# A Note for the Two Incremental Satisfiability Problem

Cristina López R., Guillermo De Ita L., and Pedro Bello L.

***Abstract*—Let *K* be a two conjunctive normal form and $\varphi$ a three conjunctive normal form, both formulas are defined over the same set of variables. It is well known that *SAT(K)* is in the complexity class P, while *SAT(φ)* is a classic NP-Complete problem. We consider the computational complexity of determining SAT($K\wedge \varphi$) as an incremental satisfiability problem (2-ISAT). We show that this problem is NP-complete even if the number of occurrences of each variable in $\varphi$ is one. Also, we propose a method to review *SAT*($K\wedge\varphi$). Our proposal is adequate to solve 2-ISAT problem. Our algorithm allows us to recognize tractable instances of 2-ISAT.**

***Index Terms*—3-Coloring, incremental satisfiability problem, 2-ISAT, NP-Complete problem, SAT problem.**

## I. Introduction

One of the fundamental problems in automatic reasoning is the satisfiability problem (SAT) that tries to determine whether a logical propositional formula *F* is (or not) satisfiable. Considering *F* as a conjunctive normal form (CNF) without restriction on the number of literals by clause, SAT(F) is a classic NP-complete problem, even if each clause has at least 3 literals.

The 2-SAT case, that determines the satisfiability of two conjunctive normal forms (2-CNF), is an important tractable case of SAT. Variations of the 2-SAT problem, e.g. in the optimization and counting area, have been essential for establishing frontiers between tractable and intractable problems.

The incremental satisfiability problem (ISAT) consists in verifying whether satisfiability is maintained when new clauses are added to an initial satisfiable formula. ISAT is considered a generalization of the SAT problem since it allows changes of the input formula over time.

We consider ISAT as a dynamic incremental set of clauses: $F_0, F_1, \ldots, F_i$ starting with an initial satisfiable formula $F_0$. Each $F_i$ results from a change in the preceding formula $F_{i-1}$, imposed by the 'outside world'.

Even though, the changes can be a restriction (add clauses) or a relaxation (remove clauses), we focus in the restriction case. In our proposal the incremental process is finished when $F_i$ is unsatisfiable or there are no more clauses to be added. ISAT can be used in a large variety of applications that need to be processed in an evolutive environment. This could be the case of applications such as reactive scheduling and planning, dynamic combinatorial optimization, review faults in combinatorial circuits, dynamic constraint satisfaction, and machine learning in a dynamic environment [1].

One idea used on ISAT methods, is to preserve the structures formed when previous formulas were processed, allowing the recognition of common subformulas that they were previously considered. More importantly, it allows the solver to reuse information across several related consecutive problems. The resulting performance improvements make ISAT a crucial feature for modern SAT solvers in real-life applications [2].

Rather than solving related formulas separately, modern solvers attempt to solve them incrementally, since many practical applications require solving a sequence of related SAT formulas [3], [4]. In this article, we consider ISAT as an incremental problem that starts with an initial satisfiable formula $K=F_0$ in 2-CNF. In a second phase, a new formula $\varphi$ in 3-CNF is added to *K*, both formulas; *K* and $\varphi$ are defined on the same set of variables. We denote this version of ISAT as the 2-ISAT problem.

As a generalization of SAT, ISAT has been considered as an NP Problem, although until now, the authors have not seen complexity theory studies about the complexity-time differences between SAT and ISAT. For example, it is known that 2-SAT is in the complexity class P. However, it is not known the computational complexity of 2-ISAT.

In [5], we have proposed a method to review the satisfiability of ($K\wedge\varphi$), and we have analysed instances of *K* and $\varphi$ that allows the existence of polynomial-time procedures. We present here, a study about the threshold for the 2-ISAT problem that could be helpful to understand the border between P and NP complexity classes for instances of 2-ISAT. In general, we show that 2-ISAT is NP-Complete, even if each variable in $\varphi$ has only one occurrence.

## II. The Graph Vertex Coloring Problem

The graph vertex coloring problem consists of coloring the vertices of the graph with the smallest possible number of colors so that no two adjacent nodes receive the same color. If such a coloring with *k* colors exists, the graph is k-colorable. The chromatic number of a graph *G*, denoted as $\chi(G)$, represents the minimum number of colors for proper coloring *G*. The *k*-colorability problem consists of determining whether

Manuscript received July 7, 2017; revised November 23, 2017. This work was supported by the Benemérita Universidad Autónoma de Puebla. Department of Computer Sciences, Puebla, México.

