New Algorithms for Enumerating All Maximal Cliques

Kazuhisa Makino[1] and Takeaki Uno[2]

[1] Division of Mathematical Science for Social Systems, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, JAPAN.
makino@sys.es.osaka-u.ac.jp

[2] National Institute or Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, JAPAN.
uno@nii.jp

Abstract. In this paper, we consider the problems of generating all maximal (bipartite) cliques in a given (bipartite) graph $G = (V, E)$ with n vertices and m edges. We propose two algorithms for enumerating all maximal cliques. One runs with $\mathrm{O}(M(n))$ time delay and in $\mathrm{O}(n^2)$ space and the other runs with $\mathrm{O}(\Delta^4)$ time delay and in $\mathrm{O}(n + m)$ space, where Δ denotes the maximum degree of G, $M(n)$ denotes the time needed to multiply two $n \times n$ matrices, and the latter one requires $\mathrm{O}(nm)$ time as a preprocessing.

For a given bipartite graph G, we propose three algorithms for enumerating all maximal bipartite cliques. The first algorithm runs with $\mathrm{O}(M(n))$ time delay and in $\mathrm{O}(n^2)$ space, which immediately follows from the algorithm for the non-bipartite case. The second one runs with $\mathrm{O}(\Delta^3)$ time delay and in $\mathrm{O}(n + m)$ space, and the last one runs with $\mathrm{O}(\Delta^2)$ time delay and in $\mathrm{O}(n + m + N\Delta)$ space, where N denotes the number of all maximal bipartite cliques in G and both algorithms require $\mathrm{O}(nm)$ time as a preprocessing.

Our algorithms improve upon all the existing algorithms, when G is either dense or sparse. Furthermore, computational experiments show that our algorithms for sparse graphs have significantly good performance for graphs which are generated randomly and appear in real-world problems.

1 Introduction

Enumerating all configurations that satisfy a given specification is a fundamental and well-studied problem in combinatorics (see e.g., [13]). From both theoretical and practical points of view, it has taken on increasing importance in many scientific fields such as artificial intelligence [10, 20], graph theory [14, 19, 21], operations research [16], data mining [2, 3], web mining [15], bioinformatics, and computational linguistics. There are several reasons to recognize enumeration as an important subject to study (see e.g., [13]). Among them, we here mention the following two reasons.

One of the reasons is that there has been beginning to study the problems whose objective functions and/or constraints are difficult to be defined mathematically. For such problems, one of the simplest way is that we first generate all the candidates (polynomially many candidates or as many candidates as computational resources can allow), and then choose one or a few from them according to a preference or plausibility relation which may be based on subjective intuition. For example, in data mining, the procedure above is usually used to find "interesting" objects, since it is difficult to define the term "interesting." Searching a webpage by keywords is another example. Search engines usually output the pages including all or some keywords as the candidates of desired webpages.

The second reason is the recent increase in computational power. 20 years ago, the computational power was too poor to enumerate all the candidates in practical time. Even if it could be, it is hard to handle a great many candidates to be enumerated. Recently, we can handle over 1 million data, and such data can be enumerated in practical time by an efficient algorithm. Hence, enumeration has been used to solve many real-world problems in diverse areas.

This paper addresses the two problems of (1) generating all maximal cliques (equivalently, all maximal independent sets or all minimal vertex covers) of a given graph and (2) generating all maximal bipartite cliques of a given bipartite graph. Since cliques are fundamental graph objects, the problem of generating all maximal cliques is regarded as one of the central problems in the field of enumeration, and has attracted considerable attention in the past (e.g., [7, 14, 16, 21]). The problems have not only theoretical interest, but also a number of potential applications in many areas (e.g., [15, 2, 3]). The next section presents two examples for generating all maximal bipartite cliques.

In 1977, Tsukiyama et al. [21] first proposed an output-polynomial (or polynomial total time) algorithm for generating all maximal cliques in a given graph $G = (V, E)$ that runs with $O(nm)$ time delay (i.e., the computation time between any consecutive output is bounded by $O(nm)$, and the first (resp., last) output occurs also in $O(nm)$ time after start (resp., before halt) of the algorithm) and in $O(n+m)$ space. Here $n = |V|$ and $m = |E|$. Lawler et al. [16] generalized this result (see [9] for further generalization). Chiba and Nishizeki [7] reduced the time complexity to $O(a(G)m)$, where $a(G)$ is the arboricity of G with $m/(n-1) \leq a(G) \leq m^{1/2}$. Johnson et al. [14] proposed an algorithm which enumerates all maximal cliques in the lexicographical order. The algorithm runs with $O(nm)$ time delay, but it uses $O(nN)$ space, where N denotes the number of all maximal cliques of a given graph.

In this paper, we propose the following two algorithms for enumerating all maximal cliques. The first one makes use of matrix multiplication, and runs with $O(M(n))$ time delay and in $O(n^2)$ space, where $M(n)$ is the time needed to multiply two $n \times n$ matrices. Since it is known that matrix multiplication can be done in $O(n^{2.376})$ time [8], our algorithm improves upon the previous algorithms for dense graphs. For example, if a given graph has $m = \Omega(n^{1.689})$ edges, our algorithm dominates all the existing ones.

