ISSN No. 0976-5697

Volume 7, No. 2, March-April 2016

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

GCP Constraint Based SAT Clause Generation in Polynomial Time

Nidhi Dahale
FCA Department
Acropolis Institute of Technology and Research
Indore (M.P), India

N.S. Chaudhari
Director and Professor
VNIT Nagpur and IIT Indore
Indore (M.P), India

Maya Ingle
Professor and Sr. System Analyst
SCSIT DAVV
Indore (M.P), India

Abstract: Graph Colorability Problem (GCP) belongs to a subset of the large family of NP–Hard combinatorial problems. Till now there exist very less deterministic methods to find the solutions to k- Color problem. Constraint analysis of the k-Color problem and formulating the constraints to the SAT clauses provides another way to find the solution to this typical problem. The recent advancement in the development of SAT solvers has really provided an easy approach to find the truth values for a SAT formula. Thus, we have performed the formulations of the constraints of GCP into ALOC, AMOC and DCOL clauses. With our formulations, we are able to generate the SAT clauses in polynomial time. The total SAT clauses that are being generated with our formulations are | V | + | V |* k*(k-1) / 2 + (m*E).

Keywords: SAT, Graph Coloring, Encodings, Clauses, Polynomial Time

I. INTRODUCTION

A proper graph coloring emphasizes on coloring the vertices of the graph G (V, E) in such a way that two vertices (u, v) of a graph which share an edge must have different colors. The minimum number of colors required to color the vertices is known as Chromatic Number for that graph and is denoted by χ**(G)**. In the proper k coloring environment, graph G (V, E) is termed to be colored with available k colors if its neighboring vertices have different colors. Deciding the k-Colorability of a graph is an NP Complete problem [1] [2].

There exist a few deterministic methods that can solve the k-Colorability of a graph in polynomial time. Satisfiability (SAT) has been a major concern to solve the k-Colorability of the graph in many ways. It has been observed that an encoding exists on the vertices and edges of the graph with 3-color that generates a Disjunctive Normal Form (DNF) clauses in SAT. This DNF expression may be enhanced to a k-CNF formula by using recursive method. Further, these k-CNF clauses with the use of some non- recursive methods are converted into the 3-CNF expression. It is observed that total clauses generated in 3-CNF formula for 3- colorable graphs are ((27*|*V*|) + (256*|*E*|)) [3]. Using some generalizations and input of this method are utilized to generates a total (k^k (k-2)*|V|) + (2^{2k}+2 *|E|)) 3CNFSAT clauses and is also applied to solve the Channel Assignment problem in cellular networks [4].

The encodings provide a very important framework of converting one problem instance to the other. In this paper, we have introduced a new encoding from k-Color graphs to SAT formula in polynomial time. This encoding results in the generation of SAT clauses on the basis of GCP constraints. We have also presented, algorithm *COLSAT* to formulate our encoding. We discuss the background details of k-Color problem and SAT in Section II. In Section III, our encoding methodology, formulations along with *COLSAT* algorithm is discussed. DIMACS provides a set of challenging datasets for the k-Color problem [5] [6]. We cover the results of our encodings on DIMACS benchmarks graphs in Section IV. Finally, we conclude with conclusions in Section V.

II. BACKGROUND

We now discuss the background details of k-Color and SAT problem in this section as follows:

2.1 k - Color Problem

Graph Coloring Problem (GCP) is a very important problem in the category of graph based NP-Hard Combinatorial problems. Formally, a GCP is denoted by {< G, k >} where G represents a graph and k denotes the minimum colors required to color the vertices V. In other words, GCP emphasizes the assignment of k colors to the vertices of the graph such that adjacent vertices receive different colors. Mathematically, a k-Coloring problem is defined on the basis of a coloring function C^k. The coloring function C^k need to assign k color values to the vertices of the graph. This assignment is stated as C^k: V → {1, 2, … , k} [7].

