Sindh Univ. Res. Jour. (Sci. Ser.) Vol.44 (2) 313-3118 (2012)

SINDH UNIVERSITY RESEARCH JOURNAL (SCIENCE SERIES)

Cliq: A Clique Finding Algorithm

Z.U.A.KHUHRO, F. N. MEMON, M.U. R. MAREE, A.P. HARRISON*

Institute of Mathematics and Computer Science, University of Sindh, Jamshoro
Corresponding author, Zain-ul-Abdin Khuhro, email: kzainkhuhro@yahoo.com Ph. No.

Received 16th May 2012 and Revised 30th June 2012

Abstract: The maximum clique problem is very important in bioinformatics, computer science and other fields. An algorithm has been designed for finding a maximum clique in a graph of any size. The algorithm is based on recursion and back-tracking techniques. This paper describes the algorithm, an example and some results. The algorithm is tested on DIMACS graphs and its performance has also been compared with an existing algorithm, the Cliquer.
Keywords: Clique problem, graphs, recursion and back-tracking.

1. INTRODUCTION

Comparing two graphs is a fundamental task in pattern recognition and has applications in many fields. Several techniques (Bron, 1973; Ostergard, 2002; Niskanen, 2010) have been proposed which transform the graph matching to an equivalent clique search problem. The clique finding problem can be defined in two ways. The first one is maximal cliques, where vertices are not a subset of the vertices of a larger clique. The second is maximum clique which defines the largest among all cliques. This paper focuses on the maximum clique problem.

Techniques which find out the maximal cliques or a maximum clique in undirected graphs typically begin at a root node and explore all neighbouring nodes. From each of the neighbouring nodes, the techniques explore their unexplored nodes and so on, until they find a clique.

In this paper a new method CLIQ for finding the cliques is developed. CLIQ has also been tested on well-known DIMACS graphs (Ostergard, 2002) and has also been compared with the Cliquer method.

The Bron-Kerbosh algorithm

This and the next two subsections are summarised from Bron, 1973. The clique detection technique described by Bron-Kerbosh (BK) requires that the input graph to be searched is in the form of a symmetrical square Boolean matrix of size J x J, where J is the number of nodes in the matrix. The diagonal elements of this matrix must be TRUE and the other elements of this matrix may be TRUE or FALSE depending upon the nodes, whether they are connected or not. The Bron-Kerbosh algorithm finds only the largest possible cliques in the matrix whose elements are TRUE, and so is the 'maximal common subgraph isomorphism algorithm'. This method aims to expand and examine all the nodes of a graph by systematically searching through every node. Nodes that have not yet been examined for their neighbours are placed in some sets for later examination.

The careful choice of pivot (the first vertex to select) is important for the efficiency of the algorithm. The pivoting strategy helps to control the number of recursive calls at each step of the algorithm. The vertices will be added to a particular clique by looking at their degrees first try the vertex with the smallest degree, secondly try the vertex whose degree in the remaining subgraph is smallest and continue trying until the vertex with smallest degree is within the subgraphs.

The recursion and back-tracking are two main features of the BK algorithm which keep the record of every node which is already connected to a growing clique. A node is recursively selected by its parent node and added to the growing clique if it is also an adjacent node to all the nodes that are already inserted in the growing clique. The location of each node is recorded in order to see whether a further node should be added. Such a record helps in back tracking the status of nodes. The recursive process ends when a node has no adjacent nodes and at this point a new clique is completed and recorded.

Without pivoting

The BK algorithm uses a depth-first search, back-tracking tree search with search ordering and

*Departments of Mathematical Sciences and Biological Sciences, University of Essex, UK

branch & bound procedures. The workings of Bron Kerbosh algorithm is further explained in **(Table 1)**. The algorithm maintains three disjoint sets of nodes R, P, and X. Set R represents the currently growing clique, set P represents prospective nodes which are connected to all nodes in R. The given information within R, P can be expanded; set X contains nodes already processed, i.e., nodes which are previously in P and hence all maximal cliques containing the processed nodes have already been reported.

