List all Maximal Cliques of an Undirected Graph: A Parallel Algorithm

Xin hua1, Maosheng Zhong1,*, Qun Liu2 and Mingwen Wang1
1School of Compute and Information Engineering, Jiangxi Normal University, Nanchang, 330022, China.
2School of Computing, Dublin City University, Dublin 9, Ireland

*Corresponding author

Abstract. Listing all the maximal cliques of an undirected graph is considered as a NP-complete problem, even if the existing fastest algorithm for listing all the maximal cliques also needs an exponential time complexity, and these algorithms are not able to be reconstructed into parallel processing algorithms, therefore, which are also unsuited to solve all maximal cliques of a complex undirected graph with a large number of maximal cliques or vertices. This paper aims to develop a parallel algorithm for listing all maximal cliques of the large size complex undirected graph. Firstly we gave some definitions, including adjacent sub-graph, suspended sub-graph and so on, then adopted a partition and recursive strategy to partition equivalently a parent-graph into smaller sub-graphs, so we can get all maximal cliques of parent-graph by recursively listing all maximal cliques of sub-graphs. we have developed a single-thread and a multi-thread program of the algorithm by using Java language. Lastly, we compared the performance of our algorithm in a benchmark data set DIMACS with the existing main algorithms. The results show that our single-thread program is basically equivalent performance with existing algorithms, but the multi-thread program has better performance than the existing algorithms. Because our method is to divide the complex problem into some smaller sub-problems and achieve the solution of the original problem by solving these sub-problems, our algorithm is easy to be implemented by using multi-thread, and also easy to be expanded to the parallel algorithm, which has better feasibility and practicability in solving the large scale complex graph.

Keywords: Listing all maximal cliques; Partition and recursive strategy; Adjacent sub-graph; Suspended sub-graph; Parallel algorithm; graph theory.

1. Introduction
A clique is a subset of vertices of an undirected graph, such that its induced sub-graph is complete, that is, every two distinct vertices in the clique are adjacent. A maximal clique is a clique to which no more vertices can be added, and a maximum clique is a clique that includes the largest possible number of vertices of the graph. Obviously, a maximum clique is also a maximal clique, not vice versa.

The clique problem arises from the real-world setting. For example, in the social network, if a vertex represents a person and the edge represents mutual acquaintance, then to find a maximal subset of persons who all know each other is a problem of finding maximal cliques in a graph [1,2]; Another example, in the bioinformatics, is to find all common sub-topologies in a set of protein structures or to search the frequent occurred structural patterns from protein structures to predict protein structure from
its molecular sequence, which also is a practical application of clique problem [3,4]. Of course, there are many other applications, such as information retrieval [5,6], data mining [7], and so on. Although the clique has been applied widely, to compute all maximal cliques of a graph is considered as a NP-complete problem [8]. According to MOON & MOSER's proof, the maximum number of maximal clique in an undirected graph with n vertices possible reach an exponential time of the vertex number n [9]. Therefore, in theory, the computing time for solving all maximal cliques also need exponential times of n. It is for this reason that many researchers turn to study the searching algorithm for finding the maximum clique. But the finds suggest that if we didn't use the parallel searching algorithm, the time complexity of the existing fastest algorithm to search the maximum clique is proportional to $O(2^{n/2})$ [10]. And when a heuristic algorithm or intelligent algorithm was used to search the maximum clique, the result may be an approximation, not the optimal value.

In many practical applications, we may not be concerned about what is the maximum clique in the graph, but more hope to solve all maximal cliques of the graph, for example, the task of recognizing all communities from the social networks means searching maximal cliques from a graph corresponding to this networks. However, because the number of the maximal clique in an undirected graph maybe reach the exponential times of the vertices number n, the computing time for solving all maximal cliques is also an exponential time. In this paper, we carried out an experiment in the computing environment with Inter(R) Core(TM) i5-4200U CPU@1.60GHz and 8.00G memory by using the Tomita algorithm, which is considered as the existing fastest algorithm to compute all maximal cliques of an undirected graph and can be downloaded by directly from the http://www.ics.uci.edu/~dstrash/quick-cliques.tar.gz, to list all maximal cliques of the brock400_1 that is a benchmark data of DIMACS and only contains with 400 vertices. The experiment result is that although the brock400_1 graph only has 400 vertices, there is 855398132259 maximal cliques and the total computing time is 1074307.85 seconds (about 12.43 days). The experiment shows that although Tomita algorithm is considered as an algorithm with the best performance at present, it appears powerless to be applied to the super large size graph, such as social networks. Therefore, we must further improve the algorithm or develop a parallel algorithm to compute all maximal cliques in a graph.

