When not specified, we always assume that, given a k-degenerate graph G, we have its degeneracy ordering, denoted by σ_G. When referring to an ordering of the vertices, we always refer to σ_G. The family of subgraphs $G_i, i \in [n]$, described in the previous section will always be constructed following the degeneracy ordering of G. Thus, these graphs have at most $k+1$ vertices, since in a degeneracy ordering $v_1, \ldots, v_n$ of the vertices of G, the inequality $|N[v_i] \cap V_i| \leq k+1$ holds.

We want to show in this section that, roughly, given some k-degenerate graph G, it is enough to compute all the maximal cliques of the induced subgraphs $G_i, i \in [n]$, to get all the maximal cliques of G. We first start by proving that the induced subgraphs $G_i, i \in [n]$, can be easily accessed. To prove that we first need to modify slightly the adjacency lists of the graph. We call this new adjacency structure the degenerate adjacency lists of G. It is defined as follows.

Definition 2. Let G be a graph given by the adjacency lists for each vertex. The degenerate adjacency list of a vertex $x \in V$ is its adjacency list in which every neighbor of x that has higher rank than x in σ_G is before every neighbor of x with lower rank in σ_G.

Lemma 1. *The degenerate adjacency lists of a graph G can be computed in time $\mathcal{O}(m)$.*

Proof. Assume that we are given the degeneracy ordering σ_G of G. Assume that we have the adjacency lists of G. Let $x \in V$ and let d_x be its degree. In time $\mathcal{O}(d_x)$ put all vertices from its adjacency lists that have lower ranking in σ_G at the end of the adjacency list. Repeat the procedure for all the vertices of the graph. This is done in total time $\mathcal{O}(m)$. $\square$

Therefore, we can transform the adjacency lists of a graph into its degenerate adjacency lists fast and without additional space. We now show, that, given a graph G and its degeneracy ordering, the vertex and edge sets of graphs $G_i, i \in [n]$, can easily be accessed.

Corollary 1. *Let G be a graph given though its degenerate adjacency lists. Let σ_G be a degeneracy ordering of G. Given $i \in [n]$, we can access the vertex set of G_i in time $\mathcal{O}(k)$. An adjacency query of vertices of G_i can be done in $\mathcal{O}(k)$ time.*

Proof. The vertex set of graph G_i corresponds to the vertices at the beginning of the degenerate adjacency list of vertex v_i. They can be found in time $\mathcal{O}(k)$, by checking at most the k first vertices of its degenerate adjacency list. Given two vertices of G_i, we can check their adjacency in time $O(k)$, by checking, at most, the k first vertices of the degenerate adjacency lists of the two vertices. $\square$

Now that we have seen how the induced subgraphs $G_i, i \in [n]$ can be accessed, we want to characterize their maximal cliques with respect to the maximal cliques of the graph. We show in the next two lemmas that the maximal cliques of graphs $G_i, i \in [n]$ which are not maximal in G can be easily described.

Lemma 2. *Let G be a k-degenerate graph, σ_G its degeneracy ordering, and let K be a maximal clique of an induced subgraph $G_i, i \in [n]$. Clique K is not a maximal clique of G if and only if there exists a maximal clique C of G which is an induced subgraph of a G_j with $j < i$ and such that K is a strict induced subgraph of C.*

Proof. Let σ_G be the degeneracy ordering of G. Assume that K is a maximal clique of an induced graph $G_i, i \in [n]$, but is not a maximal clique of G. Observe that $v_i \in V(K)$ since, by definition, v_i is connected to all the vertices of $V(G_i)\backslash v_i$. Since K_i is a clique which is not maximal, then there exists a set A of vertices such that $A \cap V(K) = \emptyset$ and the graph induced on $V(K) \cup A$ is a maximal clique of G. Let v_j be the vertex of A with lower ranking in σ_G. We have that $\sigma_G(v_j) < \sigma_G(v_i)$ since v_j is connected to v_i but does not appear in $V(G_i)$. (It does not appear otherwise $A \cap V(K) \neq \emptyset$.) Let C be the maximal clique induced on $V(K) \cup A$. Clique C is an induced subgraph of G_j with $j < i$. Observe that K does not have v_j in its vertex set. Therefore, K is a strict induced subgraph of C.

