Honours Year Project Report

Mining Algorithm for Bicliques

By

Chen Xiankun

Department of Computer Engineering

School of Computing

National University of Singapore

2007/2008

Honours Year Project Report

Mining Algorithm for Bicliques

By

Chen Xiankun

Department of Computer Engineering

School of Computing

National University of Singapore

2007/2008

Project No: H114110
Advisor: Prof Wong Lim Soon
Deliverables:
Report: 1 Volume

Abstract

There exist several existing algorithm for computing maximal bicliques (or complete bipartite) subgraphs or an undirected graph with origins in graph theory or in data mining. In this project, three algorithms, two depth first search graph algorithm, one from Tomita and one from Makino both using tree construction and pruning, and a closed-pattern-based method using frequent pattern matching are explained and implemented and their efficiency compared based on different random graphs generated.

Subject Descriptors:

F.2.2 Nonnumerical Algorithms and Problems
G.2.2 Graph Theory
G.4 Mathematical Software

Keywords:

maximal bicliques, algorithms, performance, search trees

Implementation Software and Hardware:

Linux/Unix, C++

Acknowledgement

I would like to express my heartfelt gratitude to my project supervisor, Prof .Wong Limsoon for his timely and useful advice during the past year. His advice has been very useful for the completition of this project.

I would also like to thank my family and friends who have been supportive and understanding to me for this past year.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 Introduction

1.1 Motivation

Maximal biclique (or complete bipartite) subgraphs are useful in the modelling of many real-life applications. Two applications are listed as follows. The first application comes from social networks where the data is a bipartite graph of what is known as 'affiliation' networks. Examples of such networks would be the scientific collaboration networks with the two node sets consisting of authors and papers, or the movie recommendation network where the edge set connects users to the movies they have watched. Web communities can be identified through identifying maximal bicliques from such web networks. Another application would be from the metabolic network from the biological field, where the nodes sets are enzymes and the list of reaction they participate in, the biclique subgraphs obtained would help us in determine the function of detected communities.

The problem of maximal bicliques have long been studied and there exists several algorithms for enumerating them. Some common approaches include clique algorithm (Tomita et al, 2006) where nodes are added to a search tree and branches are pruned if they are unable to generate a maximal biclique, algorithm suggested by Makino and Uno based on generating child nodes which are maximal bicliques from its parent in a tree based on a certain relation which is acyclic (Makino and Uno, 2004), and closed pattern finding with roots in data mining. However, such algorithms have not been directly compared with in terms of efficiency.

1.2 Project Objective

With the above motivation, the objective of the project is to implement some of these algorithms, model different conditions and evaluate their performance under these conditions. Time would be the main criterion used for evaluating the performance of the algorithms.

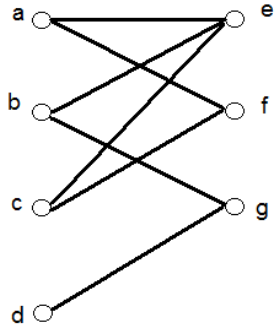
1.3 Report Structure

The rest of the paper will be organised as such. Chapter two of the paper will define notation used throughout the paper, Chapter three will explain the graph-based algorithms used and modification made. Chapter four will explain the algorithm based on data mining. Chapter five will explain the background for testing and the generation of the random graphs used for the experiments. Chapter six will present the results and chapter seven will conclude the paper.

2 General Definitions and Notations

Let G = (V, E) be a simple undirected graph with a vertex set V = $\{v_1, \ldots, v_n\}$ and an edge set E = $\{e_1, \ldots, e_m\}$ where each edge is an unordered pair (v, w) of distinct vertices. A pair of vertices v and w are said to be adjacent if $(v, w) \in E$. For a subset $W \subseteq V$, G(W) = (W, E(W)) with $E(W) = \{(v, w) \in W \times W \mid (v, w) \in E\}$ is called an induced subgraph of G by W. If there is a partition V_1 and V_2 of V such that no two vertices in V_i, where i = 1 or 2 are adjacent, G is known as a bipartite graph and we denote by G = $(V_1 \cup V_2, E)$. A vertex set K is called a bipartite clique is a induced subgraph of G such that any vertex in $K \cap V_1$ is adjacent to any vertex in $K \cap V_2$, and maximal in the sense that no other bipartite clique contains K in addition.

For a vertex v of G, $\Gamma(v) = \{u \in V \mid (u, v) \in E\}$ is also known as the neighbour of v. Also, for any vertex set S, we define $\Lambda(S)$ as the set of $v \in V \setminus S$ such that (u, v) for any $u \in S$. Let $\delta(v) = |\Gamma(v)|$ denote the degree of v and Δ the maximal degree of G. The adjacency list of G is the set of all $\Gamma(v)$. Figure 1 below illustrates the corresponding adjacency list and matrix for a graph G.



Adjacency List	
Vertex v	Γ(v)
a	{e,f}
b	{e,g}
c	{e,f}
d	{g}
e	{a,b,c}
f	{a,c}
g	{b,d}

Adjacency Matrix							
	a	b	c	d	e	f	g
a	0	0	0	0	1	1	0
b	0	0	0	0	1	0	1
c	0	0	0	0	1	1	0
d	0	0	0	0	0	0	1
e	1	1	1	0	0	0	0
f	1	0	1	0	0	0	0
g	0	1	0	1	0	0	0

Figure 1: A graph G, its corresponding adjacency list and adjacency matrix.

Note that the graph in Figure 1 is bipartite, where the partition $V_1 = \{a,b,c,d\}$ and $V_2 = \{e,f,g\}$. We also note that the subgraph $\{\{a,b\} \cup \{e\}, \{(a,e), (b,e)\}\}$ is a bipartite clique of G but not maximal. A maximal bipartite clique of G would be $\{\{a,c\} \cup \{e,f\}, \{(a,e), (c,e), (a,e),(a,f)\}\}$.

A sparse graph is a graph where $|E| = O(|V|)$ and a dense graph is a graph where $|E| = \Theta(|V^2|)$. A locally sparse graph is a graph where different vertices v_i and v_j are adjacent with probability ½ if $i - j + n \pmod n \leq r$ or $j + n - i \pmod n \leq r$, for a certain (usually small) value of r. Locally sparse graph are commonly encountered when modelling real networks due to physical or cost constraints.

For purposes of comparing, for any two vertex sets X and Y, 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.

Finally, we may be interested to find only maximal biclique subgraphs meeting a specific size condition. This is because not all maximal biclique subgraphs are interesting to us as some might be trivial or too small to be of interest. A maximal biclique subgraphs $H = \{V_1 \cup V_2, E\}$ is said to be (p, q) large if either $|V_1| \geq p$, $|V_2| \geq q$ or $|V_1| \geq q$, $|V_2| \geq p$. Note that if we want to enumerate all maximal biclique subgraphs, it is equivalent to finding all (1, 1) large maximal biclique subgraphs.

3 Graph-Based Approach

3.1 Tomita's CLIQUES Algorithm

A depth-first search algorithm based on Tomita algorithm for clique generation was considered. Q is a global variable of a set of vertices that constitutes a complete subgraph (clique) found up to this time. The algorithm begins by letting Q be an empty set and expands Q step by step by applying a recursive procedure EXPAND to V and its succeeding induced subgraphs to search for larger and larger complete subgraphs until they reach maximal cliques. Hence, if Q = $\{v_1, v_2, \ldots v_d\}$at some point, then the variable SUBG is a set $V \cap \Gamma(v_1) \cap \Gamma(v_2) \cap \ldots \cap \Gamma(v_d)$. Applying the procedure EXPAND at each stage, when SUBG = $\varnothing$, Q will be a maximal clique. Otherwise, Q U {q} is a larger complete subgraph for every $q \in$ SUBG, and we can consider the smaller subgraphs induced by the addition of each $q \in$ $SUBG_q$ where $SUBG_q = SUBG \cap \Gamma(q)$ for each q. We can then apply recursively EXPAND to $SUBG_q$ to find larger complete subgraphs containing Q U {q}.

Pruning is done with the inclusion of two more variables FINI and CAND, where FINI is a subset of SUBG that have already be processed by the algorithm and CAND being those that have not been processed. We therefore have CAND = SUBG – FINI. Letting $CAND_q = CAND \cap \Gamma(q)$ and $FINI_q = FINI \cap \Gamma(q)$, it follows that only the vertices in $CAND_q$ need to be considered as candidates for expanding the complete subgraph Q U {q} to find new larger cliques, since all the cliques containing Q U {q} U {r} where $r \in FINI_q$ and thus not in $CAND_q$ have been generated by application of the procedure EXPAND to r already.