	K_1	K_2	K_3	K_4	K_5
K_1	1	1	1	1	0
K_2	1	1	0	1	1
K_3	1	0	1	1	1
K_4	1	1	1	1	0
K_5	0	1	1	0	1

(a)

K	R_i	X_i	P_i	Action
1	1	{}	2 3 4	
2	1 2	{}	4	
3	1 2 4	{}	{}	clique
2	1 2	4	{}	
1	1	2	3 4	
2	1 3	{}	4	
3	1 3 4	{}	{}	clique
2	1 3	4	{}	
1	1	2 3	4	
1	2	1	4 5	
2	2 5	{}	{}	clique
1	2	1 5	4	
1	3	1	4 5	
2	3 5	{}	{}	clique
1	3	1 5	4	
1	4	1 2 3	{}	
1	5	2 3	{}	

(b)

Table 1: (a) An example of (Bron, 1973) matrix of order 5 x 5 and represented by K_1, K_2, K_3, K_4 and K_5 nodes. The elements of the matrix are given by 1's and 0's where 1 shows the connection among the nodes and 0 otherwise. **(b)** In this table K represents the number of nodes, R_i is the set of nodes in current growing cliques, P_i represents the prospective nodes which could be included in the set R_i. X_i represents the previously rejected nodes of R_i that are only connected to the nodes of R_i.

This means that all the nodes which are connected to every node in R is either in P or X. Thus, the purpose of the set X is to avoid the repetition of maximal cliques through the update of P. The algorithm also checks for X to be non-empty. The nodes in X may be added to R, in order to avoid reporting cliques or subgraphs of cliques that have been found previously.

In each recursive call the BK algorithm store the nodes in the two sets X_i, and P_i of the graph that are connected to every node of R_i. The algorithm then moves to next recursion by moving a candidate node from P_i to the trial set R_i, which then becomes R_{i+1}. The sets X_{i+1} and P_{i+1} are calculated from X_{i+1} and P_{i+1} by removing those nodes which are not connected to the candidate node. In the process of back-tracking the node added most recently to R_{i+1} is added to X_i and removed from P_i. The clique is shown when both X_i and P_i are empty. If only P_i is empty then R_i is a subset of a clique which is already reported.

With pivoting

The BK algorithm is inefficient in the case of unweighted graphs. Therefore, in the case of unweighted graphs, a variant has been introduced which involves a pivot vertex i_p that is chosen from P (Jain, 2011). Thus the pivot vertex i_p and vertices from P not adjacent to i_p are only considered in each recursive call of the BK algorithm. This pivot vertex saves time and allows the algorithm to backtrack more quickly in branches of the search that contain no maximal cliques.

An extension of the Bron-Kerbosh algorithm (Bron, 1973) for solving the maximum weight clique problem has been presented in Jain, 2011. This algorithm is applied to the graph matching problems in which graphs have continuous valued weights assigned to both vertices and edges.

Cliquer

This and the next subsections are summarised from Ostergard, 2002 and Niskanen, 2010. Cliquer is a set of C routines for finding cliques in an arbitrary graph. It can search for maximum cliques, maximum weight cliques or cliques whose size or weight is within a given range. Cliquer uses an exact branch-and-bound algorithm. The Cliquer method searches for maximum clique and maximum-weighted clique problems and its run time is exponential (Katharina, 2006).

Cliquer searching

Cliquer uses six functions for searching the cliques. Some of the functions return the largest weight that any clique in a graph has. Some of the functions return a single clique fulfilling weights requirements and some functions assume that all vertex weights are 1. These functions also decide how the cliques are stored and how the progress of the routine is reported. Additionally the Cliquer defines several ordering functions for selecting and storing cliques. It also defines some macros which check whether the specified condition is TRUE. If it is FALSE then the algorithm will be terminated.