The most widespread utility of GCP is as an optimization problem. GCP as an optimization problem for a graph G (V, E) is defined as follows: Let there be V- set of graph vertices denoted by |V| = n, E- set of connected edges denoted by |E| = m and a minimum positive integer k such as k ≤ n. Specifically, GCP is to find the minimum positive integer k and a coloring function C^k : V→ {1, 2, … , k} such as c(u) ≠ c(v) where u, v are the vertices of the graph G (V, E) [8].

Many other variants of optimized GCP exist in literature such as Equitable Coloring, Sum Coloring, Contrast Coloring, Harmonious Coloring, Circular Coloring, Consecutive Coloring, List Coloring etc. In optimization version of the basic problem called GCP, a conflict–free coloring with minimum number of colors is searched [9]. Intensive research conducted in this area resulted in a large number of exact and approximate algorithms, heuristics and metaheuristics. However, the reported results are often difficult to compare due to specific assumptions, different algorithms and their implementation details, tuning of parameters, computing platforms, test data sets etc. [10, 11, 12, 13, 14].

Thus, graph Coloring and its generalizations are useful tools in modeling a variety of scheduling and assignment problems. There are several interesting practical problems that can be modeled by graph coloring. Aircraft Scheduling, Bioprocesses Tasks and Frequency Assignment are also not untouched with k-Color magic [15].

2.2 SAT Problem

Let there be a function $f: \{0,1\}^n \rightarrow \{0,1\}$, i.e. $f(x_1, \dots, x_m) \in \{0,1\}$. There is an assignment 0's and1's, $a_1, \dots, a_m$, to the variables $x_1, \dots, x_m$ such that $f(a_1, \dots, a_m) = 1$. If there is such an assignment then f is satisfiable and $a_1, \dots, a_m$ is called as satisfying assignment. If no such assignment then f is unsatisfiable. Any function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ can be expressed in Conjunctive Normal Form (CNF) as $F = C_0 \wedge C_1 \wedge \dots \wedge C_{m-1}$. A clause is a disjunction of literals as $(C_i = l_{i0} \vee \dots \vee l_{i\,(s-1)})$. A literal is either a variable or a negated variable $l_{ij} = v_{ij}$ or $\neg v_{ij}$.

There exists mainly four versions of SAT namely; Max-SAT, K-SAT, 2-SAT and 3-SAT. The SAT problem converts to an optimization problem by maximizing the number of clauses that one assignment can satisfy. Such a SAT problem leads to Maximum Satisfiability Problem (MAX-SAT). K-SAT has all clauses in CNF with k literal per clause. 2-SAT is the problem which is solvable in polynomial time, with each clause is restricted to two literals only. 3SAT is the first problem to be NP-Complete, with every clause has only three literals [16].

Vivid range of other naturally occurring decision and optimization problems can be transformed into SAT instances. This is possible due to revolutionary advances in the range of SAT solvers available for solving huge SAT instances. A class of algorithms called SAT solvers can efficiently solve a large enough subset of SAT instances. The advancement of the SAT solvers is one of the reasons for rise in the number of practical applications of SAT. Many practical applications of SAT occur in model checking, Automatic Test Pattern Generation, Combinatorial Equivalence, Planning in AI, Automated Theorem Proving, Software Verification, Haplotype Inference etc. [17].

III. PROPOSED WORK

Let there be a graph G (V, E) and a positive integer k as available colors to color the vertices of the graph. Where, V is a set of n vertices $V\{v_1, v_2, \dots, v_n\}$ and E is the set of m edges $E\{e_1, e_2, \dots, e_m\}$. An encoding is performed on a graph to obtain SAT formula. To perform the encoding constraint analysis of GCP and its formalization to obtain SAT clauses is executed.

