CHAPTER 5

K-CCNF And SAT Complexity. **

5.1 Introduction

The concept of NP-complete problem had been introduced by S.A. Cook in 1971 [47]. It has been found that the complexity theory regarding the SAT problems was the first known NP-complete problem which determines whether Boolean formula having AND, NOT, OR gates and \( v_1, v_2, \ldots, v_n \) variables with parenthesis, has a satisfiable assignment.

It is known that a Boolean formula is in k-conjunctive normal form if it is in a conjunction of \( m \) clauses like \( c_1, c_2, \ldots, c_m \) and each clause contains exactly \( k \) variables or their negation. K.Subramani [99] has studied the computational of three types of queries relating to satisfiability, equivalence and hull inclusion. He has formulated to study the NP-completeness for 3CNF which are Not –All-Equal satisfiability. Satisfiability of 2-CNF formula has been checked in polynomial time algorithm and 3-CNF satisfiability formula has been found to be NP-complete.

It has already known that the SATs have many practical applications in planning, circuit design, spin-glass model, molecular biology [100-102]. Many research work on 3-SAT has been reported. Many exact and heuristic algorithms have been introduced. Exact algorithms can determine whether a problem is satisfiable or unsatisfiable and

** Published in Proc., the Second International Conference on Computer Application, Pondicherry, India, Volume 2, January 2012, pp.2-5
this type of algorithms has an exponential worst-case time complexity. Heuristic algorithms can determine a problem quickly but they are not guaranteed to give a definite solution to all problems.

There are so many examples relating to exact algorithms. It is known that the splitting algorithms reduces the problem for the input formula \( F \) to the problem for polynomial many formulas \( F_1, F_2, \ldots, F_p \) and make a recursive call for each or one of \( F_i \)'s [108]. In addition to it has been found as an example that the heuristic algorithms are stochastic local search (SLS) and evolutionary algorithms (EAs). One heuristic algorithm, known as EF_3SAT for solving 3-SAT problem has been forwarded by Istvan Borgulya [103].

Let us remind some known definitions which are related to our present discussion as follows.

**Definition 1:** Given a Boolean formula \( \varphi \), does \( \varphi \) have a satisfying assignment such that at least one literal in each clause is set to false. If such an assignment exists, then it is called a NAE-satisfying assignment and \( \varphi \) is said to be Not-All-Equal satisfiable (NAESAT).

**Definition 2:** Given a Boolean formula \( \varphi \), does \( \varphi \) have a satisfying assignment such that at least one literal in each clause is set to true. If such an assignment exists, then it is called an X-satisfying assignment and \( \varphi \) is said to be X-satisfiable (XSAT).

**Definition 3:** If a Boolean formula \( \varphi \) is unsatisfiable, then it is neither NAE-satisfiable nor X-satisfiable however, satisfiability does not imply NAE-satisfiable nor X-satisfiability.

**Definition 4:** A Boolean formula \( \varphi \) is Horn, if every clause contains at most one positive literal.
Definition 5: A 3IFF formula \( R \) is a conjunction of 2CNF clauses and of 3IFF clauses of the form \((A \lor B) \leftrightarrow C\) where \(A, B\) and \(C\) are literals of three distinct variables \(a, b, c\) respectively; variable \(a\) and \(b\) are input variables and \(c\) is the conclusion variable [7].

Definition 6: A 3IFFCNF formula \( S \) is the CNF formula obtained from a 3IFF formula \( R \) by representing each 3IFF clause \((A \lor B) \leftrightarrow C\) by the three equivalent CNF clauses \(-A \lor C, -B \lor C\) and \(A \lor B \lor \neg C\) [7].

Definition 7: The problems 2SAT (resp. 3SAT, 3IFFSAT) are special cases where \(S\) is 2CNF (resp. 3CNF, 3IFFSAT). The problem ONE-IN-THREE 3SAT has as input a 3CNF formula \(S\) and the question “Can \(S\) be satisfied such that in each clause exactly one literal evaluates to True?” is to be answered [104].

It is also known that many definitions related to SAT and CNF have been introduced. SAT theoretical approaches to categorize classes of SAT problems as polynomial time solvable or NP-complete have already been established [105]. Tractable families are distinguished by a specific clausal structure. These structures either limit the number of literals per clause [106-107] or the number of times that a variable appears across all the clause [64]. In this chapter we present a new definitions K- canonical conjunctive normal form and full K-CCNF formula with an exact algorithm to solve the problem.