The second algorithm runs with $O(\Delta^4)$ time delay and in $O(n + m)$ space, where Δ is the maximum degree of G and it additionally requires $O(nm)$ time as a preprocessing before generating the first maximal clique. This improves upon the previous algorithms when a given graph G is sparse, e.g., $n = \Omega(\Delta^{2+\epsilon})$ and $m = \Omega(n\Delta)$ for any $\epsilon > 0$. More generally, we consider graphs G having θ vertices with large degree ($> \Delta^*$). We propose an algorithm that runs $O((\Delta^*)^3(\Delta^* + \theta) + \theta^3)$ time delay and in $O((n + N^*)\theta + m)$ space, where N^* denotes the number of all maximal cliques of the subgraph of G induced by vertices with large degree, and $O(nm)$ time is required as a preprocessing. This algorithm is motivated by practical applications such as web networks, since the graphs obtained from those applications usually have a few vertices with large degree. In this paper, we implement our second algorithm, and compare it with the algorithms of Tsukiyama et al. by using graphs which are generated randomly and appear in real-world problems. We show that our algorithm is much faster than the algorithm of Tsukiyama et al.

Listing all maximal bipartite cliques is also well-studied (see e.g., [5, 11, 17, 18]). Let us first note that the generation of all maximal bipartite cliques in a bipartite graph $G = (V_1 \cup V_2, E)$ can be seen as the one of all maximal cliques in the graph $\hat{G}$ obtained from G by adding edges so that V_1 and V_2 both become cliques. This implies that the algorithms above are applicable to generate all maximal bipartite cliques. Especially, our algorithm that makes use of matrix multiplication improves upon all the existing algorithms for dense graphs. However, if we consider practical applications, we need to develop algorithms for sparse bipartite graphs. Note that $\hat{G}$ might be dense, even if G is sparse. In this paper, we propose two algorithms for sparse bipartite graphs. The first one runs in $\mathrm{O}(\Delta^3)$ time delay and in $\mathrm{O}(n+m)$ space, and the second one runs with $\mathrm{O}(\Delta^2)$ time delay and in $\mathrm{O}(n + m + N\Delta)$ space. Here both algorithms additionally require $\mathrm{O}(nm)$ time as a preprocessing. Similar to non-bipartite clique case, these algorithms improve upon previous algorithms for sparse graphs, and has good performance for computational experiments.

The rest of the paper is organized as follows. In Section 2, we present some examples of applications for our problems. Section 3 provides some preliminaries and introduces notation. Section 4 explains the algorithms of Tsukiyama et al. and Johnson et al. Section 5 presents an algorithm which uses matrix multiplication, and Sections 6 and 7 consider the problem for enumerating all maximal cliques and maximal bipartite cliques for sparse graphs, respectively. Section 8 shows some results of computational experiments.

Due to the space limitation, some proofs are omitted.

2 Applications of Maximal Clique Enumeration

In this section, we present two examples of the applications of generating all maximal bipartite graphs. Some other applications can be found in the context of concept lattice [12] and in artificial intelligence, for example.

2.1 Web Communities

Consider a directed graph $G = (V, A)$ (called *web network*) whose vertices and arcs correspond to web pages and their links, respectively. Kumar et al. [15] regarded *directed bipartite cliques* (S_1, S_2) (i.e., $S_1 \times S_2 \subseteq A$) of G as communities of web pages, i.e., the web pages in S_2 may have similar topics and web pages in S_1 may have interests in these topics, and considered generating directed bipartite cliques of G. They first construct a graph G^* with about 5,000,000 arcs by removing unnecessary vertices and arcs from G, and then enumerate all directed bipartite cliques in the reduced graph G^*. They show that directed bipartite cliques usually contain similar topics by checking them by human hands. However, since G^* contains a great number of bipartite cliques, they could enumerate only those containing at most 10 vertices.

In this setting, it is natural to regard maximal directed bipartite cliques as good representatives of communities. From a directed graph $G = (V, A)$, let us construct a bipartite (undirected) graph $\hat{G} = (V \cup \tilde{V}, E)$ such that $\tilde{V} (= \{\tilde{v} \mid v \in V\})$ is a copy of V and $(v, \tilde{u}) \in E$ if and only if $(v, u) \in A$. Then there exists a one-to-one correspondence between directed bipartite cliques in G and bipartite cliques in $\hat{G}$. Hence, our algorithms are applicable to generate all maximal directed bipartite cliques in G^*

2.2 Closed Item Sets

Let I be a set of items and $\mathcal{T}$ be a family of sets in I (i.e., $\mathcal{T} \subseteq 2^I$), where $T \in \mathcal{T}$ is called a *transaction*. For a given constant α, a subset S of I is called an (α-)*frequent set* if at least α transactions of $\mathcal{T}$ include S. In data mining, we see that frequent sets characterize database $\mathcal{T}$, and investigate the enumeration of all frequent sets to find association rules from $\mathcal{T}$, which is one of the main topics in data mining (e.g., [2, 3]). However, since a database contains a great number of frequent sets if α is small, many researchers started studying the enumeration of all closed item sets, instead of all frequent sets (e.g., [5, 17, 18, 25]). Here a frequent set S of I is called a *closed item set*, if there is no other superset S' of S such that $S' \subseteq T$ for any $T \in \mathcal{T}$ with $S \subseteq T$. Note that the number of closed item sets is usually much smaller than the one of frequent sets in database.