Another optimising used was to prune the search subtrees to be expanded. Given a certain vertex $u \in$ SUBG, assume all maximal cliques containing Q U {u} have been generated. Then, every new maximal clique containing Q, but not Q U {u}, must contain at least one vertex $q \in SUBG - \Gamma(u)$. Thus, we only need to expand Q to Q U {q} such that $q \in$

CAND - Γ(u) instead of q ∈ CAND. Choosing u ∈ SUBG to minimize |CAND - Γ(u)| will thus minimize the number of tree branches. This is achieved by finding and choosing u ∈ SUBG to maximise |CAND ∩ Γ(u)|.

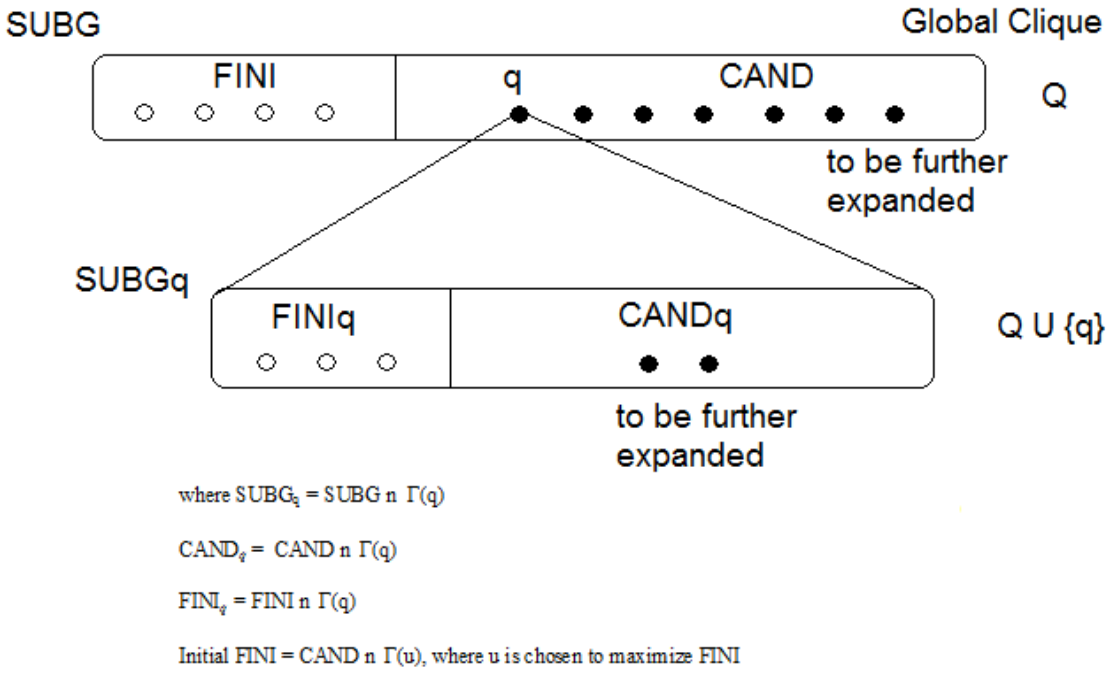


Figure 2: Illustration of procedure EXPAND for generating maximal cliques

Tomita's algorithm can be shown to have a worse-case running time of $O(n3^{n/3})$ for an n-vertex graph. This is efficient as the maximal number of cliques possible is $3^{n/3}$ and it would take O(n) to print out each clique. If we merely print out the vertex we are expanding and not the entire clique each time a clique is discovered, we would obtain a worse case timing of $O(3^{n/3})$

The algorithm is summarised as

```
procedure CLIQUES(G)
    /* Graph G = (V,E) */
begin
    Q := ∅; //Q constitutes a global clique
    EXPAND(V,V);
end of CLIQUES

procedure EXPAND(SUBG, CAND)
begin
    if SUBG = ∅
    then print Q; //Q is a maximal clique at this point
    else
        u := a vertex in SUBG that maximises |CAND ∩ Γ(u)|
        while CAND - Γ(u) ≠ ∅
        do q:= a vertex in (CAND - Γ(u));
            Q := Q ∪ {q};
            SUBG_q := SUBG ∩ Γ(q);
            CAND_q := CAND ∩ Γ(q);
            EXPAND(SUBG_q , CAND_q );
            CAND := CAND – {q};
            Q := Q – {q};
        od
    fi
end EXPAND
```

3.2 Modified CLIQUES Algorithm

Tomita's algorithm could generate maximal cliques from a general graph. To use this algorithm, we observe that to generate maximal bipartite clique from a bipartite graph is a special case of the maximal clique problem in a general graph. Given a bipartite graph $G = (V_1 \cup V_2, E)$, it is easy to transform it to a general graph where $G = (V_1 \cup V_2, E_1)$, where $E_1 = E \cup (V_1 \times V_1) \cup (V_2 \times V_2)$ and we can apply the EXPAND procedure for it. However, we eliminate any the clique set generated V where $V = V_1$, or $V = V_2$. This is because such cliques do not have any edges after removing the extra edges we added in as they comprise of vertices from one bipartite set only.

For generation of maximal bipartite from a non bipartite graph $G = (V, E)$, duplication of the non bipartite graph was made such that the new graph $G = (V_1 \cup V_2 , E_1)$ where $|V_1| = |V_2| = |V|$ and $E_1 = (V_1 \times V_1) \cup (V_2 \times V_2) \cup E_n$,and $(v_{1j}, v_{2k}) \in E_n$ if $(v_j , v_k) \in E$ where v_{1j} denotes the jth vertex in V_1, v_{2k} denotes the kth vertex in V_2 and v_j , v_k denotes the jth and kth vertex in the original graph. Essentially, this creates a duplicate set of the original graph, and duplicates the edges. We can then apply the modified algorithm above for this. However, this will cause duplicates in the maximal bicliques generated and will have to be filtered out. A simple check at this point would be only to generate it as a maximal biclique if the size of the first is no smaller than the size of the second, i.e $Q \cap V_1 \geq Q \cap V_2$ where Q is the clique generated by Tomita's algorithm. To eliminate bicliques of the same size, this is done in the output processing portion of the algorithm.

The other change to be made lies in the (p, q) - large filtering condition ($p \leq q$). At each call, the number of remaining vertices on each side of the bipartite input graph is added to the number of vertices already in the set Q (which constitutes the maximal biclique) on the same side of the bipartite input graph. It is then checked whether it is possible to form a (p, q) large biclique. If not, then we can prune the rest of this branch of the search tree. For a biclique graph input, we check that both sums are at least p and that at least one is not smaller than q. For a non-biclique graph input, we modify the condition due to symmetry and check

that the vertices belonging in V_1 is at least q and the vertices belonging in V_2 is at least p (due to only needing $Q \cap V_1 \geq Q \cap V_2$).

A bitSet was chosen to represent the adjacency list and other sets as there were a lot of union and intersection operations. The minus operation was implemented as the intersection with the complement.

The modified algorithm for the input graph being non bipartite is shown as below (the algorithm for bipartite input graph being very similar). LQ_SIZE and RQ_SIZE holds the current size of Q belonging in V_1 and V_2 respectively, and LMAX and RMAX calculates the maximum possible size of the subgraph possible for $Q \cup \{q\}$ belonging in V_1 and V_2 respectively**.**

procedure BICLIQUES(G)

/* Graph G = (V,E) */

begin

$V_c := V_1 \cup V_2$ where $|V_1| = |V_2| = |V|$;

$E_1 = (V_1 \times V_1) \cup (V_2 \times V_2) \cup E_n$ where $(v_{1j}, v_{2k}) \in$ En if $(v_j, v_k) \in E$

Q := ∅; //Q constitutes a global clique

EXPAND(V_c, V_c, 0 , 0);

end of BICLIQUES

procedure EXPAND(SUBG, CAND, LQ_SIZE, RQ_SIZE)

begin

if SUBG = ∅

then if LQ_SIZE ≥ q and RQ_SIZE ≥ p and LQ_SIZE ≥ RQ_SIZE

print Q; //Q is a maximal clique at this point

fi

else // (SUBG != ∅)

u **:=** a vertex in SUBG that maximises |CAND ∩ Γ(u)|

while CAND - Γ(u) ≠ ∅

do q:= a vertex in (CAND - Γ(u));

Q := Q ∪ {q};

$SUBG_q$:= SUBG ∩ Γ(q);

$CAND_q$:= CAND ∩ Γ(q);

if q ∈V_1

then LQ_SIZE := LQ_SIZE+1;

else RQ_SIZE := RQ_SIZE + 1;

fi

LMAX = LQ_SIZE + |$SUBG_q$∩ V_1|;

RMAX = RQ_SIZE + |$SUBG_q$∩ V_2|;

if LMAX ≥ q && RMAX ≥ p //Still possible to find a (p,q) biclique

then EXPAND($SUBG_q$, $CAND_q$, LQ_SIZE, RQ_SIZE);

fi

CAND := CAND – {q};

Q := Q – {q};

if q ∈V_1

then LQ_SIZE := LQ_SIZE - 1;

else

RQ_SIZE := RQ_SIZE - 1;

fi

od

fi

end EXPAND

3.3 Makino Algorithm

A depth first search algorithm based on papers written by Makino and Uno was also considered. Their algorithm is based on reverse search (Avis and Fukuda, 1996) and based on works done earlier by Tsukiyama et al. and Johnson et al.