Cristina, López R. is with the Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, Mexico (mail: cristyna2001@hotmail.com).

Guillermo, De Ita. L. is with Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Mexico (e-mail: deita@ccc.inaoep.mx).

Pedro. Bello L. is with Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, Mexico (mail: pb5pbello@gmail.com).

an input graph is $k$-colorable.

We denote as $K_n$ as the complete graph of n nodes. And $(t\ K_n)$ as the disjoint union of $t$ copies of $K_n$. In the case of the vertex coloring problem, 2-coloring is solvable in polynomial time, as well as to determine the 3-colorability for AT-free graphs, perfect graphs [6], and for graphs free of $K_4$ and $(2\ K_2)$ [7]. Similarly, to determine $\chi(G)$ for some classes of graphs can be solved in polynomial time, as it is the case for interval graphs, chordal graphs and comparability graphs [8]. In all those cases, special structures (patterns) have been found to characterize the classes of graphs that are colorable in polynomial-time complexity. But in general, the $k$-colorability problem, when $k \geq 3$ becomes NP-complete, even for graphs $G$ with degree $\Delta \geq 3$ [9].

Several works have been done to determine when a graph is 3-colorable, as well as to design efficient approximation algorithms, namely, given a $k$-colorable graph to try to color it efficiently with $l$ colors, $l \geq k$, where $l$ is as small as possible [10]. The most recent results for determining the 3-colorability of a graph are based on recognizing if the input graph is planar and triangle-free [11], [12], or discovering some relationship between the topology of the graph and its structural parts [6]-[8], [13]-[17].

Although determining the 3-coloring of a graph is a NP-complete problem, there is a good deal of research which looks for specific topological patterns in graphs to determine their 3-colorability. One of the first results in this line is the renowned Grötzsch's theorem [12], which guarantees that every triangle-free planar graph is 3-colorable. From this pioneer paper, several works have been developed looking for patterns which allow us to determine the 3-colorability of a graph.

For example, Lozin [7] has recently show some polynomial-time instances of $k$-colorable graphs defined by finitely many forbidden induced subgraphs. Also, Borodin has shown that planar graphs without cycles of length 4 to 7 are 3-colorable [11]. Gimbel and Thomasson [17] found an elegant 3-colorability proof for triangle-free projective planar graphs. Dvor ák [13] found a linear-time algorithm to decide whether a triangle-free graph in a general surface $\Sigma$ is 3-colorable. 3-colorability is also polynomially solvable for graphs containing no induced path on 5 vertices [15].

3-coloring has been useful to show the NP-completeness of the other combinatorial problem. In particular, we will show in following chapter, a polynomial reduction from 3-coloring to 2-ISAT, proving then, the hardness computational complexity of the 2-ISAT problem.

## III. The Computational Complexity of 2-ISAT

**Lemma 1.** 3-Coloring is polynomial reducible to SAT($K \wedge \varphi$), $K$ a 2-CNF, $\varphi$ a 3-CNF.

Proof. Let $G = (V, E)$ be a graph where $n=|V|$, $m=|E|$. We define the logical variables $x_{v,c}$ meaning that the vertex $v \in V$ has assigned the color $c \in \{1,2,3\}$. For each vertex $v \in V$, 3 logical variables $x_{v,1}$, $x_{v,2}$, $x_{v,3}$ are created. Therefore, there are $3*n$ Boolean variables in $(K \wedge \varphi)$. We define first the constraints forming $K$.

For every edge $e = \{u,v\} \in E$, $u$ and $v$ must be colored differently. This restriction is modeled by 3 binary clauses, in the following way:$(\neg x_{u,1} \vee \neg x_{v,1}) \wedge (\neg x_{u,2} \vee x_{v,2}) \wedge (\neg x_{u,3} \vee \neg x_{v,3})$. There are $3*|E|$ binary clauses of this class. The other class of binary constraints allows to define the restriction that every vertex has not more than one color. This restriction is modeled by 3 binary clauses in the following way. For each vertex $v \in V$: $(\neg x_{v,1} \vee \neg x_{v,2}) \wedge (\neg x_{v,2} \vee \neg x_{v,3}) \wedge (\neg x_{v,3} \vee \neg x_{v,1})$. There are $3*|V|$ binary clauses for this class. Both sets of $3*(|V|+|E|)$ binary clauses form the 2-CNF $K$. On the other hand, the 3-CNF $\varphi$ is formed by the clauses modeling the restriction that every vertex must be assigned at least one color. Then, for each vertex the following clause is generated: $(x_{v,1} \vee x_{v,2} \vee x_{v,3})$. $\varphi$ has 3-clauses. Furthermore, each one of the $3*n$ variables of $v(K)$ has only one occurrence in $\varphi$ .