Pasquier et al. [17, 18] proposed algorithms based on back-tracking (and pruning unnecessary branches) to enumerate all closed item sets. Their experimental results show that, if α is large, the number of closed item sets is quite small (up to about 100,000), and hence the algorithms are fast. However, since the algorithms are not output-polynomial, they are not useful if we have a number of closed item sets, for example.

For a set of transactions $\mathcal{T} \subseteq 2^I$, we construct a bipartite graph $G_{\mathcal{T}} = (V_1 \cup V_2, E)$ by $V_1 = \mathcal{T}$, $V_2 = I$ and $(u, v) \in E$ if and only if $u \in V_1$ includes $v \in V_2$. Zaki and M. Ogihara [25] showed that there exists a one-to-one correspondence between closed item sets of $\mathcal{T}$ and maximal bipartite cliques in $G_{\mathcal{T}}$. Hence our algorithms can enumerate all closed item sets in polynomial time delay. Since $G_{\mathcal{T}}$ constructed from a database $\mathcal{T}$ is usually sparse, it is shown [24] that our algorithms for sparse graphs work pretty well.

3 Definitions and Notations

This section introduces some notions and notations of graphs used in the subsequent sections.

Let $G = (V, E)$ be a graph with a vertex set $V = \{v_1, \ldots, v_n\}$ and an edge set $E = \{e_1, \ldots, e_m\}$. If there is a partition V_1 and V_2 of V such that no two vertices in $V_i, i = 1, 2$ are adjacent, then G is called *bipartite* and denoted by $G = (V_1 \cup V_2, E)$. Throughout this paper, we assume without loss of generality that G is simple and connected, since we deal with clique generation problems. We denote by A the adjacency matrix of G, i.e., A is an $n \times n$ matrix such that its element $a_{ij} = 1$ if $(i, j) \in E$, and $a_{ij} = 0$, otherwise. For a vertex subset $S \subseteq V$, $x(S)$ denotes the characteristic vector of S, i.e., the ith element of $x(S)$ is 1 if $v_i \in S$, and 0, otherwise.

For a vertex v of G, let $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$ and $\delta(v) = |\Gamma(v)|$. We call $\Gamma(v)$ the *neighbor* of v, and $\delta(v)$ the *degree* of v. We denote by Δ the maximum degree of G. Similarly, for a vertex set S, let $\Gamma(S) = \{u \in V \setminus S \mid (u, v) \in E \text{ for some } v \in S\}$, and $\Gamma(S)$ is called the *neighbor* of S. Let $\Lambda(S)$ be the set of all $v \in V \setminus S$ such that $(v, u) \in E$ for any $u \in S$. By definition, we have $\Lambda(S) \subseteq \Gamma(S)\ (\subseteq V \setminus S)$. For a vertex set S and an index i, let $S_{\leq i} = S \cap \{v_1, \ldots, v_i\}$. For two vertex sets X and Y, we say X is *lexicographically larger than* Y if the smallest vertex (i.e., a vertex with the smallest index) in $(X \setminus Y) \cup (Y \setminus X)$ is contained in X.

A vertex set $K \subseteq V$ is called a *clique* if any two vertices in K are adjacent, and a *maximal clique* if no other clique contains K in addition. For a clique K, let $C(K)$ denote the maximal clique that is the lexicographically largest among all maximal cliques

containing K. It is clear that $C(K)$ is not lexicographically smaller than K. For a bipartite graph $G = (V_1 \cup V_2, E)$, a vertex set K is called a *bipartite clique* if any vertex in $K \cap V_1$ is adjacent to any vertex in $K \cap V_2$, and *maximal* if no other bipartite clique contains K in addition.

4 Basic Algorithms

In this section, we explain the algorithms of Tsukiyama et al. [21] and Johnson et al. [14]. We view their algorithms as the enumeration algorithms based on reverse search, where *reverse search* was introduced by Avis and Fukuda [4] to solve enumeration problems efficiently. Note that our presentation of their algorithms is quite different from theirs [21, 14], which may be of independent interest.

Let K_0 denote the maximal clique that is the lexicographically largest among all maximal cliques. For a maximal clique $K\,(\neq K_0)$, we define a *parent* $P(K)$ of K by $C(K_{\leq i-1})$ such that i is the maximum index satisfying $C(K_{\leq i-1}) \neq K$. Such an index i is called the *parent index*, denoted by $i(K)$. Note that they are well-defined, since $K \neq C(K_{\leq 0})$ holds by $K \neq K_0$. Since $P(K)$ is lexicographically larger than K, this parent-child binary relation on maximal cliques is acyclic, and creates an in-tree rooted by K_0.

Lemma 1. *The parent-child relation constructs an in-tree rooted by* K_0. □

We call this in-tree the *enumeration tree* for maximal cliques of a graph G. Both algorithms [14, 21] traverse this enumeration tree. In order to traverse enumeration tree, we have to compute a parent and children of a given maximal clique efficiently.

