Data Reduction and Exact Algorithm for Clique Cover

I. Anu[1] II. JAYA PRIYADARSHNI [2] III. Bhavanesh Sharma[3] IV. Hansha Rani Gupta [3]

[1,2&3]Assistant Prof. , EE, Institute of Engineering and Technology, Alwar (Rajasthan)
[4]Assistant Prof. , EC, Alwar Institute of Engineering and Technology, Alwar (Rajasthan)

Email: anushreya17@gmail.com[1] jayapriyadarshni12@gmail.com[2]
bhavaneshsharma2009@gmail.com[3] hansaranigupta@gmail.com[4]

***Abstract*—In the computational complexity theory, Clique Cover is a graph-theoretical NP-complete problem. It has applications in diverse fields such as compiler optimization [Rajagopalan et al. 2000], computational geometry [Agarwal et al. 1994], and applied statistics [Piepho 2004; Gramm et al. 2007]. In this paper we have implemented some data reduction rules to reduce the problem set and then applied search tree algorithm to these reduced data sets to find the optimal solution in polynomial time complexity.**

I. INTRODUCTION

The problem of CLIQUE COVER is also known as KEYWORD CONFLICT problem [Kellerman1973] or COVERING BY CLIQUES problem (GT17) or INTERSECTION GRAPH BASIS (GT59) problem [Garey and Johnson 1979].Clique Cover is a graph-theoretical NP-complete problem in the computational complexity theory, along with the other hard problems such as Knapsack packing, graph coloring, Hamiltonian path, vertex cover, N-puzzle, 3-SAT, and famous Traveling salesman problem. Exponential algorithm is the best algorithm till now, although many heuristic algorithms give us reasonably good approximations of the optimal solutions. In the process of finding solution, approximated to optimal, the polynomial time data reduction techniques are very handy in reducing the problem state to smaller instances without sacrificing the possibility of finding optimal solution in reasonable time. For such reduced instances then often exhaustive search algorithms can be applied to efficiently find optimal solutions. NP-completeness is thought of as a class in which if one of such problems is solved to a polynomial time, all instances of NP-complete problems are reducible to a polynomial time. Then, it will be a very helpful in solving some very complex problems and can also be used in various applications.

1.1 Problem Description

A clique in an undirected graph G = (V, E) is a set C of vertices such that for any two vertices in C, there is an edge connecting the two.

We will also use "clique" to refer to the complete subgraph

of G induced by C and assume that the exact meaning will be made clear from context. The mathematical definition can be written as;

Given a graph G = (V, E) and an integer k, a clique is a subset U of V with |U | ³ k such that every pair of vertices in U is joined by an edge. Then, we call U a k-clique of the graph G.

Formally, as a (parameterized) decision problem, CLIQUE COVER is defined as follows:

Input: An undirected graph G = (V, E) and an integer k ≥ 0.

Question: Is there a set of at most k cliques in G such that each edge in E has both its endpoints in at least one of the selected cliques?

1.2 Definitions

The definitions used in the report are as follow:

Parameterised Problem: An instance of a parameterized problem consists of a problem instance *I* and a parameter *k* being a nonnegative integer.

Fixed Parameter Tractability: A parameterized problem is fixed-parameter tractable if it can be solved in $f(k) \cdot |I| + O(1)$ time, where f is a computable function solely depending on the parameter k, not on the input size |*I* |.

Problem Kernel: A parameterized problem such as *CLIQUE COVER* (the parameter is *k*) is said to have a problem kernel if, after the application of the reduction rules, the reduced instance has size $f(k)$ for a function f depending only on k. It is a well-known result from parameterized complexity theory that a parameterized problem is fixed-parameter tractable if and only if it admits a problem kernel [Cai et al. 1997].

1.3 Applications of Clique Cover Problem

Like instances of other graph problems, finding a maximum clique problem (or decision problem - is there a clique in a graph) can be widely seen in numerous fields of science and

engineering applications, even business application. For example, DNA molecular solution problem, data partitioning problem in embedded processor-based systems (memory chips), matching points problem in information systems, imaging process problem to remote sensing and astrophysics, and thousands more.

