[VNSGU JOURNAL OF SCIENCE AND TECHNOLOGY] Vol.4. No. 1, July, 2015
197 - 204,ISSN : 0975-5446

Determining Vertex Cover Using Polynomial Encoding Of 3sat

DAHALE Nidhi
Department of M.C.A.
A.I.T.R, Indore
nidhi.dahale@gmail.com

DR. CHAUDHARI.N.S
Director and Professor
V.N.I.T,Nagpur and IIT
Indore
nsc0183@gmail.com

DR.INGLE Maya
Professor and Sr.System
Analyst
SCSIT, DAVV Indore
maya_ingle@rediffmail.com

Abstract

It is very well reckoned that Satisfiability falls in the category of NP Complete problems. Determining Vertex Cover is also an NP Complete problem in the category of graph based problems. Reductions prove to be helpful to find the solution of one problem with the other. The paper attempts to determine the Vertex Cover in a graph by polynomially encoding the 3SATclauses. Thus, in this perspective we have formulated two algorithms kVC_2kSAT and $kSAT_23SAT$. We have reviewed the implementation of these algorithms on some prominent graphs of graph theory. It has been found that reductions prove very helpful in finding solutions to both the problems. The paper reveals the solutions for the both the NP Complete problems. Satisfiable truth values obtained for a 3SAT formula contribute to the Vertex Cover vertices of the graph.

KEYWORDS: 3SAT, kcnfclauses, minimum VC, NP Complete

1. Introduction

NP Complete problems have been one of the diverse problems from graph theory and combinatorics [1]. Determining Vertex Cover (VC) in a graph is one such NP Complete problem. It has been observed that VC problem has been studied as an aspect of approximation algorithms and at the same time, less effort have been made on exact algorithms. There exist several approaches to solve VC problems such as approximation algorithms, fixed parameter algorithms, heuristics etc. Reductions provide a way to prove any problem NP hard by reducing problems from SAT. Also, it has been observed that very few polynomial time algorithms exist to 3SAT problems. One of such approaches uses antecedents and consequents [2]. However,

there is a scope of suitable formulations to encode a graph with a VC to 3SAT as encodings play an important role in reducing one problem instance in to the other. One such encoding where the graph containing a Clique is converted to a generalize SAT formula. It has been observed here that the satisfiable values of the formula contribute in the Clique vertices of the graph [3].

The advancement in the SAT solvers led to generate more efficient SAT solvers. As a result, it turned to the fact that any problem can be reduced to or from SAT. This polynomial time reduction from a graph to 3SAT formula has been proven to be helpful to detect the Satisfiability of the generated formula as well as VC vertices of the graph. We discuss the background details of 3SAT and VC problem in Section 2. In Section 3, we propose an Algorithm I *kVC_2kSAT* for converting instance of a graph to kcnfclauses. Another Algorithm II *$kSAT_2 3CNF$* based on a non recursive method has been presented to convert kcnfclauses to 3SATclauses. Section 4 describes the steps that are undertaken to pass the generated 3SATclauses to 3SATSolver. Finally, we discuss the results in Section 5 highlighting the number of 3SATclauses and corresponding satisfiable values which contribute VC vertices of the graph.

2. Background

We now discuss firstly 3SAT and VC problem as NP complete problems in this section.