For any vertex set S and an index i, let $S_{\leq i}$ denote the vertices in S such that the index is less than or equal to i, i.e $S \cap \{v_1, \ldots v_i\}$. For a biclique K, let C(K) denote the maximal biclique that is the lexicographically largest among all maximal bicliques containing K. Note that C(K) is not lexicographically smaller than K.

First of all, the algorithm finds the lexicographically largest biclique, denoted by K_0. For each maximal biclique K ($K \neq K_0$), its parent P(K) of K is defined by $C(K_{\leq i-1})$ such that i is the maximal index satisfying $C(K_{\leq i-1}) \neq K$. This i is denoted as the parent index, denoted by i(K). This is well defined, since $K \neq C(K_{\leq 0})$ holds as $K \neq K_0$. Since P(K) is lexicographically larger than K, this parent-child binary relation on maximum bicliques is thus acyclic, and creates a tree with root at K_0. Every child node will have exactly one parent node, and so this forms our search tree.

It can be shown that K' is a child of K if and only if K' = K[i] for some i such that

a) $v_i \notin K$

b) $i > i(K)$

c) $(\Lambda(K \cap \Gamma(v_i)) - K)_{\leq i-1} = \varnothing$

where $K[i] = (K \cap \Gamma(v_i)) \cup (\Lambda(K \cap \Gamma(v_i))$. If an index satisfies (a), (b) and (c), then i is the parent index of K[i].

Adjacency List	
Vertex v	Γ(v)
0	{4,5}
1	{3,5}
2	{4,6}
3	{1}
4	{0,2}
5	{0,1}
6	{2}

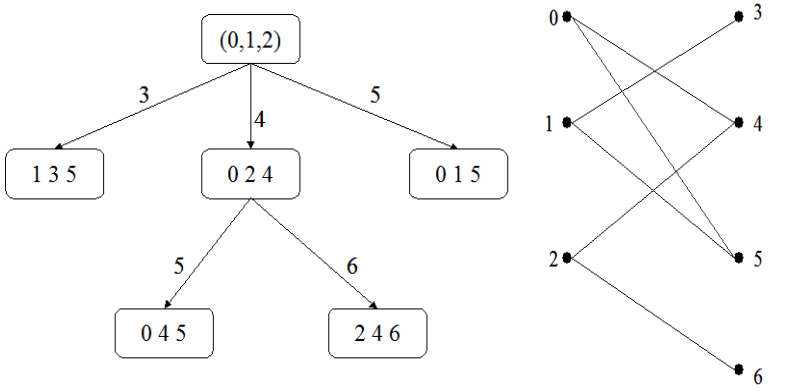


Figure 3: Tree constructed using Makino's Algorithm, where the lines show the index of the parent, and the boxes show the maximal bicliques(exception of (0,1,2)), the adjacency List and the graph.

Referring to Figure 3, applying the algorithm to K_0, we obtain 3 as a value for i. With that value, we compute $K \cap \Gamma(v_3)$ to be v_1 and $\Lambda(v_1) = \{v_1, v_3, v_5\}$ as the new child with parent index 3.

For checking of valid i's, it is easy to check the first condition, and for the second condition, the parameter i is passed in as the parent index to ease checking. Only when both conditions are satisfied do we then proceed to check for the third condition.

Once we find the set of all i, we can compute K[i] which are maximal bipartite graphs with parent as K. To optimise this, we note that $(\Lambda(K \cap \Gamma(v_i))$ has been computed earlier, and we can use it for the computation of K[i]. We then find the child nodes of K[i] recursively. The initial value of K_0 is passed in as all the vertices in V_1, and the index the size of V_1. This initial K_0 can be ignored as a maximal bipartite clique.

The algorithm is summarised as follows for a bipartite graph input (the algorithm for non-bipartite input graph is similar).

procedure MAXBICLIQUES(G)

/* Graph G (bipartite) = ($V_1 \cup V_2$,E) */

begin

ALLCHILDREN(V_1,|V_1|);

end of MAXBICLIQUES

procedure ALLCHILDREN(K, I) /* K is a maximal biclique of G, i is its index */

print K;

ALLI := ∅ //ALLI is to hold all possible values for I.

for each i ≥ I

if i satisfies $(\Lambda(K \cap \Gamma(v_i)) - K)_{\leq i-1} = \emptyset$

ALLI := ALLI ∪ {v_i};

end

for each i ∈ ALLI

compute K[i];

call ALLCHILDREN (K[i], i);

end

end of ALLCHILDREN

The algorithm is shown to be of the order of $O(\Delta^3)$, where Δ is the maximal degree of the graph. However, it is not easy to limit ourselves to the (p,q) large criteria for this algorithm except at the output stage and as such, this algorithm testing and analysis is only restricted to the case where we enumerate all maximal bicliques. Furthermore, for the case of

non-bipartite input graph, the duplication of nodes will lead to duplication of bicliques generated, and so will have to be filtered out at the output stage.

4 Closed Pattern-Based Approach

4.1 Basis

The approach views the adjacency matrix as a transactional database. This is a non-empty multi-set of transaction, where each transaction is a subset of a pre-specified set I of items. Each of these transaction can be viewed thus as the neighbours of the given vertex and I corresponds to the entire vertex set of the graph. A pattern is defined as a non-empty set of items of I. Given a transactional database and a pattern P, the transactions containing P forms the occurrence set of P denoted as $occ^{DB}(P)$, and the cardinality of this is the support of P, denoted $sup^{DB}(P)$. Furthermore, the closure of P, $CL^{DB}(P)$ is defined as the set of items shared by all the transactions containing pattern P in the database, i.e the set of items shared by all transactions in the occurrence set.

It is trivial to see that $P \subseteq CL^{DB}(P)$. However, if $P = CL^{DB}(P)$, it forms a closed pattern of DB. The occurrence set of such closed patterns is another closed pattern, and these give rise to the two vertex sets of maximal biclique subgraph. If graph $H = \{V_1 \cup V_2, E\}$ is a maximal biclique subgraph of G, V_1 and V_2 are both closed patterns of DB_G and we have $occ^{DB}(V_1) = V_2$ and $occ^{DB}(V_2) = V_1$.

4.2 Frequent Pattern Algorithm

The Frequent Pattern tree is a representation of all relevant frequency information in a database. Every branch of the tree represents a frequent itemset, and nodes are stored in decreasing order of frequencies of the corresponding items, with the leaves having the least frequent items. The tree is built in such a way that overlapping itemsets share prefixes of the corresponding branches