5.2 K-CCNF FORM:

Definition(a): A literal in a Boolean formula is an occurrence of a variable or its negation. A Boolean formula is called k-canonical conjunctive normal form or k-CCNF, if it is expressed as an AND of clauses, each of which is the OR of \(k\) literals iff there are \(k\) variables involved in the Boolean formula and in each clause all \(k\) variables appear.
Definition(b): Satisfiability of Boolean formula in k-CCNF is called full iff there are $2^k$ distinct clauses.

**Theorem 5.2.1:** Satisfiability of Boolean formulas in K-canonical conjunctive normal form belongs to NP iff $K \geq 3$.

Proof: Suppose a certificate is given to an n-canonical conjunctive normal form where $n \geq 3$. Here certificate means an assignment to the n variables. It is known that the n-conjunctive normal form belongs to NP when it can be checked in polynomial time algorithm. We now consider the following example for K-CCN form to present that this K-CCNF can also be checked in polynomial time algorithm and this will lead to proof our theorem.

Let us consider a K-canonical conjunctive normal form where $K=4$ as given below.

$$
\emptyset = (x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor \neg x_2 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor x_2 \lor \neg x_3 \lor \neg x_4) \land (\neg x_1 \lor \neg x_2 \lor x_3 \lor \neg x_4)
$$

Here one satisfying assignment to these four variables is $x_1=1$, $x_2=1$, $x_3=1$, $x_4=1$, since

$$
\emptyset = (1 \lor 1 \lor 1 \lor 1) \land (1 \lor \neg 1 \lor \neg 1 \lor \neg 1) \land (\neg 1 \lor 1 \lor \neg 1 \lor \neg 1) \land (\neg 1 \lor \neg 1 \lor 1 \lor \neg 1)
\land 1 \lor \neg (1)
= (1 \lor 1 \lor 1 \lor 1) \land (1 \lor 0 \lor 0 \lor 0) \land (0 \lor 1 \lor 0 \lor 0) \land (0 \lor 0 \lor 1 \lor 0)
= 1 \land 1 \land 1 \land 1
= 1
$$

In the above example, there are $K=4$ variables. Hence if there are n variables, each clause takes $O(n)$ times and if there are n clauses then total time taken by all clauses
will be $O(n^2)$ times. To get the final output, AND (i.e. $\land$) function will take $O(n)$ times. Hence total time required to check a satisfiability of an $n$-canonical conjunctive normal form for an assignment is $O(n^3)$, which is nothing but polynomial time.

Thus we see that if we consider a $k$-CCNF Form of Boolean formula with satisfiability condition then belong to NP.

Hence by the definition of NP problems it is clear that $k$-canonical conjunctive normal form is belongs to NP.

**Theorem 5.2.2:** Satisfiability of Boolean formula in $K$-CCNF is NP-complete iff $K \geq 3$.

**Proof:** From theorem 1, we have $K$-CCNF $\in$ NP. Now we have only to show that $SAT \leq_p K$-CCNF-SAT.

As discussed in the above theorem, we consider a satisfiable Boolean formula

$$\varnothing = ((x_1 \rightarrow x_2) \lor (x_1 \leftrightarrow x_3)) \land \neg x_2.$$ 

This is satisfiable for the assignment $x_1=1, x_2=0, x_3=1$, since

$$\varnothing = ((1 \rightarrow 0) \lor (1 \leftrightarrow 1)) \land \neg 0$$

$$= (0 \lor 1) \land 1$$

$$= 1.$$

Hence this formula $\varnothing$ belongs to SAT.

Now we reduce this formula to a $k$-CCNF where $k \geq 3$ as follows.

We introduce a variable $y_i$ for output of each operation and considering the final output as $y_1$ and parenthesizing formula as shown below.
$\emptyset = ((x_1 \rightarrow x_2) \lor (x_1 \leftrightarrow x_3)) \land \neg x_2$

$\emptyset' = y_1 \land (y_1 \leftrightarrow (y_2 \land \neg x_2))$

$\land (y_2 \leftrightarrow (y_3 \lor y_4))$

$\land (y_3 \leftrightarrow (x_1 \rightarrow x_2))$

$\land (y_4 \leftrightarrow (x_1 \leftrightarrow x_3))$, where $\emptyset'$ is the first step reduction of $\emptyset$

Here for the above satisfiable assignment of $\emptyset$, $\emptyset'$ is also satisfiable. Now we convert each clause into CNF as follows.

$\emptyset_1' = (y_1 \leftrightarrow (y_2 \land \neg x_2))$  \hspace{1cm} (1)

Table 5.1 is the truth table of "(1)"

Disjunctive normal form (or DNF) formula for $\neg \emptyset_1'$ is

$(y_1 \land y_2 \land x_2) \lor (y_1 \land \neg y_2 \land x_2) \lor (y_1 \land \neg y_2 \land \neg x_2) \lor (\neg y_1 \land y_2 \land \neg x_2)$

Applying DeMorgan's laws, we get the CNF formula.