2.1 3SAT as a NP Complete Problem

3Satisfiability (3SAT) as the original NP Complete problem, is a fundamental problem used as a main step in numerous algorithms. Thus, it plays a vital role in many applications such as design and manufacturing of integrated circuits, database applications, robotics etc. Also, it impacts in the development of the efficient computer systems at multiple levels [4]. In general, 3SAT is composed of k clauses $C_1, C_2, \ldots, C_k$ in a Conjunctive Normal Form (CNF) and is represented as $C_1 \wedge C_2 \wedge \ldots, C_k$. Here, each clause contains at the most 3 literals stating that each clause possesses the length 3. Each literal may be variable or its negation (i.e. either the variables $x_1, x_2, \ldots, x_n$ or $\sim x_1, \sim x_2, \ldots \sim x_n$, indicating ~ as negation). Every literal can take the truth value as 0 or false and 1 or true. If evaluated value of the formula for a set of values is true, the formula is called as satisfiable otherwise unsatisfiable. For example, let the expression $E = (x_1 \vee \sim x_2 \vee \sim x_3) \wedge (\sim x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$ be a 3SAT formula consisting of 3 clauses and four variables (x_1, x_2, x_3, x_4) having the length of each clause as 3. Thus, the truth assignments for the expression E may include the result data sets solution as $x_1 = x_3 =$ True and $x_2 =$ False; and $x_1 = x_2 =$ True and $x_3 =$ False etc. [5].

2.2 Vertex Cover as a NP Complete Problem

Vertex Cover problem has been considered as very important in the category of graph based NP Complete problems. It covers many real world applications such as communications, civil engineering, electrical engineering, bioinformatics etc. Also, numerous applications have been observed in construction of Phylogenetic trees, phenotype identification, microarray data, network design etc. Minimum VC has attracted researchers and practitioners because of its characteristics of NP-Completeness. Also, many difficult real world problems can be formulated as instances of VC [6]. In general, VC is defined over a graph G = (V, E) where V and E represent the vertices and edges of a graph respectively. Mathematically, it may be stated that

$$G \text{ contains } VC(S) \mid e \in E$$

There should exists at least one of the end points of e belonging to |S|, where |S| must be as small as possible [7]. For example, Fig. 2.1 represents a graph G with seven vertices (with four vertices with fill color) and eight edges. It is evident that the vertices with fill color constitute the subset S

of vertices such that every edge of graph G has an endpoint in set S of vertices thereby forming minimum VC. There exist two versions of minimum VC problem namely; decision and optimization version. In decision version, we verify for the existence of VC with a specified size for a given graph. On the other hand, optimization version deals with task of finding the minimum VC [8].

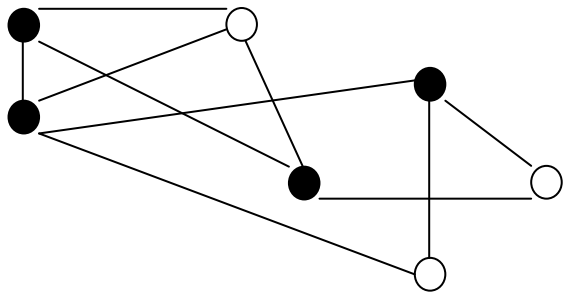

Figure 2.1: A Graph Forming Minimum VC

.

3. Proposed Work

We propose two algorithms in this paper namely; Algorithm I: *kVC_2kSAT* for converting a given graph containing VC of size k into generalized SAT formula and Algorithm II: *$kSAT_23CNF$* for converting SAT formula to 3SAT formula. The input for the Algorithm I is the adjacency matrix of the given graph. Depending upon the value at the adjacency matrix to be 1 or 0, a vertex restricted approach is followed generating kcnfclauses. The Algorithm II takes these kcnfclauses as input and generates 3SATclauses polynomially. We discuss 3SAT encoding formulation approach followed by the informal and formal description of Algorithm I and Algorithm II in the next section.

3.1 Polynomial 3SAT Encoding Formulation Approach

As per vertex restricted approach a kcnfclause is generated corresponding to each vertex of the graph having the value 1 in adjacency matrix.

$$F_v = (V_{i1} \vee V_{i2} \vee \ldots, V_{ik}) \quad (1)$$

Where V_{ic} is a vertex V_i (i = 1, 2…, n) generated as a variable of clause F_v. These generated kcnfclauses are converted to 3SATclause by a well known non-recursive method. To highlight this method we consider a clause $F = x_1 \vee x_2 \vee \ldots \vee x_k$ where (k>3) is the length of the clause. According to this method a kcnfclause is converted to a 3SATclause by introducing some extra variables

