# A novel parallel algorithm for maximal clique enumeration on multicore and distributed memory architectures

Naga Shailaja Dasari
*Department of Computer Science, Old Dominion University Norfolk, USA*
*Email: ndasari@cs.odu.edu*

Ranjan Desh
*Department of Computer Science Old Dominion University Norfolk, USA*
*Email: dranjan@cs.odu.edu*

Zubair M
*Department of Computer Science Old Dominion University Norfolk, USA*
*Email: zubair@cs.odu.edu*

***Abstract*—Maximal clique enumeration(MCE) is a fundamental problem in graph theory. It plays a vital role in many network analysis applications like community detection using social networks and also in computational biology problems like 3D protein structure alignment. MCE is a well studied problem. A number of sequential algorithms have been proposed to solve the problem such as the well known BK algorithm. A few parallel algorithms to solve the problem have been proposed most of which are based on the sequential BK algorithm. These algorithms are primarily designed for the distributed architectures. We propose a novel parallel algorithm, *pbitMCE*, and provide implementation of this algorithm on multicore distributed memory architectures. The sequential version of *pbitMCE* outperforms all known sequential MCE algorithms and *pbitMCE* also scales well on the above mentioned architectures. The two key ideas behind our algorithm are: (i) incorporating degeneracy ordering which helps in balancing the work load among the processing units (ii) design and use of a new data structure, *partial bit adjacency matrix*, that reduces the active memory usage resulting in a cache friendly algorithm. We have implemented our algorithm on a multicore machine with 32 cores and a cluster with 32 nodes which has 128 cores in total. We used the graphs from various collections including Stanford large network collection and University of Florida sparse matrix collection with the largest graph having 3.8 million vertices and 16.5 million edges. We observe good scalability: up to 31 on multicore and up to 106 on the cluster. These scalability results are more impressive noting that the sequential version of our parallel algorithm either outperforms or is close to all the known sequential algorithms.**

***Keywords*-maximal clique; parallel; multicore; distributed memory; BK;**

## I. Introduction

Maximal Clique Enumeration(MCE) is a fundamental problem in graph theory. Clique finding procedures are currently being used in many areas of science. In computational biology, they are used in 3D protein structure alignment [1], detection of protein-protein interaction complex [2], motif discovery [3] and many other applications [4] [5]. They are applied in various network analysis applications like community detection [6] [7] [8], social network analysis [9].

Definitions: Let $V$ be the set of vertices and $E$ be the set of edges of an undirected graph $G$. A clique ,$C$, is defined as a set of vertices in which every pair of vertices is connected by an edge i.e. $C$ is a complete subgraph of $G$. A clique $C$ is said to be maximal if it not contained in any other clique. The number of maximal cliques in a graph can be exponential in number of vertices. For example for Moon-Moser graphs, there can be $3^{n/3}$ maximal cliques. Maximal clique enumeration(MCE) is defined as the problem of finding all the maximal cliques in a graph. MCE is equivalent to finding all maximal independent sets in the complimentary graph and is shown to be NP-hard [10].

MCE is an extensively studied problem. Many approaches have been proposed to solve MCE. The first successful attempt to find maximal cliques was made in 1957 [11] It was applied for analysis of sociometric data. This approach was followed by some more clique finding techniques. However, the real breakthrough came in 1973 with the proposal of two separate algorithms one by Akkoyunlu [12] and the other by Bron and Kerbosch [13]. The approaches that were proposed before [12] and [13] generated cliques that included duplicates and so all the cliques had to be stored to be filtered later. Therefore, those approaches could not be used for larger problems as they ran out of memory. Both Akkoyunulu and BK algorithms generate only the maximal cliques without any duplicates and so were able to be applied for large problems.

The simplicity of BK algorithm lead to proposal of different variations and extensions to the algorithm [14] [15] [16]. Johnston [14] proposed a series of algorithms based on the BK algorithm and showed that the original BK algorithm ,in general, performs better compared to others. In [13], two variations of BK algorithm were proposed, the basic version and pivot version. Both the versions of BK algorithm works by maintaining three lists, candidate list, not list and compsub list (represented as $P$, $X$ and $R$ respectively). It proceeds by exploring a search tree using the three lists in a series of iterations. In the pivot version, a pivot node is selected from $P$. The number of recursive calls and the structure of the tree depends on the pivot node selected. Therefore, the selection of pivot node is a crucial part of the algorithm. Koch et al. [16] and Tomita et al. [15] proposed

different ways for pivot selection. Tomita et al.'s approach is shown to be worst case optimal and performs faster, in practice, than the other variations. [17] bridges the gap between the original BK algorithm, Koch et al.'s algorithms and Tomita et al.'s algorithm and showed that Tomita et al.'s algorithm is a simple modification of BK algorithm based on an observation made by Koch et al. Other approaches proposed for MCE include [18] [19] [20]. In practice, the approaches based on BK algorithm are shown to outperform the remaining approaches.

Tomita et al.'s algorithm, uses adjacency matrix representation for theoretical analysis and implementation. This limits the applicability of the algorithm to large sparse graphs as adjacency matrix representation for such graphs is not feasible due to memory limitation. To overcome the limitation, Eppstein et al. [21] [22] proposed an approach based on BK algorithm and Tomita et al.'s pivot selection rule. This algorithm is shown to provide a practical solution for large sparse graphs. Eppstein et al. showed that the initial ordering of the vertices plays a vital role in the performance of the algorithm. Eppstein et al.'s approach uses degeneracy to initially order the vertices before applying the BK algorithm. Degeneracy of a graph is the smallest number $d$ such that every subgraph of the graph has atleast one vertex with degree less than or equal to $d$. The degeneracy ordering of the vertices of a graph is the order in which every vertex has at most $d$ neighbors that have order higher than its order. Eppstein uses a subgraph structure to store the vertices and edges that are currently used in the enumeration. [22] shows that the Eppstein et al.'s algorithm is faster than the Tomita et al.'s algorithm for large sparse graphs, sometimes faster by large factor. To the best of our knowledge Eppstein et al.'s algorithm is the best sequential algorithm available so far for large sparse graphs.

