Maximal Clique Enumeration: Faster Solutions via Small Dominating Structures

by

Hamed Barghout

B.S., Computer Science, Lebanese American University, 2003

Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Division of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

LEBANESE AMERICAN UNIVERSITY

School of Arts and Sciences - Beirut Campus

Thesis approval Form (Annex III)

Student Name: Hamed Barghout I.D. #: 200101037

Thesis Title : Maximal Clique Enumeration: Faster Solutions via Small Dominating Structures

Program : MS

Division/Dept : Computer Science and Mathematics

School : Arts and Science

Approved by :

[redacted]

Faisal N. Abu-Khzam, Ph.D. (Advisor)
Associate Professor of Computer Science

[redacted]

Nashat Mansour, Ph.D.
Associate Professor of Computer Science

[redacted]

Haidar Harmanani, Ph.D.
Associate Professor of Computer Science

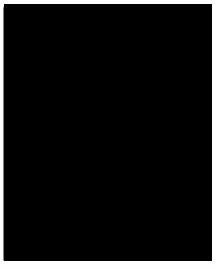
Date : 22 February 2007

Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Hamed Boraghont

Signature: 

Date: 23/02/2007

I grant to the LEBANESE AMERCIAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.

Abstract

Maximal clique enumeration is one of the most important tools for solving problems in a wide variety of application domains. Classical enumeration techniques suffer from being too slow on large scale data. New techniques, introduced in the work of Abu-Khzam, decompose the input graph into a smaller subgraph and its complement in an attempt to reduce the search space and confine the exponential-time enumeration to subgraphs that are often small in practice. Such subgraphs are called dominating structures in our work. We discuss different dominating structures for the maximal clique enumeration problem, but we focus in our implementations on two of them. The first structure is a vertex cover of the input graph and the second one is a maximal matching of the complement of the graph. We consider these two edge capturing structures on graphs of different densities. Our experimental study revealed that the vertex cover approach is faster than the well-known Bron and Kerbosch (BK) algorithm on sparse graphs, as well as graphs whose vertex covers are small. Theoretical analysis supports our findings since the run time is improved from $O(3^{n/3})$ to $O(3^{k/3})$, where k is the vertex cover size. Moreover, our second approach proved to be faster than BK on graphs of high density.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Definition of the problem

Our focus in this thesis is on maximal clique enumeration in undirected simple graphs. For a graph G = (V, E), where V is the set of n vertices and E is the set of edges, a clique can be described as a complete subgraph of G, in the sense that all vertices are pair-wise adjacent. A maximal clique is one that cannot be contained as a subgraph in any larger clique, while a maximum clique is a clique of maximum size. There are two corresponding clique detection problems; the maximum clique problem that asks for a clique of largest size in the graph, and the maximal clique enumeration problem that requires listing all maximal cliques in the graph. The maximal/mum clique problem is equivalent to the maximal/mum independent set problem, because a maximum clique in G is a maximum independent set in the complement graph of G.

1.2 Applications

Maximal clique enumeration is used to find correlations among items in a dataset, where the problem is represented as a graph of items (vertices), and when any two items satisfy certain relationship, an edge is placed between these two items. This enumeration of objects that satisfy certain specifications is taking an increasing significance in many scientific fields such as Biochemistry and Genomics (Butenko and Wilhelm 2004), Artificial Intelligence (Eiter and Makino 2002), data mining (Agrawal and Srikant 1994) and clustering (Stix 2001).

A well known sample application is Motif Finding, a Biological application that could employ maximal cliques' enumeration in order to identify motifs (or Transcription factor binding sites). These are short stretches of DNA sequences that are located in the

promoter region of a gene. The input of an (L-d) motif problem consists of t sequences; each sequence is a set of N nucleotides. The goal is to find a subsequence of length L with at most d mismatches (mutations) to subsequences of length L in each of the t input sequences.

One approach to solve this problem is called the Winnower method, which transforms the problem into a instance of Maximum Clique. The Winnower algorithm divides each sequence into N-L+1 possible L-letter substrings (signals), and then creates a t-partite graph where each part's vertices are all L-letter substrings from one sequence. Since any two instances of the mutated motif can differ in as many as 2d positions, edges of the graph connect vertices with a maximum of 2d differences. In other words Winnower constructs a graph with vertices corresponding to substrings from the sequences and puts an edge between potentially similar substrings. Now a clique in the constructed graph corresponds to a potential solution, since some maximal cliques do not correspond to a correct solution, the best way may be to enumerate them all.

Volker Stix (2001) gave another example of a gray scale of 10 levels of gray and that the human eye can distinguish between consecutive shades only if the distance between these 2 consecutive shades is more than 1. So shade number 4 belongs to the cluster of dark, grays whereas shade number 6 belongs to the cluster of light shades. Shade number 5 belongs to both clusters at the same time because it can not be distinguished from shade number 4 or shade number 6. Suppose that we have n objects given in a measure space S and a metric d(i,j), which measures the distance between object i and j in S . To find all non-distinguishable objects in the given space S, Stix uses maximal clique enumeration. He represents the objects by graph nodes, where edges are placed between vertices that cannot be distinguished, d(i,j) <= 1 then i-j is an edge in the graph. And any maximal clique will represent non-distinguishable objects in the given space.

1.3 Previous Work

The clique enumeration problem is known to be NP-hard (Gary and Johnson (1979)), Moon and Moser (1965) proved that the number of maximal cliques of a graph could be exponential with respect to the number of vertices, in fact a graph can have up to $3^{n/3}$ maximal cliques.

The most notable clique enumeration algorithm is the BK algorithm (Bron and Kerbosch 1973). BK offered a depth-first search algorithm for generating all maximal cliques in an undirected graph.

There are three sets in this algorithm that play an important role:

- COMPSUB is the set of points forming the current growing clique, (all points in COMPSUB are connected)
- CANDIDATES is the set of all points that can be added to the current COMPSUB set.
- The set NOT of points that have already been added to COMPSUB.