Conversely, assume that K is a maximal clique of G_i and C a maximal clique of $G_j, j < i$, such that K is an induced subgraph of C. Since K is a strict induced subgraph of a maximal clique of G then K cannot be a maximal clique of G. $\square$

Corollary 2. *Let G be a k-degenerate graph and let K be a maximal clique of an induced subgraph $G_i, i \in [n]$, such that K is not maximal in G. Let C be a maximal clique of G which is a subgraph of some graph G_j, $j < i$ and such that K is a subgraph of C. Let $W(K)$ and $W(C)$ be the words obtained from the vertices of cliques K and C which have been ordered following σ_G. Then $W(K)$ is a proper suffix of $W(C)$.*

Proof. Observe first that by Lemma 2, clique C is well defined. Since K is a strict subgraph of C then $V(K) \subset V(C)$. Recall that by definition, graph G_i corresponds to graph $G[N[v_i] \cap V_i]$, where v_i is the i-th vertex of the degeneracy ordering. Observe that since v_i is the vertex of $V(K)$ with smallest ranking in σ_G, then v_i appears first in $W(K)$. We also have that $v_i \in V(C)$. Assume now by contradiction that $W(K)$ is not a proper suffix of $W(C)$. This implies that there exists at least a vertex $x \in V(C) \backslash V(K)$ that appears after vertex v_i in $W(C)$. If that was not the case then $W(K)$ would have been a proper suffix of $W(C)$. This implies that vertex x appears after vertex v_i in σ_G. Observe now that x is connected to all the vertices of K since $x \in V(C)$ and $V(K) \subset V(C)$. Thus, $G[V(K) \cup \{x\}]$ is a maximal clique of G_i, which is a contradiction by maximality of K. □

To conclude the section, we prove some additional results regarding the maximal cliques of graphs $G_i, i \in [n]$, in the next three lemmas. We show first, that, essentially, every maximal clique of the graph appears in an unique such graph. We then count how many maximal cliques of these induced graphs are not maximal in the graph G itself.

Lemma 3. *Let G be a k-degenerate graph. Every clique which is maximal in some subgraph $G_i, i \in [n]$, is not maximal in any subgraph G_j with $j \neq i$.*

Proof. Let K be a maximal clique of some subgraph $G_i, i \in [n]$. Assume by contradiction that there exists a $j \in [n]$ with $j \neq i$ such that K is maximal in G_j. Assume first that $i < j$. Since vertex v_i is connected to all the vertices of graph G_i then necessarily $v_i \in V(K)$ or K is not maximal in G_i. Since we assumed $i < j$ then $v_i \notin V(G_j)$. This implies that K cannot be a subgraph of G_j, which gives a contradiction in that case. Thus assume now that $j < i$. The proof is similar. Vertex v_j which is connected to all the vertices of G_j does not belong to graph G_i. Since K is maximal in G_i and since $v_j \notin V(K)$ then K cannot be maximal in G_j. □

Lemma 4. *Let G be a k-degenerate graph, σ_G its degeneracy ordering. Every maximal clique of G is a subgraph of exactly one graph $G_i, i \in [n]$.*

Proof. Let K be some maximal clique of G. We first prove that K is a subgraph of at least a subgraph $G_i, i \in [n]$. Let $x \in V(K)$ be the vertex of K which has minimum ranking in σ_G. Observe now that clique K is subgraph of graph $G_{\sigma_G(x)}$. The fact that clique K is a subgraph of at most a graph $G_i, i \in [n]$, is a consequence of Lemma 3. □

Lemma 5. *Let G be a k-degenerate graph. Let G_i, $i \in [n]$, be the family of induced subgraphs of Definition 2. Let α denote the number of maximal cliques of G and α_i the number of maximal cliques of graph G_i. We have that $\sum_{j=1}^{n} \alpha_j \leq \alpha q$.*