3.1 Analysis of the Constraints of GCP Problem

The constraints are analyzed into three crucial variants as: At Least One Color (ALOC), At Most One Color (AMOC) and Different Color (DCOL). The constraint ALOC is use to color each vertex of the graph. In other words, no vertex should be left uncolored. AMOC is used to avoid the redundant color assignments to any vertex of the graph, as there exists a possibility of a vertex assigned with the multiple colors. DCOL is use to assign different colors to the adjacent or neighboring vertices of a graph. In order to perform encoding and obtain SAT clauses we initialize a Boolean variable x_{ij}, where $1 \le i \le |V|$ and $1 \le j \le k$ is true if the node v_i is assigned a color j. This assignment generates k |V| variables for SAT formula. We formalize the constraints ALOC, AMOC and DCOL to obtain SAT clauses as follows:

3.2. Categorize and Formalize the Constraints for SAT Clause Generation:

(I). At Least One Color (ALOC)

The intention is to have at least one color for each vertex. Out of all the variables available from the initialization at least one should be true. This generates one clause for each vertex, resulting a total of |V| clauses for the graph. The generalize representation of a clause for the vertex v_i is as follows:

$$\wedge (x_{i,1} \vee x_{i,2} \vee \dots \vee x_{i,k}) \qquad \text{(eq.1)}$$

(II). At Most One Color (AMOC)

There exists a possibility that SAT checker might come up with the assignment of more than two colors for a single vertex. To avoid such assignments, clauses are generated emphasizing the fact that vertex v_i is assigned at most one color. It is asserted definitely, that if a vertex v_i is assigned a color m, then it is not be assigned a color n i.e. $c(m) \neq c(n)$. This particular constraint gives rise to generation of simple two literal clauses of the form: $(x_{i,m} \rightarrow \neg x_{i,n}) \equiv (\neg x_{i,m} \vee \neg x_{i,n})$. For k available colors, total (k * (k-1)/2) SAT clauses are generated. For V vertices of a graph, |V| *(k * (k-1)/2) clauses is the contribution of AMOC constraint. The generalize representation for a SAT clause for the vertex v_i is as follows:

$$\wedge \wedge \qquad (\neg x_{i,m} \vee \neg x_{i,n}) \qquad \text{(eq.2)}$$
$$1 \le m \le k \quad m+1 \le n \le k$$

(III). Different Color (DCOL)

This is the most important constraint for the k-Color problem. Basically, DCOL is based upon the most important aspect of k-color problem. It is based on condition that two vertex of a graph connected by an edge have different colors. If the vertex (v_i, v_j) are connected with an edge e, then v_i and v_j should not be colored with m. Thus for every edge $(v_i, v_j) \in E$, if v_i is colored with m then the v_j should not be colored with the color m. This constraint generates a two literal clause as $(x_{i,m} \rightarrow \neg x_{j,m}) \equiv (\neg x_{i,m} \vee \neg x_{j,m})$. Further, when this constraint is placed for each color m, generates total (m * E) clauses for all edges of the graph. All the SAT clauses for the edge (v_i, v_j) are as follows:

$$\wedge \ (\neg x_{i,m} \vee \neg x_{i,m}) \qquad \text{(eq.3)}$$
$$1 \le c \le k$$

With these formulations, the total number of variables required to encode k-Color to SAT is |V| k. Besides this, our formulations also generate ALOC, AMOC and DLOC

clauses as |V|, |V| *(k *(k-1)/2) and (m*E) respectively. Thus, the Total SAT clauses (TSC) generated with these formulations are as follows:

$$TSC = |V| + |V| *(k*(k-1) / 2) + (m*E) \quad \text{(eq.4)}$$

On the basis of the above defined constraints formulations an algorithm *COLSAT* is proposed. The algorithm takes an input graph G (V, E) and available colors k. *COLSAT* generates ALOC, AMOC and DCOL clauses in polynomial time as output.