It is not difficult to see that a parent $P(K)$ is computable from a maximal clique K in linear time. However, it is not so trivial to compute from K its children. For a maximal clique K and an index i, we define

$$K[i] = C\Big((K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\}\Big). \tag{1}$$

Lemma 2. *Let K and K' be maximal cliques in G. Then K' is a child of K if and only if $K' = K[i]$ holds for some i such that*

(a) $v_i \notin K$.
(b) $i > i(K)$.
(c) $K[i]_{\leq i-1} = K_{\leq i} \cap \Gamma(v_i)$.
(d) $K_{\leq i} = C(K_{\leq i} \cap \Gamma(v_i))_{\leq i}$.

Moreover, if an index i satisfies (a) $\sim$ (d)*, then i is the parent index of $K[i]$.* □

Since $C(K)$ can be computed from a clique K in $\mathrm{O}(m)$ time, by Lemma 2, we can compute all children of a given maximal clique in $\mathrm{O}(nm)$ time. Therefore, we can traverse the enumeration tree efficiently.

The algorithm of Tsukiyama et al. traverses the enumeration tree in a depth-first manner. Their algorithm starts with a root K_0, and find its children recursively. It is not difficult to see that the algorithm requires $\mathrm{O}(nm)$ time delay and $\mathrm{O}(n + m)$ space.

The algorithm of Johnson et al. enumerates all maximal cliques in the lexicographically decreasing order. Their algorithm initializes a queue Q as $Q = \{K_0\}$, iteratively extracts the lexicographically largest element K from Q and inserts into Q all

the children which are lexicographically smaller than K. The time complexity of their algorithm is same as the algorithm of Tsukiyama et al., however, it needs $\mathrm{O}(nN + m)$ space, where N denotes the number of all maximal cliques.

5 Using Matrix Multiplication

In this section, we describe an algorithm that runs with $\mathrm{O}(M(n))$ time delay and in $\mathrm{O}(n+m)$ space, where $M(n)$ denotes the time needed to multiply two $n \times n$ matrices. The algorithm uses matrix multiplication to find all children of a maximal clique when we traverse the enumeration tree.

Let us start restating conditions (c) and (d) in Lemma 2.

Lemma 3. *Let K be a maximal clique in G. Then an index i satisfies* (c) *if and only if no index j satisfies the following three conditions.*

(c-1) $j < i$.
(c-2) $v_j \notin K_{\leq i} \cap \Gamma(v_i)$.
(c-3) v_j *is adjacent to all vertices in* $K_{\leq i} \cap \Gamma(v_i) \cup \{v_i\}$. □

Lemma 4. *Let K be a maximal clique in G. Then an index i satisfies* (d) *if and only if no index j satisfies the following four conditions.*

(d-1) $j < i$.
(d-2) $v_j \notin K$.
(d-3) v_j *is adjacent to all vertices in* $K_{\leq j}$.
(d-4) v_j *is adjacent to all vertices in* $K_{\leq i} \cap \Gamma(v_i)$. □

Let us now consider computing all indices i such that $K[i]$ is a child of K. We denote by I_{a}, I_{b}, I_{c}, and I_{d} sets of the indices that satisfy conditions (a) $\sim$ (d) in Lemma 2, respectively. It is clear that I_{a} can be constructed from K in $\mathrm{O}(n)$ time and $\mathrm{O}(n)$ space. Since $i(K)$ can be computed in $\mathrm{O}(n+m)$ time, I_{b} can be constructed in $\mathrm{O}(n+m)$ time and $\mathrm{O}(n+m)$ space. From Lemma 3, we can compute I_{c} as follows.

For $\ell = 1, 2, 3$, let $Q_{(\mathrm{c}-\ell)}$ be an $n \times n$ matrix whose (i, j) element is 1 if i and j satisfy (c-ℓ) in Lemma 3; otherwise, 0. Then it is clear that $Q_{(\mathrm{c}-1)}$ and $Q_{(\mathrm{c}-2)}$ can be computed in $\mathrm{O}(n^2)$ time and $\mathrm{O}(n^2)$ space. However, we need $\mathrm{O}(n^3)$ time to compute $Q_{(\mathrm{c}-3)}$ if a naive method is applied. In order to compute $Q_{(\mathrm{c}-3)}$ efficiently, let Q be the matrix whose ith row is $x((K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\})$, where $x(S)$ denotes the characteristic vector of a set $S \subseteq V$. Then the (i, j) element of $Q \cdot A$ is the inner product of $x((K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\})$ and $x(\Gamma(v_j))$, where we recall that A denotes the adjacency matrix of G, and hence it is $|((K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\}) \cap \Gamma(v_j)|$. We can see that the (i, j) element is equal to $|(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\}|$ if and only if v_j satisfies condition (c-3) in Lemma 3. Thus $Q_{(\mathrm{c}-3)}$ can be obtained in $\mathrm{O}(M(n))$ time and $\mathrm{O}(n^2)$ space by computing $Q \cdot A$. This implies that I_{c} can be constructed in $\mathrm{O}(M(n))$ time and $\mathrm{O}(n^2)$ space.

Similarly, I_{d} can be constructed in $\mathrm{O}(M(n))$ time and $\mathrm{O}(n^2)$ space.

Therefore we have the following lemma.