$y_1, y_2, \ldots, y_k$ as: $(x_1 \vee x_2 \vee \sim y_1) \wedge (x_3 \vee y_1 \vee \sim y_2) \wedge (x_4 \vee y_2 \vee \sim y_3) \wedge \ldots \wedge (x_{k-2} \vee y_{k-4} \vee \sim y_{k-3}) \wedge (x_{k-1} \vee x_k \vee y_{k-3})$ (2)

The expression (2) transforms each clause of length k into (k-2) clauses of length 3, which requires introducing (k-3) new variables. For example, applying (2) to a clause of length 5 yields (1 clause of length 5) = (3 clauses of length 3) with 2 additional variables. Thus, by applying (2) to (1) finally, we get a total of (k-2)*|V| 3SATclauses. Continuing in this sequence and making use of vertex restricted approach, a clause is generated a of length k in CNF corresponding to each vertex of graph having the position 0 in the adjacency matrix.

$F_{\sim v} = (\sim V_{i1} \vee \sim V_{i2} \vee \ldots \sim V_{ik})$ (3)

Where V_{ic} is a vertex V_i (i = 1, 2, … , n) generated as a variable of clause. The total number of kcnfclauses generated by these two clause generation approach is as follows:

$|F_V| + |F_{\sim V}| = (k-2)*|V| + (k-2)*|V|$ (4)

G (V, E) are polynomially encoded to 3SAT clauses.

3.2 Algorithm I: *kVC$_2$kSAT*

We present the informal, formal description and pseudo code of AlgorithmI. This Algorithm focuses on generating kcnfclauses for the input graph.

3.2.1 Informal Description

Here, we present an informal description of the Conversion algorithm *kVC$_2$kSAT*. For the input dataset of some of the prominent graphs from graph theory, we get output in terms of adjacency matrix (that consist of elements 1 and 0). The element 1 corresponds to interconnection among the vertices whereas 0 states the non-interconnection among the vertices of the graphs. We initialize kcnfclause and its Count as zero. One kcnfclause is generated for each and every row of the adjacency matrix. For the element 1 present in each row of the adjacency matrix a kcnfclause contains the column index which in turn is the vertex number of the graph for that particular position. Similarly, for the element 0 present in each row of the adjacency matrix a kcnfclause contains the negation (~) of the column index which in turn is the vertex number of the input graph. The Count is incremented as the value of j is completed. We now present the formal description depicting the logical flow of our algorithm.

3.2.2 Formal Description

Now, we present the formal description of our proposed algorithm *kVC$_2$kSAT* for generating the kcnfclauses.

Algorithm I: *kVC$_2$kSAT*

```
/* Input graph G (V, E), use its vertices and edges to generate adjacency matrix A of size n×n. */
Step 1: for i = 1 to n do
          for j = 1 to n do
               Input A[ i ][ j ];

/* Initialize clauses */
 Step 2: kcnfclauses ( ) = NULL;
          Count = 0;

/* Checking the inter-connections between vertices of graph, generating clauses and their count*/
Step3: for i =1 to n do
            Begin
          for j = 1 to n do
            Begin
          If (A[i][j] =  = 1) then
             GENERATE a kcnfclause v_ic = (v_i1 ∨ v_i2 ∨ … ∨ v_ik);
```

Else
GENERATE a kcnfclause $v_{ic} = (\sim v_{i1} \vee \sim v_{i2} \ldots \vee \sim v_{ik})$;
Count = Count + 1;
kcnfclauses = $(v_{ic}) \wedge (\sim v_{ic})$;

/* Store the kcnfclauses obtained in a file */
Step 4:
kcnfclauses.txt = kcnfclauses
End;

3.3 Algorithm II: *kSAT$_2$3CNF*

We present the informal description, formal description and pseudo code of Algorithm II. The prime focus of this Algorithm is generating 3SATclauses for the input kcnfclauses.