3.3 Algorithm: COLSAT

/* Input graph dataset G; use its vertices and edges, number of colors available k and a Boolean variable x_{ij}*/

Step 1: for i = 1 to *V* do
for j = 1 to *k* do
input(x_{ij})

/* Initialize clauses */
ALOC Clause ( ) = NULL;
AMOC Clause ( ) = NULL;
DCOL Clause ( ) = NULL;
Total Clause ( ) = NULL;

/* Generate *ALOC Clause* and their *Count* */

Step 2: for j = 1 to *k* do
begin
ALOC Clause = NULL;
for i = 1 to *V* do
begin
ALOC Clause = $\wedge$ ($x_{i,1} \vee x_{i,2} \vee \ldots \vee x_{i,k}$)
end;
ALOC Clause = *ALOC Clause* + 1;
end;

/* Generate *ALOC Clause* and their *Count* */

Step 3: for j = 1 to *k* do
begin
AMOC Clause = NULL;
for i = 1 to *V* do
begin
AMOC Clause = $\wedge$ ($\neg x_{i,m} \vee \neg x_{i,n}$)
end;
AMOC Clause = *AMOC Clause* + 1;
end;

/* Generate *DCOL Clause* and their *Count* */

Step 4: for j = 1 to *k* do
begin
DCOL Clause = NULL;
for i = 1 to *V* do
begin
DCOL Clause = $\wedge$ ($\neg x_{i,m} \vee \neg x_{i,m}$)
end;
DCOL Clause = *DCOL Clause* + 1;
end;

/* Generate *Total Clause* and their Count*/
Step 5:
begin
Total Clause = (*ALOC Clause* + *AMOC Clause* + *DCOL Clause*);
end;

Example SCG: SAT Clause Generation

In this example *COLSAT* is applied on a G (V, E) that consists of 11 vertices and 20 edges connections. The value of k i.e. the available colors to color vertices of the graph is k = 3. The small scale Dimacs dataset to elaborate this case is as follows:

(e1, e2), (e2, e3), (e3, e4), (e4, e5), (e5, e6),
(e6, e7), (e7, e8), (e8, e9), (e1, e9), (e2, e10),
(e3, e10), (e4, e10), (e5, e10), (e6, e10),
(e6, e11), (e8, e11), (e9, e11), (e1, e11)
(e10, e11), (e1, e10).

On the basis of the constraint based formulations for the k color problem, the ALOC, AMOC and DCOL constraints are generated as follows:

The ALOC clauses are generated using the eq.1 for every vertex. With the formulations, the total ALOC clauses contribution is equal to 11. The ALOC clause generation for the Dimacs dataset is depicted in Table 1. Similarly, |V|*(k*(k-1)/2) total AMOC clauses for the given dataset expending eq.2 are generated. The generation of AMOC clauses is presented in Table 1. Subsequently, to generate DCOL clauses with our formulations eq. 3 is utilized. Noticeable feature behind the generation of DCOL clauses is the basis of edge connections among the number of vertices. The obvious result of the increase in the number of DCOL clauses is the dependency on the total number of edge connections. The DCOL clause generation is depicted in Table 2.

IV. RESULTS

The implementation of our formulations are depicted on the datasets from Dimacs benchmark problems for k-Color graphs in Table 3. It must be observed that, Dimacs maintains a huge range of challenging datasets for k-Color graphs. Our results depict the generation of SAT clauses in polynomial time for k = 3 on nine different datasets. The generation of clauses for one of the data set containing 11 vertices and 20 edges has been elaborated in the example. Besides the total clauses generated, the results also state the number of variables required to generate the SAT clauses. Our results generate the ALOC, AMOC, DCOL clauses in polynomial time, providing it a novel structure. Thus, analyzing and formalizing the GCP constraints leads a polynomial time generation of SAT clauses.

V. CONCLUSION

A novel approach of encoding color graphs to CNF SAT clauses has been accomplished. With our formulations we made it possible to reduce the instances of one problem to another. The graph datasets are reduced to CNF clauses by our *COLSAT* algorithm. The polynomial generation of SAT clauses on the basis of GCP constraints was shaped by us. It was also established, that the various constraints ALOC, AMOC and DCOL of GCP laid a foundation to generate strictly correlated SAT clauses.