Table 2 explains **(Fig. 1)** in detail. The numbers in Table 2 indicate the indices of the vertices when clique is called. The vertex that is chosen on any level is underlined.

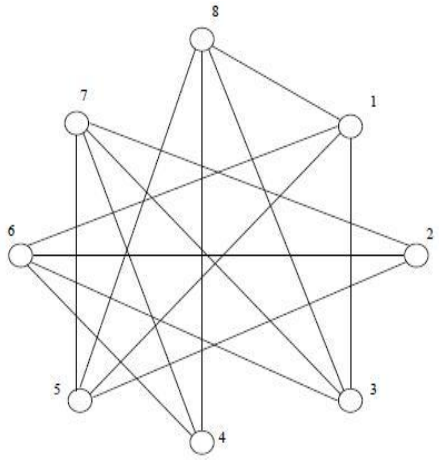

Fig. 1: An example of graph for understanding the steps of Cliquer algorithm where 8 nodes are given. Some of them are connected but others are not (Niskanen, 2010).

Table 2: Proceeding of Cliquer algorithm for example graphs (Niskanen, 2010) in Fig.1.

1: 8 (c(8) = 1; largest clique is {8})
1: 7 8
2: (empty; c(7) = 1)
1: 6 7 8
2: (empty; c(6) = 1)
1: 5 6 7 8
2: (empty; c(6) = 1)
1: 5 6 7 8
2: 7 8 (c(5) = 2; largest clique is {5, 7})
1: 4 5 6 7 8
2: 6 7 8 (prune; c(4) = 2)
1: 3 4 5 6 7 8
2: 6 7 8 (prune; c(3) = 2)
1: 2 3 4 5 6 7 8
2: 5 6 7
3: 7 (c(2) = 3; largest clique is {2, 5, 7})
1: 1 2 3 4 5 6 7 8
2: 3 5 6 8
3: 6 8
4: empty
2: 3 5 6 8 (prune; c(1) = 3)

In the Cliquer, values of a function c(i) are calculated which denotes the weight of the maximum clique in the subgraph induced by the vertices $\{v_1, v_2,..., v_n\}$. The Cliquer algorithm uses backtrack search. It searches for a clique of given size by considering each vertex starting from the last one and upto the first one, a backtrack search is carried out for the function c(i). Then it calculates c(i+1) (that is searching for a clique of size c(i)+1) and so on. Such values help to prune the search for maximum weight clique. The Cliquer algorithm's steps are shown in Table 2.

2. METHODS

The BK and Cliquer algorithms are widely used to find the maximum clique in a graph of any size. Before designing the proposed algorithm, Cliquer's approach was considered but it took so long to compute the solution. Thus in order to reduce the computational time the proposed algorithm CLIQ is developed. For reducing computational time the proposed algorithm works on strictly upper triangular matrix. The aim of this algorithm is to develop an efficient algorithm for solving the maximum clique problem and to show how the maximum clique finding algorithm can be applied to find the reasonable overlapping in biological structures.

The algorithm is simple and easy to implement which is shown in the experimental section for typical data sizes for well known graphs and random graphs. The CLIQ can also be applied in structural biological for pattern matching.

The presented algorithm CLIQ uses back tracking and recursive techniques. For this it uses an array where all possible edges of a node, to be searched for a clique, are stored in a row.

The strictly upper triangular matrix defines the data and its relationships with CLIQ. CLIQ requires that the graph is to be analysed to a strictly upper triangular matrix, or strictly lower triangular matrix. If a strictly lower triangular matrix is submitted then it is converted into a strictly upper triangular matrix by the CLIQ algorithm.