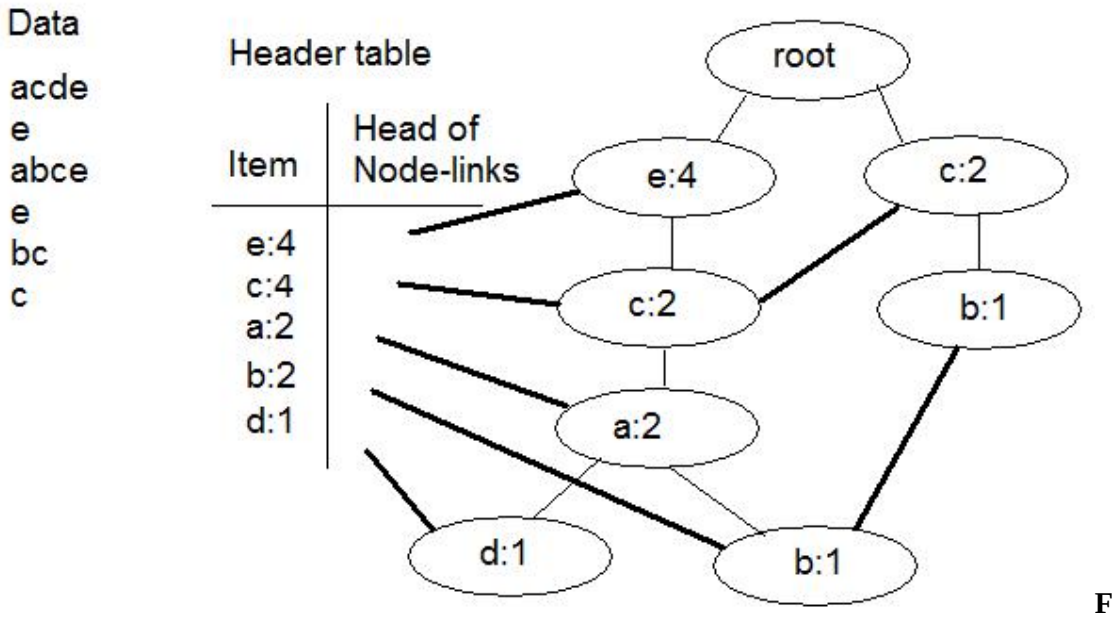


Figure 4: A FP tree constructed from data and its header table storing the links to nodes

Referring to figure 4, to construct the FP-tree, we first find all the items {a,b,c,d,e} by an initial scan. They are inserted into the header table in decreasing order of count. On the second scan, as each data is scanned, if it shares a prefix with an itemset already in the tree, it will share the prefix of the branch representing that itemset. In addition, each branch keeps track of a counter to represent the number of transactions containing the itemset represented by the path from the root to the node. For example, the d:1 leaf nodes shows that there is one and only one itemset containing {e,c,a,d} whereas the a:2 node above it shows that there are 2 itemset containing {e,c,a}. We can thus view our adjacency list as a set of items and arrange them in such a tree to find the support for a pattern.

Searching for maximal bicliques is thus equivalent to finding such closed patterns in the tree. An implementation using this algorithm for finding maximal bicliques, FP-MBC, developed by Guimei Liu was used for this comparison. For more information, refer to A Correspondence Between Maximal Complete Bipartite Subgraphs and Closed Patterns by Jinyan Li, Haiquan Li, Donny Soh, Limsoon Wong.

5 Computational Experiments

5.1 Timings

Timing for algorithms is divided into three portions, input processing time, bicliques generating time and any additional output processing time. The input processing time is the time required to read in the raw data and store it in a data structure, for the case of the graph based approach, it is the time required to generate the adjacency list, and for the case of the pattern based approach, it is the time required to construct the frequent pattern tree. The algorithm time is the time required to process the input and generate sets and the output processing time is the time required to filter off any unwanted sets. Shown is the combined time for all three portions.

5.2 Data Sets

Four sets of graphs were generated to compare the approaches. Firstly, small and large graphs were created with increasing connectivity. This was done with two parameters, n and p, where n is the number of nodes in the graph, and p is the probability for any two vertices to be connected. Since the absolute maximum number of connections possible is n * (n-1) / 2, so on average, we have around p * n * (n-1) / 2 edges in the graph. For each of these graphs, the total number of maximal bicliques, and a limited (p,q) large number of maximal bicliques was generated.

The second set of graphs generated were bipartite graphs of similar sizes. Again, two parameters, n and p are used, where n is the size of one of the bipartite set and p is the probability that any two vertices $v_1 \in V1$ and $v_2 \in V2$ are connected. Since the absolute maximum number of connections possible is n^2, on average, we have around $p * n^2$ number of edges in the graph. To keep the number of vertices similar to the first set, n is halved for this set so the graphs for this set contains lesser edges on average ($1/4 * p * n^2$ or about half the number of edges of the first set).

The third set of graphs generated were for locally random graphs with two parameters, n and r. Recall that a locally random graph with n vertices is generated such for

any two vertices v_i, v_j they are adjacent with probability ½ if $i + n - j \pmod n \leq r$ or $j + n - i \leq r$. The average number of neighbours for each vertices is therefore r. Graphs were generated for values of r = 10, 30 and all maximal bicliques were generated. For this, the average number of edges is ½ * n * r.

The fourth set is similar to the third set except that now the graphs are bipartite in nature. Again, the number of vertices in both bipartite sets is halved from the first set. However, the average number of edges is around the same at ½ * n * (r+1). As the number of bicliques generated is very huge (16,530,659 in the case of n = 500, r = 30 compared to 275,027 for similar parameters in set 3), the set only consisted the cases for which r = 10.

For each of the sets, the number of maximal bicliques generated by the conditions and the timing for the three algorithms are shown.

5.3 Hardware Specifications

The three algorithms were coded in C++ and were run on a PC of 1.60 Ghz processor with 512MB memory whose OS is Unix.

6 Results

The results are organised in the four sets with their parameters explained in Chapter 5. There are 3 algorithms used, CLIQUES is denoted as Cliques, FREQUENT PATTERN as Pattern, and MAKINO's algorithm as Makino. For each test data, the total number of maximal bicliques found is listed as Num Cli and for each algorithm, the total time is shown, followed by the average time took to find 1000 maximal bicliques.

Set 1: Varying sparseness of graph, number of vertices, and minimum size of biCliques found for general graphs as input

Table 1: Results from Set 1

n	p	Min Size	Num Cli	Cliques	Time/ 1000	Pattern	Time/ 1000	Makino	Time/ 1000
100	0.1	1	627	1.08	1.72	0.03	0.05	0.33	0.53
	0.2	1	6444	9.49	1.47	0.18	0.03	3.86	0.60
		3	3150	5.78	1.84	0.13	0.04		
	0.3	1	38873	54.27	1.40	1.06	0.03	24.78	0.64
		4	7814	27.47	3.52	0.46	0.06		
	0.4	1	419337	534.52	1.27	13.78	0.03	323.06	0.77
		5	63542	275.44	4.33	5.27	0.08		
	0.5	1	3522925	4133.60	1.17	131.70	0.04	2466.04	0.70
		6	551765	1971.30	3.57	47.46	0.09		
300	0.1	1	29836	53.67	1.80	1.20	0.04	70.70	2.37
		3	6428	25.36	3.95	0.49	0.08		
	0.2	1	851905	1422.84	1.67	37.81	0.04	3201.94	3.76
		3	807077	1237.89	1.53	37.34	0.05		
		5	244	449.83	1843.58	5.01	20.52		
	0.3	1	33453689	≥ 2 hrs		1966.59	0.06		
		7	22	≥ 2 hrs		93.46	4247.9		

500	0.1	1	212067	477.06	2.25	10.79	0.05	1270.84	5.99
1000	0.05	1	241838	775.97	3.21	21.26	0.09	2824.68	11.68

Set 2: Varying sparseness of graphs, number of vertices, and minimum size of BiCliques found for bipartite graphs as input

Table 2: Results from Set 2

n	P	Min Size	Num Cli	Cliques	Time/ 1000	Pattern	Time/ 1000	Makino	Time/1 000
100	0.1	1	218	0.10	0.44	0.03	0.07	0.02	0.09
	0.2	1	777	0.26	0.34	0.07	0.07	0.08	0.10
		3	97	0.21	2.14	0.03	0.18		
	0.3	1	4185	1.23	0.29	0.03	0.00	0.51	0.12
		4	155	0.91	5.85	0.03	0.14		
	0.4	1	16380	4.55	0.28	0.61	0.04	1.86	0.11
		5	164	2.72	16.57	0.08	0.23		
	0.5	1	94107	22.38	0.24	3.98	0.04	11.01	0.12
		6	1355	13.45	9.92	0.21	0.14		
300	0.1	1	5275	2.05	0.39	0.21	0.04	2.01	0.38
		3	285	1.48	5.21	0.04	0.09		
	0.2	1	69087	48.52	0.70	4.76	0.07	40.33	0.58
		3	49823	35.93	0.72	1.18	0.02		
		5	0	15.42		0.09			
	0.3	1	936980	289.31	0.31	105.63	0.11	638.36	0.68
		7	0	110.32		0.82			
500	0.1	1	33491	13.48	0.40	4.25	0.13	29.17	0.87
1000	0.05	1	40130	34.54	0.86	11.08	0.27	68.98	1.72

Set 3: Locally sparse graphs, varying sparseness of graphs, number of vertices for general graphs as input