Applications of the vertex cover problem arise in network security; scheduling and VLSI design .Some practical examples are as follow:

(i) Finding a clique cover in a network corresponds to locating an optimal set of nodes on which to strategically place controllers such that they can monitor the data going through every link in the network.

(ii) Algorithms for clique cover can also be used to solve the closely related problem of finding a maximum clique, which has a range of applications in biology, such as identifying related protein sequences.

II. ALGORITHMS

The algorithm for finding a k-clique in an undirected graph is highly parallel and represents a class of solving NP-complete search problems. The algorithm seems simple - systematically list all possible sets of exactly k nodes and for each such set, check whether all pairs are neighbours in a selected sub-graph. However, this brute force algorithm to find a clique in a graph is unfortunately a historically hard problem, NP-complete. The algorithm is polynomial if k is constant, but not if k varies. A better one can be done by starting with each node as a clique of one, and to merge cliques into larger cliques until there are no more possible merges to check. Two cliques A and B may be merged if each node in clique A is joined to each node in clique B. For the former method of finding clique, as you expected, a searching tree of a clique problem is spreading exponentially although it depends strongly on a structure of a graph. To reduce the problem set some data reduction rules are applied through backtracking and then by using search tree algorithm approximate solution to optimal solution is finding in reasonable time.

2.1 Data Reduction Rules

A data reduction rule replaces a Clique Cover instance by a simpler instance, such that the solution to the original instance can be reconstructed from the solution of the simpler instance. These rules replaces, in polynomial time, a given Clique Cover instance (G, k) consisting of a graph G and a nonnegative integer k by a "simpler" instance (G', k') such that (G, k) has a solution if and only if (G', k') has a solution. Rules can be described as follow:

2.1.1 Reduction Rules

Before applying the rules some initializing process are done. We inspect every edge {u, v} of the original graph. We use two auxiliary variables: We compute a set N {u, v} of u and v's common neighbours and determine whether the vertices in N {u,v} induce a clique. More precisely, we compute a positive integer c {u,v} , which denotes the number of edges interconnecting the vertices of N{u,v} . After initialization process reduction rules as follow are applied.

Rule 1: Remove isolated vertices and vertices that are only adjacent to covered edges. This is executed in O (VE) time complexity.

Rule 2: If an edge {u, v} is contained only in exactly one maximal clique C, then add C to the solution, mark its edges as covered, and decrease k by one.

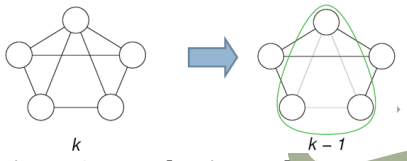

Figure 2.1: Reduction Rule 2

Rule 3: If there is an edge {u, v} whose endpoints have exactly the same closed neighborhood, that is, N [u] = N [v], then mark all edges incident to u as covered. To reconstruct a solution for the unreduced instance, add u to every clique containing v.

Rule 4: If all the neighbours of vertex v, x with N(x) \ N (v) = ø have at least one vertex p such that N(p) ⊆ N(v), then delete v.

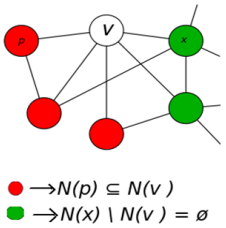

Figure 2.2: Reduction Rule 4

2.2 Search Tree Algorithm

In the process of finding optimal solution for our problem, search tree algorithm is applied on the reduced data set that we get on applying data reduction rules on the problem set.

In search tree algorithm we first choose an uncovered edge e ϵ E .Then for each maximal clique C that contains e, mark all edges in C as covered, decrease k by one, and call the algorithm recursively.

This exhaustive algorithm can solve all instances in a benchmark from applied statistics within a second (up to 124 vertices and 4847 edges) and can solve sparse instances with hundreds of vertices and tens of thousands of edges within minutes.

