CLISAT - CLIQUE ENCODINGS IN IMPLEMENTATION OF SAT

NIDHI DAHALE, DR.N.S.CHAUDHARI, DR.MAYA INGLE

Abstract: Satisfiability is the core problem among the NP complete problem. The solutions to this difficult problem have become an easy task due to the development of efficient SAT solvers. Many difficult problems can be solved with the help of the Satisfiability (SAT) problem or by making use of it, by just encoding the problem to the Satisfiability problem. Clique problem (for k>3) is a well known NP Complete problem, till now there are very less known deterministic methods that can solve a Clique problem in polynomial time. To solve this efficiently we go through the propositional Satisfiability. However to go through the propositional Satisfiability the Clique instances are to be converted to SAT.In other words the need of encoding a graph problem instance to a formula which consists of clauses and literals. In this paper we propose *CLISAT* that converts graph instances to SAT CNF clauses and with the satisfiable values to the formula we obtain the maximum clique vertices for graph datasets.

Keywords: SAT, CNF, Clique, NP-Complete, Chromatic number, DIMACS

Introduction: Determining the Clique in a graph is an NP Complete problem [1]. Finding out Cliques in a graph has always attracted researchers since a long time. The problem to find out Cliques have been found to be very important as there exist many challenging applications of this problem in the area of data mining, bioinformatics, web page clustering, text mining and analysis of social networking patterns. The literature reveals that many efforts have been made in finding Cliques in the graph [2]. Although a vast amount of work has already been done in Clique finding algorithms. Generally at the same time, it has been observed that Satisfiability has been used to prove any problem as NP Complete. There does not exist any idea of encoding Clique into Satisfiability (SAT). Here, we propose an approach of finding cliques in a graph through the truth values of the solutions of Satisfiability problem. The encodings proposed by us makes it feasible to find the solutions to Satisfiability as well as Clique.

A Clique of a graph is a set of vertices, any two of which are adjacent. Algorithms for finding the maximal and maximum Cliques have been in focus over a long time. The maximal Cliques are the type of Cliques whose vertices are not a subset of the vertices of a larger Clique. Differently the maximum Cliques are the largest among all Cliques in a graph [3]. In the maximum Clique problem, one desire to find one maximum clique of an arbitrary undirected graph. In this paper, our approach is to find out the maximum Cliques in a graph through Satisfiability clause generation. This problem of finding maximum Cliques is computationally equivalent to some other important graph problems, for example, the maximum independent (or stable) set problem and the minimum vertex cover problem [4].

The advancement in the SAT solvers led to generate efficient SAT solvers. As a result, it turned in to the fact that any NP Complete problem can be reduced to/from SAT. In this work, we perform a reduction from graph to CNF Satisfiability clauses in polynomial time. After this reduction, the literals of all clauses are passed onto a SAT solver that results in generation of Clique vertices of the given graph. We discuss the background details of SAT and Clique problem in Section 2. Section 3 describes our proposed encoding algorithm i.e. *CLISAT* to reduce a given graph to propositional SAT formula (consisting of clauses and literals). An example illustrating the working of algorithm is also presented in this section. In Section 4, we enlighten the results of our methodology on Dimacs Challenge Clique Graph datasets. Finally, we conclude with conclusions in this paper.

Background: We now discuss Satisfiability along with its versions and Clique problem in detail in this section.

Satisfiability and its Versions: SAT problem is the core problem among the large family of intractable NP Complete problems. There exist two versions of SAT namely; decision version and search version. Here, we deal with the search version of SAT. It is basically composed of the set of m variables x_1, x_2,…,xm. These variables are also known as literals and can occur in a positive (Q = x) or negative (Q = ~x) state. Further, these literals collectively form set of distinct clauses consisting of only literals concatenated together with the symbols either $\vee$ (logical OR) or $\wedge$ (logical AND) symbols only. For example, let the clauses be C_1= ($x_2 \vee x_4 \vee x_5 \vee x_7$) and C2= ($x_1 \vee x_2 \vee x_{3\vee} x_6 \vee x_7$). The Conjunctive Normal Form (CNF) ($C_1 \wedge C_2$) for these two clauses is ($x_2 \vee x_4 \vee x_5 \vee x_7$) $\wedge$ ($x_1 \vee x_2 \vee x_{3\vee} x_6 \vee x_7$). The CNF formula thus is a conjunction of clauses ($C_1 \wedge C_2$ …$\wedge C_{n)}$. The goal of the Satisfiability problem is to assign truth values to the variables of the clauses that make the formula satisfiable [5].

