N-SAT in P with Non-Coherent Space Factorization

F.L.B. Périat

22 August 2019

1 Introduction

We know since Cook that Boolean satisfiability problems with at least three literals in each clauses are in NP[1] and are NP-complete. With proving that 3-SAT (or more) is in P, corollary proves that P = NP[2].

2 Definitions and Processing

2.1 Notations

First, when literal of variable a is negative, $\bar{a}$ notation will be employed.
Then, *or* operator will be noted as $+$, while *and* won't be represented by any symbol ($\cdot$ may be employed in hardly ever cases).
Furthermore, we will use 0 and 1 to represent *true* and *false*, respectively, also in Boole algebra.
At last, for a better understanding, equations in a Boolean ring will be ended by [2], to differentiate it from Boole algebra and we will use the $=$ relation as an hypothetical equality.

2.2 De Morgan's Law Substitution

As we potentially work with conjunctive normal form of SAT problems, the first operation in this document is applying the De Morgan's Law on the whole CNF problem. For example, if we first had something like :

$$(v_1 + v_2 + \bar{v_3})(\bar{v_1} + \bar{v_2} + v_3) = 1$$

We obtain, after substitution :

$$\bar{v_1}\bar{v_2}v_3 + v_1v_2\bar{v_3} = 0$$

This is the form we will work with after, called disjunctive form. To be short and concise will we use functions notation for unspecified sets of one or more disjunctive clauses, like : $f(...)$, $g(...)$, etc.

2.3 Boolean Ring

The *not* operator on a variable, noted $\bar{a}$, can be equivalently replaced by $(1-a)[2]$ in Boolean rings.
As the *and* operator is idempotent, we won't make any change with Boole algebra.
or operator won't be represented by it's exact form; to simplify equations, we will let the additive operator as it is. We can allow this because in the disjunctive form, there won't be any solution if one additive element equals 1, as in case if several elements equal 1 (the exact form for $a+b$ would be $(a+b-ab)[2]$).

2.4 Empty Problem

A problem with no clause, has only one valid solution. In other words, every combination (of no variable) lets the problem satisfied (is this document we mean by combination a specific assignment of 0 or 1 for each variables). An empty problem is, in its disjunctive form : 0, so $0 = 0$.

2.5 Trivial Clause

2.5.1 Definition

A clause is called trivial if it contains simultaneously any literal and its negation. For example clause $\bar{v_1}\bar{v_2}v_2$ is trivial, because literals v_2 and $\bar{v_2}$ are in it.

2.5.2 Lemma

Every Trivial clause can be removed from the problem, without altering the result, as long as we consider the removed variables contained in the problem.

2.5.3 Proof

A problem with a trivial clause can be represented as:

$$f(...) + g(...)a\bar{a} = 0$$

In the Boolean ring we obtain:

$$f(...) + g(...)a(1-a) = 0[2]$$

The part $a(1-a)[2] = a - a = 0$, so we find that: $f(...) + g(...) \cdot 0 = 0$ is equivalent to $f(...) = 0$, a problem without trivial clauses. We can have many trivial clauses in a problem and remove them, since $0 + 0 = 0$.
A special case, is when we only have trivial clauses in a problem; we obtain a empty problem, which is always valid. We only have to consider the solution, as every combination of the removed variables.

2.6 Coherent Space

2.6.1 Definition

If we consider the problem as a set of clauses without any trivial clauses, a coherent space is a subset of the problem (including the whole problem set itself), where no variable exists simultaneously in its positive and negative state. For example : $abc + \bar{d}ac$ is a coherent space, while $\bar{a}f(...) + ag(...) + h(...)$ is not.

2.6.2 Lemma

If a coherent space would be a SAT problem, it would always have one obvious solution.

2.6.3 Proof

If the coherent space is an empty set, as showed before, its obvious solution is a combination of no variable.
Else if we assign to every variable (each different, so no conflict, hence the name of coherent space) the state of it's negation (1 if negative, 0 otherwise), then the problem will necessarily be satisfied. For example in the last given coherent space; $abc + \bar{d}ac$, transformed into a problem : $abc + \bar{d}ac = 0$, we obtain an equation without 1 value. As a matter of fact, only $(1 - a) \to (1 - 1) = 0[2]$ and $a \to 0[2]$ values will never achieve to a $1[2]$ value, implying $0 = 0[2]$.