The search tree internal nodes correspond to all non-maximal cliques and the leaves represent (or hold) maximal cliques. At each node of the search tree, BK selects a vertex v from CANDIDATES and adds it to COMPSUB, this will need the creation of a NEW_CANDIDATES and NEW_NOT, where NEW_CANSISATES is the intersection of CANDIDATES and the neighbors of v, N(v). The same applies for NEW_NOT = NOT $\cap$ N(v). This last step is very important in avoiding the generation of non-maximal cliques.
In particular, if a search-tree node, say X, has a couple of children such that the first resulted from adding vertex u to COMPSUB, then neighbors of u that are examined in the second child of X must not consider the vertex u again, so u is kept in the NOT set of descendants of X as long as neighbors of u are added to COMPSUB. A maximal clique is declared in a leaf node only if both sets NOT and CANDIDATES are empty.

A simple description of the BK Algorithm can be the following (Initially CANDIDATES contains all vertices, NOT is empty, and COMPSUB is empty)

```
Expand_BK (COMPSUB, NOT, CANDIDATES)
{
If CANDIDATES is empty and NOT is empty then
        Generate the maximal clique COMPSUB
        Return
Else
    For each vertex u in CANDIDATES do the following
        Add u to COMPSUB
        New_CANDIDATES = CANDIDATES ∩ N(u)
        New_NOT = NOT ∩ N(u)
        Expand_BK (COMPSUB, New_NOT,New_CANDIDATES)
        Remove u from CANDIDATES'
        Add u to NOT
}
```

More recently, Tomita et al. (2004) proved that the worst case scenario for the BK algorithm is $O(3^{n/3})$, thus BK achieves the best possible worst-case performance as a function of n, the number of vertices in G.

An improved version of the BK algorithm (Improved-BK) was also published in the same paper as the Base BK (Bron and Kerbosch 1973), it has been proven to be faster than Base BK with dense graphs while the Base BK proved to be twice as fast as the improved BK on sparse graphs, and slower when the edge density increases. Improved BK has one main difference from the Base BK, this difference lies in the choice of the selected vertex v at each node of the search tree. If the previously selected candidate is in array position p then the next selection will be for candidate in place p+1, this is how the Base BK works. The improved BK does not select the vertex in position p+1 but instead it chooses a vertex with the largest number of connections to the other vertices in CANDIDATES. This modification is useful only if the time spent finding the maximum degree vertex is less than the time spent on exploring the eliminated branches of the search tree.

Tsukiyama et al. (1977) came with an algorithm to generate all maximal independent sets in a graph G, and many researchers presented new algorithms to enumerate maximal cliques based on Tsukiyama's algorithm. Makino and Uno (2004) presented new algorithms that are based on Tsukiyama's algorithm, one of these algorithm enumerates all maximal cliques of G in $O(\mu\Delta^4)$ where Δ is the maximal degree of G and μ is the number of independent sets in $\check{G}$, but Tomita et al. (2004) showed that, in practice, the BK is much faster on sparse graphs than those based on the Tsukiyama et al method.

Kose et al. (2001) algorithm took a different approach, it starts from the fact that any clique of size p where $p \geq 2$, is composed of 2 cliques of size p-1 that have p-2 vertices in common. This algorithm requires an enormous amount of memory. Abu Khzam et al. (2005) proved that the BK is much faster on sparse graphs derived from microarray data, and after the implementation of a multi-threaded version of Kose's algorithm with load balancing; Zhang et al. (2005) achieved a speed up of 383 on sparse graphs, but even with this speed up the improved BK remains faster.

1.4 The Dominating Structures Methodology

In order to achieve faster enumeration of maximal cliques, we adopt the dominating structure approach of Dr. Abu-Khzam. We take a closer look at the structure of input graphs. We consider a special type of subgraphs to which we can restrict the exponential part of the enumeration process.

Such subgraphs, called here dominating structures, are problem-dependent in general since different dominating structures can be associated to different problems. A dominating structure is a subgraph that captures all non-trivial maximal cliques, in the sense that any clique of size three or more has all but very few vertices in the said subgraph. This idea was first mentioned by Abu Khzam et al. (2005).

Examples of such MCE dominating structures include:

- Maximal matching, since the complement of (the vertex set of a maximal matching induces an edgeless subgraph.
- Vertex cover, or set of vertices whose complement induces and edge-less subgraph. Thus, a maximal matching is a special vertex cover.
- Feedback vertex set, which is any set of vertices whose complement induces an acyclic subgraph (tree of forest). Vertex covers are special feedback vertex sets.
- Triangle vertex cover (AKA. triangle vertex deletion set), which is a set of vertices whose complement induces a triangle-free subgraph. A feedback vertex set is a special triangle cover.

We will discuss the first two examples in the second chapter. In the third chapter, we apply the Maximal Matching dominating structure on the complement of the graph instead of working on the initial graph, We present the experimental results and analysis in chapter four. The last chapter is devoted to concluding remarks and a description of the ongoing research in the MCE project.

Chapter 2

The Vertex Cover Dominating Structure

Recall that a vertex cover is a set of vertices that induce a subgraph whose complement is edge-less. In other words, if G = (V,E) is a given undirected simple graph and S is a vertex cover of G, then any edge of G has at least one of its endpoints in S. It follows that any clique of G has at most one endpoint in the set I = V − S. This property is enough to treat S as a dominating structure for MCE. We can take every element of I and enumerate all maximal cliques in the subgraph induced by its neighborhood in S, then we enumerate all maximal cliques of S that do not contain any element of I. This way, we reduce the exponential factor in the enumeration time of MCE to a function that is exponential in the size of a vertex cover only. This leads to the following.

Lemma 1: Let G be a simple undirected graph on n vertices and let k be the size of a vertex cover of G. Then the number of maximal cliques in G is bounded above by $(n - k + 1)3^{k/3}$.

A simple proof of this lemma uses the fact that BK order is $3^{n/3}$ and when we apply BK on a graph of size k (size of the dominating structure) then its order is $n^{k/3}$. And in the proposed algorithm we are running BK for each element in I, |I| = n-k, and one additional time without any element from I. So in total we are running BK on a graph of size at most k+1 for (n-k+1) times.