Proof. Let max_i denote the number of maximal cliques of $G_i, i \in [n]$, which are maximal in G and $Nmax_i$ the number of maximal cliques of $G_i, i \in [n]$, which are not maximal in G. We have that $\alpha_i = max_i + Nmax_i$. By Lemma 4, every maximal clique of G is a subgraph of exactly one graph $G_i, i \in [n]$. This implies that $\sum_{j=1}^{n} max_j = \alpha$. Let X be the set of cliques which are maximal in some graph $G_i, i \in [n]$, but not maximal in G and let $x \in X$. By Lemma 2, the word obtained from the vertices of x which have been ordered following σ_G is a proper suffix of the word obtained from ordering the vertices, following σ_G, of some maximal clique of G. This implies that X is of size at most $(q-1)\alpha$. To conclude the proof, Lemma 3 implies that clique x is maximal in an unique graph $G_i, i \in [n]$, which implies that $\sum_{j=1}^{n} Nmax_j \leq (q-1)\alpha$. Thus, in total, we have that $\sum_{j=1}^{n} \alpha_j \leq \alpha + \alpha(q-1) = \alpha q$. □

4. Algorithms for maximal clique enumeration

We start by presenting the first algorithm, which has enumeration time depending only on the degeneracy of the graph. The second algorithm is a slightly modified version of the first one. The outline of the first algorithm is the following. We consider each subgraph $G_i, i \in [n]$, iteratively, starting from G_1 up to G_n. We find all its maximal cliques and we search for them in a generalized suffix tree. If there is a match, the clique is rejected, otherwise it is outputted and its proper suffixes are inserted into the generalized suffix tree. The procedure is described in Algorithm 1. Its correctness is proved in Theorem 1 and its time complexity in Theorem 2. The second algorithm is presented in Theorem 3.

Theorem 1. *Given a graph G, Algorithm 1 outputs exactly all its maximal cliques, without duplication.*

Proof. By Lemma 4, every maximal clique of the graph is a subgraph of exactly one graph $G_i, i \in [n]$. Thus, every maximal clique K of the graph is considered exactly once in Line 6 of the algorithm. If K is matched in the generalized suffix tree at Line 7 then the vertices of K ordered following σ_G form a proper suffix of some clique of the graph. This contradicts the fact that K is maximal in G. Thus, every maximal clique is outputted exactly once. Moreover, all the proper suffixes of all the maximal cliques are stored in the generalized tree. By Corollary 2, the word obtained from a maximal clique in some

Algorithm 1:

```
Data: A graph $G$.
Result: All the maximal cliques of $G$.
1  Compute $k$ the degeneracy of $G$ and $\sigma_G$.
2  Compute the degenerate adjacency lists of $G$.
3  Initialize $T$ an empty generalized suffix tree.
4  for $j = 1$ to $n$ do
5      Compute all maximal cliques of graph $G_j$.
6      for every maximal clique $K$ of graph $G_j$ do
7          Order the vertices of $K$ following $\sigma_G$
8          Search for $K$ in $T$.
9          if there is a match then
10             Reject it.
11         else
12             Insert the proper suffixes of $K$ in $T$.
13             Output $K$.
```

graph $G_i, i \in [n]$ which is not maximal in G forms a proper suffix of the world obtained from some maximal clique of G. Thus, all such cliques will be rejected in Line 9 of Algorithm 1. In conclusion, we proved that only the maximum cliques of G are outputted, without duplication. □

Theorem 2. *Given a graph G, Algorithm 1 has setup time $\mathcal{O}(m + s(k+1)n)$ and enumeration time $\alpha\mathcal{O}(f(k+1)q)$, where $s(k+1)$ (resp. $f(k+1)$) is the preprocessing time (resp. enumeration time) of maximal clique enumeration in a $(k+1)$-order graph.*