2.3 Backtracking based Solution

To find the solution of Clique Cover problem we integrate backtracking with the data reduction and search tree algorithm.

Backtracking based solution can be applied with two variations such as:

(i) Permutated all possible sub-graphs and checks if every node in a sub-graph is connected to every other node in that sub-graph and the size of clique is equal to k.

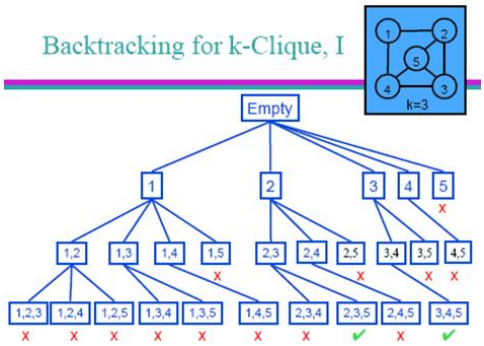

Figure: 2.3

(ii) Permutated only sub-graphs in which every node in a sub-graph is connected to every other nodes in that sub-graph, and check if the size of clique is equal to k. In this manner, cutoffs occur in pruning.

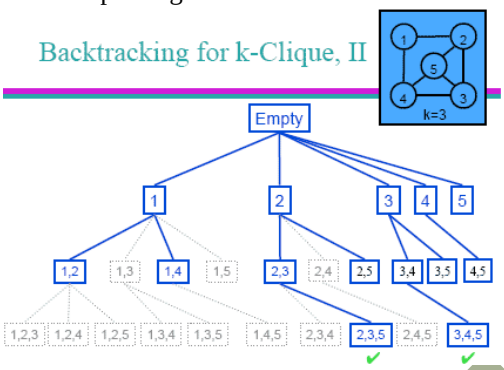

Figure 2.4

III. SOFTWARE WORK BENCH DESCRIPTION

3.1 Graphical User Interface

The GUI of the workbench is shown in **Figure 3.1**. Various options shown are described below:

Size: This field is used to provide size of the clique by user.

Nodes: This field is used to give the number of nodes by user from which the graph is generated.

Generate Graph: By clicking this button the graph will be generated on the canvas from the number of nodes used in Nodes field and the random file provided by the user.

Generate Solution: By clicking this button our algorithm will run and generate the cliques for given nodes and for given random file which is shown on the canvas.

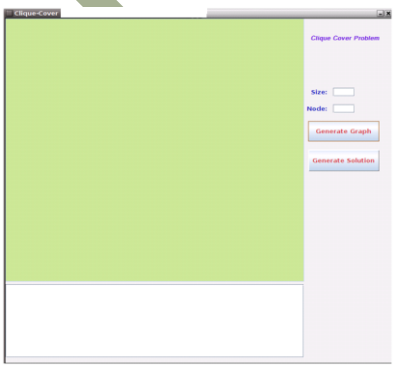

Figure 3.1 GUI for Clique Cover

The canvas represented in green in **Figure 3.1** is used to show the graph generated with the given number of nodes and the random file generated from random file generator. Text area is used to show the result of the solution generated. Graph generation is shown in **Figure 3.2.** In **Figure 3.3** generation of all possible cliques of given size is represented.