Some parallel approaches were also proposed for maximal cliques enumeration. The first parallel approach is proposed in [23] and is referred to pCliques. This approach is based on the approach of Kose et al. [24]. It generates the cliques in increasing order of the clique size. The cliques of size $k+1$ are generated from the cliques of size $k$. However, this approach is very memory intensive and so is not practical for large graphs. v *peamc* [25] is another parallel algorithm proposed for MCE. *peamc* uses triangle structure as the basis. At each level it generates a set of vertices that form triangles with the current vertex and its neighbors. It proceeds recursively by selecting a vertex from the set of vertices generated from the previous level. By selecting the vertices in the increasing order *peamc* avoids the chance of generating duplicate cliques. However, it might generate non-maximal cliques. It uses an additional filtering step to check if the clique is maximal. *peamc* uses a simple strategy for parallelizing. The vertices are distributed to the available computing nodes and the nodes independently work on the vertices assigned. Most real world graphs follow power-law degree distribution. *peamc* does not work well for these graphs as the load is not balanced. Some computing nodes terminate earlier while other nodes are still working.

[26] provides a state of the art parallel MCE algorithm. We refer to the algorithm as PSMCE. PSMCE parallelizes the original BK algorithm. It uses different strategies to efficiently parallelize the BK algorithm. All the information required to explore a search tree rooted at a node is stored in a structure called candidate path structure which essentially consists of cand list, not list, the current vertex and the level of the search tree node. Like *peamc*, PSMCE also distributes the vertices to the available computing nodes. However, unlike *peamc*, it solves the imbalance problem using dynamic load balancing techniques. This is done by using a stack structure which stores the candidate path structures. In each iteration, a candidate path structure is taken from the stack and the candidate path structures corresponding to its children in the search tree are generated and added to the stack. Whenever a process finishes the work assigned to it, it requests work from another process which responds by removing some candidate path structures from its stack and sending to the requesting process. By dynamically balancing the load PSMCE ensures that all the computing nodes terminate at the same time and hence was able to achieve good scaling.

*dMaximalCliques* [27] is a distributed algorithm for maximal cliques. It uses Tsukiyama's algorithm [18] for enumerating the maximal cliques. Similar to *peamc*, this algorithm does not include any strategies for dynamic load balancing and hence was not able to scale well. There also exists algorithms for extremely large graphs [28] [29] [30] [31] that do not fit in memory. These are out of scope of this paper.

In this paper we propose a new algorithm, *pbitMCE*, for efficiently enumerating all maximal cliques. *pbitMCE* is identical in spirit to Eppstein et al.'s algorithm. Like, Eppstein et al.'s algorithm, *pbitMCE* also degeneracy ordering of vertices and also Tomita et al.'s pivot selection rule. Eppstein et al. uses a dynamic subgraph structure to store the current working set. However, these subgraph structures need to be computed at each level. We propose a new structure, *partial bit adjacency matrix(pbam)*, to represent the subgraph. *pbam* is a set of bit vectors. By using *pbam*, the two major tasks involved in the enumeration process, pivot selection and set intersections, take $O(d(|P| + |X|))$ and $O(|P| + |X|)$ time respectively, while Eppstein et al.'s approach takes $O(|P|(|P| + |X|)$ and $O(|P|^2(|P| + |X|)$ respectively. Moreover, using bit vectors reduces the size of $pbam$ and in most cases $pbam$ is small enough to fit in lower levels of cache. Note that *pbitMCE* and Eppstein et al.'s algorithm, both address the poor data locality problem faced by most graph algorithm by storing the data that is required in the current level and lower levels in a subgraph.

We have implemented *pbitMCE* on multicore architecture

and distributed memory architecture. Although degeneracy ordering is not explicitly used for parallelizing, it significantly contributed to load balancing. By using degeneracy ordering, the cand list size is limited to $d$. By limiting the cand list size, the size of the search tree corresponding to the high degree nodes reduces while that if low degree nodes increase which results in a balanced load.

The experimental results show that the sequential implementation of *pbitMCE* outperforms or is close to all the sequential algorithms. We have used graphs from different collections including Stanford large network data collection [32] and University of Florida spaprse matrix collection [33]. The results show that *pbitMCE* scales upto 31 on a 32 core multicore machine and upto 106 on a 32 nodes cluster with 128 cores in total.

The rest of the paper is organized as follows: the algorithms on which *pbitMCE* is based on: the BK algorithm , the Tomita et al.'s algorithm and the Eppstein et al.'s algorithm are described in sections II-A, II-B and II-C respectively. In section III we describe the *pbitMCE* algorithm. Section III-B gives the details of the $pbam$ structure used in *pbitMCE* and explains how it is constructed. Sections III-C and III-D describe how $pbam$ can be used to efficiently perform the pivot selection and set intersection: the two major tasks in enumeration. In section III-E, we describe how the approach can be further refined by using hierarchical renumering and the parallelization is given in section III-G. We then present the experimental results in section IV.

## II. Background

In this section we describe various algorithms and concepts that *pbitMCE* is based on: the BK algorithm, Tomita et al.'s algorithm (referred to as TTT), Eppstein et al.'s(referred to as ELS) algorithm and degeneracy. *pbitMCE* is identical in spirit to the ELS algorithm which is based on TTT algorithm. The TTT algorithm in turn is based on the BK algorithm. The ELS algorithm and *pbitMCE* both use an initial ordering of vertices based on degeneracy.