Proof. Computing the degeneracy of G in Line 1 is done in $\mathcal{O}(m)$ time. Computing the degenerate adjacency lists is done in $\mathcal{O}(m)$, by Lemma 1. To compute all the maximal cliques of every graph $G_i, i \in [n]$, we can use any output sensitive algorithm of Table 1. The chosen algorithm has preprocessing time $s(|V(G_i)|) = \mathcal{O}(s(k+1))$ and enumeration time $f(|V(G_i)|) = \mathcal{O}(f(k+1))$ for each graph $G_i, i \in [n]$, since these graphs have at most $k+1$ vertices. We first preprocess every such graph G_i in total time $\mathcal{O}(s(k+1)n)$. Thus, the preprocessing phase takes time $\mathcal{O}(m + s(k+1)n)$. Then, we enumerate all the maximal cliques of the graphs $G_i, i \in [n]$, in total time $(\sum_{j=1}^{n} \alpha_j) * \mathcal{O}(f(k+1))$, where α_j is the number of maximal cliques of graph G_j. By Lemma 5, $\sum_{j=1}^{n} \alpha_j \leq q\alpha$. Thus, enumerating all the maximal cliques of the graphs $G_i, i \in [n]$ takes total time $\alpha\mathcal{O}(f(k+1)q)$. Searching and inserting the generated cliques in the suffix tree takes total time $\alpha\mathcal{O}(q^2)$. In conclusion, Algorithm 1 has preprocessing time $\mathcal{O}(m + s(k+1))$ and enumeration time $\alpha\mathcal{O}(f(k+1)q)$, as claimed. □

Algorithm 2:

```
1  for $j = 1$ to $n$ do
2      Compute all maximal cliques of graph $G_j$.
3      for every maximal clique $K$ of graph $G_j$ do
4          for every vertex $x \in K$ do
5              if any of its neighbors in $G$ with lower rank than $v_j$ in $\sigma_G$ is adjacent to all the vertices in $K$ then
6                  reject $K$.
7              else
8                  output $K$.
```

Theorem 3. *Algorithm 1 can be modified to run with processing time $\mathcal{O}(m + s(k+1)n)$ and enumeration time $\alpha\mathcal{O}(f(k+1)qk\Delta)$, where $s(k+1)$ (resp. $f(k+1)$) is the preprocessing time (resp. enumeration time) of maximal clique enumeration in a $(k+1)$-order graph. In that case, it requires $\mathcal{O}(g(k+1))$ space, where $g(k+1)$ is the space needed for maximal clique enumeration in a $(k+1)$-order graph.*

Proof. We modify Algorithm 1 as follows. Lines 3–13 are replaced with the code of Algorithm 2. We first prove the correctness of this new algorithm. As in the proof of Theorem 1, computing all the cliques of graphs $G_i, i \in [n]$, yields all the maximal cliques of G and some cliques which are maximal in graphs $G_i, i \in [n]$, but not in G. By Corollary 2, the word obtained from a maximal clique in some graph $G_i, i \in [n]$, which is not maximal in G, forms a proper suffix of the world obtained from some maximal clique of G. Therefore, if a clique K is not maximal in some graph $G_i, i \in [n]$, there exists at least a common neighbor of all the vertices in K which has rank in σ_G lower than v_i's. This is exactly the condition that is checked in the code proposed in Algorithm 2.

We now prove its time complexity. The preprocessing phase is unchanged, thus its complexity is the same as for Algorithm 1. For the enumeration phase, the new algorithm does some additional computations. Namely, we check, for every maximal clique K computed in graphs $G_i, i \in [n]$, if its vertices have a common neighbor with lower rank in σ_G than vertex v_i. Therefore, there are at most $\mathcal{O}(\Delta)$ such possible vertices to check. For each such vertex, we need to do at most q adjacency queries. Using Corollary 1, an adjacency query takes $\mathcal{O}(k)$ time. Thus, we perform at most $\mathcal{O}(kq\Delta)$ additional actions, per maximal clique of the graphs $G_i, i \in [n]$. Overall, this yields the claimed complexity for the enumeration phase of this new algorithm.

Concerning the space complexity, the new algorithm considers every graph G_i, $i \in [n]$, iteratively. For each such graph we need to store its vertex set, which is of size at most $\mathcal{O}(k)$. Adjacency queries can be done using the degenerate adjacency lists of the graph, and do not require additional memory. Therefore, this new algorithm only requires the space to store the vertex set of each G_i, $i \in [n]$, iteratively, and the space to find each clique in a graph of order $k+1$. □

5. Application to other clique problems

The p-CLIQUE problem asks the question whether an input graph G contains a clique with p vertices. This is a well known NP-complete problem [19] and is also known W[1]-complete with respect to the clique size p, see [11]. In the next theorem, we show that the problem is fixed-parameter tractable with respect to p and the inductive independence number k.