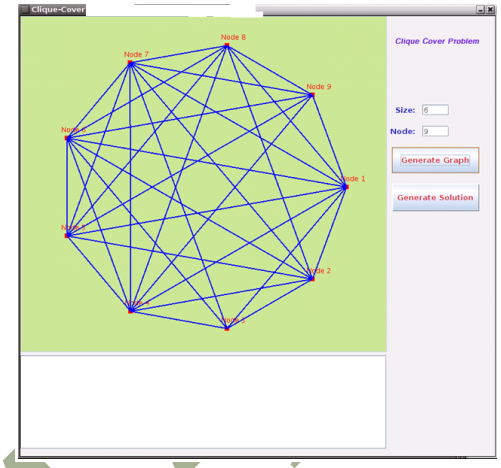

Figure 3.2 Graph generation for 9 Nodes

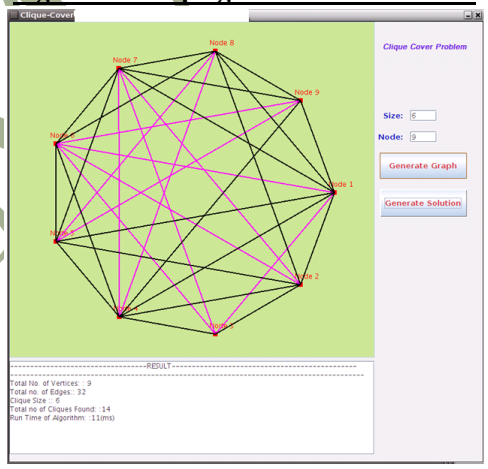

Figure 3.3 Cliques of size 6 for graph with 9 nodes

In **Figure 3.3** on the canvas generated cliques are displayed by BLACK edges and the edges that are not the part of the solutions are displayed by PINK color. In the Text Area result is shown for 9 node graph with clique size 6. First line shows the total number of vertices , second line displays total edges in the graph, this list is provided by the user generated from random graph generator and saved in data_n.txt in the same directory. For the purpose of generating clique's vertices that are the part of the cliques is saved in file name myfilen.txt. This is automatically generated file and generated differently as it takes n from Nodes field and generate file at run time.

Solution of our problem exist when all the edges in E of graph G(V,E) are the part of cliques. In **Figure 3.3** we get some edges displayed in pink means that are not the part of our solution. In **Figure 3.4** we generate cliques of size 5 with 9 nodes and in **Figure 3.5** we generate it for size 4.

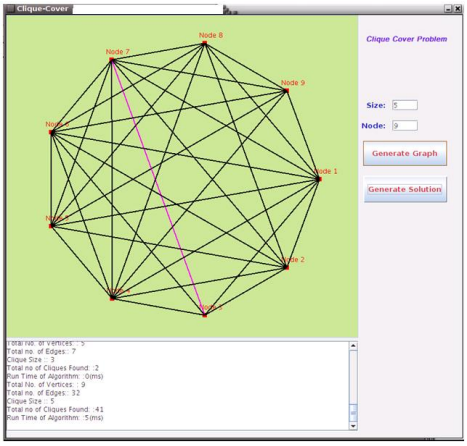

Figure 3.4 Cliques of size 5 for 9 nodes

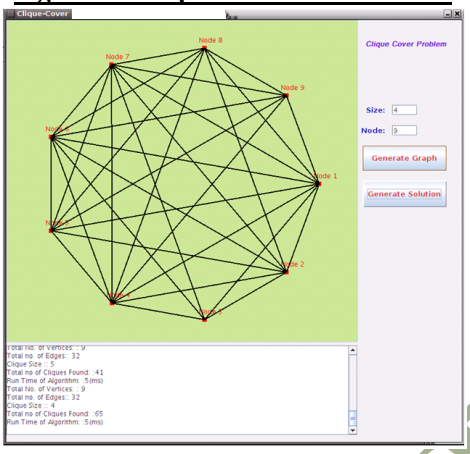

Figure 3.5 Cliques of size 4 for 9 nodes

In **Figure 3.4** we get one edge spare which is not in our solution, but in **Figure 3.5** for clique size of 4 we get all the edges for graph G(V,E) in our solution space. For 9 nodes we get 65 cliques of size 4 as our solution. For graph having 5 vertices we get 7 cliques of size 2 as our solution as for size 3 two edges spare for being a part of solution, which is shown in **Figure 3.6** .In **Figure 3.7** solution for clique size 2 is displayed.