### A. The BK algorithm

The BK algorithm is a recursive backtracking algorithm. It was proposed by Bron Kerborsh in 1973 [13]. It is one of the most successful and efficient algorithms used in practice for maximal clique enumeration. It maintains three lists through out the enumeration process: *candidate* list, *not* list and *compsub* list, commonly denoted as $P$, $X$ and $R$ respectively. The set of vertices in $R$ form a clique, but it might not be maximal. The idea is to extend $R$ until it forms a maximal cliques. $P$ contains the set of vertices that are adjacent to all the vertices in $R$ and are potential candidates to be added to the maximal clique. $X$ list also contains the vertices that are adjacent to all the vertices in *compsub* list but adding these vertices to the clique results in duplicate or non-maximal cliques. The BK algorithm is presented in

Figure 1. The BK algorithm

```
1: procedure BK(P, X, R)
2: if P is empty then
3:    if X is empty then
4:       R is a maximal clique
5:    end if
6: else
7:    for each vertex v in P do
8:       call BK(P ∩ N{v}, X ∩ N{v}, R ∪ {v})
9:       P = P \ v
10:      X = X ∪ v
11:   end for
12: end if
```

Figure 1. $N(v)$ in the algorithm represents the adjacency list of a vertex $v$. In each call to BK procedure, it iterates $P$ times once for each vertex in $P$. In each iteration, a vertex from $P$ is added to $R$, new sets of vertices of $P$ and $X$ are computed and input to a subsequent recursive call. The BK algorithm can be viewed as exploring a search tree. Each node in the tree node contains the three lists: $P$, $X$ and $R$. The algorithm proceeds by exploring the search tree in a depth first style, backtracking when $P$ becomes empty. This algorithm is the basic version of the BK algorithm. Another version of BK was also proposed in [13], which involves pivoting. Tomita et al.'s algorithm is based on the pivoting version of BK. It is explained in the next section.

### B. The Tomita et al.'s algorithm

The algorithm described in section II-A is the basic version of the BK algorithm. Another variation of the BK algorithm exists that involves a technique called pivoting. We have seen that the BK algorithm makes $P$ recursive calls. Pivoting improves the basic BK algorithm by reducing the number of recursive calls. This is done by selecting a vertex $u$ called pivot , and all the subsequent maximal cliques must contain a non-neighbor of $u$. This reduces the number of recursive calls by $|P \cap N(u)|$. A number of variations for selecting the pivot vertex were proposed [15] [16]. The variation by Tomita et al. is shown to be best in theory and practice. Tomita et al.'s pivot selection method is to select a vertex $u$ from $P \cup X$ that maximizes $|P \cap N(u)|$. Obviously, this results in minimum number of recursive calls among all possible pivots. The TTT algorithm is shown in Figure 2.

The analysis and the implementation of the TTT algorithm presented in [15] is based on adjacency matrix representation of the input graph. By using adjacency matrices, the pivot selection(line 7 in Figure 2 and set intersections(in line 9 in Figure 2) can be computed in time $O(|P|(|P| + |X|))$. The worst case time complexity for TTT is shown to be $O(3^{n/3})$. However, most real world graphs are large sparse graphs and the adjacency matrix representation is impractical for such graphs.

Figure 2. The Tomita et al.'s algorithm

```
1: TTT(P, X, R)
2: if P is empty then
3:    if X is empty then
4:       R is a maximal clique
5:    end if
6: else
7:    choose a pivot vertex u in P ∪ X
8:    for each vertex v in P \ N{u} do
9:       call TTT(P ∩ N{v}, X ∩ N{v}, R ∪ {v})
10:      P = P \ v
11:      X = X ∪ v
12:   end for
13: end if
```

Figure 3. The Eppstein et al.'s algorithm

```
1: BKMulti(P, X, R)
2: if P is empty then
3:    if X is empty then
4:       R is a maximal clique
5:    end if
6: else
7:    for each vertex v_i in the degeneracy ordering
      v_0, v_1, ..., v_{n-1} do
8:       P = N(v_i) ∩ {v_{i+1}, ..., v_{n-1}}
9:       X = N(v_i) ∩ {v_0, ..., v_{i-1}}
10:      call TTT(P, X, {v_i})
11:   end for
12: end if
```

### C. The Eppstein et al.'s algorithm

To overcome the drawback of the TTT algorithm, Eppstein et al. proposed a new algorithm, referred to as ERS algorithm, that is based on TTT but uses adjacency list representation of the graph. The ERS algorithm is given in Figure 3. Unlike the BK and TTT algorithms, ERS algorithm explores multiple search trees. This makes it more suitable for suitable choice for being used in a parallel approach. ERS uses an initial ordering of vertices based on degeneracy. Degeneracy of a graph $G$ is the smallest number $d$ such that every subgraph of $G$ has atleast one vertex with degree at most $d$. Degeneracy ordering is the ordering of vertices such that every vertex has at most $d$ neighbors that come later in the ordering. Degeneracy ordering of vertices in a graph can be computed in linear time by repeatedly removing the vertex with smallest degree [34]. Further details on degeneracy ordering are given in Section II-D.

For each vertex $v$ in the graph, an initial list of vertices in $P$, $X$ and $R$ are computed based on the degeneracy ordering of vertices and input to the TTT procedure. The set $P$ is computed by adding all the neighbors of $v$ that come later in the ordering. Similarly, the set $X$ is computed by adding all the neighbors of $v$ that come before $v$ in the ordering. Degeneracy ordering makes sure that there are no more than $d$ neighbors that come later in the ordering. Therefore, the size of $P$ is limited to $d$ in the outermost call to TTT (line 10 in Figure 3). This reduces the number of recursive calls within the outermost TTT call.

ELS further enhances the approach by using a subgraph representation, $H_{P,X}$(we refer to as *hypergraph)*, that contains all the vertices in $P \cup X$ at the current level, and edges connecting a vertex in $P$ to a vertex in $P \cup X$. Using the hypergraph representation, ELS computes the pivot vertex in $O(|P|(|P|+|X|)$ and the set intersections in $O(|P|^2(|P|+|X|)$. The running time of ELS is shown to be $O(dn3^{d/3})$ which is worst case optimal for large sparse graphs.

A similar approach that explores multiple search trees is used in PSMCE [26] with a difference in the ordering and the underlying algorithm used. PSMCE uses the initial ordering of the vertices as is, that is the ordering based on the vertex numbers while ELS uses degeneracy ordering. While ELS uses the Tomita et al.'s pivoting rule, i.e selects a pivot vertex $u$ from $P \cup X$ that maximizes $P \cap N(u)$, PSMCE selects a pivot vertex $u$ from $P$ that maximizes $P \cap N(u)$.