Because the Tomita algorithm, when it executes a current search step, always need trace back the previous execution steps and remove some vertices, which has been used to prevented from generating the subset of a maximal cliques, it is difficult to be updated to a parallel algorithm. In this paper, aiming at developing a parallable algorithm for listing all the maximal cliques of the large size and complex undirected graph, we applied the partition & recursive strategy and partitioned equivalently the parent-graph into smaller sub-graph by giving some new definitions, including adjacent sub-graph and suspended sub-graph, and then get all the maximal cliques of the parent-graph by computing all maximal cliques of the sub-graph.

We had developed a single-thread and a multi-thread program of the algorithm by using Java language. Lastly, we compared the performance of our algorithm in a benchmark data set DIMACS with the existing main algorithms. The results show that our single-thread program is basically equivalent performance with existing algorithms, but the multi-thread program has better performance than the existing algorithms.

Because our algorithm is to divide the complex problem (large size graph) into some smaller sub-problems (sub-graphs) and achieve the solution of the original problem by solving the sub-problems, it is easy to be implemented by using the multi-thread, and obviously, it is easy to be expanded to a parallel algorithm and has a good practicability to list all maximal cliques of the large size graph or networks. The rest of this paper is organized as follows. In section 2, we described some related research of clique problems. In section 3 and section 4, we present some definitions, designed the corresponding algorithms, and described its implementation and some experimental results.

2. Related Work
In 1949, when Lucy & Perry researched the application of the matrix analysis method to the group structure, they firstly introduced the terminology—Clique, and since then many researchers began to explore the clique problem. At present, there are four main research topics on the clique problem,
namely, listing all maximal cliques, finding a maximum clique, finding a maximum weight clique in a weighted graph and solving the decision problem of testing whether a graph contains a clique larger than given size. Essentially, all these topics can come down to solving all the maximal cliques in an undirected graph. Because a n-vertex graph has at most $3^{n/3}$ maximal cliques, computing all maximal cliques in a graph is also considered as a NP-complete problem [8]. On the problem of finding the maximum clique in an undirected graph, Bomze, Pardalos and Xue have reported a detailed review[10,13]. Next, we will summarize the related work of listing all maximal cliques in an undirected graph. Bron & Kerbosch presented a depth-first search algorithm for generating all maximal cliques in a graph, and their experiment results show that the computing time for generating all maximal cliques is almost independent of the number of vertices of the graph and the time complexity of the algorithm is $O(3.14^{n/3})$ at Moser & Moon graph with n vertices [14]. Johnston [15] and Koch [3] improved Bron and Kerbosch's algorithm. Tsukiyama et al. designed an algorithm to generate the maximal independent subsets of a graph and its time complexity is $O(\mu m n)$ (where $\mu$, $m$, $n$ are the number of maximal independent subsets, edges and vertices in a graph respectively) [16], and Lawler et al. also obtained the same conclusion [17]. Based on Tsukiyama's algorithm, Chiba & Nishizeki proposed a new algorithm to generate all maximal cliques, the time complexity of which is $O(\mu a(G) m n)$ [18] (where $a(G)$ is the arboricith degree for a connected graph $G$, $a(G) \leq O(m^{3/2})$, $\mu$ is the number of the maximal clique), and Makino & Uno also presented an algorithm to solve all maximal cliques, the time complexity of which is $O(\Delta^2 n)$ [19] ($\Delta$ is the maximum degree of vertex of a graph). Tomita et al. proposed a depth-first search algorithm combined with the pruning technology to generate all maximal cliques of a graph, the worst-case time complexity of which is $O(3^{n/3})$ [11]. For the problem of Tomita's algorithm is slower than the theory speed on the sparse connected graph, David & Darren presented an algorithm to compute all maximal cliques on the sparse connected graph. At the same time, they found that the algorithm of Tomita used an adjacency matrix, which required too much space for large sparse graphs, so they implemented the Tomita's algorithm by using adjacency lists to store the sparse connected graph [20]. All above works show that the time complexity of all existing algorithms for generating all maximal cliques of an undirected graphs are exponential time of vertices n. By comparing the experimental result reported from the relevant literature, Tomita's algorithm is the fastest at the same experimental data. Actually, it is difficult to essentially improve the time complexity of the algorithm because a n-vertex graph has at most $3^{n/3}$ maximal cliques. In this paper, we describe a parallable partition & recursive algorithm to list all maximal cliques in an graph and the time complexity of our algorithm is $O(\mu m n)$ (where $\mu$ is the number of the maximal clique). The O ($\mu m n$) shows that the time complexity of our algorithm is linear with the number of maximal clique in graphs with same n-vertices. Furthermore, because the strategy of our algorithm is to partition a large size and complex graph into some smaller size sub-graphs and to achieve the solution of parent-graph by computing all maximal cliques of the sub-graphs, it is easy to be implemented by using the multi-thread or to be expanded to a parallel algorithm so that it has a good practicability to list all maximal cliques of a large size graph or networks.