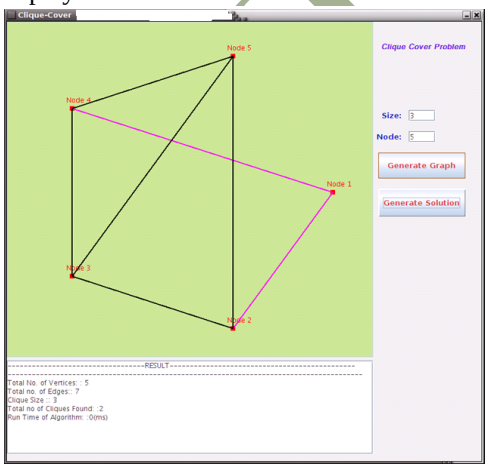

Figure 3.6 Cliques of size 3 for 5 nodes

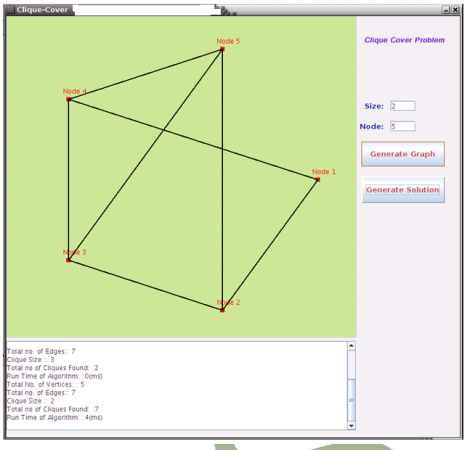

Figure 3.7 Cliques of size 2 for 5 nodes

3.2 Class Diagram and Description

The main class which provides the GUI front end is the CliqueDisplay. **Figure 3.8** shows the class diagram of Clique, which is important class to generate solution for clique cover problem.

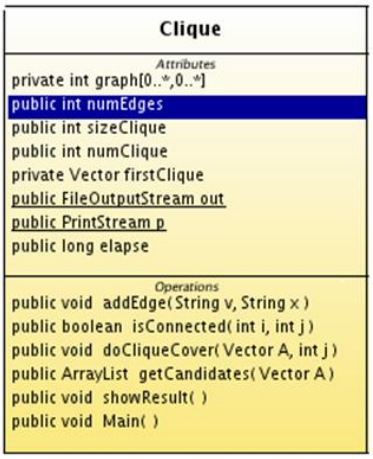

Figure 3.8 Clique Class Diagram

Figure 3.9 represents the class diagram for main class that represents our graphics user interface.