### D. Degeneracy ordering

Similar to ELS algorithm, *pbitMCE* also computes the initial ordering of vertices. We have seen that by using degeneracy ordering the number of recursive calls to the underlying TTT algorithm is reduced. In *pbitMCE*, the advantage of using degeneracy ordering is two fold. The first is that as in ELS, it reduces the number of recursive calls improving the overall performance. The second advantage comes in the context of parallelization. Although not explicitly, degeneracy ordering significantly contributes to the load balancing. Most of the real world graphs are power law degree distributed graphs i.e few vertices have relatively very high degree compared to the majority of vertices. These high degree vertices, in general are contained in a large number of maximal cliques and the search trees corresponding to these vertices are large in size. Therefore these vertices tend to consume majority of the total time taken for maximal clique enumeration. In these graphs, there is a wide variation in the sizes of the search trees associated with the vertices and so the time taken to explore the search trees.

We have seen that the degeneracy ordering ensures that each vertex has no more than $d$ neighbors that have greater order than itself. This limits the candidate list size to $d$ which in turn limits the size of the search tree. Let $M_v$ be the set of maximal cliques that are generated by exploring the search tree of a vertex $v$ based on initial ordering of vertices. The time taken for $v$ depends on the number and size of the cliques in $M_v$. Typically, the high degree vertices have large number of clique in $M_v$. By using degeneracy most of the cliques in $M_v$ of high degree nodes are moved to $M_v$

of low degree nodes. This reduces the time taken by high degree vertices and increases the time taken by low degree vertices resulting in a much balanced load. Figure II-D depicts the variation in the time taken by different vertices with and without degeneracy ordering. This is obtained by running Tomita et al.'s algorithm on a random graph with 700 vertices and 0.3 edge density using a single process. It can be seen from the figure that the time taken varies from 0 to 1.2 when degeneracy ordering is not used and from 0 to 0.15 when it is used. It can be seen from the figure that without using when degeneracy is not used, some of the vertices take longer time compared to the majority of the vertices. When degeneracy is used, there is not much difference in the time taken in the time taken by different vertices. *pbitMCE* is the first parallel approach that takes advantage of degeneracy ordering.

## III. The *pbitMCE* algorithm

*pbitMCE* is identical in spirit to the ELS algorithm. Like ELS algorithm *pbitMCE* also uses degeneracy based initial ordering of vertices. We have seen that ELS uses a subgraph representation $H_{P,X}$ that stores, at each level, the vertices in $P \cup X$ and the edges connecting a vertex in $P$ to a vertex in $P \cup X$. However, these *hypergraphs* need to be computed at each level. The total time taken for computing *hypergraphs* $O(|P|^2(|P| + |X|))$. In *pbitMCE* we use a different representation of the subgraph $H_{P,X}$ called partial bit adjacency matrix(*pbam*. By using *pbam* the pivot vertex can be computed in $O(d(|P| + |X|))$ and set intersection can be done in $O(|P| + |X|)$ compared to the time taken by ELS $O(|P|(|P| + |X|)$, $O(|P|^2(|P| + |X|)$ respectively.

By using subgraph representation, both ELS and *pbitMCE* store all the information required to process a call at each level in the subgraph structure used. By doing so, they avoid the need to access the larger adjacency list representation of the input graph in which all the information is scattered throughout the graph. By storing only the necessary information in the subgraph, ELS and *pbitMCE* address the problem of poor data locality that many graph algorithms face. Furthermore, the $pbam$ structure used in *pbitMCE* uses bit representation which considerably reduces the current working set size making it cache friendly. In most cases, $pbam$ is small enough to fit in *L1* cache. Note that the time taken to construct the subgraph should be kept to a minimal for the algorithm to be efficient. Further details of $pbam$ are given in section III-B. Figure 5 gives the *pbitMCE* algorithm. The notation used and the steps involved are explained in the following sections.

Figure 5. The *pbitMCE* algorithm

```
1: pbitMCE(P, X, R)
2: if P is empty then
3:    if X is empty then
4:       R is a maximal clique
5:    end if
6: else
7:    for each vertex v in the degeneracy order do
8:       P = postAdjList[v]
9:       X = preAdjList[v]
10:      Construct pbam(P, X, B)
11:      call TTT(P, X, {v}, B )
12:   end for
13: end if
```

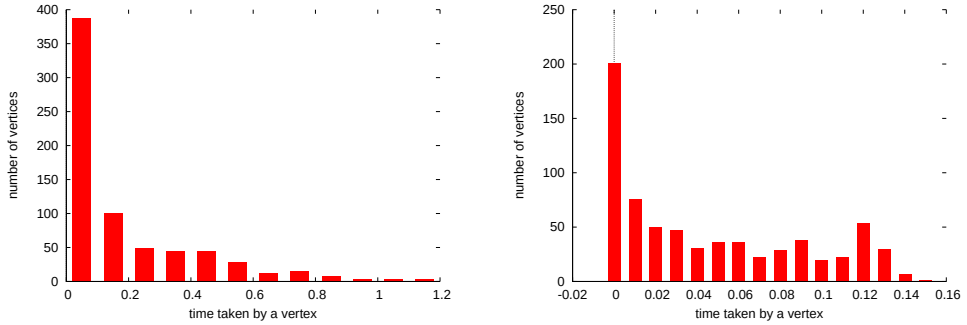
### A. Pre-processing

After reading the input file and constructing the initial adjacency list, the degeneracy ordering of the vertices is generated. As we have seen in Section II-D, the degeneracy ordering places the vertices such that each vertex has no more than $d$ neighbors that come later in the ordering. While exploring the BK search tree corresponding to a vertex $v$, the neighbors of $v$ that come before $v$ in the degeneracy order are accessed only once, to construct the root node. So to avoid unnecessary processing of those vertices in the further enumeration process, we partition the adjacency list into two separate lists: pre-adjacency list and post-adjacency list denoted as $preAdjList$ and $postAdjList$ respectively. The pre-adjacency list of a vertex $v$, $preAdjList[v]$, is the list of all neighbors of $v$ that come before $v$ in the degeneracy order. Similarly, $postAdjList[v]$ is the list of all neighbors of $v$ that come later in the degeneracy order. Obviously, the maximum size of $postAdjList[v]$ is $d$, the degeneracy of the graph. The initial adjacency list can be discarded at this point as it is no longer required.

### B. Partial bit adjacency matrix

As in ELS, *pbitMCE* also uses Tomita et al.'s TTT algorithm as the basis. We have seen that the TTT uses adjacency matrix representation of the graph. The constant lookup time for adjacency matrix makes it, in general, a preferred method of representation of a graph. However, for large sparse graphs, which most real world graphs are, the adjacency matrix representation is not feasible because of the memory limitations. We propose a method by which we can take advantage of the constant lookup time of adjacency matrix representation and yet meeting the memory constraints.