Lemma 1 poses a couple of important questions:

- What properties of the input graph determine (or bounds) the number of its maximal cliques?
- How could we design a structure-aware enumeration that makes the best use of such properties?

Since the Maximal Matching is a special form of Vertex Cover and all what apply on Vertex Cover applies also on maximal matching, we will refer in this section the Vertex Cover dominating structure but it can be replaced by a Maximal Matching in the same graph. When it is better to choose a Maximal Matching over a Vertex Cover or the reverse will be discussed more in the experimental section of this paper.

2.2 Algorithm Pros and Cons

If all the work performed by the BK algorithm is spent on delivering maximal cliques, then all we are saying here is that BK is faster on such graphs. However, we are more concerned about fast implementations. So we proceed to the investigation of the benefits of using a structure-aware implementation. We observed that, while the idea of restricting the enumeration to a subgraph induced by a vertex cover is simple, there are diverse types of implementations of such technique.

2.2.1 Method 1

If we follow the discussion preceding Lemma 1, we iteratively consider each element, say v, of I and enumerate all maximal cliques in the neighborhood of v. Then we have to enumerate the maximal cliques of G_S, the subgraph induced by a vertex cover S. None of these cliques is contained in the neighborhood of any vertex of I. This latter task is hard to perform without duplicating the bulk of the effort spent in the first pass through the

elements of I. Yet, this algorithm was faster than BK on dense graphs. A simple description of the algorithm is the following:

We assume the set S is given together with the graph G represented by an adjacency matrix. We start by creating a new adjacency matrix for Gs of size $(c+1) \times (c+1)$ where c is the size of the cover. Initially Gs contains all the vertices of S, and we leave the last row and column empty to be filled each before each iteration with a different vertex from I.

We use the same sets, as in the BK algorithm. COMPSUB, NOT and CANDIDATES will consist of vertices of graph Gs, |S|, plus 1, initially NOT COMPSUB and CANDIDATES are empty.

For every vertex u not in VC

    Put u in COMPSUB

    Adjust CANDIDATES to contain only neighbors of u

    Add u to Gs

    Run the BK-algorithm on Gs with the updated CANDIDATES set

    Remove u from COMPSUB

Run a special-BK on Gs without adding any vertex from I, and initialize CANDIDATES to contain all vertices in the S.

The special-BK is different from the known Base-BK algorithm in one thing: it has one additional test before declaring that a maximal clique is found in the leaf nodes It checks if the current clique in COMPSUB can be extended by the addition of any vertex from outside the cover, if this is possible, then this solution is not maximal because it has already been tested in another instance of the algorithm and we ignore such clique.

We noticed that using this implementation we have already some sort of independence between tasks. Each task generated from a vertex in I plus its neighbors in Gs and a task that contains Gs only. So we implemented a simple parallel version of this algorithm to see its performance on a small cluster. The drawback was when Gs contains most of the edges of G, then running BK on Gs for (n-k+1) times is a multiplication of the same effort and resulted in bad results on certain graphs.

2.2.2 Method 2

Because of duplication of efforts in the above method, we proposed another implementation that proved to be much faster during our experimental study.

The main idea is to enumerate all maximal cliques of Gs, and then expand each maximal clique by adding a vertex from I. But this is not enough since some non-maximal cliques of Gs can be expanded through I. If we try to keep track of all non-maximal cliques that are neighbors of at least one vertex of I, then enormous book-keeping would be needed. A better approach, and the one we use here, is to follow the same BK search, restricted to Gs. In what follows, N(v) denotes the neighborhood of vertex v in G. After reducing the graph representation to Gs, the algorithm proceeds by applying the same BK search. We use the same COMPSUB and CANDIDATES arrays that consist of vertices of Gs only. Also we use an array Outsider_Degree to keep track of the degrees of elements of I, which we call the outsiders.

At each node of the search tree, the following steps are performed (as part of the function Expand_VC, which is the main routine of our algorithm):

```
Expand_VC (COMPSUB, NOT, CANDIDATES)
{
If CANDIDATES is empty then
        For each outsider vertex t do
                If (Outside Degree(t) = |COMPSUB|) And (t has no neighbors in NOT) then
                        Generate the maximal clique ({t} U COMPSUB)
                        Flag P=1 (initially Flag P is zero)
        If P = 0 And NOT is empty then
                Generate the maximal clique COMPSUB
        Return
Else
   For each vertex u in CANDIDATES
      For each outsider vertex t do
        If t is not in (N(u) U Outside_NOT) And degree (t) = |COMPSUB| And N(t) ∩ (NOT U
        CANDIDATES) is empty then
                Generate the maximal clique formed by ({t} U COMPSUB)
                Add t to Outside_NOT
        Else if t is in N(u) then
                Increment degree of t
   – Add u to COMPSUB
   – New_CANDIDATES = CANDIDATES ∩ N(u)
   – New_NOT = NOT ∩ N(u)
   – Expand_VC(COMPSUB, New_NOT and New_CANDIDATES)
   – Remove u from COMPSUB
   Update OutsideDegree (by decrementing the degrees of neighbors of u)
   –    Add u to NOT
}
```

The correctness of our algorithm follows easily from the discussion that preceded the pseudo-code of Expand_VC. As for the run time, note that we used BK on the subgraph induced by S, so the number of nodes of our search tree is in $O(3^{k/3})$, where k is the cardinality of S. Moreover, at each node of the search tree we spend a linear time of (n-k) to update the degree of outsiders, one time when adding u from CANDIDATES to COMPSUB and one time after removing u from COMPSUB, and when we find an outsider that form a maximal clique with the current growing COMPSUB, we check if $N(t) \cap (NOT \cup CANDIDATES)$ is empty what takes a complexity of k, the upper bound of this is when all outsiders are forming maximal cliques with the current COMPSUB. So the total run time is in $O(((n-k)k+(n-k)) (3^{k/3}))$.