Table 3: Results from Set 3

n	r	Min Size	Num Cli	Cliques	Time/ 1000	Pattern	Time/ 1000	Makino	Time/1 000
100	10	1	929	2.38	2.56	0.04	0.05	0.42	0.45
300	10	1	2950	7.97	2.70	0.10	0.03	5.11	1.73
	30	1	143828	185.47	1.29	4.27	0.03	408.36	2.84
500	10	1	4543	11.33	2.49	0.16	0.03	11.11	2.45
	30	1	275027	338.90	1.23	8.22	0.03	1263.38	4.59
1000	10	1	8929	59.78	6.70	0.27	0.03	44.42	4.97
	30	1	540128	2200.42	4.07	16.08	0.03	5162.93	9.56
2000	10	1	18842	143.80	7.63	0.81	0.04	468.48	24.86
3000	10	1	27376	496.00	18.12	1.52	0.06	1030.41	37.64
5000	10	1	46113	1030.11	22.34	3.67	0.08	1549.01	33.59

Set 4: Locally sparse graphs, varying number of vertices for bipartite graphs as input

Table 4: Results from Set 4

n	R	Min Size	Num Cli	Cliques	Time/ 1000	Pattern	Time/ 1000	Makino	Time/ 1000
100	10	1	5122	2.26	0.44	0.15	0.03	2.89	0.56
300	10	1	11347	10.85	0.96	0.28	0.02	20.91	1.84
500	10	1	24181	22.86	0.95	0.89	0.04	76.99	3.18
1000	10	1	43284	83.99	1.94	1.37	0.03	268.55	6.20
2000	10	1	93377	365.75	3.92	2.59	0.03	1188.75	12.73
3000	10	1	139625	1390.30	9.96	3.60	0.03	2035.19	14.58
5000	10	1	222643	2428.71	10.91	6.32	0.03	3712.31	16.67

7 Summary

7.1 Analysis of Results obtained

The algorithm chosen based on graphs are very simple and consequently easy to implement. However, they scale poorly in timing when compared to the algorithm based on FREQUENT PATTERN. For example, the rise in timing per 1000 bicliques found for PATTERN does not increase as much with either the increase in number of vertices or for the density of the graph.

It is clear that the time taken to find a maximal biclique is shortened by increasing the (p,q) large filter condition, however, for the FREQUENT PATTERN algorithm, it is shortened quite proportionally to the number of maximal bicliques found. On the other hand, CLIQUES only shows a slight improvement when we tighten the filter criteria. This is because the elimination condition for the filter takes place only near the end of its subtree usually and thus, prunes lesser of the tree whereas the FREQUENT PATTERN algorithm searches for larger patterns and thus have less choices for these patterns, pruning the tree more from the very start.

For the case where the total number of vertices in a bipartite graph is compared to a normal graph (Set 2 vs Set 1), the graph based approach gives some fairly competent timings. As the density of the graph increases, we find that the CLIQUES algorithm has a linear factor in timing when compared to FREQUENT PATTERN. However, the algorithm suggested by Makino starts off faster than CLIQUES but seems to fare poorly when the vertex sets becomes much larger due to the need to construct maximal bicliques before calling the recursive method.

Comparing local sparseness with normal sparse graph, we find that both FREQUENT PATTERN and CLIQUES do better with increasing sparseness (lower r values) compared with the original graph. There are less cliques or pattern to search for, and thus give better results.

7.2 Conclusion

In conclusion, the paper has explained several existing fast algorithms for solving the maximal biclique subgraph problem and has implemented and tested them with randomly generated graph sets. Graph based approach based on finding nodes and using them to compute maximal bicliques in a tree does not scale well with increasing nodes compared to finding frequent patterns and matching them when searching for relatively sparse graph, as most of the graphs presented here are. This could also be due to the idea that graph based work was done mainly on enumerating maximal cliques from a graph and extending it to bicliques would add a lot of redundant edges that would lessen the efficiency of the algorithm.

7.3 Future work

There are other algorithms both in the area of data mining and graph based. For example, Linear time Closed Itemset Miner (Takiake Uno et al) which were not implemented in this study. Also, there already exist algorithms for parallel computation to search for maximal bicliques. It might be useful to evaluate the performance of such parallel algorithms as well as the portability of existing algorithms for parallel computer.

References

David Eppstein, “Arboricity and Bipartite Subgraph Listing Algorithms”, Department of Information and Computer Science, University of California, Irvine, CA 92717, 2004

Etsuji Tomita, Akira Tanaka, Haruhisa Takahashi “The worst-case time complexity for generating all maximal cliques and computational experiments”, *Theoretical Computer Science archive Volume 363, Issue 1 (October 2006),* 28-42, 2006

Gösta Grahne and Jianfei Zhu “Efficiently Using Prefix-trees in Mining Frequent Itemsets”, Concordia University, Montreal, Canada, 2003

Jinyan Li, Guimei Liu, Haiquan Li, Limsoon Wong “Maximal Biclique Subgraphs and Closed Pattern Pairs of the Adjacency Matrix: A 1-to-1 Correspondence and Mining Algorithms”***,*** *IEEE Transactions on Knowledge and Data Engineering, 19(2):*1625--1637, December 2007

Jinyan Li, Haiquan Li, Donny Soh, Limsoon Wong **“**A Correspondence Between Maximal Complete Bipartite Subgraphs and Closed Patterns**”,** *Proceedings of 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 146--156, Porto, Portugal, October 2005

Kazuhisa Makino and Takeaki Uno “New Algorithms for Enumerating All Maximal Cliques”, *Algorithm Theory - SWAT 2004*, 260-272, 2004

Kumlander, Deniss “Some practical algorithms to solve the maximum clique problem”, Tallinn University of Technology, Faculty of Information Technology, Department of Informatics , Estonia, 2005

Panos M. Pardalos, Jonas Rappe, Mauricio G.C Resende, “An exact parallel algorithm for the maximal clique problem”, *High Performance Algorithms and Software in Nonlinear Optimization*, pp. 279-300, 1999

Sune Lehmann, Martin Schwartz, Hansen Lars Kai, “Biclique Communities”, arxiv.org/abs/0710.4867, Center for Complex Network Research and Department of Physics, Northeastern University, Boston, 2007

Takiake Uno, Masashi Kiyomi, Hiroaki Arimura, “Efficient mining algorithms for frequent/closed/maximal itemsets”, *IEEE ICDM’04 Workshop FIMI’04 (International Conference on Data Mining, Frequent Itemset Mining Implementation),* 2004

Vicky Choi, “Faster Algorithms for Constructing a Galois Lattice, Enumerating All Maximal Bipartite Cliques and Closed Frequent Sets”, arXiv:cs/0602069v1 , Department of Computer Science, Virginia Tech, USA, 2006

Y. Cheng and G.M. Church, “Biclustering of expression data”, *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology (ISMB),* pages 93-103, 2000.

Appendix A1: Comparison of Timings for density

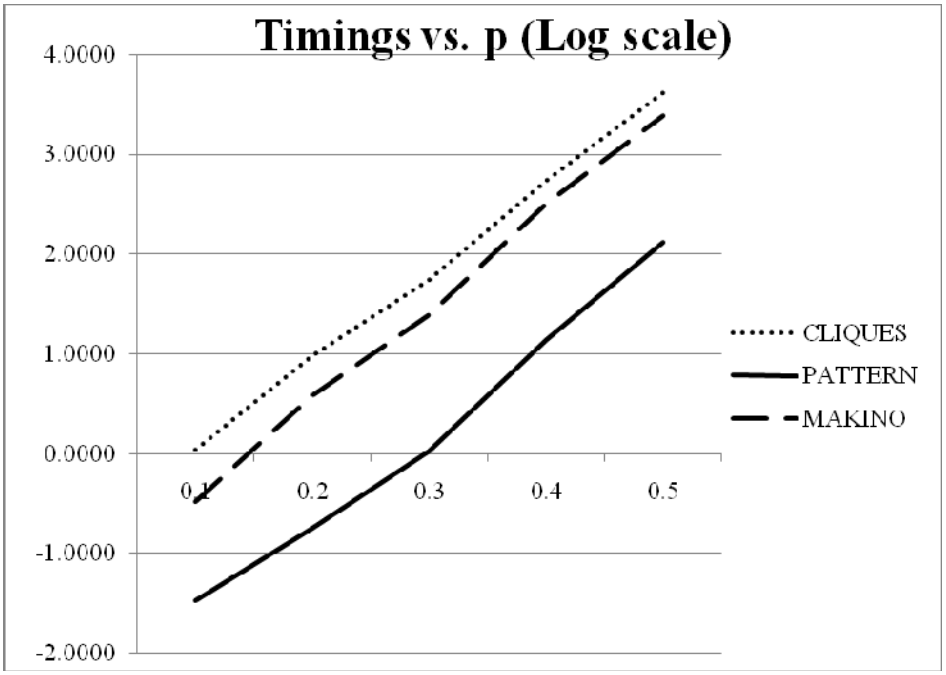


Figure 5a: Time taken for different algorithms vs density of graph for a general graph