Notice that to process a TTT call(Figure 2), the only information that is required is the list of vertices in $P$ and $X$ and their adjacency lists i.e a subgraph, denoted by $S_v$, with the vertex set $P \cup X$ and the edge set $\{(u, v)|u \in P,\ v \in \{P \cup X\}\}$. Note that we did not include the edges connecting a vertex in $X$ with another vertex in $X$ since such edges are never used in the TTT algorithm. This subgraph is represented using a partial adjacency matrix (*pbam*). *pbam* is essentially a set of bit vectors of size $P$, each corresponding to a vertex in subgraph. Each bit in a bit vector corresponds




(a) without degeneracy ordering (b) with degeneracy ordering

Figure 4. Plot showing distribution of time taken by vertices

to a vertex in $P$. If a vertex $u$ in $P \cup X$ is connected to a vertex in $v$ in $P$, then the bit corresponding to $v$ is set to 1 in the bit vector corresponding to $u$. Before each initial call to TTT in line 11 in Figure 5, we construct a *pbam* and pass it to the TTT procedure. The construction of *pbam* is described in the following section.

*1) Renumbering:* To construct partial adjacency matrix we use a technique called renumbering. Renumbering maps the vertices in $P \cup X$ which are numbered in the range $[0 \ldots |V|-1]$ to a new number in the range $[0 \ldots |P \cup X|-1]$. Each vertex in the set $P$ is assigned a unique number in the range $[0 \ldots |P|-1]$ and each vertex in the set $X$ is assigned a unique number in the range $[|P| \ldots |P \cup X|-1]$. Let $\eta(v)$ denote the new number assigned to a vertex $v$. Once the new numbers are assigned the key values pairs $v, \eta(v)$ are added to a hash table. The partial adjacency matrix can then be constructed using the procedure in Figure 6. Let $B_{\eta(v)}$ represent a bit vector of a vertex $v$. If a vertex $u$ in $P \cup X$ is connected to a vertex $v$ in $P$ then a bit is set at index $\eta(v)$ in bit vector $B_{\eta(u)}$. The bits are set by iterating through the post-adjacency lists of the vertices in $P \cup X$. The structure is called a partial bit adjacency matrix because like adjacency matrix it only takes a constant lookup time but it only has a portion of the adjacency matrix. Assuming constant time to retrieve a value from hash table, *pbam* can be constructed in $O((p+x)d)$ time and $O(d * M)$ space where $M$ is the maximum degree.

To give an idea of the memory requirement, consider the cit-Patents graph from the Stanford large network collection(Table I). It has 3.7 million vertices and 16.5 million vertices and has degeneracy 64. So the maximum size of a bit vector is 8 bytes. The maximum degree for the graph is 793. So the maximum size required for *pbam* is $793 * 8 = 6344$ bytes. It is small enough that the complete structure can fit in *L1* or *L2* cache. $pbam$ also reduces the number of operations required for the two major tasks in TTT algorithm: pivot selection and set intersections. The details of how *pbam* is used for efficient computation of these two tasks is given in the following sections.

Figure 6. Construction of partial bit adjacency matrix

```
1: procedure Constructpbam(P, X, B)
2: for each vertex v in X do
3:     x = η(v)
4:     for each vertex u in postAdjList[v] do
5:         y = η(u)
6:         if y < |P| then
7:             set bit at index y in B_x
8:         end if
9:     end for
10: end for
11: for each vertex v in P do
12:     x = η(v)
13:     for each vertex u in postAdjList[v] do
14:         y = η(u)
15:         if y < |P| then
16:             set bit at index y in B_x
17:             set bit at index x in B_y
18:         end if
19:     end for
20: end for
```

### C. Pivot selection

Once the *pbam* is constructed it is passed to the TTT procedure. We have seen in section II-B that pivoting significantly improves the performance of the algorithm by reducing the number of recursive calls. In TTT, a vertex $u$ is selected as pivot from $P \cup X$ that maximizes $|P \cup N(u)|$ which requires a total of $|P \cup X|$ set intersections. Pivot selection is a crucial part of the algorithm and also takes a significant portion of the total enumeration time. We have seen that *pbam* is simply a set of bit vectors. The set $P$ is also represented as a bit vector. To compute the pivot vertex, we perform logical AND operation between a bit vector corresponding to each vertex in $P \cup X$ and the bit vector of $P$. The number of bits set in each operation gives the value of $|P \cup N(u)|$. The vertex with maximum number of bits set is the pivot vertex.

### D. Set Intersections

Another expensive task in TTT is the set intersections (line 9 in Figure 2). Since *pbam* only takes constant lookup time, the set intersections can be done in $O(|P|+|X|)$. The hypergraph representaion used in ELS on the other hand takes $O(|P|^2(|P|+|X|)$. This is a significant improvement over ELS.

### E. Optimization

We have seen that pivot selection process involves performing logical AND operation between two bit vectors. The maximum size of a bit vector is $d$ bits i.e $d/8$ characters if character array is used to represent bit vector. This requires $d/8$ logical AND operations to intersect two bit vectors. This is faster for higher levels of the search tree. However as we reach the lower levels, the size of $P$ reduces and the bit vector of $P$ is mostly sparse. Even then intersection of two bit vectors require $d/8$ logical AND operation. This is highly inefficient. To overcome this drawback we introduce hierarchical renumbering.

*1) Hierarchical renumbering:* After the candidate list size reduces to a certain value $\tau$ , we construct a new partial bit adjacency matrix by using the renumbering logic explained in section III-B1. The new *pbam* constructed is much smaller in size compared to the previous one. This new smaller matrix is used in further processing instead of the old larger *pbam*. This addresses the problem of requiring $d/8$ logical operations. It now only requires $\tau/8$ operations instead. This process can be repeated when the candidate list size is further reduced. However, if this process is repeated more than required, there is a possibility that the time taken for renumbering and constructing the partial adjacency matrices dominates the overall time taken for enumeration. By experimenting, we have settled down on two levels of renumbering one at $\tau = 128$ and the other at $\tau = 32$ i.e. whenever the candidate list size reduces to 128 or 32 we do renumbering and construct new partial adjacency matrices.