Recursion and back-tracking

Suppose a graph G has n nodes/vertices that are labelled as $\{v_1, v_2, v_3...vi,...,v_n\}$. The algorithm initially starts with the v_1 and one of its adjacent nodes. The v_1 and its adjacent nodes are stored in a growing clique one after another. Then an adjacent node of the last node in the growing clique (also called adjacent of the adjacent node of v_1) is selected in such a way that the newly selected node is also adjacent to all the nodes stored in the growing clique. This selection of adjacent to the adjacent node continues until a node without any adjacent node is found that means a new clique is completed. The recursion method is used to select the adjacent to the adjacent of a node and back tracking helps to maintain the recursion.

This algorithm also uses the branch and bound technique. At each step of the algorithm, if the v_i is not connected to its previous v_{i-1} node and there are no further neighbour nodes available then all the nodes less than v_i will form a clique. The algorithm also remembers these bounds, i.e. every clique. Thus for each starting node, all possible cliques are recorded but not stored in an array. This recording of all the cliques helps to find the maximum clique in the graph. Thus before back-tracking a new maximum clique will be compared with the previous one. Subsequently, the largest of all maximal cliques is the maximum clique of the graph.

Clique finding process

The CLIQ algorithm finds cliques from the square matrix M_{ij} where i,j={1,2,3,...n}. The rows and

columns of the M represent the nodes of the graph and each element of M represents a connection between the two nodes. The number of nodes of the graph is the order of the matrix. The matrix stores the edges with 0's and 1's as its elements. 0 shows the absence of an edge between node i and node j and 1 shows the presence of an edge between the two nodes.

CLIQ Algorithm

Suppose N contains all the nodes of a graph G. CLIQUE is a growing clique which builds up a clique node by node. Repeat the following procedure for every node of G that is stored in N.

1. Add an unprocessed node of N in an array CLIQUE
2. Store the last node of CLIQUE in TOP-NODE
3. If there is no any unprocessed adjacent node of TOP-NODE, a new clique is completed and available in CLIQUE. Then

a) Store the new clique in the MAX-CLIQUE if it is larger than the previously found largest clique

b) Remove the latest node of CLIQUE

Otherwise

a) add at the end of CLIQUE an unprocessed adjacent node of TOP-NODE that is also an adjacent node of every node stored in CLIQUE

4. Repeat steps 2 and 3 until CLIQUE is not empty.

Example for finding the maximum clique

In each row an adjacent node of the node N is stored in the matrix DATA. The number of adjacent nodes are then stored in ROW_LENGTHS [I]. For the next node of N, the DATA set will be checked whether a particular node of N is connected to previous nodes in DATA set. If the next node is available then search will go on and the DATA set will be checked to find every other node until the DATA set is empty. Here, the interest is to record only the maximum size of clique. Further explanation of these sets is given in **(Table 3),** which are extracted from **(Fig. 2).**

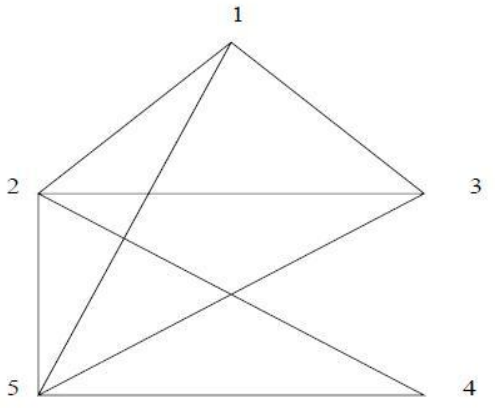

Fig. 2: An example of clique, where 1,2,3,5 are connected and form a clique of size 4.