Table 1: ALOC and AMOC Clauses for Example SCG

Vertex	ALOC Clauses	AMOC Clauses
1	$(x_{1,1} \vee x_{1,2} \vee x_{1,3})$	$(\neg x_{1,1} \vee \neg x_{1,2}) \wedge (\neg x_{1,1} \vee \neg x_{1,3}) \wedge (\neg x_{1,2} \vee \neg x_{1,3})$
2	$(x_{2,1} \vee x_{2,2} \vee x_{2,3})$	$(\neg x_{2,1} \vee \neg x_{2,2}) \wedge (\neg x_{2,1} \vee \neg x_{2,3}) \wedge (\neg x_{2,2} \vee \neg x_{2,3})$
3	$(x_{3,1} \vee x_{3,2} \vee x_{3,3})$	$(\neg x_{3,1} \vee \neg x_{3,2}) \wedge (\neg x_{3,1} \vee \neg x_{3,3}) \wedge (\neg x_{3,2} \vee \neg x_{3,3})$
4	$(x_{4,1} \vee x_{4,2} \vee x_{4,3})$	$(\neg x_{4,1} \vee \neg x_{4,2}) \wedge (\neg x_{4,1} \vee \neg x_{4,3}) \wedge (\neg x_{4,2} \vee \neg x_{4,3})$
5	$(x_{5,1} \vee x_{5,2} \vee x_{5,3})$	$(\neg x_{5,1} \vee \neg x_{5,2}) \wedge (\neg x_{5,1} \vee \neg x_{5,3}) \wedge (\neg x_{5,2} \vee \neg x_{5,3})$
6	$(x_{6,1} \vee x_{6,2} \vee x_{6,3})$	$(\neg x_{6,1} \vee \neg x_{6,2}) \wedge (\neg x_{6,1} \vee \neg x_{6,3}) \wedge (\neg x_{6,2} \vee \neg x_{6,3})$
7	$(x_{7,1} \vee x_{7,2} \vee x_{7,3})$	$(\neg x_{7,1} \vee \neg x_{7,2}) \wedge (\neg x_{7,1} \vee \neg x_{7,3}) \wedge (\neg x_{7,2} \vee \neg x_{7,3})$
8	$(x_{8,1} \vee x_{8,2} \vee x_{8,3})$	$(\neg x_{8,1} \vee \neg x_{8,2}) \wedge (\neg x_{8,1} \vee \neg x_{8,3}) \wedge (\neg x_{8,2} \vee \neg x_{8,3})$
9	$(x_{9,1} \vee x_{9,2} \vee x_{9,3})$	$(\neg x_{9,1} \vee \neg x_{9,2}) \wedge (\neg x_{9,1} \vee \neg x_{9,3}) \wedge (\neg x_{9,2} \vee \neg x_{9,3})$
10	$(x_{10,1} \vee x_{10,2} \vee x_{10,3})$	$(\neg x_{10,1} \vee \neg x_{10,2}) \wedge (\neg x_{10,1} \vee \neg x_{10,3}) \wedge (\neg x_{10,2} \vee \neg x_{10,3})$
Total	11	33

VI. REFERENCES

[1] Garey, R. Johnson D. S. "Computers and Intractability - A Guide to the Theory of NP Completeness", Freeman, 1979.

[2] Karp R. M. "Reducibility Among Combinatorial Problems", In: Miller R. E. and Thatcher J. W. (Eds.), Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[3] Alexander Tsiatas, "Phase Transitions in Boolean Satisfiability and Graph Coloring", May 2008, Department of Computer Science, Cornell University.

(www.cseweb.ucsd.edu/users/atsiatas/phase.pdf)

[4] Prakash C. Sharma and Narendra S. Chaudhari, "A Graph Coloring Approach for Channel Assignment in Cellular Network via Propositional Satisfiability", International Conference on Emerging Trends in Networks and Computer Communications (ETNCC) at Udaipur, 22-24 April 2011, pp. 23-26.

[5] DIMACS Implementation Challenges, http://dimacs.rutgers .edu/Challenges/.