Chapter 3

Enumerating maximal Independent Sets using the Maximal Matching Structure

A maximal matching of a graph forms a dominating structure that divides the graph into two subgraphs, a set of independent vertices and a set of matching edges. A set M of edges in a graph G is a matching if no two edges in M have an endpoint in common, and this set M is maximal if it can not be increased by any additional edge. This gives us a set M containing only the maximal matching edges and a set I containing the remaining independent vertices of the graph. There could be additional edges in the graph that connect vertices in I to vertices in M, but no edge will exist connecting two vertices in I, otherwise we will be able to mark this edge as a matching edge and then M will not be maximal. Also there may be additional edges connecting vertices in M, which can be as dense as a complete subgraph.

Although we could use the maximal matching of G as a dominating structure and enumerate all maximal cliques in G directly, we choose here to go in a different way, we will pick a maximal matching for the complement of G, $\check{G}$. Note that maximal clique enumeration problem can be transformed into enumeration of maximal independent sets in the complement graph since any maximal independent set in $\check{G}$ will be a maximal clique in G. The proposed algorithm will limit the enumeration to the set M, and then adds all the remaining vertices from the set I.

3.2 Algorithm Pros and Cons

The proposed algorithm will work on the set M. For each edge (u,v) in M, at most one of the vertices u and v parties a member of an independent set, simply because they are connected. So for any independent set I, and for each edge (u,v), we have three possibilities: either u is in I or v is in I or both u and v are not in I. We try all corresponding combinations (of the three possibilities) among all edges in M. These combinations will be used to generate independent sets that are maximal in G_M, the subgraph induced by edges of M, but may be extended by adding vertices from I. After generating all possible maximal independent sets in G_M we add all remaining vertices in I and we declare a maximal independent set in Ğ. The order of this algorithm is clearly 3^k where *k* is the number of edges in the maximal matching set M. We now describe this algorithm in details.

Initially, we suppose that we have the complement of the initial graph in an adjacency matrix, we call it Ğ, and we use similar notations as those used in BK. The set COMPSUB contains the growing independent set, the set NOT contains vertices that have been already added to the global set COMPSUB (initially empty) and the set CANDIDATES is initialized to contain all vertices. Moreover, we have the Maximal Matching of Ğ in set M. **EnumerateI,** below, enumerates all maximal independent sets. It takes three arguments: the next edge in M, the CANDIDATES and NOT sets.

```
EnumerateI(CANDIDATES, NOT, matching_edge u-v)
{
    If there are no more edges in M then
            Add to the new-independent array all remaining vertices in CANDIDATES
            New_NOT = NOT – the neighbors of the added vertices from I
            If NOT is empty then print this maximal independent set otherwise this result is not
            maximal and it was already printed (so ignore this case).
    Else
        if u is in CANDIDATES
            Add u to COMPSUB
            NEW_CANDIDATES = CANDIDATES - N(u)
            New_NOT = NOT – N(u)
            EnumerateI(NEW_NOT,NEW_CANDIDATES, Next_matching_Edge)
            Remove u from COMPSUB and add it to NOT

        if v is in CANDIDATES
            Add v to COMPSUB
            NEW_CANDIDATES = CANDIDATES - N(v)
            New_NOT = NOT – N(v)
            EnumerateI(NEW_NOT,NEW_CANDIDATES, Next_matching_Edge)
            Remove v from COMPSUB and add it to NOT

        NEW_NOT = NOT + ( {u,v} ∩ CANDIDATES)
        EnumerateI(NEW_NOT,CANDIDATES, Next_matching_Edge)
}
```

Theoretically, the run time of our algorithm depends on the size of the maximal matching: the smaller the matching size the faster is the enumeration. In order to improve upon the (somewhat generic) worst-case scenario of the BK strategy, the input graph should have a maximal matching whose size is smaller than n/3. This being the case because the worst-case run time of BK is $O(3^{n/3})$.

The density of the graph plays another major role in determining which algorithm is faster. Recall that BK is faster on sparse graphs than it is on dense ones. The value of k could affect the density in some extreme cases. For example, suppose we have k=n/4. This makes the graph density of G at most 50% (50% of the vertices are not part of the matching, so they induce an edgeless subgraph). In this case, the density of the complement graph (which is the one BK takes as input) is at least 50%.

Again, let $\check{G}$ be the complement of the input graph G of BK. Another interesting scenario is when $\check{G}$ has n/3 edges in M. The vertex set of M contains 2n/3 vertices. So I contains n/3 vertices. Again, there are 0 edges among vertices in I, but vertices in M can form a complete subgraph. This yields the following formula:

Maximum number of edges in $\check{G}$ = (maximum number of edges in M) + (maximum number of edges from M to I) = $(2n/3\ (2n/3-1))/2 + 2n/3(n/3) = n(n-1)/3$

We know that the maximum number of edges in a graph of n vertices is n(n-1)/2. Then (Maximum number of edges in $\check{G}$) / (maximum number of edges in any graph) = 2/3.
It follows that the maximum density of $\check{G}$ is 66%. So the input to BK has a minimum density of 33%.
The above numbers are only a guide for us. They show that graphs of large density could have complements of small maximal matching, which is a favorable condition for our independent set enumeration algorithm. In practice, there are many other factors that may affect the speed-up. We realized, based on experimental study, that the local density of the subgraph induced by the vertex set of M (G_M) plays a major role determining the speed-up/slow-down from using our algorithm. This will explained in the next chapter.

Chapter 4

Experimental Analysis

4.1 BK verses Vertex Cover approaches

In our experiments, we generate random graphs with different sizes, vertex cover sizes, and densities. Each graph was generated based on these factors. In other words, our random generator takes the vertex cover size, k, as input instead of generating a graph and, later, finding the minimum vertex cover. This reverse engineering technique was necessary for us because we wanted to know the relationship between the ratio k/n and the speedup. Moreover, we are not interested in finding the smallest vertex cover.