Table 3: (a) An example of matrix M which is a strictly upper triangular matrix of 5 x 5 nodes where 1 shows the connection between nodes and 0 otherwise. b) In table, array N represents a node in matrix M and in array DATA. Each row of DATA contains all the adjacent nodes which are connected to the corresponding node of graph and an array ROW_LENGTHS stores total number of adjacent nodes of DATA in its corresponding row.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(a)

N	DATA	ROW_LENGTHS
1	2,3,5	**3**
1	3,5	2
2	3,5	2
2	4,5	2
3	5	1
4	5	1
5	0	0

(b)

3. RESULTS AND DISCUSSION

DIMACS data tests

The CLIQ algorithm has been tested on DIMACS data (Johnson, 1996). DIMACS graphs are special benchmark graphs in the second DIMACS Implementation Challenge and can be obtained electronically from Rutgers, 2011 and have been described in Hasselberg, 1993. The DIMACS graphs are used in many papers as benchmark graphs, such as (Balas, 1996; Kuznetsova, 2001; Ostergard, 2002; Tomita, 2003; Kumlandersimple, 2006).

Different types of DIMACS graphs have been used for testing (Table 4). These types of graphs are available at the website (Instances).

CFat (From Panos Pardalos pardalos@math.uflorida.edu). Graphs describe the fault diagnosis problems. The reference for this type of graphs is Berman and Pelc, (Berman, 1990).

Joh (From Panos Pardalos pardalos@math.uflorida.edu). Graphs define a coding theory. The Johnson graph is with n, w, d descriptions that have a node for every binary vector of length n and defined with exactly w 1s. Vertices are adjacent if and only if their Hamming distance is at least d. Clique is represented by a feasible set of vectors for each code.

Kel (From Peter Shor shor@research.att.com). Graphs are defined for Keller's conjecture on tilings using hypercubes (Lagarias, 1992).

Ham (From Panos Pardalos pardalos@math.uflorida.edu) this is also a coding theory graph. This type of graph is also defined with n and d has a node for each binary vector of a certain length n. The proposed two nodes are adjacent if and only if the corresponding bit vectors are having at least d distance which is a Hamming distance.

San (From Laura Sanchis laura@cs.colgate.edu) Examples are based on the ``Test Case Construction for Vertex Cover Problem," which were part of a technical report to be published. The graphs show that the generators create examples with known clique size.

Bro (From Mark Brockington. brock@cs.ualberta.ca) Examples from Mark Brockington and Joe Culberson's are generated that attempts to ``hide" cliques in a graph.

Stein (From Carlo Mannino mannino@iasi.rm.cnr.it). The formation of the set covering formula of the Steiner Triple Problem is created using Mannino's code to convert set covering problems to clique problems.

A comparison of CLIQ and Cliquer

The experiments have been carried out to compare the performance of the presented algorithm CLIQ with Cliquer (Niskanen, 2010). This comparison tests the performance of CLIQ algorithm in terms of average time taken to find the maximum clique in different graphs used as benchmarks in the second DIMACS implementation Challenge (Johnson, 1996).

For many instances, the number of search tree nodes is presented that define the nature of an algorithm, but cannot be used to evaluate the quality of it. Some algorithms traverse many nodes and spend little time in each node, whereas others traverse few nodes and spend more time in each. Thus the CPU time alone should be the measure of the efficiency of algorithm (Ostergard, 2002).

Results present a ratio of time in seconds and common node size which corresponds to the maximum clique. The C language programming code for Cliquer has been downloaded from the website of Niskanen, 2010 and has been run on Linux operating system. The implementation of CLIQ is also written in C language and it has also been run on Linux operating system.

The results in **(Table 4)** show that the CLIQ algorithm's performance is good as compared to Cliquer, i.e. it uses lower time per node. For example, for c-fat type of graphs, the CLIQ algorithm has lower time to Cliquer. Also in Table 4, it can be observed that the performance of the CLIQ is much better than the Cliquer for San graphs; Cliquer shows ratio 21.145 for san400_0.7_3.clq whereas the CLIQ algorithm shows ratio 1.846 for the same graph.

Table 4 also shows the worst cases of Cliquer where the ratio is 117.95 for brock200_1.clq whereas the CLIQ reports the ratio 1.882 for the same graph. Thus in most of the comparisons between the two methods, the CLIQ algorithm worked well.