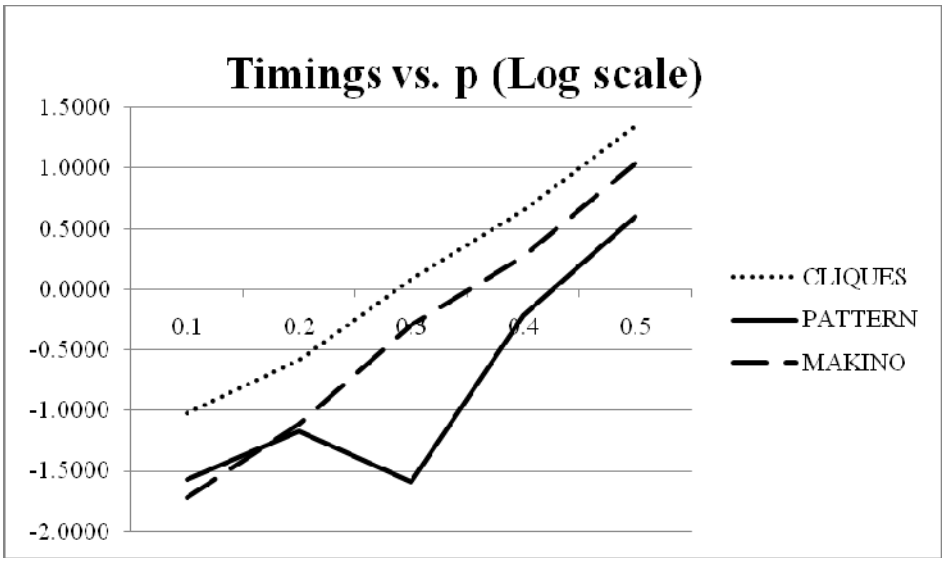


Figure 5b: Time taken for different algorithms vs density of graph for a bipartite graph

The above graphs (from set 1 and 2) compares the three algorithms based on running time for a fixed n of 100 and varying the probability p, the density of the graph. There is a

clear pattern that the time taken by all three algorithms increases at an exponential rate with respect to the density of the graph. Also for small n, PATTERN runs the fastest, followed by Makino's and finally CLIQUES.

Appendix A2: Comparison of Timings for number of vertices

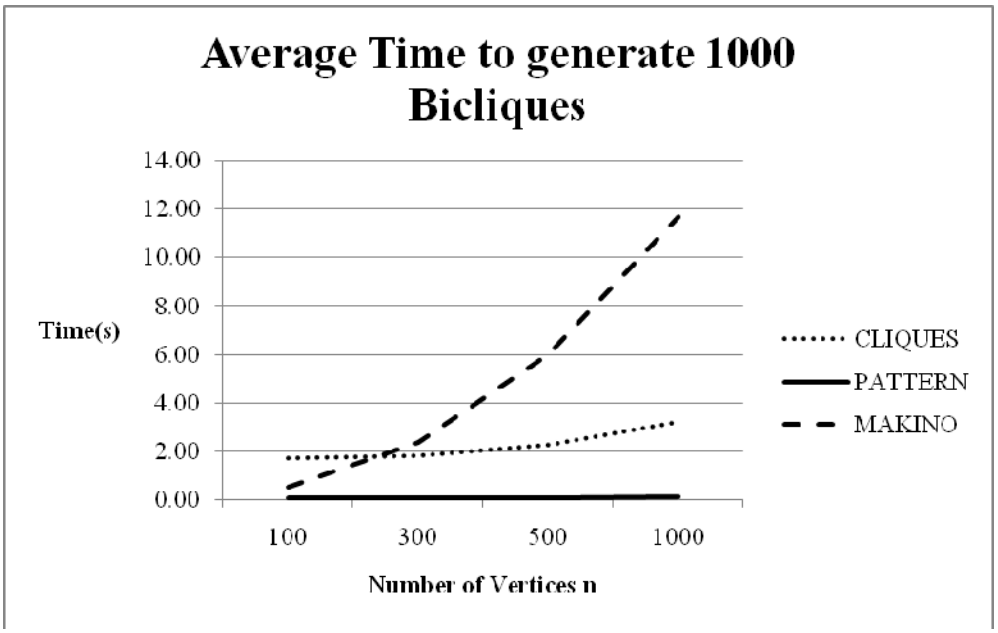


Figure 6a: Time taken for different algorithms vs number of vertices for a general graph

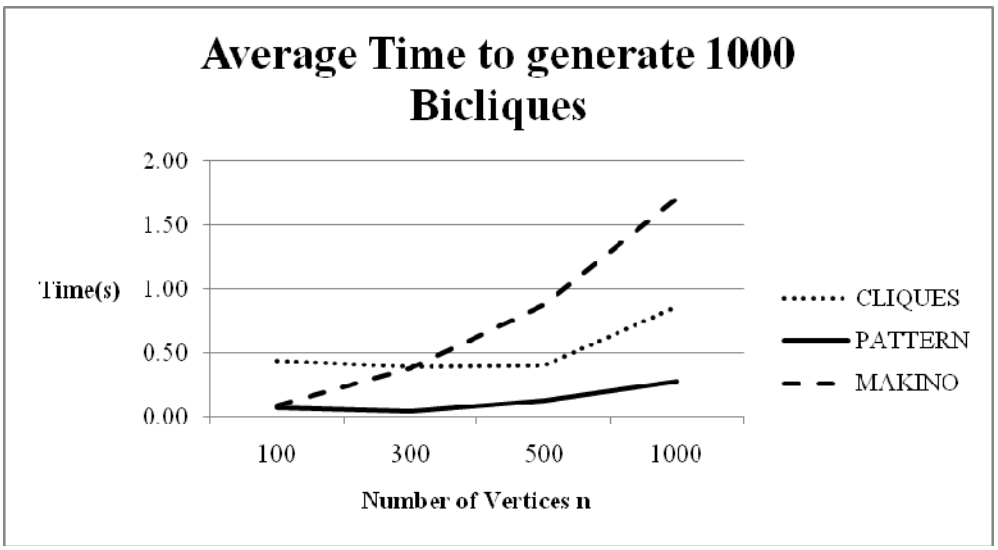


Figure 6b: Time taken for different algorithms vs number of vertices for a bipartite graph

The figures above compare how the time taken to generate maximal bicliques varies based on the number of vertices n, taken from Set 1 and 2. Although Makino's algorithm generates maximal bicliques very fast for small n, the average time increases as we increase the number of vertices n compared to CLIQUES and PATTERN, whereas for the other two, the increase in time per thousand bicliques is not very significant.

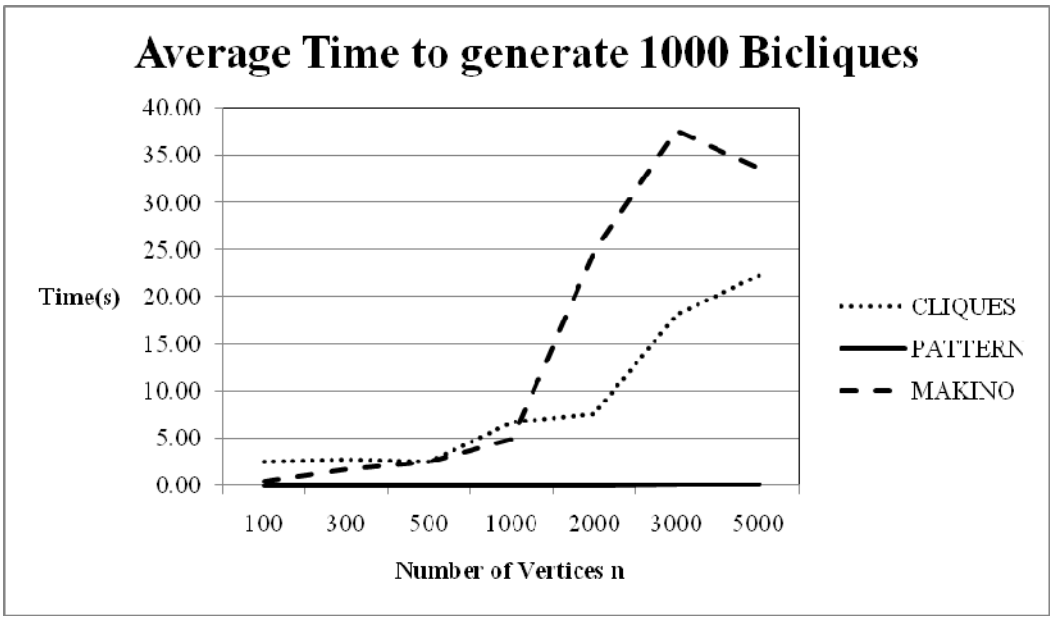


Figure 7a: Time taken for different algorithms vs number of vertices for a sparse general graph.