Lemma 5. *Let K be a maximal clique of a graph G, and let I denote the set of all indices i such that $K[i]$ is a child of K, i.e., $I = I_{\mathrm{a}} \cap I_{\mathrm{b}} \cap I_{\mathrm{c}} \cap I_{\mathrm{d}}$. Then I can be computed in* $\mathrm{O}(M(n))$ *time and* $\mathrm{O}(n^2)$ *space.* □

We are now ready to describe our algorithm formally.

Algorithm ALLMAXCLIQUES
Input: A graph $G = (V, E)$.
Output: All maximal cliques of G.

Step 1. Compute the lexicographically largest maximal clique K_0 of G.
Step 2. Call ALLCHILDREN(K_0) and halt. □

Procedure ALLCHILDREN(K) /* K is a maximal clique in G. */

Step 1. Output K and compute the set I of all indices i such that $K[i]$ is a child of K.
Step 2. **For each** $i \in I$ **do**
Compute $K[i]$ and call ALLCHILDREN($K[i]$).
end.
Step 3. Return. □

Theorem 1. *For a given graph $G = (V, E)$, we can generate all maximal cliques of G with $O(M(n))$ time delay and in $O(n^2)$ space.* □

6 Algorithms for Sparse Graphs

In many practical applications, the given graphs G are sparse and only a few vertices have large degree. Such examples can be found in web networks [1]. In such cases, $\Omega(n)$ time delay is not effi cient enough. We fi rst consider the simplest case in which all vertices have small degree, i.e., Δ is small. We develop an algorithm for generating all maximal cliques with $O(\Delta^4)$ time delay and in $O(n + m)$ space, where $O(nm)$ time is required as a preprocessing.

Since $|K| \leq \Delta + 1$ holds for any clique K, given a clique K, we can compute $C(K)$ in $O(\Delta^2)$ and $O(n + m)$ space by repeatedly augmenting K. Therefore, we can compute $K[i]$ and $P(K)$ in $O(\Delta^2)$ time and $O(n + m)$ space.

The following lemma shows that any maximal clique $K (\neq K_0)$ has at most Δ^2 children.

Lemma 6. *For a maximal clique $K(\neq K_0)$, let K' be a child of K. Then $v_{i(K')} \in \Gamma(K_{\leq i(K')})$ holds.* □

Note that K_0 in general has $\Omega(n)$ children, and hence we compute them in $O(nm)$ time as a preprocessing.

Let us now describe an algorithm that runs with $O(\Delta^4)$ time delay and in $O(n+m)$ space. The algorithm is similar to ALLMAXCLIQUES in Section 5, but different in the following two points.

First, we do not construct I in Step 1 of Procedure ALLCHILDREN. If we store I in the algorithm, we require $O(n^2)$ space in general, since we need $O(n)$ space for each I and the depth of the recursion is $O(n)$. Instead, we check if $K[i]$ is a child of K in the lexicographic order of i's, and store the current i. This reduces the space to $O(n)$.

Second, we do not always output a maximal clique K before recursively calling Procedure ALLCHILDREN. From Lemma 6, Step 1 of Procedure ALLCHILDREN checks at most Δ^2 indices i, if $K \neq K_0$. Since each check can be performed in $O(\Delta^2)$ time, ALLCHILDREN requires $O(\Delta^4)$ time without considering its recursive calls. Thus, if we do not modify the algorithm, it runs $O(n\Delta^4)$ time delay, since the depth of the

recursion is $O(n)$. To reduce the time complexity, Procedure ALLCHILDREN outputs K before all its recursive calls, if the depth of the current recursion is odd; output K after all its recursive calls, otherwise. Although we skip the details, due to the space limitation (see [23] for more details), this reduces the delay to $O(\Delta^4)$.

Theorem 2. *For a given graph $G = (V, E)$, all maximal cliques of G can be generated with $O(\Delta^4)$ time delay and in $O(n + m)$ space, where $O(nm)$ time is required as a preprocessing.* □

We next consider a more general case. Let $G = (V = \{v_1, \ldots, v_n\}, E)$ be a graph such that $\delta(v_i) \leq \Delta^* (\ll \Delta)$ holds for $i = 1, \ldots, n - \theta$. Namely, only θ vertices in G have large degree ($> \Delta^*$). Let $V^* = \{v_{n-\theta+1}, \ldots, v_n\}$ and $G[V^*]$ denotes the subgraph of G induced by V^*. We divide the family $\mathcal{F}$ of all maximal cliques into two subfamilies $\mathcal{F}_1$ and $\mathcal{F}_2$, where $\mathcal{F}_1$ has all maximal cliques that are contained in V^* and $\mathcal{F}_2 = \mathcal{F} \setminus \mathcal{F}_1$. Our algorithm first generates all maximal cliques in the graph $G[V^*]$ and keeps them in the memory. This can be done in $O(\theta^3 N^*)$ time, by preparing the adjacency matrix of $G[V^*]$ as a preprocessing, where N^* denotes the number of all maximal cliques in $G[V^*]$. Note that this generates all maximal cliques in $\mathcal{F}_1$, but may generate some non-maximal cliques of G. Therefore, we remove them after generating all maximal cliques in $\mathcal{F}_2$. We remark that each non-maximal clique of G in $\mathcal{F}_1$ is contained in a maximal clique in $\mathcal{F}_2$, but no maximal clique in $\mathcal{F}_2$ contains more than one maximal clique in $\mathcal{F}_1$.