The speed-ups that we recorded looked somewhat sporadic (initially). We could have two different experiments such that the first graph has a smaller vertex cover than the second, yet our Expand_VC code could be faster on the second graph. Our investigation of this behavior revealed that the density of the graph Gs (induced by the cover) plays a major role. This is not a surprise, given that BK is faster on sparse graphs and given that we are employing BK on Gs. But the question was not completely resolved as such, since we could have covers whose size is 50% and others of size 90%, yet the latter ones yield better results. Further investigation showed that the ratio of edges of Gs to the edges in the input graph is responsible for this behavior. Again, the main reason for this was obvious. BK spends most of its computation time on dense regions of the graphs, if a vertex cover is denser than (or at least as dense as) the graph then the time complexity of our algorithm may not be better than that of BK.

In all of the following experiments we used n = 10000 for the graph size. We ran the two codes on graphs of densities in the range 1% through 5%. To guarantee the ratio k/n in generated graphs, we generated our graphs by first generating edges between the k

vertices of the cover (using a specified number of edges), then we generate edges that connect cover vertices to the outsiders.

The following experiments were performed on a Pentium 4 3.4 GHz machine, with 1 GB of RAM. The operating system is a Rocks 4.2 Beowulf Cluster Solution. The following table shows experiments on graphs of density 2%. It gives an idea on how the charts were generated. The times reported here are system times, all expressed in seconds.

In what follows, the term speed-up is the ratio of the BK Time to the time of our algorithm.

Table 4.1 BK Vs Expand_VC

VC size	VC density	Number of maximal cliques	BK time	Expand VC time	Speed-up	Edges in VC	VC/G
3000	0.222274091	135415	157	4	39.25	9999	1%
3000	2.445015005	1344282	321	43	7.465116	109989	11%
3000	4.667755919	2063446	486	143	3.398601	209979	21%
3000	6.890496832	2600506	715	406	1.761084	309969	31%
3000	9.113237746	3589543	1178	1064	1.107143	409959	41%
3000	11.33597866	6446735	2282	2638	0.865049	509949	51%
5000	0.080008002	78923	151	4	37.75	9999	1%
5000	0.880088018	702551	219	28	7.821429	109989	11%
5000	1.680168034	1089588	274	63	4.349206	209979	21%
5000	2.48024805	1392318	326	118	2.762712	309969	31%
5000	3.280328066	1670109	384	205	1.873171	409959	41%
5000	4.080408082	1978022	463	343	1.349854	509949	51%
5000	4.880488098	2349259	591	547	1.080439	609939	61%
5000	5.680568114	2788454	795	865	0.919075	709929	71%
5000	6.48064813	3491065	1131	1333	0.848462	809919	81%
7000	0.040818076	55401	150	4	37.5	9999	1%
7000	0.448998837	560570	203	27	7.518519	109989	11%
7000	0.857179597	914205	246	50	4.92	209979	21%
7000	1.265360358	1084232	272	79	3.443038	309969	31%
7000	1.673541118	1193014	287	117	2.452991	409959	41%
7000	2.081721879	1285585	306	164	1.865854	509949	51%
7000	2.489902639	1426684	374	276	1.355072	609939	61%
7000	2.8980834	1626276	389	340	1.144118	709929	71%
7000	3.30626416	1946547	462	443	1.042889	809919	81%
7000	3.714444921	2382438	552	591	0.93401	909909	91%
9000	0.024691632	115725	163	5	32.6	9999	1%
9000	0.271607956	1033626	278	39	7.128205	109989	11%
9000	0.51852428	1575958	354	64	5.53125	209979	21%
9000	0.765440605	1807936	386	85	4.541176	309969	31%
9000	1.012356929	1790508	381	105	3.628571	409959	41%
9000	1.259273253	1639598	362	132	2.742424	509949	51%
9000	1.506189577	1468631	335	162	2.067901	609939	61%
9000	1.753105901	1320061	311	196	1.586735	709929	71%
9000	2.000022225	1262704	299	239	1.251046	809919	81%
9000	2.246938549	1414438	327	311	1.051447	909909	91%