The SAT problem converts to an optimization problem by trying to maximize the number of clauses that one assignment can satisfy, which leads to formation of the Maximum Satisfiability Problem (MAX-SAT). Another version of SAT is K-SAT where each clause in CNF is limited to only k literals. Other variation in Satisfiability problem is 2-SAT which is solvable in polynomial time, in which each clause is restricted with two literals only. 3SAT is believed to be NP-Complete, and in this each clause is restricted to three literals per clause [6].

Clique as a NP Complete Problem: Clique problem is a very important problem in the category of graph based NP Complete problems. Formally a Clique problem is defined by a graph {<G, K>} where G represents a graph and K denotes the size of the Clique contained in the graph G [7]. In other words it could be stated that whether the

Graph G contains Clique of size K. More formally it is defined that if a graph G = (V, E) has a clique of size K then whether there exists a subset V' with |V'| = K of the vertices V such that U, V in V'is the edge (U, V) is in E. In other words it is said that the induced sub graph on the vertices V' is the complete graph of K vertices.

To elaborate the definitions of clique problem, a figure below represents a graph G with five vertices. All these five vertices have connections among themselves. All the five vertices v1, v2, v3, v4 and v5and are connected to each other through the connecting edges. It is visible that vertex v1 is connected to v2, v3, v4 and v5. Similarly the vertex v2 have interconnections with all the other vertices v1, v3, v4 and v5. This is true for all the other vertices also. Thus the size of the clique is K= 5 and it forms a complete sub graph with all the five vertices.

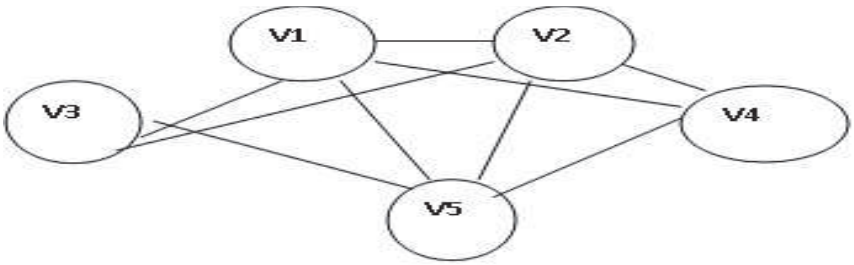

Fig 2.1 Undirected Graph Representing Clique

Every NP Complete problem always occurs in two versions. The version is either a decision version which decides in the form of "Yes" or "No". Another form of version is the search version, which finds out the solution for a particular NP Complete problem [8]. The Clique problem also occurs in its two versions, where its decision version provides an answer that whether clique exists or not. On the other hand the search version with the help of Clique finding algorithms finds the set of maximum or minimal Cliques that exists in a graph. The main emphasis of the Clique finding algorithm is on the size of the Clique which exists in the undirected graph. We define the Clique problem as follows:

Problem: CLIQUE

Input: Let <G, K> be a graph where G = (V, E) is a graph and K is number of vertices to form a CLIQUE.

Decision version: In any undirected graph, we find whether there exists a clique or not. The result may be in "yes" or "no" form.

Search Version: In any undirected graph G, we find a clique C in G with |C | =K, where K is an integer. In this version, a Clique in a graph is searched out.