### F. Analysis

Let $p$ be the size of the set $P$ and $x$ be the size of the set $X$ at a certain level. We have seen that the construction of *pbam* takes $O((p+x)d)$ time, assuming constant time to retrieve a value from hash table. Also the time taken for pivot selection and set intersections is bounded by $O((p+x)d)$ and $O(p+x)$ respectively. Given these individual complexities the running time for *pbitMCE* can be given in similar terms as given in [21]. Let $D(p,x)$ be the running time of TTT($P$, $X$, $\{V_i\}$). Then $D$ satisfies the following recurrence relation:

$$D(p,x) \leq \begin{cases} max_k\{kD(p-k,x)\} + d(p+x) & \text{if } p > 0 \\ c & \text{if } p = 0 \end{cases}$$

Eventhough, in [21], in the above relation the second term is replaced by $c_1p^2(p+x)$ which is greater than $d(p+x)$, solving both the recurrence relations result in same complexity $O((d+x)3^{p/3})$. And so the complexity of the *pbitMCE* is same as the ELS algorithm i.e. $O(dn3^{d/3})$. However, in practice, the use of *pbam* significantly reduces the number of operations and so *pbitMCE* is able to achieve better performance compared to ELS.

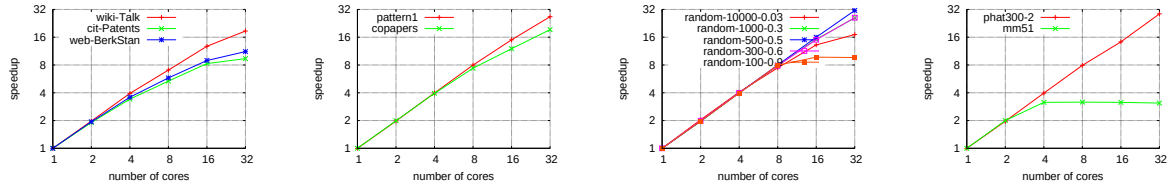
### G. Parallelizing

we have seen that the *pbitMCE* proceeds by exploring multiple independent search trees. Parallelizing *pbitMCE* is as simple as distributing the search trees among the computing nodes. However, different search trees take different time to explore, which leads to an unbalanced distribution of work load. Also, since it is impossible to pre-determine the size of a search tree, the load balancing should be done dynamically. In the following sections we explain how the dynamic load balancing is achieved for both shared memory and distributed memory architectures.

OpenMP is used for parallel implementation on shared memory architecture. All the preprocessing is done by the parent thread i.e. the parent threads reads the input graph, constructs the initial adjacency list, computes degeneracy order and builds the pre and post adjacency lists. The task of enumeration is then distributed to all threads by OpenMP using dynamic scheduling. All the threads are assigned a chunk of vertices. The threads independently work on the vertices assigned by exploring their search trees. When a thread finishes its work it is assigned another chunk of vertices.

In case of distributed memory architecture, the parallel implementation is done using MPI. Each computing task or process preforms the preprocessing. This is done by each process because each process needs the data, i.e pre adjacency list, post adjacency list and degeneracy order, for further processing and since the data is repeatedly accessed, it is efficient to keep it local than to access remotely. The actual work of enumeration is then distributed to the computing tasks by equally assigning the vertices to them. The computing tasks then work independently by exploring the search trees corresponding to the vertices assigned to them. When a computing node finishes all the work assigned to it, it selects another computing node at random and requests for work. The node which receives the request shares its work by assigning some of its vertices to the requesting node. This way the work is dynamically balanced between the computing nodes. A computing node terminates when it does not receive work from any of the processes.

PSMCE [26], which is considered as the state of art parallel approach for MCE, is similar to the ELS algorithm given in Figure 3 but it uses initial ordering of the vertices and BK algorithm instead of TTT algorithm. It performs a more refined dynamic load balancing. In addition to sharing search trees between the computing nodes, portions of an individual search tree are also shared between the nodes. This




(a) Graphs from collection 1 (b) Graphs from collections 2 (c) Graphs from collections 3 (d) Graphs from collections 4

Figure 7. Scaling on multicore architecture

is done by using a special representation called candidate path structure. Each search tree node is represented using a candidate path structure and the candidate path structures are stored in a queue. When a computing node receives a work request it shares some of the candidate path structures from the queue with the requesting process. Since PSMCE does not use any sort of reordering of vertices, the time taken by individual vertices varies widely and so a more refined dynamic load balancing is required. By using the degeneracy ordering *pbitMCE* is able to avoid such refined level of load balancing and is also able to achieve good scaleup.

## IV. Experimental Results

We implemented all our algorithms in C language. The results for sequential algorithms including that of ELS, TTT were obtained on Intel Xeon X5550 machine. The programs were compiled using gcc and O3 optimization flag is used for *pbitMCE*.

We performed our experiments on both real world networks and random graphs from four different collections. For comparison we have used the graphs used in [22] and [15]. Out of the dataset used in [22] we have selected only large graphs for which Eppstein's approach took more than 10 sec. For first collection, we have selected three graphs wiki-Talk, cit-Patents and web-BerkStan from the Stanford large network data collection [32]. The second collection contains two graphs: pattern1 and copapers which are obtained from Univeristy of Florida sparse matrix collection [33]. We have used five random graphs random-n-p also used in [22] where $n$ is the number of vertices and $p$ is the edge density. These graphs are part of third collection. The last collection consists of two graphs: phat300-2 which is a graph from the collection of DIMACS benchmark graphs [35] and mm51 which is a Moon-Moser graph with 51 vertices [36]. Table I gives the details of the graphs, $a$, $d$, $|C|$ and $M$ in the table denotes the average degree, degeneracy, the number of maximal cliques and maximum degree of the graph respectively.