2.7 Factorization into Coherent Space

2.7.1 Definition

When our SAT problem is not a coherent space, we can factorize it into three subset by one of the variables that exists in the set as positive and negative. The form of a non-coherent space is :

$$af(...) + \bar{a}g(...) + h(...)$$

Where variable a is not contained in any subsets f,g or h.

2.7.2 True Subset

If we factorize a literal by itself, we obtain the variable *and* 1. This 1 value is called; in this document, a true subset. It is always equal to 1 hence its name. (If it it was once equal to 0 we would not have any variable, because $0 \cdot a = 0$)

2.7.3 Lemma

If we add (+) a true subset to a coherent space (called a lonely true subset), we obtain a non-coherent space.

2.7.4 Proof

The true subset is always true, so we can replace it with true value $a + \bar{a}$ where a is an arbitrary variable, what add a variable positive and negative states to the set, so it is not coherent anymore. We will not make any substitution in the factorization.

2.7.5 Lemma

We can always recursively factorize a set of clauses, so that the leaves of the obtained tree are either coherent space or non-coherent space within a least one lonely true subset, but no positive and negative state of any variable.

2.7.6 Proof

If we have to deal with a coherent space or a non-coherent space without any variable with positive and negative states, we stop factorizing, otherwise, we choose an arbitrary variable that exist simultaneously positive and negative in the set : we split the set into three distinct subsets; one subset (called positive subset) of clauses that were containing the positive literal of the chosen variable, one subset (negative subset) made of the clauses that were containing its negative state and the last subset is the remaining clauses (without the chosen variable). The chosen variable will be a node of the tree and nonexistent in the subsets. The three tree children nodes will be the potential chosen variables for the subsets factorization, non-coherent spaces with lonely true subsets or by opposition with non-coherent spaces; coherent spaces.

2.8 Solution Path

2.8.1 Definition

We will see that when we have obtained a tree, after have the multiple subsets factorizations, we can follow a conditional solution path, which may allow us to find a combination of the variable which satisfied every clause. But first, let see a few lemmas.

2.8.2 Lemma

When we factorize a subset, we allow either the positive subset, or the negative one to be equals to 1. If the subset, is the problem, both can not be equal to 1: we obtains an *and* operator between the two subsets relation.

2.8.3 Proof

If the positive or negative subset will be equal to 1, we can assign value 0 or 1, respectively, to the factorized variable :

$$f(..)a + g(...)\bar{a} = 0$$

where $a = 0$ if $f(...) = 1$, else if $g(...) = 1$ then $a = 1$. If the subset is the problem and both $f(...)$ and $g(...)$ are equal to value 1, then we can not find a solution. Otherwise, the deal will be with the parents nodes.

2.8.4 Lemma

The continual remaining subset has to be equal to 0, in the other case, there is no solution.

2.8.5 Proof

As we said before, we can only deal with 1 value subsets when the clauses are factorized. In the case of the remaining of remaining ... of remaining, we have not removed any literal from this last subset, then if it is equal to 1, nothing can make it equal to 0.

2.8.6 Lemma

To find a solution for the SAT problem, the continual remaining subset must have by addition with a leave; a solution. Furthermore, we can not have every leave owning lonely true subsets, whatever is in the continual remaining.

2.8.7 Proof

At first, if we only have leaves within lonely true subsets, we can forget the validity of the continual remaining subset : due to the matter of fact that only 1 as leaves, the parents nodes will be equal to 1.
If we now have no solution between the continual remaining subset, we have to make that last subset equals to 0, to have a solution, but all over leaves will equal 1 because there is no solution with them, and we find us in the previous case.

3 Computation and Complexity

There are multiple possibilities of computations and structure, depending on several conditions, whose would have different time and space complexities in P. Here we have chosen one which is clear and concise.

3.1 Structure

In our abstract structure, we create a set of clauses, whose are also sets of literals. Variable represented by integers (called key) from 1 to n, where n is the number of variables. Positive state of a variable will be the opposite of its key (-key), since negative literal will be its key itself.

3.2 Functions

3.2.1 Solve Coherent Space Complexity

If we have to find the obvious solution of a coherent space, we iterate on the distinct variables and choose the assignment as in Lemma 1.6.2. This can be done in time $O(n)$, where n is the size of the coherent space.