This reduction can be performed in polynomial time on the size $n$ and $m$, since it consists in creating $3*(n+m)$ binary clauses for $K$, and $(3*n)$ 3-clauses for $\varphi$. We also have that $G$ has a 3-Coloring if and only if $(K \wedge \varphi)$ is satisfiable.

In order to design reductions from 3-Coloring to 2-ISAT, we must consider the dynamic nature of 2-ISAT. Our proposal of 2-ISAT works in two consecutive phases. In the first phase, the input is a 2-CNF $K$. The purpose of this phase is to determine SAT($K$), as well as to build any computational or logical structure, denoted as $A_K$ and derived from $K$, that could be helpful for processing the input formula of the following phase. The main restriction on the construction of $A_K$ is to spent only polynomial time on the size $|K|$, such that all the phase 1 runs upper bounded by a polynomial on $|K|$.

In the second phase of 2-ISAT, the input is a 3-CNF $\varphi$ where all variables in $\varphi$ have already appeared in $K$. This is denoted as $v(\varphi) \subseteq v(K)$. The purpose of this phase is to determine SAT($K \wedge \varphi$). Although $K$ has been already processed, it can be used again as well as $A_K$, in order to accelerate the revision of $SAT(K \wedge \varphi)$.

**Theorem 1.** 2-ISAT is NP-Complete

Proof. The membership of 2-ISAT in NP comes from the fact that SAT is in NP, since a nondeterministic algorithm needs to guess only a satisfying assignment for $v(K)$, and check in polynomial time whether that assignment setting satisfies $(K \wedge \varphi)$, since $v(\varphi) \subseteq v(K)$. This happens independently of the two phases of 2-ISAT.

We have seen that 3-Coloring is polynomial reducible to SAT$(K \wedge \varphi)$, and since 3-Coloring is NP-complete, then SAT($K \wedge \varphi$) is also NP-complete. If a structure $A_K$ allows to determine SAT($K \wedge \varphi$) in upper-bounded time by a polynomial on $|K \cup \varphi|$, then 3-Coloring would be solvable in polynomial time. Thus, if 2-ISAT is polynomial solvable, then 3-Coloring too. This shows that 2-ISAT is NP-complete too.

From the above theorem, it is not expected (unless P=NP) to build an efficient algorithm for determining 2-ISAT, although in the first phase, $K$ has been already processed and some structures $A_K$, describing the logical dependencies among variables in $K$, have been built.

Notice that each component of SAT($K\land\varphi$) is polynomial-time solvable, that is, is solvable in linear time when $K$ is a 2-CNF [18]. Meanwhile SAT($\varphi$), where $\varphi$ is a 3-CNF and each one of its variables occurrs only one time on $\varphi$, is a trivial solvable problem, since every variable is pure in $\varphi$. Therefore, $\varphi$ is always satisfiable. However, SAT($K\land\varphi$) is a NP-complete problem. Therefore, it is not expected to build a structure $A_K$ that allows to solve SAT($K\land\varphi$) efficiently. For example, if $A_K$ is the set of models of $K$, then to review SAT($K\land\varphi$) can be performed efficiently, since it consists on delete from $A_K$ all falsifying assignment of each clause in $\varphi$. However, the set of models $A_K$ does not have always a polynomial size on $|K|$. In the worst case, the construction of the set of models of $K$ would require an exponential time on $|K|$. Nevertheless, we can look for restrictive cases of 2-ISAT that can be solved efficiently.

## IV. The Transitive Closure of a 2-CNF

The fact that in a 2-CNF formula a clause is equivalent to a pair of implications can be straightforward established as follows: if $\{x, y\}\in F$ then $\{x, y\}$ is equivalent to both $\neg x \rightarrow y$ and $\neg y \rightarrow x$. The arrow $\rightarrow$ has the usual meaning of implication in classical logic.

**Definition 1.** Let $F$ be a 2-CNF and $L$ its set of literals. The relation $\rightarrow_R \subset LxL$ is defined as follows: $x \rightarrow_R Y$ if and only if $x \rightarrow y$.

**Definition 2.** Let $F$ be a 2-CNF, a partial assignment $s$ of $F$ is a feasible model for , if $s$ does not falsify any clause in $F$.