If the total input length is n and s is some parameter of the instance, then, a problem is considered fixed-parameter tractable with respect to s if there is an algorithm that optimally solves it in $f(s)n^{\mathcal{O}(1)}$ time, where $f()$ is allowed to be any arbitrary computable function which is independent of n, and the exponent in $n^{\mathcal{O}(1)}$ is required to be independent of k. For example, a solution running in time $2^{\mathcal{O}(r)}n^3$ is considered fixed-parameter tractable (with respect to s) while $n^{\mathcal{O}(s)}$ is not.

Theorem 4. *Given an inductive r-independent graph and a r-inductive ordering of its vertices, there is an algorithm solving the p-clique problem in $\mathcal{O}(p^2(p+r-1)^{rp}n)$ time.*

Proof. We prove that the p-clique problem can be solved on a graph $G_i, i \in [n]$ in time $\mathcal{O}(p^2(p+r-1)^{rp})$. We use Ramsey's theorem which states that for any two positive integers i, c there exists a positive integer $R(i,c)$ such that any graph with at least $R(i,c)$ vertices contains either an independent set on i vertices or a clique on c vertices or both. It is shown [18, p.65] that $R(i,c) \le \binom{i+c-2}{c-1}$. With $i = r+1$ and $c = p$ then if a graph with at least $\binom{p+r-1}{p-1} = \binom{p+r-1}{r} \le (p+r-1)^r$ vertices does not contain an independent set of size $r+1$, then it has a clique with p vertices.

Let G be our inductive r-independent graph and let $v_1, \ldots, v_n$ be its r-independent ordering. We know by definition that for each i, $\alpha(G_i) < r+1$. Therefore, if for some i, $|V(G_i)| \ge (p+r-1)^r$ then G_i contains a clique on p vertices by Ramsey's theorem. This can be checked in time $\mathcal{O}(n)$.

Now suppose that for each i, $|V(G_i)| < (p+r-1)^r$. We iterate over the vertices in the r-independence order and check for every vertex i if the graph G_i contains a clique of size p. This procedure will return a clique of size p, if any, in G, since such a clique appears in at least one graph $G_i, i \in [n]$ (if x is the vertex of the clique with smallest rank (say $\sigma(x)$) in the inductive independence ordering, then the clique is included in $G_{\sigma(x)}$). To achieve this, we generate all subsets of size p in time $\mathcal{O}\binom{(p+r-1)^r}{p} = \mathcal{O}((p+r-1)^{rp})$ and check if all the $\mathcal{O}(p^2)$ edges are present, which can be done in $\mathcal{O}(p^2)$ with an adjacency matrix or $\mathcal{O}(p^2)n$ with adjacency lists. This takes $\mathcal{O}(p^2(p+r-1)^{rp})$ in total. Therefore, we can conclude that the algorithm runs in time $\mathcal{O}(p^2(p+r-1)^{rp}n)$. □

In the next theorem, we are interested in the question of finding a maximum clique in an input graph. We improve on an existing algorithm of Buchanan et al. [4] for this problem, with complexity $\mathcal{O}(nm + nT_k)$. Here k denotes the degeneracy of the input graph.

Theorem 5. *Given a k-degenerate graph G, there is an algorithm finding a maximum clique in $\mathcal{O}(m + T_k(n-k))$ time, where T_k is the time complexity of finding a maximum clique in a graph of order k.*

Proof. We compute first a degeneracy ordering and the degeneracy adjacency lists of the graph in time $\mathcal{O}(m)$, using Lemma 1. For each of the $n-k$ graphs $G_i, i \in [n-k]$, since they are of size k, we find their maximum clique in time T_k. We also compute the maximum clique of the graph induced on the last k vertices of the degeneracy ordering. Among these $n-k+1$ computed cliques we return the largest one. It is a maximum clique of the graph. This holds since such a clique appears in at least one of these graphs; if x is the vertex of a maximum clique with smallest rank (say $\sigma(x)$) in the degeneracy ordering, then this clique is included in $G_{\sigma(x)}$. □