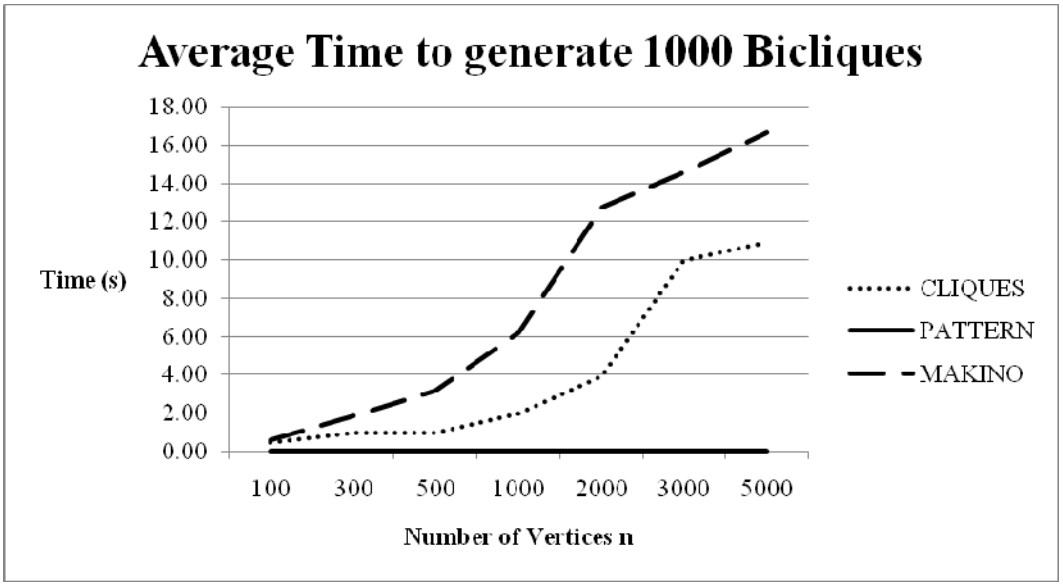


Figure 7b: Time taken for different algorithms vs number of vertices for a sparse bipartite graph.

The graphs above, taken from Set 3 and 4 compare how the time taken to generate maximal bicliques vary based on the number of vertices n, with a fixed r of 10. Makino algorithm does indeed seem to have a poor performance when we increase the size of the number of vertices.

Mining Algorithms for bicliques

Chen Xiankun, Department of Computer Engineering,
School of Computing, National University of Singapore

Abstract: There exist several existing algorithm for computing maximal bicliques (or complete bipartite) subgraphs or an undirected graph with origins in graph theory or in data mining. In this project, three algorithms, two depth first search graph algorithm, one from Tomita and one from Makino both using tree construction and pruning, and a closed-pattern-based method using frequent pattern matching are explained and implemented and their efficiency compared based on different random graphs generated.

Introduction

A maximal biclique (or complete bipartite) subgraph is an induced subgraph K = {V_1 U V_2, E} of a general graph G such that any vertex in V_1 is adjacent to all vertices in V_2, and maximal in the sense that we cannot find another complete bipartite subgraph K_1 of G such that K is contained in K_1.

Maximal bicliques are useful to model many real-life applications, such as social affiliation networks, and metabolic network from the biological field, with the edges representing enzymes-reactions relations.

There exists several fast algorithms, with origins in graph theory or in data mining. The aim of the project is to implement some of these algorithms, model different conditions for testing, and evaluate their performance (running time) under the different conditions.

Tomita's CLIQUES algorithm (Graph)

A global variable Q representing vertices of the graph is used to store the biclique found until this point while a temporary local variable SUBG were used to store the remaining possible candidates vertices to add for maximal bicliques due to the biclique Q at this time. Another temporary variable CAND holds the set of unprocessed candidates vertices of SUBG while FINI holds the set of processed candidates.

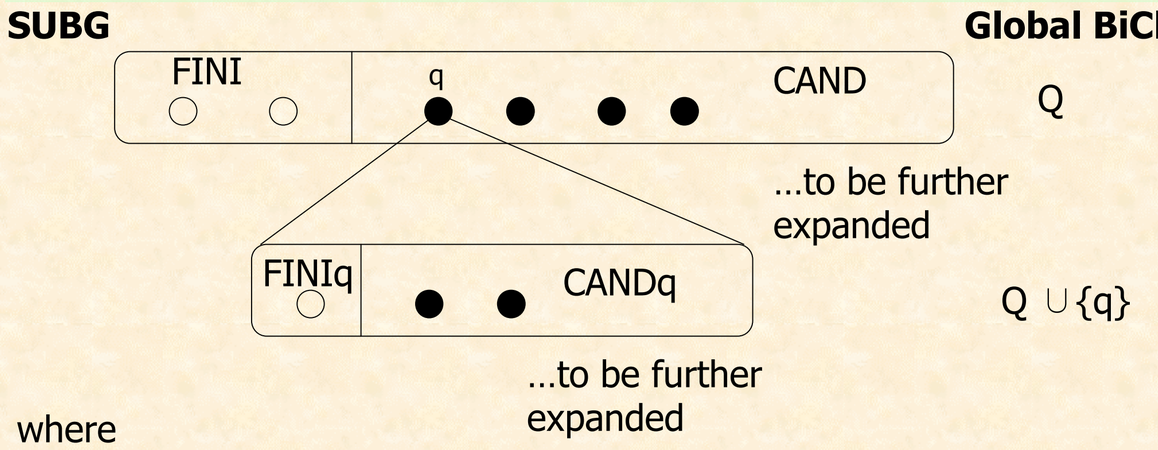


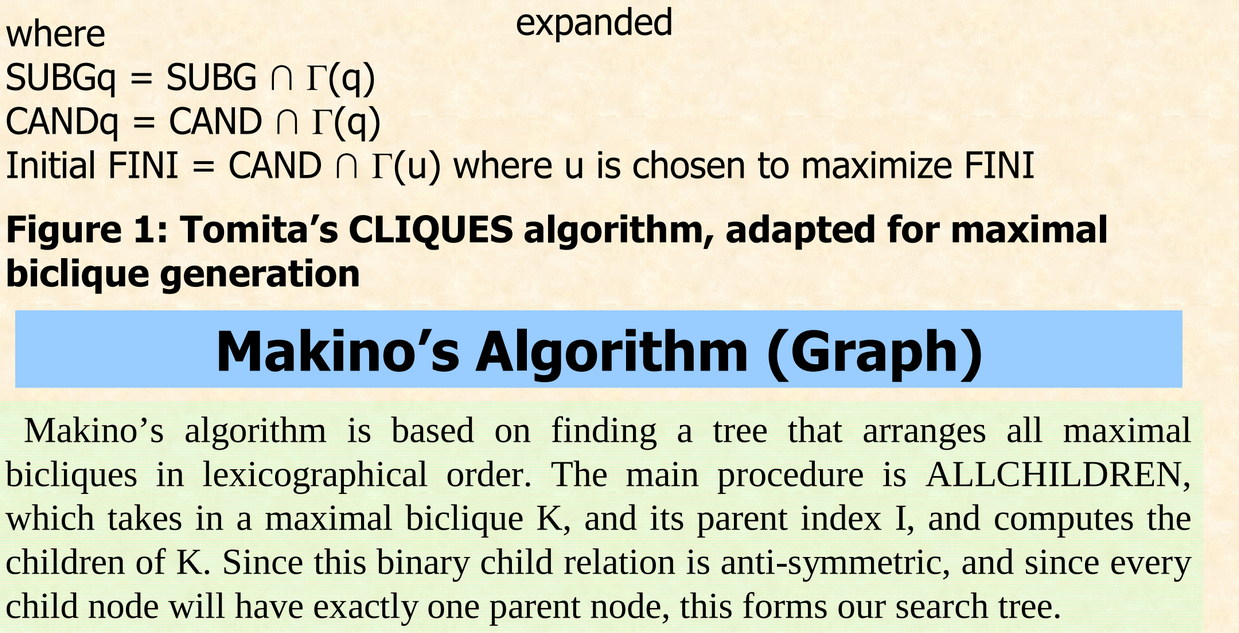
Figure 1: Tomita's CLIQUES algorithm, adapted for maximal biclique generation

Makino's Algorithm (Graph)

Makino's algorithm is based on finding a tree that arranges all maximal bicliques in lexicographical order. The main procedure is ALLCHILDREN, which takes in a maximal biclique K, and its parent index I, and computes the children of K. Since this binary child relation is anti-symmetric, and since every child node will have exactly one parent node, this forms our search tree.