We consider now the transitive closure of $x \rightarrow_R$, denoted by "$\Rightarrow$". This new relation $\Rightarrow$ can always be constructed inductively from $x \rightarrow_R$. For any feasible model $s$ of $F$ where $x$ and $y$ occur in $F$; if $x\Rightarrow y$ and $x$ is true in $s$ then it is straight forward to show that $y$ is true in $s$. It is said that $y$ is forced to be true by $x$. Let $T(x)$ be the set of literals forced to be true by $x$, that is $T(x) = \{ x \} \cup \{ y: x\Rightarrow y\}$.

It is clear that, if $x$ is a literal occurring in a formula $F$, and if $\neg x \in T(\neg x)$ then $x$ cannot be set to true in any model of $F$. Analogously, if $x \in T(\neg x)$ then $x$ cannot be set to false in any model of $F$.

**Definition 3**. Let $F$ be a 2-CNF, for any literal $x \in F$, it is said that $x \in T(x)$ is inconsistent if $\neg x \in T(x)$ or $\perp\in T(x)$, otherwise $T(x)$ is said to be consistent.

Unit clauses in 2-CNF can be expressed as implications, that is, if $F$ has unit clauses $\{u\}$ then $u\equiv u\lor\perp$, hence $\perp\in T(\neg u)$. As a consequence, in formulas with unit clause $\{u\}$ follows that $T(\neg u)$ is inconsistent. Let $F$ be a 2-CNF with $n$ variables and $m$ clauses, it has been shown that for any literal $x \in F$, $T(x)$ and $T(\neg x)$ are computed in polynomial time over $|F|$, in fact, for all $l \in Lit(F)$, $T(l)$ is computed with time complexity $O(n*m)$ [19].

For any literal $x$ in a 2-CNF, the sets $T(x)$ and $T(\neg x)$ allow to determine which variables have a fixed logical values in every model of $F$, that is to say, the variables that are true in every model of $F$ and the variables that are false in every model of $F$. The properties of the sets $T(x)$ and $T(\neg x)$ will be established as a lemma.

**Lemma 2.** Let $F$ be a 2-CNF and $x$ a variable in $F$.

1. If $T(x)$ is inconsistent and $T(\neg x)$ is consistent then $x$ is true in every model of $F$.

2. If $T(\neg x)$ is inconsistent and $T(x)$ is consistent then $x$ is true in every model of $F$.

3. If both $T(\neg x)$ and $T(x)$ are inconsistent then $F$ does not have models and $F$ is unsatisfiable.

4. If both $T(\neg x)$ and $T(x)$ are consistent then $x$ does not have a fixed valued in each model of $F$.

Proof. 1. Suppose $\neg x$ is false in a model of $F$, so $x$ should be true in that model of $F$. However, $T(x)$ is inconsistent, so $x \Rightarrow \neg x$ and $x$ cannot be true in the model of $F$ contradicting the assumption. Hence, any model of $F$ has to assign false to $x$ and true to $\neg x$. The other cases are proved similarly.

From properties (1) and (2) of lemma 4 we formulate the following definition:

**Definition 4.** A base for the set of models of a 2-CNF $F$, denoted as $S(F)$, is a partial assignment of $F$ which consists of the variables with a fixed truth value.

We denote by Transitive Closure($F$) to the procedure which computes the sets $T(x)$ and $T(\neg x)$ for each $x \in v(F)$. The transitive procedure applied on a 2-CNF $F$ allows to build bases for the set of models of $F$. If a base $S(F)$ is such that $|S(F)| = |v(F)|$, then each variable of $F$ has a fixed truth value in every model of $F$, so there is just one model.

**Definition 5.** Let $F$ be a 2-CNF and $x$ a literal of $F$. The reduction of $F$ by $x$, also called forcing $x$ and denoted by $F(x)$, is the formula generated from $F$ by the following two rules.

a) removing from $F$ the clauses containing x (subsumption rule),

b) removing $\neg x$ from the remaining clauses (unit resolution rule).

A reduction is also sometimes called a unit reduction. The reduction by a set of literals can be inductively established as follows: let $s=\{l_1,l_2,...,l_k\}$ be a partial assignment of $v(F)$. The reduction of $F$ by $s$ is defined by successively applying definition 5 for $l_i$, $i = 1,...,k$. That is reduction of $F$ by $l_1$ gives the formula $F(l_1)$, following a reduction of $F(l_1)$ by $l_2$, giving as a result the formula $F$ [l1], [l2] and so on. The process continues until $F[s] = [l_1,l_2,...,l_k]$ is reached. In case that $s = \varphi$ then $F[s]=F$.