Formally our algorithm can be described as follows.

Algorithm ALLMAXCLIQUES*
Input: A graph $G = (V, E)$ such that the degree of v_i $(i = 1, \ldots, n - \theta)$ is at most Δ^*.
Output: All maximal cliques of G.

Step 1. Generates all maximal cliques in the graph $G[V^*]$ and store them in Q.
Step 2. Compute the lexicographically largest maximal clique K_0 of G.
Step 3. Call ALLCHILDREN(K_0), output all sets in Q, and halt. □

Procedure ALLCHILDREN$^*(K)$ /* K is a maximal clique of G contained in $\mathcal{F}_2$. */

Step 1. **if** K contains a clique K' in Q **then** remove K' from Q.
/* K contains at most one clique in Q, which is not a maximal clique of G. */
Step 2. Output K, compute $I = \{i \mid v_i \in \Gamma(K_{\leq i})\}$, and let $I^* := \emptyset$.
/* I is the set of candidates i such that $K[i]$ is a child of K. */
Step 3. **For each** $i \in I$ **do**
if $(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\} \not\subseteq V^*$
then begin check conditions (a) $\sim$ (d) in Lemma 2.
if they are satisfied **then** $I^* := I^* \cup \{i\}$.
end.
Step 4. **For each** $i \in I^*$ **do**
Compute $K[i]$ and call ALLCHILDREN$(K[i])$.
/* Note that $K[i] \not\subseteq V^*$ by $(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\} \not\subseteq V^*$. */
Step 5. Return. □

Let us show the correctness of the algorithm via a series of lemmas.

Lemma 7. *Let K ba a maximal clique that is contained in V^*. Then any descendant of K is contained in V^*.* □

From this lemma, $\mathcal{F}_2$ (i.e., the set of all maximal cliques containing a vertex in $V \setminus V^*$) forms a connected component of the enumeration tree that contains K_0. When we generate all maximal cliques in $\mathcal{F}_2$, we need not to traverse any descendant of a maximal clique K contained in V^*. Therefore, Step 4 of Procedure ALLCHILDREN* checks if $K[i] \not\subseteq V^*$ before going to the recursion.

The next lemma, together with Lemma 6 shows that the number of candidates i such that $K[i]$ is a child of K ($\neq K_0$) is small (see I in Step 2 of ALLCHILDREN*).

Lemma 8. *Let K ($\neq K_0$) ba a maximal clique that contains a vertex in $V \setminus V^*$. Then we have $|\{i \mid v_i \in \Gamma(K_{\leq i})\}| \leq (\Delta^* + 1)(\Delta^* + \theta)$.* □

Let us then consider constructing $K[i]$ from K and i in Step 4 of ALLCHILDREN*.

Lemma 9. *Let K be a clique including a vertex $v \in V \setminus V^*$. Then $C(K)$ can be computed in $\mathrm{O}((\Delta^*)^2)$ time and $\mathrm{O}(n\theta)$ space.* □

From this lemma, if $(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\} \not\subseteq V^*$, then we only need $\mathrm{O}((\Delta^*)^2)$ time to construct $K[i]$ and to check conditions (a) $\sim$ (d) in Lemma 2. However, we note that $\Omega(n)$ time is needed to construct $K[i]$ if $(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\} \subseteq V^*$. The following lemma overcomes such diffi culty.

Lemma 10. *Let K be a maximal clique in G. If $(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\} \subseteq V^*$, then $K[i]$ is either not a child of K or a maximal clique contained in V^*.* □

Base on this lemma, Step 3 of ALLCHILDREN* checks if $(K_{\leq i} \cap \Gamma(v_i)) \cup \{v_i\} \subseteq V^*$ before checking the conditions in Lemma 2.

We are now ready to present our theorem.

Theorem 3. *Let G be a a graph with $n - \theta$ vertices of degree at most Δ^*. Then all maximal cliques of G can be enumerated with amortized $\mathrm{O}((\Delta^*)^3(\Delta^* + \theta) + \theta^3)$ time delay and in $\mathrm{O}((n + N^*)\theta + m)$ space, where N^* denotes the number of all maximal cliques in $G[V^*]$, and $\mathrm{O}(nm)$ time is required as a preprocessing.* □

We remark that θ is small in practical cases. For example, we have $\theta \leq \log n$ in web networks, where it is called *power law* [1, 15]. Therefore, the memory required in the application is not so large.

7 Enumeration of All Maximal Bipartite Cliques

In this section we consider enumerating maximal bipartite cliques in a bipartite graph.