3.2.2 Get Factorization Variable Complexity

When we choose a variable for factorization, we can for example take the first one we find which have simultaneously positive and negative states. For that computation, let make a set called literals.
Iterating on each literal of the subset, if negative literal is in literals, return the variable (positive literal) of it, else add it in the set. If no return has been made, return 0. (meaning we can not factorize). As set insertion complexity is $O(\log(x))$, where x is the current size of the last set. So Adding a percent of n literals in the set before a return will make a time complexity of $\int_0^n \log(x)dx = O(n\log(n))$.

3.2.3 Factorization Complexity

When we iterate on the clauses set, we search each time if either positive or negative factorization variable is in them. Searching can be executed in $O(\log(l))$ in a subset of l literals, then for c clauses we obtain $O(c\log(l))$. As we are making a tree where each node is a factorized variable, we remove a percent of c literals from clause of l literals, what give us also $O(c\log(l))$. As the number of literals n is approximately $n = c \cdot l$; both c and l are fractions of n and to simplify, lets say that factorization time complexity is $O(n\log(n))$

3.2.4 Global Factorization

In the middle case we get approximately, after a factorization, three subsets of sizes $\frac{n}{3}$ of the initial size n. Then, in the worst case, we have to wait that $3^d \approx n$, with d the depth of all subsets on the tree. If we call by f complexity of getting factorization key and factorizing which is $O(n\log(n))$, we have the global factorization complexity of :

$$3^0 f + 3^1 f + 3^2 f + ... + 3^d f = O(f3^d)$$

But since $f \approx log_3(n)$, we obtain $O(nf) = O(n^2\log(n))$

4 Example of Computation

To resolve this 3-SAT conjunctive normal form problem:

$$(x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (t \vee \bar{x} \vee \bar{y}) \wedge (\bar{t} \vee \bar{y} \vee \bar{z}) = 1$$

We first make De Morgan's law substitution to obtain :

$$\bar{x}yz + x\bar{y}z + xy\bar{z} + \bar{t}xy + tyz = 0$$

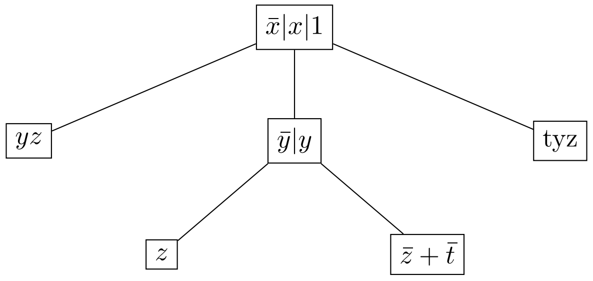
As we do not have to deal with a coherent space and there is not lonely true subset, we are going from left to right, to find first variable with positive and negative states, here : we stop at x because we already met $\bar{x}$. So x will be the factorization variable. Then after factorization we find :

$$x(\bar{y}z + y\bar{z} + \bar{t}y) + \bar{x}(yz) + (1)tyz = 0$$

We can factorize the positive subset by the first factorizable variable founded, here y.

$$x(y(\bar{z} + \bar{t}) + \bar{y}(z)) + \bar{x}(yz) + (1)tyz = 0$$

The tree would be like :



We look at the remaining, it is coherent with z leave, so we assign $z = 0$, then we follow the solution path, as $z\bar{y} = 0$ we can assign $y = 0$. We reiterate, as $z\bar{y}x = 0$ we can assign $x = 1$. We have reached the root, that means this combination is sufficient to satisfied every clauses.

5 Conclusion

With this processing, we can experiment finding a solution, or not in time complexity $O(n^2 \log(n))$ which is polynomial, with also a linear auxiliary space complexity, however we found an other computation, harder to explain, that permits a $O(n^2)$ time complexity. We have not proved this was the minimal bound, but we assuredly proved that N-SAT problems can be solved in polynomial time.

References

1. Richard M. Karp, Reducibility among Combinatorial Problems , dans Complexity of Computer Computations, Springer US, 1972, p. 85–103.
2. Stephen A. Cook, The Complexity of Theorem-Proving Procedures , dans Conference Record of Third Annual ACM Symposium on Theory of Computing (STOC), 1971, p. 151-158