[6] Graph Coloring Instances, http://mat.gsia.cmu.edu/ COLOR/instances.html.

[7] M. Anathanarayanan, S.Lavanya, "Fuzzy Graph Coloring Using α Cuts", International Journal of Engineering and Applied Sciences, March , 2014,Vol.4,No.90.

[8] Jensen, T. R.—Toft, B., "Graph coloring problems", Wiley Interscience, 1995.

[9] Kubale, M. (Ed.), "Graph Colorings, American Mathematical Society, 2004. DOI: 10.1090/conm/352

[10] Croitoriu, C.—Luchian, H.—Gheorghies, O.—Apetrei A, "A new Genetic Graph Coloring Heuristic", Computational Symposium on Graph Coloring and Generalizations COLOR'02.In: Constraint Programming, Proceedings of the International Conference, P'02, 2002.

[11] Filho, G. R.,—Lorena, L. A. N.,"Constructive Genetic Algorithm and Column Generation: An Application to Graph Coloring", Proc. Asia Pacific Operations Research Symposium, APORS'2000, 2000.

[12] Fleurent, C.—Ferland, J. A., "Genetic and Hybrid Algorithms for Graph Coloring", Annals of Operations Research Vol. 63, 1996, pp. 437–461.DOI: 10.1007/BF02125407.

[13] Khuri, S.Walters, T.Sugono, "Grouping Genetic Algorithm for Coloring Edges of Graph", Proc. 2000 ACM Symposium on Applied Computing, 2000, pp. 422–427. DOI: 10.1145/335603.335880.

[14] Kubale M, "Introduction to Computational Complexity and Algorithmic Graph Coloring", GTN, Gda´nsk, 1998.

[15] Daniel Marx, "Graph Colouring Problems and their Applications in Scheduling", Periodica Polytechnica Ser El. Eng Vol.48, No.1, pp. 11-16 (2004).

[16] E´en, N., S¨orensson N., "An extensible SAT-solver" In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003, LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004).

[17] A. Slater, Investigations into Satisfiability Search", PhD thesis, NICTA, Australian National University, Acton, Australia, 2003.

[18] N.Een and A. Biere, "Effective Preprocessing in SAT through Variable and Clause Elimination", In Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05).