For a bipartite graph $G = (V_1 \cup V_2, E)$, let $V_1 = \{v_1, \ldots, v_{n_1}\}$ and $V_2 = \{v_{n_1+1}, \ldots, v_n\}$. We assume without loss of generality that no vertex v satisfi es $\Gamma(v) = V_1$ or V_2. Recall that the generation of all maximal bipartite cliques in G can be regarded as the one of all maximal cliques in the graph $\hat{G}$ obtained from G by adding edges so that V_1 and V_2 both become cliques. We denote by $\hat{\Gamma}$ and $\hat{C}$ as Γ and C for $\hat{G}$; e.g., for a vertex v, $\hat{\Gamma}(v) = \Gamma(v) \cup V_1$ if $v \in V_1$, and $\hat{\Gamma}(v) = \Gamma(v) \cup V_2$ if $v \in V_2$. We frequently use $\hat{\Gamma}$ and $\hat{C}$ instead of Γ and C to follow the results obtained in the previous sections. For example, we defi ne $K[i]$ by

$$K[i] = \hat{C}\Big((K_{\leq i} \cap \hat{\Gamma}(v_i)) \cup \{v_i\}\Big) \tag{2}$$

Before describing our algorithms, let us present several good properties for bipartite graphs to reduce the complexity of our problem.

Lemma 11. *Let K $(\neq K_0)$ be a maximal bipartite clique in G. If $i > i(K)$, then we have*

(i) *$K[i]$ can be represented as*

$$K[i] = \Big(K \cap \Gamma(v_i)\Big) \cup \Big(\Lambda(K \cap \Gamma(v_i))\Big). \tag{3}$$

(ii) *$K[i]_{\leq i-1} = K_{\leq i} \cap \hat{\Gamma}(v_i)$ is equivalent to* (c') *$\Lambda(K \cap \Gamma(v_i)) - K = \emptyset$.*
(iii) *$K_{\leq i} = \hat{C}(K_{\leq i} \cap \hat{\Gamma}(v_i))_{\leq i}$ is always satisfied.* □

From Lemmas 2 and 11, we can reduce the delay to $\mathrm{O}(\Delta^3)$ time for maximal bipartite cliques.

Theorem 4. *Let G be a bipartite graph. Then all maximal bipartite cliques can be generated with $\mathrm{O}(\Delta^3)$ time delay and in $\mathrm{O}(n+m)$ space, where $\mathrm{O}(nm)$ time is required as a preprocessing.* □

Moreover, the delay can be improved, if we use additional space.

Theorem 5. *Let G be a bipartite graph. Then all maximal bipartite cliques can be generated with $\mathrm{O}(\Delta^2)$ time delay and in $\mathrm{O}(n + m + N\Delta)$ space, where N denotes the number of all maximal bipartite cliques in G and $\mathrm{O}(nm)$ time is required as a preprocessing.* □

8 Computational Experiments

To evaluate the performance of our algorithms, we implement our algorithms for sparse graphs in Theorems 2 and 4. We also implement the algorithm of Tsukiyama et al., and adapt it for bipartite graphs. Our codes are written in C, and the programs run in a PC of Pentium III 500MHz with 256MB memory, whose OS is Linux. We examine these algorithm by using graphs that are generated randomly and taken from word data of newspapers in computational linguistics. Their experimental results can be found in the table below.

Our random graphs are generated as follows. For given r and n, we construct a graph with n vertices such that v_i and v_j is adjacent with probability $1/2$ if $i + n - j \pmod n \leq r$ or $j + n - i \pmod n \leq r$. Bipartite graphs are constructed similarly, where we have $|V_1| = |V_2|$. We examine the cases of $r = 10, 30$ and $n = 1000, 2000, 4000, \ldots, 256000$. Exp. 1 and 2 (resp., Exp. 3) represent the results for generating all maximal cliques. (resp., all maximal bipartite cliques). Exp. 1 (resp. Exp. 3) shows the computational time to generate 10000 maximal cliques (resp., maximal bipartite cliques), as well as the number of all maximal cliques (resp., all maximal bipartite cliques), where the computational time in the table is expressed in seconds, and we only output the first 10000 cliques, if the computational time is over 3 hours. Exp. 2 shows the the computational time of our algorithm per a maximal clique. We also construct graphs G such that a few vertices of G have large degree, by adding 40

vertices and edges adjacent to such vertices with probability $1/2$ to graphs that are generated randomly for $r = 10$. Similarly to Exp. 2, Exp. 4 shows the the computational time of our algorithm per a maximal clique. Finally, we examine our algorithm for real data P1, P2 and P3 which are taken from computational linguistics. The result is shown in Exp. 5.

Exp. 1: maximal cliques, $r = 10, 30$

# vertices		1000	2000	4000	8000	10000	16000	32000	64000	128000	256000
Tsukiyama	$r = 10$	378	761	1410	3564	5123					
Tsukiyama	$r = 30$	1755	4478	9912	21085	25345					
Ours	$r = 10$	0.64	0.65	0.72	0.73	0.72	0.74	0.75	0.81	0.82	0.82
Ours	$r = 30$	4.41	4.44	4.47	4.56	4.51	4.54	4.55	4.91	4.88	4.88
# output	$r = 10$	2774	5553	11058	22133	27624	44398	89120	179012	357657	716978
# output	$r = 30$	20571	41394	83146	168049	209594	336870	675229	1352210	2711564	5411519

Exp. 2: maximal cliques , # vertices = 10000

r	10	20	40	80	120	160	240	320	480	640
Ours	0.23	0.31	0.51	1	1.7	2.4	4.1	5.7	9.8	14