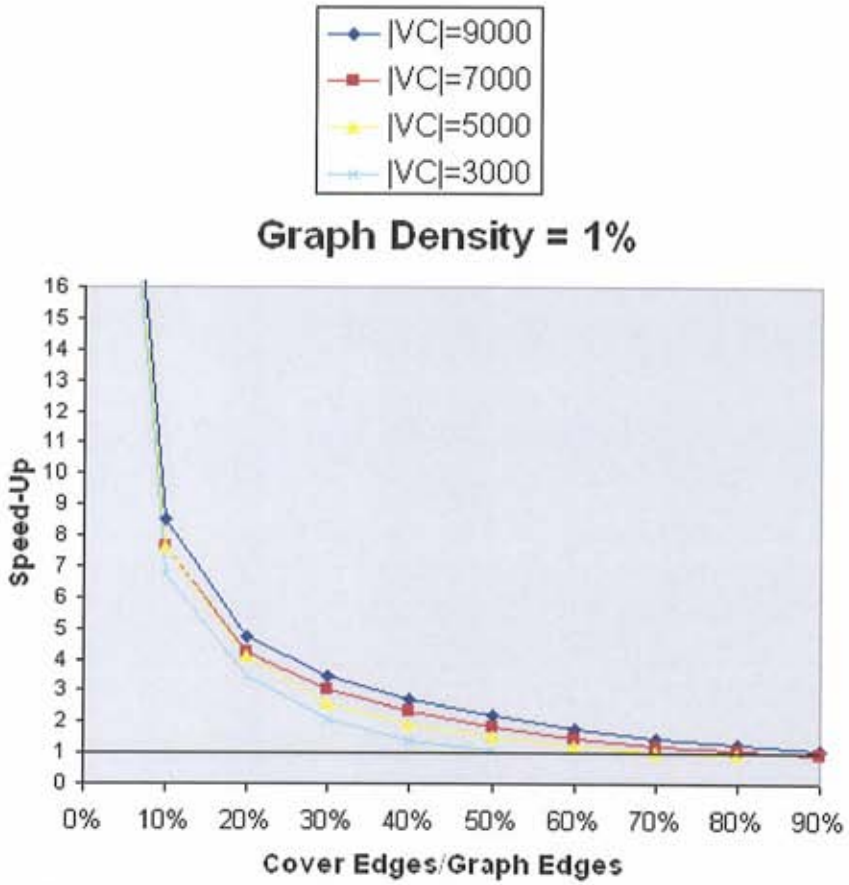


Chart 4.1 Speed up of Expand_VC over BK for graph density 1%

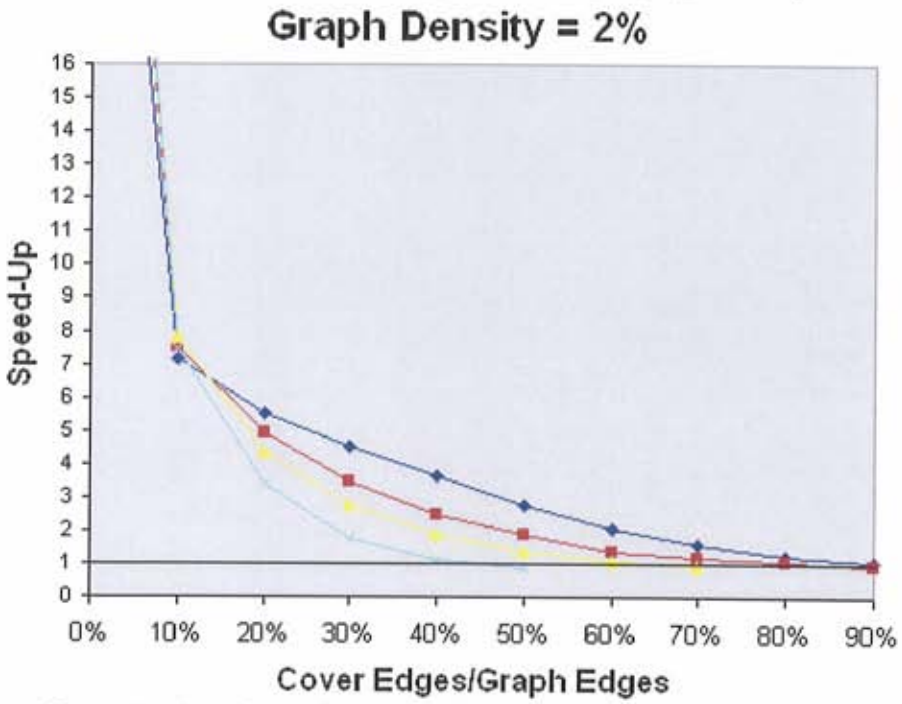


Chart 4.2 Speed up of Expand_VC over BK for graph density 2%

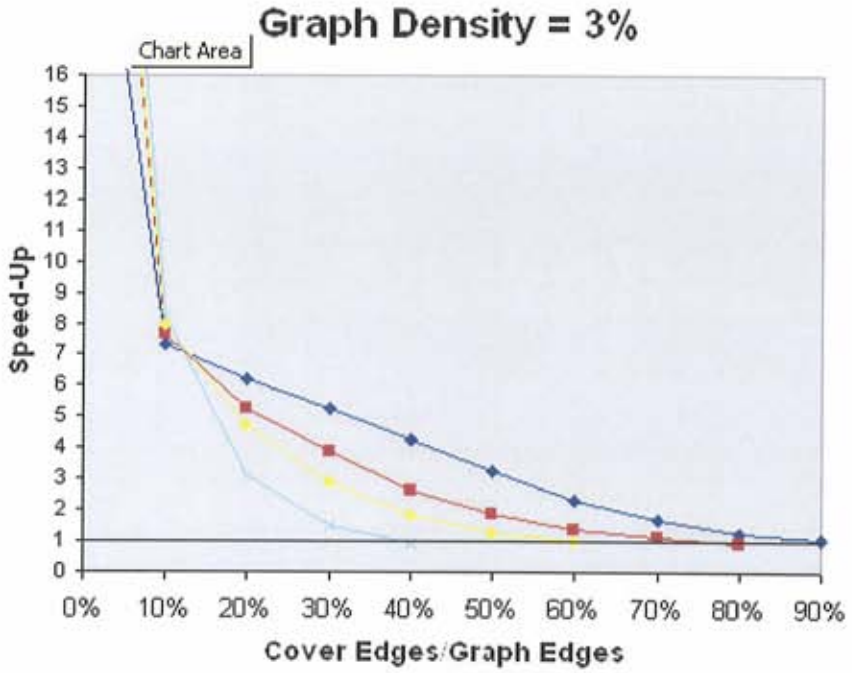


Chart 4.3 Speed up of Expand_VC over BK for graph density 3%

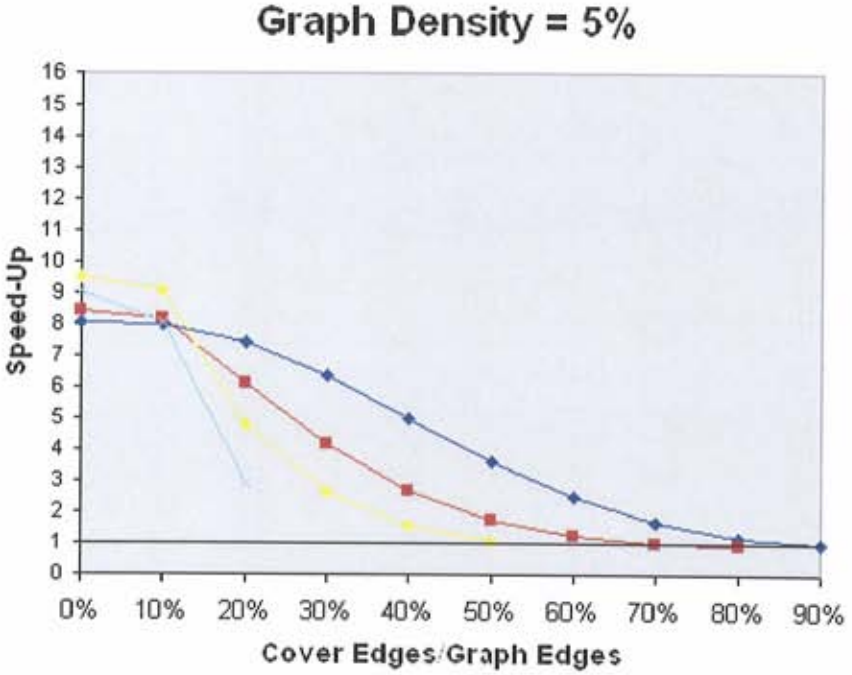


Chart 4.4 Speed up of Expand_VC over BK for graph density 5%

When we apply the same experiments for the first approach implementation we notice that the speed-up in regards to the BK algorithm is only in the cases where the ratio of the cover edges to the total edges in the graph is less than 10%.

