$\neg$SAT
A Dummy Approach to P vs NP

Charles Dana, H23, Discrete Mathematician

February 20, 2024

Abstract

In this article, will try and introduce a framework for SAT Solving. Under the assumption that the SAT Instance φ is hard to solve, we will try to generate the opposite SAT Instance $\vartheta = 1 - \varphi$. A SAT Instance by definition is the intersection of the negation of partial assignments.

$$\neg \left(\bigwedge \bigvee \right) = \bigvee \bigwedge \neg$$

What is an AND ($\wedge$) statement, if not the assertion that the variables are assigned. What is an OR ($\vee$) statement, if not an AND Statement alongside with dummy variables. How many dummy variables do you need to describe 2^n OR assertions? n suffice.

1 Introduction

1.1 Introducing a simple φ Instance

```
p cnf 2 2
1 -2 0
-1 0
```

$\varphi = \{(0,0)\} \times \{0,1\}^{\mathbb{N}}$

1.2 Introducing a simple $\vartheta = \neg\varphi$ Instance

$\vartheta = \{(1,1),(0,1),(1,0)\} \times \{0,1\}^{\mathbb{N}}$

```
p cnf 2 1
1 2 0
```

Technically, this is very elegant, because as we can see the negation of the unique clause gives the unique solution, without using dummy variables. But rest assured, a SAT Instance is way more complex than this.

But let's not cheat and use 3 as a dummy variable.

```
p cnf 3 3
-1 3 0
2 3 0
1 -3 0
```

$\vartheta = \{(1,1,\cdot),(0,1,\cdot),(1,0,\cdot)\} \times \{0,1\}^{\mathbb{N}}$ This was made possible without relying on previous knowledge on the initial φ solution set, and so long you ignore the assignation of x_3, the created instance is isomorphic to $\neg\varphi$. If you are being smart, you will not use one dummy variable per clause, but rather $\log(m)$ dummy variables, where the assertion contradiction is to be made on the remainder.

Proposition Let φ be a $K-$SAT instance with n variables and m clauses. There exists a SAT instance ϑ with $n + \lceil \log(m) \rceil$ variables and at most $2^{\lceil \log Km \rceil}$ clauses, where:

$$\varphi = \neg\vartheta$$

Proof By python code.

```
def reversat(SAT, show=False):
    if show:
        print(SAT)
    VARS = max((abs(x) for clause in SAT for x in clause))
    VTH = []
    NUM_CLAUSES = len(SAT)
    m = 1
    while 2**m < NUM_CLAUSES:
        m = m + 1
    for idx in range(len(SAT)):
        clause = SAT[idx]
        for l in clause:
            bnr = "0" * (m - len(bin(idx).replace("0b",""))) + bin(idx).replace("0b", "")
            VTH += [[-l] + [(VARS + k + 1) * [-1,1][int(bnr[k])] for k in range(len(bnr))]]
    idx = len(SAT)
    while idx < 2**m:
        bnr = "0" * (m - len(bin(idx).replace("0b",""))) + bin(idx).replace("0b", "")
        VTH += [[(VARS + k + 1) * [-1,1][int(bnr[k])] for k in range(len(bnr))]]
        idx += 1
    if show:
        print(VTH)
    return VTH
```