3. Basic Definitions

Given an undirected graph $G=\langle V,E \rangle$, where $V=\{1,2, \ldots, n\}$ is a non-empty finite set of vertices of $G$, and $E \subseteq V \times V$ is a set of edges (unordered pairs of vertices), where vertex $u$ is adjacent to vertex $v$ if and only if $(u,v)$ is in $E$, so, $(u,v)$ is also called a edge of $G$. If a subset of vertices $U \subseteq V$ and every pair of vertex in $U$ is adjacent in $G$, the induced sub-graph of the subset $U$ in $G$ is a complete sub-graph of $G$, denoted by $G_U$, and correspondingly the subset $U$ is called a clique. A maximal clique is a clique to which no more vertices can be added and a maximum clique is a clique that includes the largest possible number of vertices in $G$.

Next, we will introduce several new definitions as follows.

**Definition 1.** The set of adjacent vertices $U_{v_k}^+$ of vertex $v_k$ on graph $G$ is defined as a set, which contains vertex $v_k$ and vertices that are adjacent to $v_k$ of the graph $G$, namely,
\[ U^+_v = \{ v_j \mid (\forall v_j) ((v_j = v_k) \lor (v_j \in V \land (v_k, v_j) \in E)) \}. \]

**Definition 2.** The set of suspended vertices \( U^-_v \) of vertex \( v_k \) on graph \( G \) is defined as a set, which contains vertices \((V - U^+_v)\) that are not adjacent to \( v_k \) of the graph \( G \) and another vertices that are simultaneously adjacent to \( v_k \) and certain vertex of \( V - U^+_v \), that is, \( U^-_v = \{ v_j \mid (\forall v_j) \ (v_k, v_j) \notin E \lor (\forall v_j) (\exists v_p) \ (v_j \in U^+_v \land v_p \in V - U^+_v \land (v_j, v_p) \in E) \}. \)

**Definition 3.** The adjacent sub-graph \( G^+_\nu \) of the vertex \( v_k \) on graph \( G \) is defined as a projection sub-graph of the set of adjacent vertices \( U^+_\nu \) on graph \( G \).

**Definition 4.** The suspended sub-graph \( G^-_\nu \) of the vertex \( v_k \) on graph \( G \) is defined as a projection sub-graph of the set of suspended vertices \( U^-_\nu \) on graph \( G \).

According to the above definitions, we can draw a conclusion, that is, if an undirected graph \( G \) is partitioned an adjacent sub-graph \( G^+_\nu \) and a suspended sub-graph \( G^-_\nu \) by the vertex \( v_k \), Let \( C^\nu = \{ C_i \mid C_i \) is a maximal clique, which at most contain a vertex of \( V - U^-_\nu \) or \( V - U^+_\nu \}\), and \( C = \{ C_j \mid C_j \) is a maximal clique of the graph \( G \}\), then \( C^\nu = C \).

4. Algorithm Design and Experiment

**4.1 Main Idea**

According to the above conclusion, when an undirected graph \( G \) is partitioned into an adjacent sub-graph \( G^+_\nu \) and a suspended sub-graph \( G^-_\nu \) by the vertex \( v_k \) of \( G \), each maximal clique induced from \( G^+_\nu \) and \( G^-_\nu \), which included at least a vertex of \( V - U^-_\nu \) or \( V - U^+_\nu \), is also a maximal clique in \( G \). Therefore, we can computing all maximal cliques of \( G \) by using partition and recursion strategy, the main idea is that, if \( G \) is not a complete graph, then we choose a non-(n-1)degree vertex \( v_k \) partition \( G \) into an adjacent sub-graph \( G^+_\nu \) and a suspended sub-graph \( G^-_\nu \), then recursively compute all maximal cliques of \( G^+_\nu \) that at least include a vertex of \( V - U^-_\nu \) and all maximal cliques of \( G^-_\nu \) that at least include a vertex of \( V - U^+_\nu \).