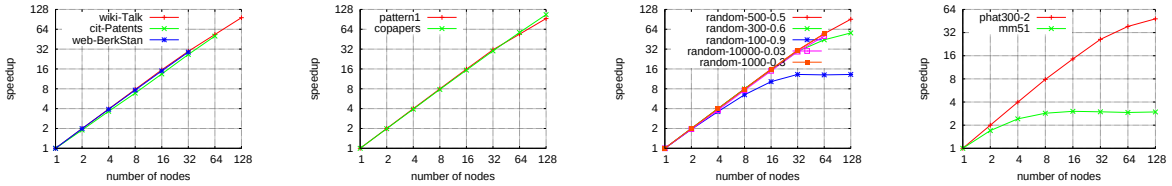
We compare our sequential algorithm with that of Eppstein et al.'s ELS algorithm and Tomita et el's TTT algorithm. The code for both the algorithms is obtained from [37]. Table II compares *pbitMCE* with three approaches TTT, hybrid and ELS. TTT is the Tomita et al.'s approach, hybrid and ELS are the Eppstein et al.'s approach without using the subgraphs and with using the subgraphs respectively. As adjacency matrix version of TTT is not applicable to the graphs used in the experiments, adjacency list version is used. The time for all three approaches does not include the time taken to build the initial adjacency list. It can be seen from the table that *pbitMCE* outperforms or close to all the other algorithms.

Table II
Sequential Algorithm Comparison

| dataset | TTT | hybrid | ELS | *pbitMCE* |
|---|---|---|---|---|
| wiki-Talk | >60 mins | 344.8 | 126.62 | 36.5 |
| cit-Patents | 18.46 | 32.41 | 38.77 | 21.84 |
| copapers | 31.5 | 24.01 | 26.25 | 18.44 |
| pattern1 | 19.66 | 21.83 | 41.66 | 28.27 |
| phat300-2 | 680.04 | 242.37 | 95.11 | 23.13 |
| web-BerkStan | 56.25 | 18.83 | 13.97 | 6.01 |
| mm51 | 81.22 | 123.63 | 35.89 | 9.62 |
| randaom-10000-0.03 | 13.33 | 75.02 | 13.01 | 4.7 |
| random-1000-0.3 | 206.64 | 177.52 | 41.95 | 12.92 |
| random-500-0.5 | 1533.7 | 1052.25 | 242.02 | 72.9 |
| random-300-0.6 | 753.53 | 522.24 | 142.93 | 41.16 |
| random-100-0.9 | 778.47 | 691.56 | 236.65 | 55.43 |

### A. Results for parallel implementation

For shared memory experiments we used a multicore machine with four Intel Xeon 7560 processors having 8 cores each adding up to 32 cores. The implementation is done in C programming language using OpenMP. We have compiled the code with intel compiler and set the processor affinity to compact. For experiments on distributed memory we used a cluster consisting of 32 nodes, each with Intel Xeon E5504 processor. MPI is used for inter process communication and the code is compiled using mpicc. The scalability results for all the experiments on multicore architecture and distributed memory architecture are shown in Figures III-F and III-G respectively. Note that these results do not include the time taken for preprocessing, i.e degeneracy ordering and partitioning adjacency list to pre and post adjacency lists. This is because preprocessing is done sequentially. As it can be seen *pbitMCE* scales upto 106 times using 128 process on distributed memory and upto 31 times using 32 cores on a multicore architecture. As seen in Figure III-F.7d and Figure III-G.8d that *pbitMCE* does not scale beyond 3 for mm51 graph. The minimum time that *pbitMCE* can take to enumerate all the cliques




(a) Graphs from collections 1 (b) Graphs from collections 2 (c) Graphs from collections 3 (d) Graphs from collections 4

Figure 8. Scaling on distributed memory architecture

Table I
GRAPHS

| dataset | $\|V\|$ | $\|E\|$ | $a$ | $\|C\|$ | $d$ | $M$ |
|---|---|---|---|---|---|---|
| pattern1 | 19242 | 4652095 | 241.7 | 17732 | 754 | 6029 |
| copapers | 540486 | 15245729 | 28.2 | 139340 | 336 | 3299 |
| wiki-Talk | 2394385 | 4659565 | 1.9 | 86333306 | 131 | 100029 |
| cit-Patents | 3774768 | 16518947 | 4.4 | 14787032 | 64 | 793 |
| web-Berksten | 685231 | 6649470 | 9.7 | 3405813 | 201 | 84230 |
| phat300-2 | 300 | 21928 | 73.1 | 79917408 | 98 | 229 |
| mm51 | 51 | 1224 | 24 | 129140163 | 48 | 48 |
| randaom-10000-0.03 | 10000 | 1499357 | 149.9 | 3733699 | 262 | 360 |
| random-1000-0.3 | 1000 | 149998 | 149.998 | 15671489 | 266 | 349 |
| random-500-0.5 | 500 | 62571 | 125.1 | 103686974 | 225 | 291 |
| random-300-0.6 | 300 | 27013 | 90.1 | 72454791 | 162 | 204 |
| random-100-0.9 | 100 | 4473 | 44.73 | 240998654 | 81 | 97 |

is $max(T_i)$ for $0 \leq i \leq |V|$, where $T_i$ is the time taken for a vertex $i$. *pbitMCE* cannot scale after achieving the minimum time. To achieve further scaleup, a more refined level of load balancing is required, like in PSMCE. However, mm51 is a Moon-Moser graph. In reality such type of graphs are highly unlikely to occur. Noting that the sequential version of *pbitMCE* outperforms or close to all the known sequential algorithms, the scalability results of *pbitMCE* are more impressive.

## REFERENCES


[1] Y. Chen and G. M. Crippen, "A novel approach to structural alignment using realistic structural and environmental information," *Protein Science*, vol. 14, no. 12, p. 29352946, 2005.

[2] B. Zhang, B.-H. Park, T. V. Karpinets, and N. F. Samatova, "From pull-down data to protein interaction networks and complexes with biological relevance," *Bioinformatics*, vol. 24, no. 7, pp. 979–986, 2008.

[3] K. L. Jensen, M. P. Styczynski, I. Rigoutsos, and G. Stephanopoulos, "A generic motif discovery algorithm for sequential data." *Bioinformatics*, vol. 22, no. 1, pp. 21–28, 2006.

[4] X. Yang, J. Zola, and S. Aluru, "Large-scale metagenomic clustering on map-reduce clusters," *Journal of Bioinformatics and Computational Biology*, vol. 11, no. 1, p. 1340001, 2013.