6. Concluding remarks

Given a k-degenerate graph G, Lemma 5 bounds the number of non-maximal cliques of the graphs G_i, $i \in [n]$. Therefore, counting the number of maximal cliques of each of these graphs and outputting their sum yields a number of maximal cliques for G within a factor q of the optimal, where we recall that q is the maximum clique size. This holds, since, by Lemma 4, every maximal clique of G appears in exactly one G_i, $i \in [n]$, and that these graphs have at most $\mathcal{O}(\alpha q)$ maximal cliques, by Lemma 5.

Thus, these observations yield the following very simple algorithm. For each graph G_i, $i \in [n]$, compute its number of maximal cliques. Output the sum of these numbers. Its time complexity is bounded by $\mathcal{O}(t(k+1)n)$ where $t(k+1)$ is the time complexity of counting the maximal cliques of a $(k+1)$-order graph. There are a few algorithms for counting maximal cliques (which is equivalent to counting independent set in the complement graph). To the best of our knowledge, the fastest algorithm has been presented by Gaspers et al. [14], and runs in time $\mathcal{O}(1.3642^n)$ for a graph of order n. Using this result, we get an algorithm running in time $\mathcal{O}(1.3642^{k+1}n)$ with approximation ratio q. This might be interesting for sparse graphs with many small cliques. We are not aware of any other parameterized algorithm for counting maximal cliques.

We presented the first output sensitive algorithm for maximal clique enumeration whose enumeration time depends only on the degeneracy of the graph. We were not able to prove that it has polynomial time delay. Our intuition is that in its current state, our algorithm has time delay $\mathcal{O}(\alpha q)$. Thus, we first ask whether this is true or not and if yes, if there is a way to modify our approach as to get a polynomial time delay. This algorithm also requires that the maximal cliques are stored in memory.

We then presented a second algorithm, with a nicer space complexity, but with an additional $\mathcal{O}(k\Delta)$ factor in the time complexity. We therefore ask if it is possible to combine the best features of both algorithms. That is, can we design an algorithm with enumeration time depending only on the degeneracy of the graph, and which does not require to store the maximal cliques.

Finally we presented some new algorithms for the p-clique and maximum clique problems, parameterized by the inductive independence number and the degeneracy of the graph, respectively.

Acknowledgement

This research was supported by Grant No. 2016049 from the United States-Israel Binational Science Foundation (BSF).