Proposed Work: Let there be a graph G = (V, E) where V is the set of n vertices $\{v_1, v_2 \ldots v_n\}$ and E is the set of m edges $\{e_1, e_2, \ldots, em\}$ connecting these vertices. This dataset of vertices and the connecting edges between them are stored as an input file. The Dimacs Clique datasets [9] are usually large, for example may contain 800 vertices and 20816 edges connecting these vertices. The adjacency matrix A[n][n] acts as a storage for 1's and 0's for connections and non connections among the edges. Our proposed algorithm *CLISAT* generates clauses traversing all the n rows of A[n][n] for each value. We discuss the informal description of this algorithm followed by formal description in this section.

Informal Description: Here, we present an informal description of the CLISAT algorithm. For input dataset of size *n* X *n* using Dimacs, we get output in terms of adjacency matrix (having elements as 0 or 1). The element 0 corresponds to non-interconnections whereas 1 states interconnections among the vertices. We initialize CNF clause i.e. *cnfclause* and its count viz. *count* in the loop as zero. If the element at a particular position is found to be 1, then its column index is stored in *temp*. If the element at a position is found to be 0, temp stores (~i V ~ j), where i and j are row and column index of adjacency matrix. The count is incremented as each value of j is completed. Thus, we present now the formal description depicting the logical flow of our algorithm.

Formal Description : Now, we present the formal description of our proposed *CLISAT* algorithm for generating the CNF clauses and later on finding the satisfiable values for these clauses as follows:

Algorithm: ***CLISAT:*** /* Input graph dataset G; use its vertices and edges to generate adjacency matrix A of size *n* X *n*. */

Step 1: for i = 1 to *n* do
for j = 1 to *n* do
input A[i][j];
/* Initialize clauses */
cnfclause ( ) = NULL;
count = 0;
/* Checking inter-connections between vertices, generating clauses and their count */
Step 2: for i = 1 to *n* do
begin
temp = NULL;
for j = 1 to *n* do
begin
If (A[i][j] = = 1) then
temp = *temp* Λ (*temp* V j);
else
temp = *temp* Λ (~i V ~ j);
end;
count = *count* + 1;
cnfclause( ) = *cnfclause*( ) Λ *temp*;
end;

Following steps are used by SAT solver for announcing

the stiafiable values associated with the clauses generated above. */

Step 3: Store clauses *cnfclause*( ) in Literals.txt file

Step 4: Convert Literals.txt into SAT solver input format using solver tool

Step 5: Solver announces satisfiable values (T or F) for given input.

The CNF expression obtained after execution of *CLISAT* algorithm is passed to SAT solver. There exist many general purpose solvers as open source for testing Satisfiability of CNF formulae. These SAT solvers help us to verify the Satisfiasbility of CNF formula. We have used Glucose [10] and MiniSAT [11] solver tools to check the Satisfiability of the given formula.

Implementation Results: To understand the functionality, we present the case study with the Dimacs graph file containing three vertices namely; (v_1, v_2, v_3) and two edges as e 1 2 and e 1 3. The edge e 1 2 represents that there exists a connection between vertex (v_1) and vertex (v_2). After execution of *CLISAT*, we get results stepwise as shown in Table 1. It may be observed that the end of looping structures lead to produce the desired CNF clause for given graph dataset. This CNF clause is passed to the SAT solver thereby producing the satisfiable values as 1 = true, 2 = false and 3 = true. These values returned by the SAT solver makes the formula unsatisfiable. Thus, resulting in no maximum Clique vertices in the graph.

We have implemented our formulation on five datasets (from Dimacs benchmark problems for Clique graphs) as depicted in Table 2. It must be observed that Dimacs provides a huge range of challenging datasets for finding Cliques in a graph [12]. For some datasets of graphs having varying number of vertices and edges, the satisfiable values have been found. These values represent the maximum Clique vertices of the input graph. For a graph containing 400 vertices and 59765 edges, 20283 total clauses have been generated as highlighted in Table 2. It is worth to state that out of these total clauses, 33 vertices possess the satisfiable values.

Conclusion: We formulated a novel approach of encoding graphs to CNF SAT clauses. With this formulation, it is possible to reduce the instances of one problem as Clique to another as SAT. The graph datasets are reduced to CNF clauses by our *CLISAT* algorithm first. The satisfiable values that are announced by the SAT solver are observed to be same as the maximum Clique vertices for the graph datasets. The implementation results are quite robust in the sense outperformance on huge graph datasets.