[5] M. C. Schmidt, A. M. Rocha, K. Padmanabhan, Z. Chen, K. Scott, J. R. Mihelcic, and N. F. Samatova, "Efficient alpha, beta-motif finder for identification of phenotype-related functional modules." *BMC Bioinformatics*, vol. 12, p. 440, 2011.

[6] Z. Chen, W. Hendrix, and N. F. Samatova, "Community-based anomaly detection in evolutionary networks," *J. Intell. Inf. Syst.*, vol. 39, no. 1, pp. 59–85, 2012.

[7] Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova, "Detecting and tracking community dynamics in evolutionary networks," in *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ser. ICDMW '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 318–327. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2010.32

[8] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu, "Community detection in large-scale social networks," in *WebKDD/SNA-KDD '07: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*. New York, NY, USA: ACM, 2007, pp. 16–25.

[9] N. Berry, T. Ko, T. Moy, J. Smrcka, J. Turnley, and B. Wu, "Emergent Clique Formation in Terrorist Recruitmen." National Conference on Artificial Intelligence, 2004. [Online]. Available: http://www.cs.uu.nl/people/virginia/aotp/papers/seldon-aaai-final.pdf

[10] E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Generating all maximal independent sets: NP-hardness and polynomial-time algorithms." *SIAM J. Comput.*, vol. 9, no. 3, pp. 558–565, 1980.

[11] F. Harary and I. C. Ross, "A procedure for clique detection using the group matrix," *Sociometry*, vol. 20, pp. 205–215, 1957.

[12] E. A. Akkoyunlu, "The enumeration of maximal cliques of large graphs," *SIAM J. Comput.*, vol. 2, no. 1, pp. 1–6, 1973.

[13] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, Sep. 1973. [Online]. Available: http://doi.acm.org/10.1145/362342.362367

[14] H. C. Johnston, "Cliques of a graph-variations on the Bron-Kerbosch algorithm," *International Journal of Parallel Programming*, vol. 5, no. 3, pp. 209–238, Sep. 1976. [Online]. Available: http://dx.doi.org/10.1007/bf00991836

[15] E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theor. Comput. Sci.*, vol. 363, no. 1, pp. 28–42, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2006.06.015

[16] I. Koch, "Enumerating all connected maximal common subgraphs in two graphs," *Theor. Comput. Sci.*, vol. 250, no. 1-2, pp. 1–30, Jan. 2001. [Online]. Available: http://dx.doi.org/10.1016/S0304-3975(00)00286-3

[17] F. Cazals and C. Karande, "Note: A note on the problem of reporting maximal cliques," *Theor. Comput. Sci.*, vol. 407, no. 1-3, pp. 564–568, Nov. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2008.05.010

[18] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM J. Comput.*, vol. 6, no. 3, pp. 505–517, 1977.

[19] K. Makino and T. Uno, "New algorithms for enumerating all maximal cliques," in *SWAT*, 2004, pp. 260–272.

[20] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, Feb. 1985. [Online]. Available: http://dx.doi.org/10.1137/0214017

[21] D. Eppstein, M. Lffler, and D. Strash, "Listing all maximal cliques in sparse graphs in near-optimal time," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6506, pp. 403–414.

[22] D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," in *Proceedings of the 10th international conference on Experimental algorithms*, ser. SEA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 364–375. [Online]. Available: http://dl.acm.org/citation.cfm?id=2008623.2008656

[23] Y. Zhang, F. N. Abu-Khzam, N. E. Baldwin, E. J. Chesler, M. A. Langston, and N. F. Samatova, "Genome-scale computational approaches to memory-intensive applications in systems biology." in *SC*. IEEE Computer Society, 2005, p. 12.

[24] F. Kose, W. a. Weckwerth, T. Linke, and O. Fiehn, "Visualizing plant metabolomic correlation networks using clique-metabolite matrices." *Bioinformatics*, vol. 17, no. 12, pp. 1198–1208, 2001.

[25] N. Du, B. Wu, L. Xu, B. Wang, and P. Xin, "Parallel algorithm for enumerating maximal cliques in complex network," in *Mining Complex Data*, 2009, pp. 207–221.

[26] M. C. Schmidt, N. F. Samatova, K. Thomas, and B.-H. Park, "A scalable, parallel algorithm for maximal clique enumeration," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 417 – 428, 2009.

[27] L. Lu, Y. Gu, and R. L. Grossman, "dmaximalcliques: A distributed algorithm for enumerating all maximal cliques and maximal clique distribution." in *ICDM Workshops*, W. Fan, W. Hsu, G. I. Webb, B. L. 0001, C. Zhang, D. Gunopulos, and X. Wu, Eds. IEEE Computer Society, 2010, pp. 1320–1327.

[28] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 1240–1248. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339724

[29] B. Wu, S. Yang, H. Zhao, and B. Wang, "A distributed algorithm to enumerate all maximal cliques in mapreduce," in *Proceedings of the 2009 Fourth International Conference on Frontier of Computer Science and Technology*, ser. FCST '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 45–51. [Online]. Available: http://dx.doi.org/10.1109/FCST.2009.30

[30] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by h*-graph." in *SIGMOD Conference*, A. K. Elmagarmid and D. Agrawal, Eds. ACM, 2010, pp. 447–458.

[31] ——, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043652.2043654

[32] J. Leskovec, "Stanford Large Network Dataset Collection." [Online]. Available: http://snap.stanford.edu/data/

[33] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[34] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," 2003, cite arxiv:cs/0310049. [Online]. Available: http://arxiv.org/abs/cs/0310049

[35] D. J. Johnson and M. A. Trick, Eds., *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993*. Boston, MA, USA: American Mathematical Society, 1996.

[36] J. Moon and L. Moser, "On cliques in graphs," *Israel Journal of Mathematics*, vol. 3, no. 1, pp. 23–28, 1965. [Online]. Available: http://dx.doi.org/10.1007/BF02760024

[37] D. Strash, "Quick cliques: A package to efficiently compute all maximal cliques in sparse graphs." [Online]. Available: http://www.dcs.gla.ac.uk/ pat/jchoco/clique/enumeration/quick-cliques/doc/index.html