References

[1] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.
[2] V. Batagelj, M. Zaversnik, An $\mathcal{O}(m)$ algorithm for cores decomposition of networks, CoRR, arXiv:cs.DS/0310049, 2003.
[3] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph, Commun. ACM 16 (9) (September 1973) 575–577, https://doi.org/10.1145/362342.362367, http://doi.acm.org/10.1145/362342.362367.
[4] A. Buchanan, J.L. Walteros, S. Butenko, P.M. Pardalos, Solving maximum clique in sparse graphs: an $o(nm + n2^{d/4})$ algorithm for d-degenerate graphs, Optim. Lett. 8 (5) (2014) 1611–1617, https://doi.org/10.1007/s11590-013-0698-2.
[5] F. Cazals, C. Karande, A note on the problem of reporting maximal cliques, Theoret. Comput. Sci. 407 (1–3) (2008) 564–568.
[6] L. Chang, J.X. Yu, L. Qin, Fast maximal cliques enumeration in sparse graphs, Algorithmica 66 (1) (2013) 173–186.
[7] N. Chiba, T. Nishizeki, Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1) (1985) 210–223.
[8] C. Comin, R. Rizzi, An improved upper bound on maximal clique listing via rectangular fast matrix multiplication, arXiv preprint, arXiv:1506.01082, 2015.
[9] A. Conte, R. Grossi, A. Marino, L. Versari, Sublinear-space bounded-delay enumeration for massive network analytics: maximal cliques, in: 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), 2016, pp. 148:1–148:15.
[10] T.H. Cormen, C. Stein, R.L. Rivest, C.E. Leiserson, Introduction to Algorithms, 2nd edition, McGraw-Hill Higher Education, 2001.
[11] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer Science & Business Media, 2012.
[12] D. Eppstein, M. Löffler, D. Strash, Listing all maximal cliques in large sparse real-world graphs, J. Exp. Algorithmics 18 (November 2013) 3, https://doi.org/10.1145/2543629, http://doi.acm.org/10.1145/2543629.
[13] S. Fortunato, Community detection in graphs, Phys. Rep. 486 (3) (2010) 75–174, https://doi.org/10.1016/j.physrep.2009.11.002, http://www.sciencedirect.com/science/article/pii/S0370157309002841.
[14] S. Gaspers, D. Kratsch, M. Liedloff, On independent sets and bicliques in graphs, in: International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, 2008, pp. 171–182.
[15] G. Goel, J. Gustedt, Bounded arboricity to determine the local structure of sparse graphs, in: International Workshop on Graph-Theoretic Concepts in Computer Science, Springer, 2006, pp. 159–167.
[16] D. Gusfield, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology, Cambridge University Press, 1997.
[17] D.S. Johnson, M. Yannakakis, C.H. Papadimitriou, On generating all maximal independent sets, Inform. Process. Lett. 27 (3) (1988) 119–123, https://doi.org/10.1016/0020-0190(88)90065-8.
[18] S. Jukna, Extremal Combinatorics: With Applications in Computer Science, Springer Science & Business Media, 2011.
[19] R.M. Karp, Reducibility among combinatorial problems, in: Complexity of Computer Computations, Springer, 1972, pp. 85–103.
[20] A. Lancichinetti, S. Fortunato, Community detection algorithms: a comparative analysis, Phys. Rev. E 80 (Nov 2009) 056117, https://doi.org/10.1103/PhysRevE.80.056117, https://link.aps.org/doi/10.1103/PhysRevE.80.056117.
[21] D.R. Lick, A.T. White, d-degenerate graphs, Canad. J. Math. 22 (1970) 1082–1096, http://www.smc.math.ca/cjm/v22/p1082.
[22] K. Makino, T. Uno, New algorithms for enumerating all maximal cliques, in: Scandinavian Workshop on Algorithm Theory, Springer, 2004, pp. 260–272.
[23] G. Manoussakis, An output sensitive algorithm for maximal clique enumeration in sparse graphs, in: 12th International Symposium on Parameterized and Exact Computation, IPEC 2017, September 6–8, 2017, Vienna, Austria, 2017, pp. 27:1–27:8.
[24] E.M. McCreight, A space-economical suffix tree construction algorithm, J. ACM 23 (2) (April 1976) 262–272, https://doi.org/10.1145/321941.321946, http://doi.acm.org/10.1145/321941.321946.
[25] J.W. Moon, L. Moser, On cliques in graphs, Israel J. Math. 3 (1) (1965) 23–28.
[26] K. Shin, T. Eliassi-Rad, C. Faloutsos, Patterns and anomalies in k-cores of real-world graphs with applications, Knowl. Inf. Syst. 54 (3) (2018) 677–710.
[27] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments, Theoret. Comput. Sci. 363 (1) (October 2006) 28–42, https://doi.org/10.1016/j.tcs.2006.06.015.

[28] S. Tsukiyama, M. Ide, H. Ariyoshi, I. Shirakawa, A new algorithm for generating all the maximal independent sets, SIAM J. Comput. 6 (3) (1977) 505–517.
[29] E. Ukkonen, On-line construction of suffix trees, Algorithmica 14 (3) (1995) 249–260, https://doi.org/10.1007/BF01206331.
[30] P. Weiner, Linear pattern matching algorithms, in: Switching and Automata Theory, 1973. SWAT '08. IEEE Conference Record of 14th Annual Symposium on, Oct 1973, pp. 1–11.
[31] M. Xiao, H. Nagamochi, Exact algorithms for maximum independent set, in: Leizhen Cai, Siu-Wing Cheng, Tak-Wah Lam (Eds.), Algorithms and Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 328–338.
[32] Y. Ye, A. Borodin, Elimination graphs, ACM Trans. Algorithms 8 (2) (April 2012) 14.