For some graphs, such as the Johnson graphs, the speed of both algorithms is the same. **(Fig. 3),** illustrate all these cases.

Table 4: The table shows the result of comparison, for DIMACS graphs, of two algorithms, the CLIQ and Cliquer. Column one represent the names of DIMACS graphs. Second column gives the codes of those graphs, whereas third column describes the number of nodes in each graph. Fourth column represents the edges and fifth column states the clique size in each DIMACS graph. CLIQ and Cliquer columns illustrate the ratio of time in seconds and clique size, given in column five and six for the algorithms CLIQ and Cliquer respectively. This ratio is multiple of 10^3.

DIMACS	CLIQUE	BENCHMARKS				
DIMACS graphs	Codes of graphs	Nodes	Edges	Clique Size	CLIQ	Cliquer
c-fat200-1.clq	(CFat)	200	1534	12	0.333	0.333
c-fat200-2.clq	(CFat)	200	3235	24	0.166	0.166
c-fat200-5.clq	(CFat)	200	8473	58	0.413	8.137
c-fat500-1.clq	(CFat)	500	4459	14	0.285	0.571
c-fat500-10.clq	(CFat)	500	46627	126	3.047	0.253
c-fat500-2.clq	(CFat)	500	9139	26	0.615	0.307
c-fat500-5.clq	(CFat)	500	23191	64	1.000	0.250
johnson16-2-4.clq	(Joh)	120	5460	8	0.500	0.500
johnson8-2-4.clq	(Joh)	28	210	4	0.00	0.00
johnson8-4-4.clq	(Joh)	70	1855	14	0.285	0.285
keller4.clq	(Kel)	171	9435	11	0.888	0.035
hamming10-2.clq	(Ham)	1024	518656	512	11.507	1.140
hamming6-2.clq	(Ham)	64	1824	32	0.000	0.000
hamming8-2.clq	(Ham)	256	31616	≥ 79	1.812	1.812
hamming8-4.clq	(Ham)	256	20864	16	1.500	1.250
san200_0.7_2.clq	(San)	200	13930	18	2.400	0.666
san200_0.9_1.clq	(San)	200	17910	70	1.309	1.657
san200_0.9_2.clq	(San)	200	17910	60	1.846	21.768
san400_0.7_3.clq	(San)	400	55860	22	1.300	21.145
brock200_1.clq.b	(Bro)	200	14834	21	1.882	117.95
brock200_2.clq.b	(Bro)	200	9876	12	1.200	1.666
brock200_3.clq.b	(Bro)	200	12048	15	1.666	1.013
brock200_4.clq.b	(Bro)	200	13089	17	1.714	3.223
MANN_a9.clq	(Stein)	45	918	16	0.000	1.500

However, there are some cases, such as hamming10-2.clq, where CLIQ shows ratio 11.507, considerably worse than the Cliquer ratio 1.140.

The complexity of the proposed algorithm is $O(N^3)$ whereas the complexity of Cliquer is exponential (Katharina, 2006). Though the CLIQ algorithm in

theory has a higher runtime than the more technical one when applied to real data and the runtimes were nonetheless impressive. But in comparing with the Cliquer (Niskanen, 2010) in order to compute the maximum/maximal cliques in general graphs as well as in DIMACS graphs presented in Table 4, it can be seen that for many instances of DIMACS test cases, Cliquer could not report results in less time than the time of CLIQ.

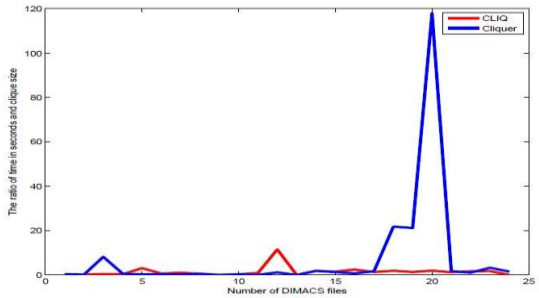

Fig. 3: Comparison of Cliquer and CLIQ algorithm, where x-axis shows the number of benchmark graphs downloaded from the DIMACS website and y-axis shows the ratio of time in seconds and clique size.