We could use a parameter driven search that will eliminate vertices and edges from the graph if they are not part of any maximal clique of a certain user-defined parameter p. p will be the minimum required clique size, vertices and edges that are not part of any p clique should be removed from the graph. This kind of graph pruning was mentioned in more than one paper including Abu Khzam et al. (2005) and the Winnower algorithm that focuses on finding spurious edges to eliminate them before the clique search (Pevzner et al. 2000). But using these techniques will not allow us to compare our algorithm to the BK algorithm. In all experiments we used the BK algorithm exactly as presented in the initial paper and more details about the implementation were also mentioned in (Tomita et al 2004).

Table 4.2Comparing method 1 Vs method 2

Number of Vertices	Graph Density	VC size	VC Density	Number of maximal Cliques	BK Time	Expend VC	First Approach	VC/G
10000	5	1000	4	10533488	2863	42	234	0.79928
10000	5	1000	15	60356877	27457	674	3345	2.9973
10000	5	2000	4	10928783	2770	116	798	3.19872
10000	5	2000	8	19930002	6911	506	5676	6.39744
10000	2	5000	2	1261017	303	80	8567	24.9975
10000	2	5000	1	708848	223	35	809	12.4987

All of the above experiments use a maximal matching of the graph, since a maximal matching is a special vertex cover, and these graphs are generated synthetically, the result will not differ if the dominating structure used was a cover or a matching. The best way to choose what to use between the vertex cover or the maximal matching is the characteristics of the graph induced from this cover or from this matching. As we noticed in the experiments, the smaller the percentage of edges in the dominating structure (Matching or Cover) the better the speed-up.

4.2 BK Vs Maximal Matchings to Enumerate Maximal Independent Sets in the Complement Graph

In our experiments, we also generated random graphs with different sizes, maximal matching sizes, and densities. Each graph was generated based on these factors. But here we generate the complement also to apply BK on it. It is almost the same idea as in the above synthetic graph generation. Again, we needed this reverse engineering technique because we wanted to know the relationship between the ratio of edges in the matching to the edges in the graph and the speedup. Moreover, we are not interested in finding the smallest maximal matching.

Since in these experiments we are interested in dense graphs, we had a problem with BK because it could take hours before it finishes on dense graphs, even if the graph size does not exceed 100. So we had two sets of experiments: one on graphs of size 50 and another on graphs of size 100, the results below display CPU time, expressed in seconds. The experiments where done on the same machine as the above (Pentium 4 3.4 GHz with 1 GB of RAM).

Table 4.3 EnumerateI Vs BK on graph size 50

G Vertices	G Density	Comp Density	MM Size	Edges in MM / Edges in Ğ	Maximal Cliques	BK Sec	MM Sec
50	10.00	90.00	23	90%	34	0	0
50	15.00	85.00	22	80%	59	0	0
50	15.00	85.00	22	90%	46	0	0
50	20.00	80.00	20	60%	101	0	0
50	20.00	80.00	20	70%	96	0	0
50	30.00	70.00	18	50%	197	0.001000	0
50	30.00	70.00	18	60%	148	0.001000	0
50	30.00	70.00	18	70%	64	0.002000	0
50	40.00	60.00	16	40%	271	0.022000	0
50	40.00	60.00	16	50%	178	0.022000	0
50	40.00	60.00	16	60%	90	0.024000	0
50	50.00	50.00	13	10%	378	1.043000	0.005000
50	50.00	50.00	13	20%	341	1.045000	0
50	50.00	50.00	13	40%	132	1.014000	0
50	50.00	50.00	13	50%	47	1.155000	0
50	50.00	50.00	16	10%	1759	0.078000	0.134000
50	50.00	50.00	16	20%	805	0.020000	0.008000
50	50.00	50.00	16	40%	573	0.018000	0
50	50.00	50.00	16	50%	435	0.019000	0
50	50.00	50.00	16	60%	220	0.026000	0
50	50.00	50.00	16	70%	108	0.041000	0
50	50.00	50.00	16	80%	39	0.072000	0
50	60.00	40.00	10	10%	264	69.190000	0
50	60.00	40.00	10	20%	182	65.706000	0
50	60.00	40.00	13	10%	844	1.075000	0.008000
50	60.00	40.00	13	20%	804	1.079000	0.001000
50	60.00	40.00	13	40%	326	1.034000	0
50	60.00	40.00	13	50%	151	1.234000	0
50	60.00	40.00	13	60%	60	1.460000	0
50	60.00	40.00	16	10%	3811	0.141000	0.268000
50	60.00	40.00	16	20%	1363	0.027000	0.024000
50	60.00	40.00	16	40%	1221	0.022000	0.001000
50	60.00	40.00	16	50%	924	0.028000	0
50	60.00	40.00	16	60%	601	0.038000	0
50	60.00	40.00	16	70%	331	0.094000	0
50	60.00	40.00	16	80%	146	0.218000	0
50	70.00	30.00	9	10%	531	273.598000	0
50	70.00	30.00	9	20%	214	266.805000	0
50	70.00	30.00	9	40%	25	278.297000	0
50	70.00	30.00	13	10%	2228	1.087000	0.023000
50	70.00	30.00	13	20%	2323	1.183000	0.003000
50	70.00	30.00	13	40%	934	1.499000	0
50	70.00	30.00	13	50%	423	1.560000	0
50	70.00	30.00	13	60%	225	2.609000	0
50	70.00	30.00	13	70%	116	4.594000	0
50	70.00	30.00	13	80%	50	9.306000	0