**4.2. Algorithm of Listing all Maximal Cliques in an Undirected Graph**

According to the above idea of partition and recursive strategy, let the adjacent matrix \( M \) denote an undirected graph \( G (V, E) \), we can design the following algorithm 1 for listing all maximal cliques of \( G \).
Algorithm 1. Algorithm of Listing all Maximal Cliques.

Input: adjacent matrix $M$ and vertices set $V$ of $G$
Output: all maximal cliques $c_i$ of $G$.

Procedure: ListAllMaximalCliques($M, V$)

{ 
  if (Each element of $M$ is 1)
    Print the vertices set $V$
  else
    
      (1) Partition $G$ into $G_{U_+}$ and $G_{U_-}$ (namely, Partition $V$ into and $V_{U_+}$
      (2) ListAllMaximalCliques ($M_{U_+}, V_{U_+}$)
        //computing all maximal cliques of $G_{U_+}$
        //that at least include a vertex of $V - U_{U_+}$
      (3) ListAllMaximalCliques ($M_{U_-}, V_{U_-}$)
        //computing all maximal cliques of $G_{U_-}$
        //that at least include a vertex of $V - U_{U_-}$


Take the undirected graph shown in Fig.1 for example to demonstrate the process of the algorithm described above, which can be shown by the binary tree as the Fig.2 (the rectangular block in Fig.2 can be regarded as a node of the binary tree)

![Figure 1](example.png)

**Figure 1.** An undirected graph - an example.
(In the figure, $U^+$ represents the set of the adjacent vertices and $U^-$ represents the set of the suspended vertices; $\blacktriangleleft$ represents that the corresponding vertices set is a maximal clique and $\triangledown$ represents that the vertices set is not a maximal clique)

Figure 2. Running process of algorithm 1 in the graph shown in fig.1.

4.3. Algorithm Implementation and Experiment

In order to further verify the performance of our algorithm, we have implemented the single-thread version (Partition-Recursion-List-Maximal-Clique-Single-Thread, abbreviated as PR-LMC-ST) and multi-thread version (Partition-Recursion-List-Maximal-Clique-Multi-Thread, abbreviated as PR-LMC-MT) for our algorithm to list all maximal cliques in an undirected graph by using Java programming language in this paper.

In this paper, we carry out comparative experiments between the above algorithm program and the program given by [20], which can be downloaded by http://www.ics.uci.edu/~dstrash/quick-cliques.tar.gz, in a medium scale data of benchmark data set DIMACS at Inter (R) Core (TM) i5-4200U CPU @ 1.60GHz, 8.00G memory computing environment.

The literature [20] has given four comparative programs, including two algorithm program codes designed by Tomita et al. [11], that is, tomita and maxdegree, which are implemented respectively by using adjacent matrix and adjacent list, and two algorithm program codes are proposed by Eppstein et al. [20,21], that is, hybrid and degen. The following are experiment results of all above program codes in the benchmark data set DIMACS.

| (a) | (b) |
| --- | --- |

Figure 3. Relationship between the CPU computing time for listing maximal cliques and the number of maximal cliques.
Firstly, we have run the above 6 program codes in 6 graphs, that is brock200_2, brock200_3, brock200_4, c-fat200-1, c-fat200-2 and c-fat200-5, all of which have 200 vertices from benchmark data set DIMACS but different number of maximal cliques, and the purpose is to study the relationship between the CPU computing time for listing all maximal cliques and the number of maximal cliques of the undirected graph with the same number of vertices. Then, we also have investigated the relationship between the CPU computing time for listing all maximal cliques and the number of maximal cliques of the undirected graph by using some undirected graphs with different number of vertices. Figure 3(a) shows the relationship between the CPU computing time and the number of maximal clique of the undirected graph with same number of vertices of all above algorithm program, and Figure 3(b) shows the relationship between CPU computing time and the number of maximal clique of the undirected graph with different number of vertices.

As we can see from Figure 3(a), 6 curves corresponding to 6 programs are basically close to the straight line, which indicates that the relationship between the computing time and the number of maximal cliques in 6 algorithms are basically linear when these graphs have same number of vertices. In Figure 3(b), 6 curves corresponding to 6 programs also are basically straight line, but irregular because of the influence of the number of vertices. Moreover, it can be seen from Figure 3(a), the lines corresponding to 6 algorithm programs have different slope, which indicates that different algorithms need consume different time for computing same number of maximal cliques, namely, the slope can represent performance of different algorithms, and the lower the slope is, the higher the performance of the algorithm is. According to Figure 3(a), algorithm PR-LMC-MT, which is implemented by multi-thread, has better performance than Tomita.