4. CONCLUSION

A new algorithm CLIQ which finds the maximum clique in a graph is presented in this paper. This algorithm mainly uses recursive and back-tracking approaches. A comparison of the CLIQ algorithm with the Cliquer algorithm shows that the CLIQ performs better than Cliquer in many cases.

For all DIMACS instances the CLIQ can find a maximum size of clique given for every DIMACS graph. At the same time the CLIQ is efficient for the problems of c-fat, Johnson, Hamming, San, Brock and MANN (Table 4). The results show that the algorithm performs well and solutions have equal quality with those obtained by Cliquer.

The DIMACS problems are benchmark results and the CLIQ's performance on DIMACS problems is encouraging. For many instances the algorithm performs well but in some instances the algorithm's results are not good. This may bring some questions into light that when the benchmark graphs are tested, are the algorithms tuned for each instance? (Ostergard, 2002). In the process of tuning, the vertices (nodes) of the graph are reordered in order to speed up the algorithm. Thus in testing CLIQ algorithm, no such tuning of the vertices is carried out but the same algorithm is used in all instances.

In this study, the algorithm CLIQ is only used for searching for the maximum clique and does not use other aspects of graphs, such as assigning the weight to edges or colouring the vertices or edges. In future, different searching aspects may be considered, such as by selecting different vertices with their corresponding weights.

REFERENCES:

Balas, E., S. Ceria, G. Cornuejols, and G. Pataki, (1996) Polyhedral methods for the maximum clique problem. *American Mathematical Society*, **13**, 11–28.

Berman, P. and A. Pelc, (1990) Distributed probabilistic fault diagnosis for multiprocessor systems, *20th International Symposium, IEEE*, 340–346.

Bron, C. and J. Kerbosch, (1973) Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, **16**, 575–577.

Gendreau, M., P. Soriano and L. Salvail, (1993) Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, **41**, 385–403.

Hasselberg, J., P. Pardalos and G. Vairaktarakis, (1993) Test case generators and computational results for the maximum clique problem. *Journal of Global Optimization*, **3**, 463–482.

Jain, B. and K. Obermayer, (2011) Extending Bron Kerbosch for Solving the Maximum Weight Clique Problem. *Arxiv Preprint arXiv*: 1101.1266.

Johnson, D. and M. Trick, (1996) Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, vol. **26**. *American Mathematical Society.*

Katharina, L., K. Michael, S. Stephan and N. Kay, (2006) On the maximal cliques in c-max-tolerance graphs and their application in clustering molecular sequences, *Algorithms for Molecular Biology* **1**(1), 9.

Kumlander, D, (2006) A Simple and efficient algorithm for the maximum clique finding reusing a heuristic vertex colouring. *IADIS International Journal on Computer Science and Information System*, **1**, 32–49.

Kuznetsova, A. and A. Strekalovsky, (2001) On solving the maximum clique problem. *Journal of Global Optimization*, **21**, 265–288.

Lagarias, J. and P. Shor, (1992) Keller's cube-tiling conjecture is false in high dimensions. *Bull. Amer. Math. Soc (NS)*, **27**.

Niskanen, S. and P. Ostergard, (2010) Cliquer User's Guide, *Version 1.21*

Ostergard, P. (2002) A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, **120**, 197–207.

Rutgers (2011) Dimacs graphs. URL ftp://dimacs:rutgers.edu/pub/challenge/ graph/benchmarks/clique/.

Tomita, E. and T. Seki, (2003) An efficient branch-and-bound algorithm for finding a maximum clique. *Discrete Mathematics and Theoretical Computer Science*, 278–289.