Closed Pattern Based Approach

The approach views the adjacency matrix as a transactional database. Each transaction (neighbours) is a subset of a pre-specified set I (vertex set of graph). Given a pattern P (non-empty set of items of I), the occurrence set of P (transactions containing P), the closure of P (the set of items shared by all the occurrence set of P), will form the two vertex sets of a maximal biclique. The problem thus reduces to harvesting such patterns P from a frequent pattern tree.



Results

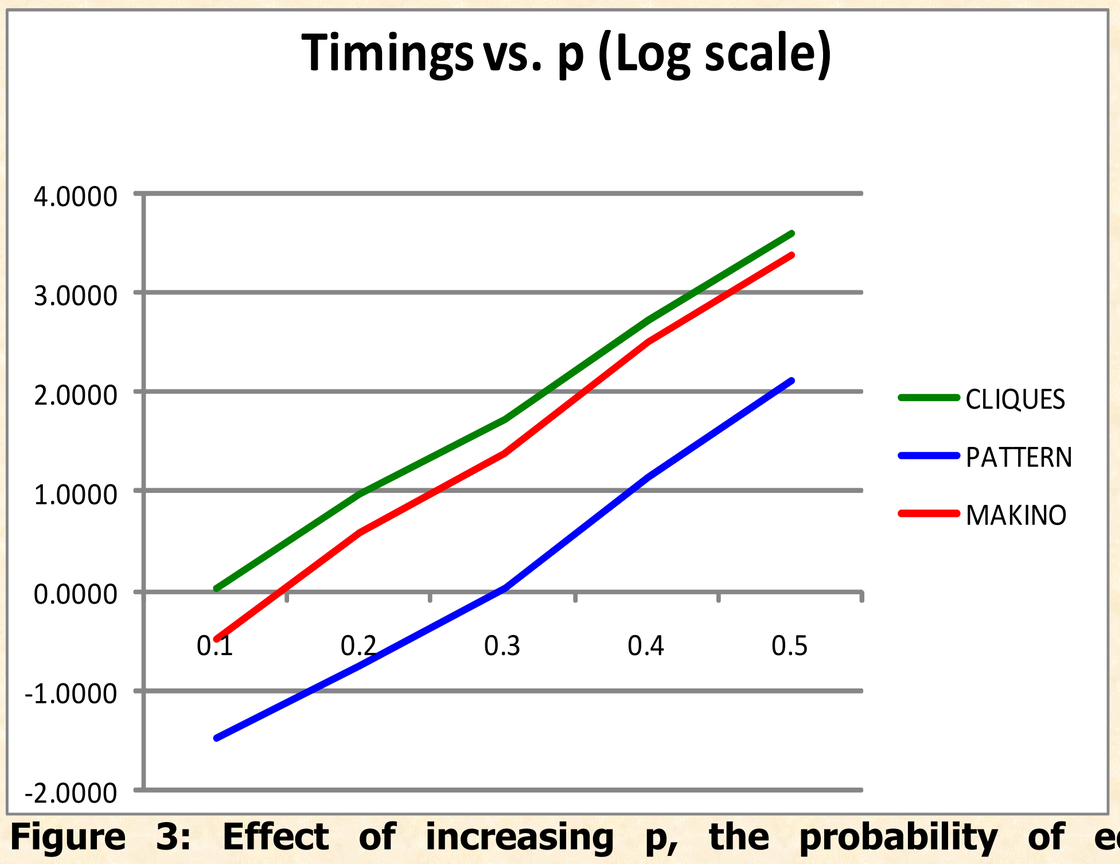


Figure 3: Effect of increasing p, the probability of edge between any two vertices on execution time for the 3 algorithms

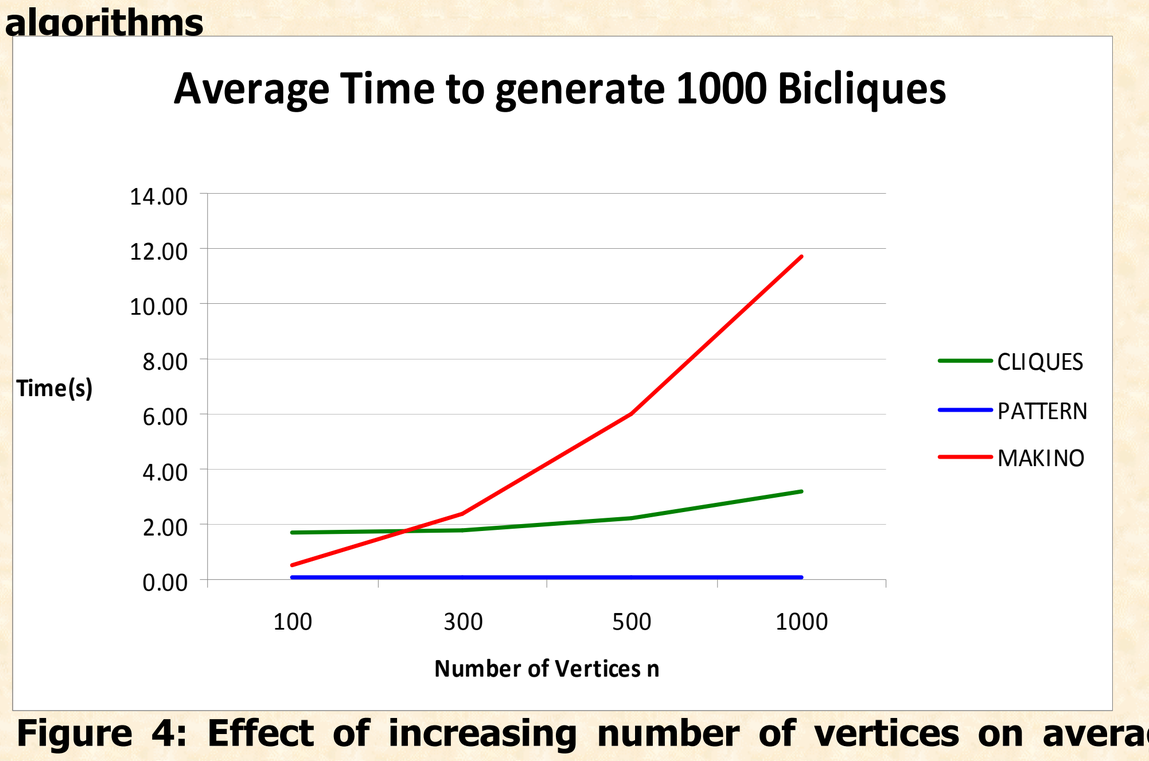


Figure 4: Effect of increasing number of vertices on average time to find 1000 bicliques for the 3 algorithms

Also tested was adapting the algorithm for a minimum size (number of vertices on both sides of the biclique) for each biclique generated. Makino could not be properly adapted to fit this, while pattern algorithm fared better than cliques.

Other observations noted include that locally sparse graphs tend to be faster to evaluate for all algorithms as compared to random sparse graphs, as well as input graphs which are bipartite as compared to a general graph.

Conclusion and Future Work

Graph based approach based on finding nodes and using them to compute maximal bicliques in a tree does not scale well with increasing nodes compared to finding frequent patterns and matching them when searching for relatively sparse graph. This could be due to graph based work was done mainly on enumerating maximal cliques from a graph and extending it to bicliques (by assuming all vertices in one bipartite set are interconnected) would add a lot of redundant edges that would lessen the efficiency of the algorithm.

There are other algorithms both in the area of data mining and graph based, for example, Linear time Closed Itemset Miner which were not considered in this study. Also, there already exist algorithms for parallel computation to search for maximal bicliques. It might be useful to evaluate the performance of such parallel algorithms as well as the portability of existing algorithms for parallel computation.