Finally, we show the experiment results of different algorithm programs in medium or small size undirected graphs of the DIMACS data set, as shown in Table 1. In Table 1, the first four columns of each row are respectively, data for experiment (an undirected graph), number of vertices, number of edges and number of maximal cliques in this graph, and the following six columns are the experiment results, namely, CPU time (in seconds), of the above 6 algorithm programs for listing all maximal cliques in current graph. As can be seen from Table 1, PR-LMC-MT is faster than existing algorithms in all of data which contain a large number of maximal cliques, such as brock200_2, brock200_3, brock200_4, hamming6-2, johnson16-2-4, keller4 and so on.

Table 1. CPU time of different algorithm programs to list all maximal cliques in the DIMACS data

| Data for experiment | n    | e    | \(\mu\) (\#cliques) | tomita | max degree | hybrid | degen | PR-LMC-ST | PR-LMC-MT |
|--------------------|------|------|----------------------|--------|------------|--------|-------|-----------|-----------|
| brock200_2         | 200  | 9876 | 431586               | 1.485  | 3.844      | 3.281  | 2.251 | 2.118     | 1.081     |
| brock200_3         | 200  | 1204 | 4595644              | 16.375 | 47.937     | 38.281 | 25.046| 31.832    | 16.358    |
| brock200_4         | 200  | 1308 | 19645556            | 78.859 | 213.499    | 171.7  | 103.2 | 148.376   | 75.562    |
| c-fat200-1         | 200  | 1534 | 37.0                | 0      | 0          | 0      | 0     | 0         | 0.008     |
| c-fat200-2         | 200  | 3235 | 18.0                | 0      | 0          | 0.015  | 0.015 | 0.001     | 0.004     |
| c-fat200-5         | 200  | 8473 | 7.0                 | 0      | 0          | 0.015  | 0.015 | 0.001     | 0.003     |
| c-fat500-1         | 500  | 4459 | 80.0                | 0      | 0          | 0.015  | 0.015 | 0.005     | 0.076     |
| c-fat500-2         | 500  | 9139 | 40.0                | 0.015  | 0.016      | 0.016  | 0      | 0.004     | 0.038     |
| hamming6-2         | 64   | 1824 | 1281402             | 4.884  | 6.39       | 6.531  | 4.953 | 10.554    | 4.836     |
| hamming6-4         | 64   | 704  | 464.0               | 0      | 0          | 0.015  | 0      | 0.001     | 0.009     |
| johnson16-2-4      | 120  | 5460 | 2027025             | 16.093 | 30.671     | 36.593 | 23.265| 18.775     | 10.359    |
| johnson8-2-4       | 28   | 210  | 105.0               | 0      | 0          | 0      | 0     | 0.001     | 0.004     |
| johnson8-4-4       | 70   | 1855 | 114690             | 0.328  | 0.578      | 0.656  | 0.562 | 0.458      | 0.302     |
| keller4            | 171  | 9435 | 10284321            | 19.39  | 33.922     | 34.764 | 25    | 29.197     | 12.807    |
| MANN a9            | 45   | 918  | 590887              | 1.443  | 1.766      | 1.765  | 1.546 | 1.941      | 1.408     |

Generally, algorithm Tomita has a good performance in the small size graph, but the algorithm Tomita cannot be implemented by multi-thread and also cannot be updated to a parallel algorithm because of the
execution characteristics of itself. So, it is difficult to be applied to the large size graph, such as large scale social networks. The performance of algorithm ListAllMaximalCliques proposed in this paper is basically between that of existing algorithms. However, the strategy of algorithm ListAllMaximalCliques is to partition “equivalently” a large size and complex graph into some smaller size sub-graphs and to get all maximal cliques of parent-graph by computing all maximal cliques of the sub-graphs. Therefore, our algorithm ListAllMaximalCliques is a parallable algorithm, which provides a prerequisite for the community identification and network analysis in large scale social networks, so the algorithm ListAllMaximalCliques has a better practicability to list all maximal cliques of a large size graph or networks.