Exp. 3: maximal bipartite cliques, $r = 10, 30$

# vertices		1000	2000	4000	8000	10000	16000	32000	64000	128000	256000
Tsukiyama	$r = 10$	104	214	446	966	1260					
Tsukiyama	$r = 30$	282	582	1190	2455	3100					
Ours	$r = 10$	0.33	0.32	0.3	0.3	0.27	0.3	0.3	0.34	0.34	0.35
Ours	$r = 30$	1.08	1.08	1.09	1.1	1.09	1.11	1.12	1.22	1.22	1.26
# output	$r = 10$	2085	4126	8316	16609	20862	33586	67143	134911	270770	541035
# output	$r = 30$	9136	18488	40427	68597	101697	165561	322149	625385	1233989	8351277

Exp. 4: including 40 vertices with large degree, $r = 10$

# vertices	1000	2000	4000	8000	10000	16000	32000	64000	128000	256000
Ours	1.07	1.14	1.12	1.31	1.21	1.36	1.74	2.62	4.02	7.8
# output	9136	18488	40427	68597	101697	165561	322149	625385	1233989	2307135

Exp. 5: Real world data

	# vertices(V_1, V_2)	# edges	# max cliques	time
P1	22677,18484	247003	2700737	291
P2	33347,32757	233450	1892469	255
P3	20433,4297	127713	11860169	974

From the results in Exp. 1 and 3, we can see that our algorithms are much faster than the algorithm of Tsukiyama et al. The computational time of the algorithm of Tsukiyama et al. is linear to the number of vertices, but the one of our algorithm does not depend the number of vertices, since the maximum degree is small. From Exp. 2, we can see that the computational time of our algorithm per a maximal clique is close to $O(\Delta)$, which is almost linear in the output size. Exp. 4 shows that the computational time does not increase so much, even if the graphs contain some vertices of large degree. Exp. 5 shows that problems P1, P2 and P3 can be solved effi ciently. We note that the algorithm of Tsukiyama et al. did not terminate for these problems by 3 hours.

References

1. A.-L. Barabasi, "LINKED – The New Science of Networks," Perseus Publishing, 2002.
2. R. Agrawal and R. Srikant, Fast algorithms for mining association rules in large databases, *Proc. VLDB '94*, pp. 487–499, 1994.
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast discovery of association rules, *In Advances in Knowledge Discovery and Data Mining*, MIT Press, pp. 307–328, 1996.

4. D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete App. Math.*, 65 (1996) 21–46.
5. E. Boros, V. Gurvich, L. Khachiyan and K. Makino, On the complexity of generating maximal frequent and minimal infrequent sets, *Annals of Math. and Artif. Int.*, 39 (2003) 211-221.
6. E. Boros, K. Elbassioni, V. Gurvich, L. Khachiyan and K. Makino, Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities, *SIAM J. Comput.*, 31 (2002) 1624–1643.
7. N. Chiba and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.*, 14 (1985) 210-223.
8. D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progression, *Journal of Symbolic Computation*, 9 (1990) 251–280.
9. T. Eiter, G. Gottlob and K. Makino, New results on monotone dualization and generating hypergraph transversals, *SIAM J. Comput.*, 32 (2003) 514–537.
10. T. Eiter and K. Makino, On computing all abductive explanations, *Proc. AAAI '02*, AAAI Press, pp. 62–67, 2002.
11. D. Eppstein, Arboricity and bipartite subgraph listing algorithms, *Info. Proc. Lett.*, 51 (1994) 207-211.
12. B. Ganter and R. Wille, *Formal Concept Analysis*, Springer, 1996.
13. L. A. Goldberg, *Effi cient algorithms for listing combinatorial structures*, Cambridge University Press, New York, 1993.
14. D. S. Johnson, M. Yanakakis and C. H. Papadimitriou, On generating all maximal independent sets, *Info. Proc. Lett.*, 27 (1998) 119–123.
15. S. R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, Trawling the web for emerging cyber-communities, *Proc. the Eighth International World Wide Web Conference*, Toronto, Canada, 1999.
16. E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Generating all maximal independent sets, NP-hardness and polynomial-time algorithms, *SIAM J. Comput.*, 9 (1980) 558–565.
17. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, Discovering frequent closed itemsets for association rules, *Proc. the 7th ICDT Conference*, *LNCS* 1540, Springer, pp. 398-416, 1999.
18. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, Closed set based discovery of small covers for association rules, *Proc. 15emes Journees Bases de Donnees Avancees*, pp. 361-381, 1999.
19. R. C. Read and R. E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks*, 5 (1975) 237-252.
20. B. Selman and H. J. Levesque, Support set selection for abductive and default reasoning, *Artif. Int.*, 82 (1996) 259–272.
21. S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM J. Comput.*, 6 (1977) 505–517.
22. T. Uno, Fast algorithms for computing web communities and frequent sets by using maximal clique generations, In preparation.
23. T. Uno, Two general methods to reduce delay and change of enumeration algorithms, Technical Report of National Institute of Informatics, Japan, 2003.
24. T. Uno, T. Asai, H. Arimura and Y. Uchida, LCM: An efficient algorithm for enumerating frequent closed item sets, Workshop on Frequent Itemset Mining Implementations (FIMI'03).
25. M. J. Zaki and M. Ogihara, Theoretical foundations of association rules," *3rd SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, June 1998.