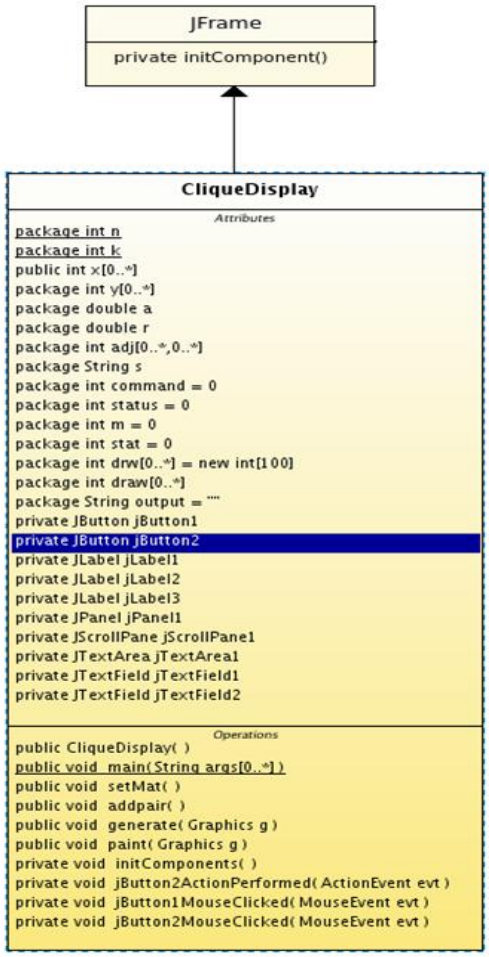

Figure 3.9 Class diagram for GUI

3.3 Sub Diagram for Clique Cover User Interface

Apart from the GUI class some package diagrams are displayed in **Figure 3.10**

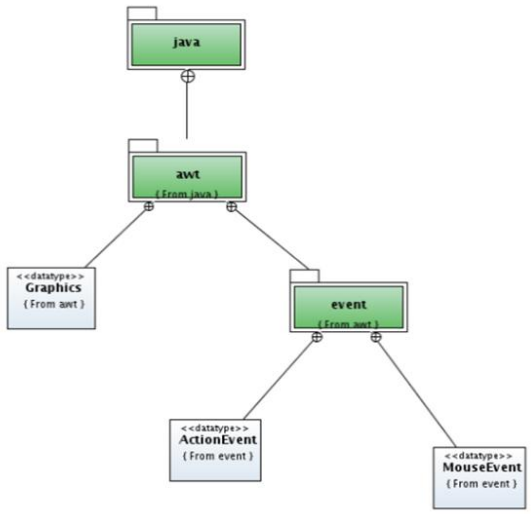

Figure 3.10 Package Diagram

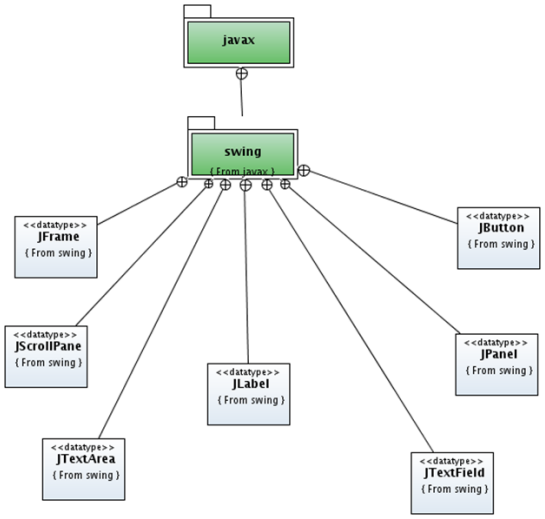

Figure 3.11 Component Diagram

Figure 3.11 shows connectivity between various components of swing package. Swing is the java extension used to make user interfaces. It's simpler to use than AWT, which is part of its appeal, and have a lot of neat things pre-defined.

IV. RESULTS AND DESCRIPTION

For the purpose of comparison we consider the following algorithms-

4.1 Kouakou Algorithm:

If an on-line algorithm LA guarantees the competitive ratio C_{LA} for the minimum clique cover problem, then its corresponding algorithm b-LA, for b-LCC, guarantees the competitive ratio $C_{LA} / 1+C_{LA}$

Let us consider Algorithm b-LA solving b-LCC. Let Ns be the number of saturated cliques returned by b-LA and Nu the number of unsaturated cliques. Then, the total number λb (G) of cliques returned by b-LA satisfies:

λb (G) = Ns + Nu …… (1)

Let denote by χb (G) (resp. λ(G)) the optimal value for the b-clique cover problem (resp. the number of cliques returned by LA for LLC). χb (G) can also be defined as the b-bounded co- chromatic number. Then, we have Nu $\leq \lambda$(G) and Ns $\leq \chi b$ (G). Moreover, relation (1) implies

λb (G) $\leq \chi b$ (G) + λ(G) $\leq \chi b$ (G) +1/C_{LA} * k(G),

where the last inequality of expression (2) holds since LA is C_{LA} -competitif. Let us also note that, for a given instance graph G, every solution of b-LLC is a feasible solution of LLC. Therefore, as they are minimization problems, we have k(G) $\leq \chi b$ (G). Relation (2) becomes:

λb (G) $\leq \chi_b$ (G) + C_{LA} *χ_b (G)