3.3.1 Informal Description

Here, we present the informal description of the *kSAT$_2$3CNF* algorithm which is based on one of the non-recursive method. The input is the kcnfclauses obtained by the algorithm *kVC$_2$kSAT*. In output the polynomially 3SATclauses are obtained. The aim is to convert each kclause to a 3SATclause. The kclause that is generated by the conversion algorithm *kVC$_2$kSAT* may contain any number of literals. The input kcnfclauses are stored in a file called as kcnfclauses.txt that contains all the kcnfclauses. Another file 3cnfclauses.txt is maintained to store 3SATclauses. Depending upon the condition that whether the number of literals in a k clause are equal to or greater than three the additional literals are appended.

3.3.2 Formal Description

Here, we present the formal description of the algorithm. The input for this algorithm is kcnfclauses obtained from *kVC$_2$kSAT*. The final results of 3cnfclauses are stored in a file 3cnfclauses.txt.

Algorithm II: *kSAT$_2$3CNF*
/*Input kcnfclauses.txt obtained from *kVC$_2$kSAT*; use all the k clauses from this file*/
Step1: for i = 1 to n
READ (kcnfclauses.txt);

/*Initialize the file and array used for storing the resulting 3cnfclauses */
Step2: 3cnfclauses.txt = NULL;
New [ ] = NULL;

/*Depending on the value of k or the number of literals in a k clause proceed*/
Step3: If k = 3
3cnfclause.txt = kcnfclause;
Else
If k >3
Use some additional variables $\{y_{i1}, y_{i2}, \ldots, y_{i,k-3}\}$.
Generate a new clause as
Newclause = $(x_1 \vee x_2, y_{i1}) \wedge (\sim y_{i1} \vee x_3 \vee y_{i2}) \wedge (\sim y_{i2} \vee x_4 \vee y_{i3}) \wedge \ldots \wedge (x_{k-1}, x_k, y_{i,k-2})$;
New [ ] = Newclause;
3cnfclause.txt = New [ ];

End;

3.5 Encoding a Graph to 3SAT- An Illustration

We demonstrate the working and flow of encodings with an example. Let us consider a graph consisting of ten vertices and fifteen edges in the form of dataset as

(e 1 e2, e1 e5, e1 e7, e2 e3, e2 e8, e3 e4, e3 e9, e4 e5, e4 e10, e5 e6, e6 e8, e6 e9, e7 e9, e7 e10, e8 e10).

kVC$_2$kSAT proceeds with the generation of kcnfclauses. These kcnfclauses are generated by the vertex restricted approach (1) and (3) combinedly. After generation of the kcnfclauses brightens the role of polynomial encoding of the kcnfclauses to 3SATclauses, which is accomplished by algorithm *kSAT$_2$3SAT*.

As per the vertex restricted approach (1), the kcnfclause is obtained as:

F_v = (2∨ 5∨ 7) ∧ (1∨ 3∨ 8) ∧ (2∨ 4∨ 9) ∧ (3∨ 5∨ 10) ∧ (1∨ 4∨ 6) ∧ (5∨ 8∨ 9) ∧ (1∨ 9∨ 10) ∧ (2∨ 6∨ 10) ∧ (3∨ 6∨ 7) ∧ (4∨ 7∨ 8).

Similarly the vertices are encoded as per the vertex restricted approach (3), and are stored in $F_{\sim v}$ as:

$F_{\sim v}$ = (~3∨ ~4∨ ~6∨ ~8∨ ~9∨ ~10) ∧ (~4∨ ~5∨ ~6∨ ~7∨ ~9∨ ~10) ∧ (~1∨ ~5∨ ~6∨ ~7∨ ~8∨ ~10) ∧ (~1∨ ~2∨ ~6∨ ~7∨ ~8∨ ~9) ∧ (~2∨ ~3∨ ~7∨ ~8∨ ~9∨ ~10)∧ (~1∨ ~2∨ ~3∨ ~4∨ ~7∨ ~10)∧ (~2∨ ~3∨ ~4∨ ~5∨ ~6∨ ~8)∧ (~1∨ ~3∨ ~4∨ ~5∨ ~7∨ ~9)∧ (~1∨ ~2∨ ~4∨ ~5∨ ~8∨ ~10) ∧ (~1∨ ~2∨ ~3∨ ~5∨ ~6∨ ~9).