$\emptyset_1'' = (\neg y_1 \lor \neg y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor \neg x_2) \land (\neg y_1 \lor y_2 \lor x_2) \land (y_1 \lor \neg y_2 \lor x_2)$ which is equivalent to the original clause $\emptyset_1'$.

$\emptyset_2' = (y_2 \leftrightarrow (y_3 \lor y_4))$  \hspace{1cm} (2)

Table 5.2 is the truth table for "(2)" From table 2, disjunctive normal form (or DNF) formula for $\neg \emptyset_2'$ is
Applying DeMorgan's laws, we get the CNF formula.

\[ \varnothing_2^* = (\neg y_2 \lor y_3 \lor y_4) \land (y_2 \lor \neg y_3 \lor \neg y_4) \land (y_2 \lor y_3 \lor \neg y_4) \land (y_2 \lor \neg y_3 \lor y_4) \] which is equivalent to the original clause \( \varnothing_2' \).

\[ \varnothing_3' = (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \] (3)

Table 5.3 is the truth table for "(3)".

**Table 5.1 Truth Table**

<table>
<thead>
<tr>
<th>( Y_1 )</th>
<th>( Y_2 )</th>
<th>( X_2 )</th>
<th>( (y_1 \leftrightarrow (y_2 \land \neg x_2)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
From table 5.3, disjunctive normal form (or DNF) formula for \( \neg \varphi_3' \) is

\[
(y_3 \land x_1 \land \neg x_2) \lor (-y_3 \land x_1 \land x_2) \lor (-y_3 \land \neg x_1 \land x_2) \lor (-y_3 \land \neg x_1 \land \neg x_2)
\]

Applying DeMorgan's laws, we get the CNF formula.

\[
\varnothing_3'' = (\neg y_3 \lor \neg x_1 \lor x_2) \land (y_3 \lor \neg x_1 \lor \neg x_2) \land (y_3 \land x_1 \lor x_2) \lor (\neg y_3 \land \neg x_1 \land \neg x_2) \text{ which is equivalent to the original clause } \varnothing_3'.
\]

\[
\varnothing_3' = (y_4 \leftrightarrow (x_1 \leftrightarrow x_3)) \tag{4}
\]

Table 5.4 is the truth table of "(4)". Disjunctive normal form (or DNF) formula for \( \neg \varphi_4' \) is

\[
(y_4 \land x_1 \land \neg x_3) \lor (y_4 \land \neg x_1 \land x_3) \lor (\neg y_4 \land x_1 \land x_3) \lor (\neg y_4 \land \neg x_1 \land \neg x_3)
\]

Applying DeMorgan's laws, we get the CNF formula.

\[
\varnothing_4'' = (\neg y_4 \lor \neg x_1 \lor x_3) \land (\neg y_4 \land x_1 \lor x_3) \land (y_4 \lor \neg x_1 \lor \neg x_3) \land (y_4 \land x_1 \lor x_3) \text{ which is equivalent to the original clause } \varnothing_4'.
\]

Table 5.2 Truth Table

<table>
<thead>
<tr>
<th>( Y_2 )</th>
<th>( Y_3 )</th>
<th>( Y_4 )</th>
<th>((y_2 \leftrightarrow (y_3 \lor y_4)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 5.3 Truth Table

<table>
<thead>
<tr>
<th>$y_3$</th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$(y_3 \leftrightarrow (x_1 \rightarrow x_2))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 5.4 Truth Table

<table>
<thead>
<tr>
<th>$y_4$</th>
<th>$X_1$</th>
<th>$X_3$</th>
<th>$(y_4 \leftrightarrow (x_1 \leftrightarrow x_3))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Now we convert each \( \phi_i \) CNF formula into 7-CCNF as there are 3 original variables and 4 introduced variables.

It is known that if there is a clause \((p \lor q \lor r)\), a new variable \(s\) can be included as \((p \lor q \lor r \lor s) \land (p \lor q \lor r \lor \neg s)\). Applying this rule repetitively on \( \phi_1'', \phi_2'', \phi_3'', \phi_4'' \) CNF formula can be converted into 7-CCNF formula \( \phi_1''', \phi_2''', \phi_3''', \phi_4''' \) respectively.

Hence \( \phi_1''', \phi_2''', \phi_3''', \phi_4''' \) are satisfiable as it was considered originally as satisfiable the SAT formula \( \phi \) is satisfiable.