Table 1: Stepwise Execution of CLISAT Algorithm

i	j	A(i,j)	*temp*	*Cnfclause*
1	1	0	Null	(~1∨~1)
1	2	1	(~1∨~1)∧2	(~1∨~1)∧2
1	3	1	(~1∨~1)∧(2∨3)	(~1∨~1)∧(2∨3)
2	1	1	(~1∨~1)∧(2∨3)∧1	(~1∨~1)∧(2∨3)∧1
2	2	0	(~2∨~2)	(~1∨~1)∧(2∨3)∧(1)∧((~2∨~2)
2	3	0	(~1∨~1)∧(2∨3)∧(1)∧((~2∨~2)∨(~2∨~3)	(~1∨~1)∧(2∨3)∧(1)∧(~2∨~2)∨(~2∨~3)
3	1	1	(~1∨~1)∧(2∨3)∧(1)∧((~2∨~2)∨(~2∨~3)∧(1)	(~1∨~1)∧(2∨3)∧(1)∧(~2∨~2)∨(~2∨~3)∧(1)
3	2	0	∧(~2∨~3)	(~1∨~1)∧(2∨3)∧(1)∧(~2∨~2)∨(~2∨~3)∧(1) ∧(~2∨~3)
3	**3**	**0**	**(~1∨~1)∧(2∨3)∧(1)∧((~2∨~2)∨(~2∨~3)∧(1) ∧(~2∨~3)∧(~3∨~3)**	**(~1∨~1)∧(2∨3)∧(1)∧((~2∨~2)∨ (~2∨~3)∧(1) ∧(~2∨~3)∧(~3∨~3)**

Table 2: CNF-SAT Clauses Generated on Dimacs Dataset

Dataset Name	No. of Vertices	No. of Edges	Total Clauses	No. of Satisfiable Literals
brock200_2.b	200	9876	10128	12
brock200_4.b	200	13089	6948	17
brock400_2.b	400	59786	20278	29
brock400_4.b	**400**	**59765**	**20283**	**33**
brock800_2.b	800	20816	110725	24

References:

1. M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, 1962, pp. 394-397.
2. Bron C and J.Kerbos, "Algorithm 457: finding all cliques of an undirected graph", communications of ACM, 16, 1973, pp.575-577.
3. David R Wood, "An Algorithm for finding a

maximum clique in a graph", Operation Research Letters, 21, 1997, pp.211-217.
4. L.Babel, "Finding Maximum Cliques in arbitrary and in special graphs", Computing, 46, 1991, pp. 321-341.
5. Rino Sanchez, "The Boolean Satisfiability Problem (SAT)", 605.721-Design and Analysis of Algorithms, vol.1, No.1, April 2010.
6. S.Dasgupta,C.H.Papadimitriou and U.V.Vazirani, "Algorithms, chapter8-NP problems", pp. 247-282.
7. Kuznetsova.A. and A.Strekalovesky, "On solving the Maximum Clique Problem", Journal of Global Optimization, 21, pp.265-288.
8. R.G.Jeroslow and J.Wang , "Solving Propositional Satisfiability Algorithms" ,Annals of Mathematics and and Artificial Intelligence, vol. 1,1990, pp.167-187.
9. DimacsImplementation,allenges,http://dimacs.rugters.edu/challenges/.
10. http://www.labri.fr/perso/lsimon/glucose
11. N. Een, N. Sorensson. "An Extensible SAT-solver" in SAT 2003, pp. 502-508.
12. http://mat.gsia.cmu.edu/COLOR03/clq.html.

* * *

Nidhi Dahale/Assistant professor AITR , Indore(M.P)/ nidhi.dahale@gmail.com
Dr.N.S.Chaudhari/Director VNIT Nagpur & Professor CSE IIT, Indore (M.P)/nsc183@gmail.com
Dr. Maya Ingle/Professor SCSIT DAVV, Indore(M.P)/maya_ingle@rediffmail.com