The encoded clauses so obtained by the vertex restricted approach (1) are all in 3SAT, so there is no need to process these clauses any further with the help of Algorithm II. The encoded clauses obtained by the vertex restricted approach (3) are all the k clauses of length 6. There is a need to convert each clause to a 3SATclause by using the *kCNF$_2$3SAT* algorithm. So according to our formulation to encode each clause of length k requires (k-2) clauses each of length 3 and (k-3) additional variables. This particular case also contains each clause of length k = 6, which further generates 4 clauses of length 3 with the introduction of three additional variables. The 3SAT clauses generated by the non recursive method (2) are as follows:

(~3∨ ~4∨ ~y_1) ∧ (~6∨ y_1∨ ~y_2) ∧ (~8∨ y_2∨ ~y_3) ∧ (~9∨ ~10∨ y_3) ∧ (~4∨ ~5∨ ~y_1) ∧ (~6∨ y_1∨ ~y_2) ∧ (~7∨ y_2∨ ~y_3) ∧ (~9∨ ~10∨ y_3) ∧ (~1∨ ~5∨ ~y_1) ∧ (~6∨ y_1∨ ~y_2) ∧ (~7∨ y_2∨ ~y_3) ∧ (~8∨ ~9∨ y_3) ∧ (~1∨ ~2∨ ~y_1) ∧ (~6∨ y_1∨ ~y_2) ∧ (~7∨ y_2∨ ~y_3) ∧ (~8∨ ~9∨ y_3) ∧ (~2∨ ~3∨ ~y_1) ∧ (~7∨ y_1∨ ~y_2) ∧ (~8∨ y_2∨ ~y_3) ∧ (~9∨ ~10∨ y_3) ∧ (~1∨ ~2∨ ~y_1) ∧ (~3∨ y_1∨ ~y_2) ∧ (~4∨ y_2∨ ~y_3) ∧ (~7∨ ~10∨ y_3) ∧ (~2∨ ~3∨ ~y_1) ∧ (~4∨ y_1∨ ~y_2) ∧ (~5∨ y_2∨ ~y_3) ∧ (~6∨ ~8∨ y_3) ∧ (~1∨ ~3∨ ~y_1) ∧ (~4∨ y_1∨ ~y_2) ∧ (~5∨ y_2∨ ~y_3) ∧ (~7∨ ~9∨ y_3) ∧ (~1∨ ~2∨ ~y_1) ∧ (~4∨ y_1∨ ~y_2) ∧ (~5∨ y_2∨ ~y_3) ∧ (~8∨ ~10∨ y_3) ∧ (~1∨ ~2∨ ~y_1) ∧ (~3∨ y_1∨ ~y_2) ∧ (~5∨ y_2∨ ~y_3) ∧ (~6∨ ~9∨ y_3).

According to our formulation approach, the total numbers of clauses generated are as follows: By the first vertex restricted approach, the clause of length 3 are obtained. Thus, the total clauses generated are 10, as if we put k = 3 in the formula (3-2)*10 = 10.

By the second approach of vertex constraint, each clause of length 6 are generated, thus the total clauses generated are 40, as we put k=6 in the formula (6-2)*10 = 40. Thus using the equation (5) we are able to perform the polynomial encoding of a graph to 3CNFSAT formula.

$|F_V| + |F_{\sim V}| = (k-2)*|v| + (k-2)*|v|$ (5)

10 + 40 = 50, this is a polynomial encoding of graph to a 3SAT formula.

4. Use Of 3sat Solver

To check the satisfiability of the resulting 3SAT formula, the 3SAT formula so obtained is passed to the SAT solver. There are many 3SAT solver tools available to check the satisfiability of the SAT formula. We have made use of the online MiniSAT solver to check its satisfiability. The solver accepts the 3SAT formula in the solver compatible format or the Dimacs format. The following steps indicate the use of 3SAT solver to accept 3SAT formula and announce the results.