We have reduced the SAT formula \( \phi \) into 7-CCNF formula. Constructing \( \phi' \) from \( \phi \) introduces at most 1 variable and 1 clause per connective. Constructing \( \phi'' \) from \( \phi' \) can introduce at most 8 clauses into \( \phi'' \) for each clause from \( \phi' \), since each clause has at most 3 variables and the truth table for each clause has at most \(2^3=8\) rows. The construction of \( \phi''' \) from \( \phi'' \) introduces at most (i.e. \(2^7=128\) - \(3=125\)) 125 clauses into \( \phi''' \) for each clause of \( \phi'' \). Thus, the size of the resulting formula \( \phi''' \) is polynomial in the length of the original formula. Each of the construction can easily be accomplished in polynomial time.

Thus total time taken in the reduction = \(O(m)\) to construct \( \phi' \), where \(m = \text{no of connective} + O(2^3) + O(2^7) = O(2^7)\) which is polynomial.
In reducing in case we introduce $y_i$ variables, and suppose there are $k$ variables in the original formula then in k-CCNF formula value of $n$ will be at least $i + m$.

Definition: Satisfiability of Boolean formula in k-CCNF is called full iff there are $2^k$ distinct clauses.

**Lemma:** Full k-CCNF formula is unsatisfiable.

**Proof:** Let us assume a 3-CCNF formula

\[(x \lor y \lor z) \land (x \lor \neg y \lor z) \land (\neg x \lor y \lor z) \land (x \lor \neg y \lor \neg z) \land (\neg x \lor y \lor \neg z) \land (\neg x \lor \neg y \lor \neg z)\]

Now for these three variables there will be $2^3 = 8$ input combinations. If we apply these 8 input combinations to the above formula then for each input combination the output is 0. Hence it is proved that full k-CCNF formula is unsatisfiable.

In circuit design we can replace a full k-CCNF circuit by a simple circuit which will produce constant 0.

**5.3 Algorithm for k-CCNF formula:**

**INPUT:** Consider any 3-CCNF formula.

**OUTPUT:** Find whether it is satisfiable or not

For $k$ variables, there will be $2^k$ input combinations, and we have to check the satisfiability of the formula for each input combination.

For $u = 0, 1, 2, \ldots, 2^k - 1$
Convert decimal $u$ to $k$ digit binary number like $b_1 b_2 \ldots b_k$

For $i=1, 2, \ldots, k$

\{ $x_i = b_i$ \}

Read the $k$-CCNF formula and assign $C_1, C_2, \ldots, C_m$ to clauses serially. //for example $C_1 = (x_1 \lor x_2 \lor \neg x_3), C_2 = (x_1 \lor \neg x_2 \lor x_3), C_3 = (x_1 \lor \neg x_2 \lor \neg x_3) //$

For $j = 1, 2, \ldots, m$

\{

Read clause $C_j$

For $i = 1, 2, \ldots, k$

\{

If (negation of $x_i$)

Change the value of $x_i$ (0 to 1 or 1 to 0)

If ($x_i == 0$)

\{

\[
i = i + 1
\]

if ($i > k$)

\{

$C_j = 0$

\}
For $j = 1, 2, ..., m$
{
    If($C_j = 0$)
    
    Goto print

    Else
    
    $j = j + 1$

    Goto print
}

Printf satisfied for $x_1, x_2, ..., x_k$
Print: printf unsatisfied for $x_1, x_2, \ldots, x_k$

\[
u = u + 1\]

}

Complexity Analysis of the Algorithm:

To convert decimal number $u$ to binary number, complexity = $O(k)$, where $2^{k-1} \leq u \leq 2^k$

Algorithm to convert a decimal number $u$ to binary number

```c
int i, num, k
i = 1;

While (num > 1)
{
    a[i] = num % 2
    i++;
    num = num /2;
}
K = i -1;

For (j = 1; j < i; j++)
{
    B[j] = a[k];
```
Here complexity of the decimal to binary conversion is = k,

It is observed that k = 4 and u = 8 when $2^3 \leq u \leq 2^4$

k = 6 and u = 32 when $2^5 \leq u \leq 2^6$

k = 4 and u = 9 when $2^3 \leq u \leq 2^4$

k = 5 and u = 16 when $2^4 \leq u \leq 2^5$

k = 5 and u = 25 when $2^4 \leq u \leq 2^5$

Hence complexity of the conversion of a decimal number 'u' to binary number = O(k),

where $2^{k-1} \leq u \leq 2^k$

The worst-case time Complexity of the main algorithm = $2^k(k + k + mk + m) = O(2^k mk)$.

Example: $\varnothing = (x_1 \lor x_2 \lor \neg x_3) \land (x_1 \lor \neg x_2 \lor x_3) \land (x_1 \lor x_2 \lor \neg x_3)$

Ans: Applying the above algorithm we can test that the $\varnothing$ is satisfiable.