5. Conclusion
To list all maximal cliques in an undirected graph is considered as a NP-complete problem. The computing time of the existing algorithm to solve all maximal cliques of a graph is basically linear relationship with the number of maximal clique in this graph. However, a n-vertex graph maybe has at most $3^{n/3}$ maximal cliques, so the time complexity of the existing algorithm to compute all maximal cliques is also an exponential time of number of vertex n, which is only adapt to the undirected graph with a few maximal cliques or a few vertices, and when there are a large number of maximal cliques in a large size and complexly undirected graph, it appears to be powerless even if it is the existing fastest algorithm. In this paper, aiming at developing a parallable algorithm for computing all maximal cliques of the large size and complex undirected graph, we present a partition and recursive algorithm to compute all maximal cliques of a graph by adopting the partition and recursive strategy and describing some new definitions, such as adjacent sub-graph, suspended sub-graph, and so on, to partition equivalently a larger size and complex parent-graph into several smaller size sub-graphs. The idea of our algorithm is to partition a complex problem into some smaller sub-problems and to get the solution of the complex problem by solving these sub-problems, therefore, it is easy to be implemented by multi-thread, and also is easy to be expanded for a parallel algorithm, which has better feasibility and practicability to computing all maximal cliques of a large size and complex graph.

Acknowledgement
This work was supported by the National Natural Science Foundation of China (Grant No. 61877031, 61462027 and 61462045) and the Science and Technology Project of Jiangxi Province Education Department(GJJ170206).

References
[1] F. Harary and I. C. Ross. A procedure for clique detection using the group matrix, Sociometry (American Sociological Association), 20 (3): 205–215, 1957.
[2] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA 99, 7821 – 7826, 2002.
[3] I. Koch. Enumerating all connected maximal common subgraphs in two graphs. Theor. Comput. Sci. 250(1-2), 1–30,2001.
[4] I. Koch, T. Lengauer and E. Wanke. An algorithm for finding maximal common subtopologies in a set of protein structures. J. Comput. Biol. 3(2), 289 – 306 ,1996
[5] K. Raghunath and K.D.Susant. A Study on High Dimensional Clustering by Using Clique, International Journal of Computer Science & Informatics, 1(2): 22-25, 2011
[6] M.C.Paull and S.H.Unger. Minimizing the number of states in incompletely specified sequential switching functions, IRE Trans. Electr. Compute, Vol. EC-8:356-367, 1959.
[7] R.E.Bonner. On some clustering techniques, IBM J. Res. Develop., 1964. vol.8:22-32.
[8] M. K. Richard. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum. pp. 85–103, 1972.
[9] J.W. Moon and L. Moser. On cliques in graphs. Israel Journal of Mathematics, 1965, 3(1):23-28
[10] I. M. Bomze, M. Budinich, P.M. Pardalos and M. Pelillo. The maximum clique problem, Handbook of Combinatorial Optimization 4, Kluwer Academic Publishers, pp. 1–74, 1999
[11] E. Tomita, A. Tanaka and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science, 363:28-42, 2006.

[12] R.D. Luce, A.D. Perry. A method of matrix analysis of group structure, Psychometrika, 14 (2): 95–116, 1949.

[13] P.M. Pardalos, J. Xue, The maximum clique problem, J. Global Optim. 4: 301–328, 1994.

[14] C. Bron and J. Kerbosch. Algorithm 457, finding all cliques of an undirected graph, Comm. ACM, 16: 575–577, 1973.

[15] H.C. Johnston. Cliques of a graph—variations on the Bron–Kerbosch algorithm, Int. J. Comput. Inform. Sci. 5: 209–238, 1976.

[16] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa. A new algorithm for generating all the maximal independent sets, SIAM J. Comput. 6: 505–517, 1977.

[17] E.L. Lawler, J.K. Lenstra and K.A.H.G. Rinnooy, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, SIAM J. Comput. 9: 558–565, 1980.

[18] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms, SIAM J. Comput. 14:210–223, 1985.

[19] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques, SWAT 2004, Lecture Notes in Computer Science, Vol. 3111:260–272, 2004.

[20] D. Eppstein and D. Strash. Listing All Maximal Cliques in Large Sparse Real-World Graphs, Proceedings of the 10th International Conference on Experimental Algorithms (SEA 2011), LNCS vol. 6630, pp. 403-414.

[21] D. Eppstein, M. L’offler and D. Strash. Listing all maximal cliques in sparse graphs in near optimal time. In: Cheong, O., Chwa, K.-Y., Park, K. (eds.) ISAAC 2010. LNCS, vol. 6506, pp. 403–414. Springer, Heidelberg (2010)