4.1.1 Hardness Result

The algorithm considers an interval graph G = (V, E) of which the n vertices will be revealed (one-by-one) as it progress with the resolution. The algorithm can be thinking of

like an on-line problem as a game between two players. One player, the adversary, generates requests on-line according to some specified mechanism and is charged a cost for its selections according to some specified rules. For example, for a minimization problem, the adversary is charged the minimum (off-line) cost to perform any request sequence it generates. The other player is the algorithm, which responds to requests by making a decision, and incurs some total cost over the course of the game. In the model, the adversary has complete knowledge of the algorithm and it can first decide how many requests it wants to generate; then, it internally simulates the algorithm on each possible sequence of that length to find the sequence that maximizes the ratio between the on-line algorithm's cost and the adversary's cost. The algorithm emphasize that, as we deal with a minimization problem, the adversary's goal is to maximize the ratio of the algorithm's cost to the adversary's cost, while the on-line algorithm's goal is to minimize it.

4.2 Running time of Backtracking based solution

The measurements are made in two ways that the environmental variables are controlled alternatively. For each environmental variable, the running time is measured upon various different graphs.

System Summary: Intel Pentium D 2.80GHz with 4Mb L2 Cache, 1.5 Gb RAM.

The number of edges is fixed to 4000, and the running time is measured upon different graph size - the number of vertices in a graph varies. For a certain range of vertices, the running time is moderately increasing in polynomial behavior.

Specifically, in the **Figure 4.1**, when the number of vertices lies in the range of 0 to 220, the running time obeys a polynomial behavior. However, if the number of vertices exceeds about 220, the running time increases with an astronomical growth rate. Although we can't see in the plot, the running time will probably increase with a polynomial growth rate again up to the point where the number of vertices reaches about 4100. Then, after that point, it will jump with another astronomical growth rate. To visualize, the asymptotic behavior may have a stair-like-shape: switching a polynomial growth and a vertical growth alternatively, having a certain amount of term. This is because although the k-clique algorithm looks like a polynomial algorithm, it actually takes an exponential time, depending on the value of k.

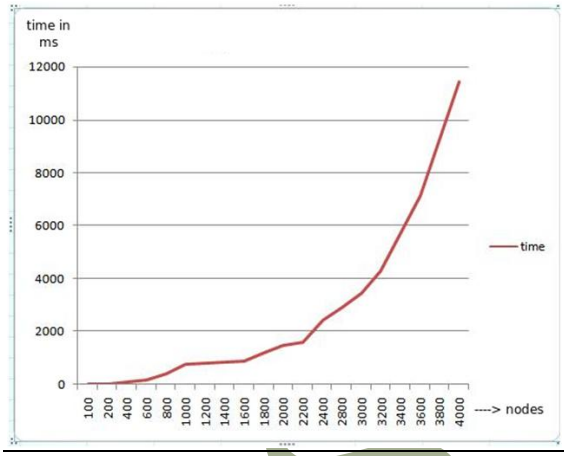

Figure 4.1 Graph between run time and number of nodes

Indeed, we can be convinced if we notice that the "critical" points such as 220, 4000, etc are the numbers of vertices which change the size of clique, from 4 to 5 and 5 to 6, respectively. Thus, the asymptotic behavior obeys O(nk) worst-case time complexity, where k = O(log n) in this particular instance of the problem.

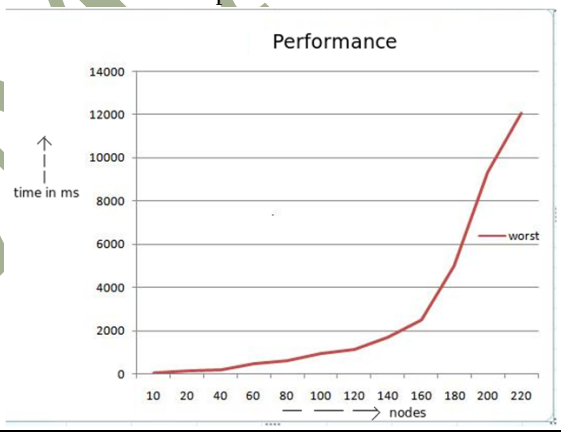

Figure 4.2 Graph between run time and number of nodes

In **Figure 4.2**, the number of vertices is fixed to 4000, and the running time is measured on different graph density – the number of edges in a graph varies. In this case, since the number of vertices is fixed to 4000, the value of k is also fixed to 4. Thus, as we expect, the running time increases in a polynomial time. In fact, a polynomial of degree of 4 is used for the best-fit curve, and it reasonably fits the data. Therefore, the time complexity is O(n4) in the worst-case.