Table 2: DCOL Clause Generation for Example SCG

Edge	DCOL	Edge	DCOL
(e1, e2)	$(\neg x_{1,1} \vee \neg x_{2,1}) \wedge (\neg x_{1,2} \vee \neg x_{2,2}) \wedge (\neg x_{1,3} \wedge \neg x_{2,3})$	(e3, e10)	$(\neg x_{3,1} \vee \neg x_{10,1}) \wedge (\neg x_{3,2} \vee \neg x_{10,2}) \wedge (\neg x_{3,3} \wedge \neg x_{10,3})$
(e2, e3)	$(\neg x_{2,1} \vee \neg x_{3,1}) \wedge (\neg x_{2,2} \vee \neg x_{3,2}) \wedge (\neg x_{2,3} \wedge \neg x_{3,3})$.	(e4, e10)	$(\neg x_{4,1} \vee \neg x_{10,1}) \wedge (\neg x_{4,2} \vee \neg x_{10,2}) \wedge (\neg x_{4,3} \wedge \neg x_{10,3})$
(e3, e4)	$(\neg x_{3,1} \vee \neg x_{4,1}) \wedge (\neg x_{3,2} \vee \neg x_{4,2}) \wedge (\neg x_{3,3} \wedge \neg x_{4,3})$	(e5, e10)	$(\neg x_{5,1} \vee \neg x_{10,1}) \wedge (\neg x_{5,2} \vee \neg x_{10,2}) \wedge (\neg x_{5,3} \wedge \neg x_{10,3})$
(e4, e5)	$(\neg x_{4,1} \vee \neg x_{5,1}) \wedge (\neg x_{4,2} \vee \neg x_{5,2}) \wedge (\neg x_{4,3} \wedge \neg x_{5,3})$	(e6, e10)	$(\neg x_{6,1} \vee \neg x_{10,1}) \wedge (\neg x_{6,2} \vee \neg x_{10,2}) \wedge (\neg x_{6,3} \wedge \neg x_{10,3})$
(e5, e6)	$(\neg x_{5,1} \vee \neg x_{6,1}) \wedge (\neg x_{5,2} \vee \neg x_{6,2}) \wedge (\neg x_{5,3} \wedge \neg x_{6,3})$.	(e6,e11)	$(\neg x_{6,1} \vee \neg x_{11,1}) \wedge (\neg x_{6,2} \vee \neg x_{11,2}) \wedge (\neg x_{6,3} \wedge \neg x_{11,3})$.
(e6, e7)	$(\neg x_{6,1} \vee \neg x_{7,1}) \wedge (\neg x_{6,2} \vee \neg x_{7,2}) \wedge (\neg x_{6,3} \wedge \neg x_{7,3})$	(e8,e11)	$(\neg x_{8,1} \vee \neg x_{11,1}) \wedge (\neg x_{8,2} \vee \neg x_{11,2}) \wedge (\neg x_{8,3} \wedge \neg x_{11,3})$
(e7, e8)	$(\neg x_{7,1} \vee \neg x_{8,1}) \wedge (\neg x_{7,2} \vee \neg x_{8,2}) \wedge (\neg x_{7,3} \wedge \neg x_{8,3})$	(e9,e11)	$(\neg x_{9,1} \vee \neg x_{11,1}) \wedge (\neg x_{9,2} \vee \neg x_{11,2}) \wedge (\neg x_{9,3} \wedge \neg x_{11,3})$
(e8, e9)	$(\neg x_{8,1} \vee \neg x_{9,1}) \wedge (\neg x_{8,2} \vee \neg x_{9,2}) \wedge (\neg x_{8,3} \wedge \neg x_{9,3})$	(e1,e11)	$(\neg x_{1,1} \vee \neg x_{11,1}) \wedge (\neg x_{1,2} \vee \neg x_{11,2}) \wedge (\neg x_{1,3} \wedge \neg x_{11,3})$
(e1, e9)	$(\neg x_{1,1} \vee \neg x_{9,1}) \wedge (\neg x_{1,2} \vee \neg x_{9,2}) \wedge (\neg x_{1,3} \wedge \neg x_{9,3})$	(e10,e11)	$(\neg x_{10,1} \vee \neg x_{11,1}) \wedge (\neg x_{10,2} \vee \neg x_{11,2}) \wedge (\neg x_{10,3} \wedge \neg x_{11,3})$
(e2, e10)	: $(\neg x_{2,1} \vee \neg x_{10,1}) \wedge (\neg x_{2,2} \vee \neg x_{10,2}) \wedge (\neg x_{2,3} \wedge \neg x_{10,3})$	(e1,e10)	$(\neg x_{1,1} \vee \neg x_{10,1}) \wedge (\neg x_{1,2} \vee \neg x_{10,2}) \wedge (\neg x_{1,3} \wedge \neg x_{10,3})$

Table 3: ALOC, AMOC, DCOL and Total Clauses Generated for K=3 on Dimacs Datasets.

S.No	Dataset	V	E	Total Variables Generated	ALOC Clauses	AMOC Clauses	DLOC Clauses	Total Clauses
1.	**myciel 13**	**11**	**20**	**33**	**11**	**33**	**60**	**104**
2.	**myciel 14**	23	71	69	23	69	213	305
3.	**queen5_5**	25	160	75	25	75	480	580
4.	**queen6_6**	36	290	108	36	108	870	1014
5.	**myciel 15**	47	236	141	47	141	708	896
6.	**queen7_7**	49	476	147	49	147	1428	1624
7.	**myciel 16**	95	755	285	95	285	2265	2645
8.	**mugg100**	100	166	300	100	300	498	898
9.	**miles250**	128	387	384	128	384	1161	1673