G Vertices	G Density	Comp Density	MM Size	Edges in MM / Edges in $\bar{G}$	Maximal Cliques	BK Sec	MM Sec
50	70.00	30.00	16	10%	4327	0.393000	0.822000
50	70.00	30.00	16	20%	3964	0.052000	0.062000
50	70.00	30.00	16	40%	4020	0.051000	0.004000
50	70.00	30.00	16	50%	3015	0.062000	0.002000
50	70.00	30.00	16	60%	1769	0.119000	0.001000
50	70.00	30.00	16	70%	911	0.316000	0
50	70.00	30.00	16	80%	494	0.769000	0
50	80.00	20.00	7	10%	311	3967.035000	0
50	80.00	20.00	7	20%	105	3786.916000	0
50	80.00	20.00	10	10%	3308	75.541000	0.003000
50	80.00	20.00	10	20%	1564	71.298000	0
50	80.00	20.00	10	40%	292	98.068000	0
50	80.00	20.00	10	50%	143	124.182000	0
50	80.00	20.00	10	60%	75	154.903000	0
50	80.00	20.00	10	70%	38	262.466000	0
50	80.00	20.00	13	10%	11920	1.500000	0.071000
50	80.00	20.00	13	20%	7834	2.252000	0.017000
50	80.00	20.00	13	40%	3213	3.044000	0.002000
50	80.00	20.00	13	50%	1175	8.550000	0.001000
50	80.00	20.00	13	60%	728	11.358000	0
50	80.00	20.00	13	70%	404	19.270000	0
50	80.00	20.00	13	80%	223	33.731000	0
50	80.00	20.00	16	10%	16358	1.314000	2.298000
50	80.00	20.00	16	20%	13983	0.395000	0.338000
50	80.00	20.00	16	40%	9750	0.665000	0.036000
50	80.00	20.00	16	50%	10051	0.567000	0.013000
50	80.00	20.00	16	60%	3976	1.661000	0.006000
50	80.00	20.00	16	70%	2798	2.192000	0.003000
50	80.00	20.00	16	80%	1743	5.809000	0.002000

Table 4.4 EnumerateI Vs BK on graph size 100

G Vertices	G Density	Comp Density	Current MM	Edges in MM / Edges in $\bar{G}$	Maximal Cliques	BK Sec	MM Sec
100	10	90	46	90%	210	0	0
100	15	85	44	80%	351	0.001	0.001
100	15	85	44	90%	126	0.005	0
100	20	80	40	60%	680	0.17	0.002
100	20	80	40	70%	477	0.15	0
100	30	70	36	50%	1222	38.205	0.003
100	30	70	36	60%	755	37.161	0.001
100	30	70	36	70%	207	37.787	0
100	40	60	32	40%	2168	>1h	0.005
100	40	60	32	50%	1254	>1h	0.001
100	40	60	32	60%	367	>1h	0
100	50	50	26	10%	30813	>1h	3.224
100	50	50	26	20%	4653	>1h	0.038
100	50	50	26	40%	789	>1h	0.001
100	50	50	26	50%	148	>1h	0
100	50	50	32	10%		>1h	>1h
100	50	50	32	20%	52951	>1h	2.395
100	50	50	32	40%	7119	>1h	0.02

As a conclusion of our experimental study on this MM algorithm, the best results were obtained when we have a small and dense matching. The speed-up decreases when the matching is large and more than 80% of the graph edges are not in the graph induced by vertices of M.

Chapter 5

Concluding Remarks

Real data is rich in structural properties. A blind application of generic enumeration methods that work for general graphs is unfair for many such data sets. As can be seen from our experimental results, obtaining a high speed-up depends on finding small sparse dominating structures. Our proposed method takes into consideration this fact and benefits from such structures when they exist, by reducing the graph into smaller subgraphs that dominate all non-trivial cliques. Vertex covers were used as MCE dominating structures because any maximal clique in a graph has all but at most one vertex outside a vertex cover. Many other examples of MCE dominating structures exist and are the subject of future potential improved MCE algorithms.

References

Abu-Khzam F., Baldwin N., Langston M. and Samatova N., *On the Relative Efficiency of Maximal Clique Enumeration Algorithms, with Application to High-Throughput Computational Biology*, International Conference on Research Trends in Science and Technology, Beirut, Lebanon, 2005.

Agrawal R. and Srikant R., *Fast algorithms for mining association rules in large databases*, Proc. VLDB, "487-499", 1994.

Bron C. and Kerbosch J.. *Finding all cliques of an undirected graph.* Communications of the ACM, 16:575–577, 1973.

Butenko S. and Wilhelm W., *Clique-detection Models in Computational Biochemistry and Genomics*, Department of Industrial Engineering, *Texas A&M University, TAMUS 3131*, 2005.

Eiter T. and Makino K., *On computing all abdicative explanations*, Proc AAAI 02, AAAI Press, 62-67, 2002.

Garey M. and Johnson. D., *Computers and Intractability*. W. H. Freeman, New York, 1979.

Kose F., Weckwerth W., Linke T., and Fiehn O.. *Visualizing plant metabolomic correlation networks using clique-metabolite matrices.* Bioinformatics, 17:1198–1208, 2001.

Makino K. and Uno. T., *New algorithm for enumerating all maximal cliques.* In SWAT, volume 3111 of *Lecture Notes in Computer Science*, pages 260–272. Springer Verlag, 2004.

Moon J. and Moser L. *On cliques in graphs*. Israel J. Math, 3:23–28, 1965.

Stix V., *Finding all maximal cliques in dynamic graphs*, Department of Information Business, Augasse 2-6, Austria, 2001

Tomita E., Tanaka A. and Takahashi H., *The worst-case time complexity for generating all maximal cliques.* In 10th Int. Computing and Combinatorics Conf. (COCOON), 2004.

Tsukiyama S., Ide M., Aviyoshi H. and Shirakawa I., *A new algorithm for generating all the maximum independent sets*. SIAM Journal on Computing, 6:505–517, 1977.