This does not end the beauty of mathematics where the **Figure 4.3** will make the revealing of the fact that as the "k" grows the complexity grows but at the time n/2 is reached the time the complexity is maximum and after that it decrease because the value of ${}^{n}C_{k}$ starts decreasing.

To conclude, even though the k-clique algorithm using backtracking may depend strongly on the structure of a graph, it still has the time complexity of O(nk) in the worst-case, where the size of clique, k, is a variable. (Where n can be either the number of vertices or edges since either one of them can be interchangeably expressed as a function with respect to

the other). Thus, it is an exponential algorithm, which is NP-complete.

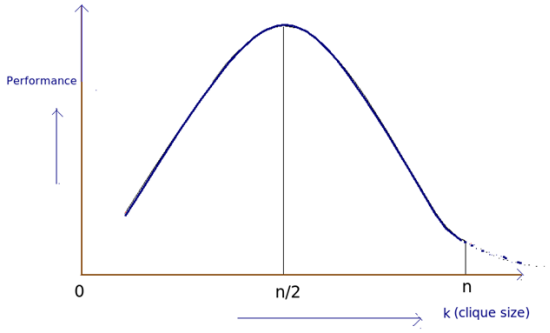

Figure 4.3 Performance vs clique size

V. CONCLUSION AND FUTURE WORKS

The implementation of Clique Cover problem using backtracking depends strongly on the structure of a graph, it still has the time complexity of O(nk) in the worst-case, where the size of clique, k, is a variable and n can be either the number of vertices or edges since either one of them can be interchangeably expressed as a function with respect to the other. Thus, it is an exponential algorithm, which is NP-complete.

The backtracking based solution is compared with Kouakou algorithm which works fine only up to 200-300 nodes but takes exponential time for higher number of nodes. Whereas our algorithm works fine upto 4000 and higher number of nodes.

In this paper Clique Cover problem is solved by backtracking method in which we consider a problem instance at a time. Parallel algorithms can be used to make the algorithm run faster. By applying parallel algorithms several problem instances will be considered simultaneously which can reduce the run time of the algorithm to a great extent.

VI. REFERENCES

- *Data Reduction, Exact, and Heuristic Algorithms for Clique Cover by Jens Gramm , Jiong Guo ,Falk Hffner Rolf Niedermeier*
- *Fast Exact and Heuristic Methods for Role Minimization Problems by Alina Ene,William Horne ,Nikola Milosavljevic.*
- *A HARDWARE ALGORITHM FOR THE MINIMUM P - QUASI CLIQUE COVER PROBLEM by Shuichi Watanabe, Junji Kitamichi, Kenichi Kuroda.*
- *Reducibility among Combinatorial Problems by Richard Karp.*
- *Approximating the minimum clique cover and other hard problems in subtree filament graphs by J. Mark Keila , Lorna Stewartb*
- *On-line algorithm for the minimal b-clique cover problem interval graphs by B. Kouakou.*
- *Polynomial-Time Data Reduction for DOMINATING SET by JOCHEN ALBER, MICHAEL R. FELLOWS and ROLF NIEDERMEIER.*
- *A Stochastic Local Search Approach to Vertex Cover by Silvia Richter, Malte Helmert, and Charles Gretton.*
- *www.nada.kth.se*
- *www.new.dli.ernet.in*
- *www.geometrylab.de*
- *www.liafa.jussieu.fr*
- *ieeexplore.ieee.org*
- *www.portal.acm.org.*