Step1: To pass the 3cnfclauses.txt to the 3SAT solver, in a file (Solver.txt) separate each variable of the clause.

Step 2: Pass the clauses to the solver in the CNF compatible format.

a. Execute the command
   p cnf number of variables number of clauses.
b. Each separated variables of the clause ends with a zero character.
c. Each clause ends with a zero character.

Step 3: The solver returns satisfiable values for the 3CNFSAT formula.

Step 4: Verify the satisfiable values against the vertices of the Vertex Cover.

Step 5: If there is a Vertex Cover in a graph, the formula is satisfiable.

5. Results And Discussion

We have implemented formulation on some important graphs of graph theory [9]. Table 4.1 depicts the encoding results of Algorithm I and Algorithm II. Graph acts as an input and the processing of Algorithm I yields the kcnfclauses. These clauses so obtained are processed with Algorithm II and the 3SATclauses are generated. 3SATclauses are passed to the 3SAT Solver which provides the satisfiable values for the formula. On checking these satisfiable values against the minimum VC, it is observed that both are the same. In Table 4.1 one of the input graph is the Peterson graph, which consists of 10 vertices and 60 edges. Processing this graph, on Algorithm I and Algorithm II the total 3SATclauses obtained are 60. On passing these clauses of the 3SAT formula to the Solver, the 6 satisfiable values fetched are the minimum VC for the Peterson graph.

We formulated a novel approach of encoding graphs to 3SATclauses. With this we made it possible to encode a graph to a 3SAT formula. With our approach we are able to encode graphs to first the kcnfclauses and then finally to 3SATclauses. The satisfiable values that are announced by the 3SAT solver are same as the minimum VC vertices. This encoding makes it possible to find out the solutions to 3SAT problem and the minimum Vertex Cover problem.

Table 4.1: Execution of 3SAT Formula on Various Graphs

S. No	Graph Name	Number of Vertices	Number of Edges	3SAT Encoded Clauses	Number of Satisfiable Values	Vertex Cover Vertices
1.	*Kuratowski*	6	12	12	03	03
2.	*Octahedron*	6	24	24	04	04
3.	*Bondy*	7	20	20	04	04

	Murty(G1)					
4.	***Peterson***	**10**	**60**	**60**	**06**	**06**
5.	*Bondy Murty(G2)*	11	34	34	07	07
6.	*Hershel*	11	14	14	05	05
7.	*Icosahedron*	12	36	36	09	09
8.	*Bondy Murty(G3)*	14	14	14	07	07
9.	*Bondy Murty(G4)*	16	39	39	07	07
10.	*Ramsey*	17	102	102	08	08

References

[1] Karp, Reducibility Among Combinatorial Problems, Complexity of Computer Computations. (1972) Plenum Press .85-103.

[2] N.S Chaudhari, Computationally Hard Problems: 3SAT and its Polynomial Solvability Journal of Indian Academy of Mathematics. 31 (2) December (2009) 407-444.

[3] N.Dahale, N.S.Chaudhari, M. Ingle, CLISAT-Clique Encodings in Implementation of SAT, Mathematical Sciences International Journal. 3(2) (2014) 565-567.

[4] R.Sanchez, The Boolean Satisfiability Problem (SAT), Design and Analysis of Algorithms. 1(1) (2010).111 - 117.

[5] Biere, Sinz, Carsten, Decomposing SAT Problems into Connected Components, Journal of Satisfiability, Boolean Modelling and Computation. 2 (2006). 210 - 208.

[6] Hassin, R.Levin, The Minimum Generalized Vertex Cover Problem, ACM Transactions on Algorithms. (2). 66-78.

[7] Aho,Hopcroft,J.E.Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading MA

[8] M. Pelikan, Hierarchical Bayesian Optimization Algorithm: Towards a New Generation of Optimization Algorithms, Springer Verlag, 2005.

[9] J.A. Bondy, U.S.R. Murty, Graph Theory with Applications, Elsevier Science Publishing Co. (1976).