Let $F$ be a 2-CNF formula and s a partial assignment of $F$. If a pair of contradictory unitary clauses is obtained while $F[s]$ is being computed, then $F$ is falsified by the assignment $s$. Furthermore, during the computation of $F[s]$, new unitary clauses can be generated. Thus, the partial assignment $s$ is extended by adding the already found unitary clauses, that is, $s=s \cup \{u\}$ where $\{u\}$ is a unitary clause. So, $F[s]$ can be again reduced using the new unitary clauses. The above iterative process is generalized, and we call to this iterative process Unit Propagation ($F$, $s$). For simplicity, we will abbreviate

Unit Propagation $(F, s)$ as $UP(F, s)$.

As a result of applying $UP(F, s)$, we obtain a new assignment $s'$ that extend to $s$, and a new subformula $F'$ formed by the clauses from $F$ that are not satisfied by $s'$. We denote $(F', s')=UP(F, s)$ to the pair resulting of the application of Unit Propagation on $F$ by the assignment $s$. Notice that if $s$ falsifies $F$ then $s'$ could have complementary literals and $F'$ contains the null clause. And when $s$ satisfies $F$, then $F'$ is empty.

## V. Incremental Satisfiability Problem

The incremental satisfiability problem (ISAT) involves checking whether satisfiability is maintained when new clauses are added to an initial satisfiable knowledge base $K$. ISAT is considered as a generalization of SAT since it allows changes of the input formula overtime. Different methods have been applied to solve ISAT, among them, variations of the branch and bounds procedure, denoted as IDPL methods, which are usually based in the classical Davis-Putnam-Loveland (DPL) method. In a IDPL procedure, when adding new clauses, the procedure maintains the search tree generated previously for the set of clauses $K$. Rather than solving related formulas separately, modern solvers attempt to solve them incrementally since many practical applications require solving a sequence of related SAT formulas [3], [4].

From now on, let us consider that $K$ is a 2-CNF and $\varphi$ is a 3-CNF. We consider that $\varphi$ consists of clauses that effectively decrease the set of models of $K$. We present now, an algorithm to solve 2-ISAT.

```
**Algorithm 1. Adv**
**Input:** $K, \phi$
**Output:** a message "SAT" or "UNSAT"
$S = Base(K)$
$Cola = \varnothing$
**while** $\phi \neq \varnothing$ **do**
 **for all** $x \in \upsilon(\phi)$ **do** /* Apply the reduced rules on $K$ and $\phi$ { includes $\phi = \phi[S]$} */
    $S_1 = S \cup T(x)$
    $S_2 = S \cup T(\neg x)$ /*{$S_1, S_2$ are consistent because $x, \neg x \notin S$}*/
    $(F_4, K_1, S_4) = UP(\phi, S_1)$
    $(F_5, K_2, S_5) = UP(\phi, S_2)$
    $M_4 = K \cup K_1 \cup S_4$
    $M_5 = K \cup K_2 \cup S_5$ /* {test for inconsistency} */
    **if** ((($Nil \in F_4$) or ($M_4$ is unsatisfiable)) and (($Nil \in F_5$) or ($M_5$ is unsatisfiable) )) **then**
       **return** ('UnSat')
    **end if**
    **if** $(F_4 = \varnothing) \vee (F_5 = \varnothing)$ then
       **return** ('Sat')
    **end if**
    **if** ($Nil \in F_4$ or $M_4$ is unsatisfiable) **then**
       $S = S \cup T(\neg x)$
       $Cola = Append(Cola, M_5)$
       $\phi = F_5$
    **end if**
    **if** ($Nil \in F_5$ or $M_5$ is unsatisfiable) then
       $S = S \cup T(x)$
       $Cola = Append(Cola, M_4)$
       $\phi = F_4$
    **end if**
    **for all** $l \in S_4$ **do** /* Updating the sets $T[X's]$ in $S_4$ */
       $T(x) = T(x) \cup T(l)$
       $T(\neg l) = T(\neg l) \cup T(\neg x)$
    **end for**
    **for all** $l \in S_5$ **do** /* Updating the sets $T[X's]$ in $S_5$ */
       $T(\neg x) = T(\neg x) \cup T(l)$
       $T(\neg l) = T(\neg l) \cup T(x)$
    **end for**
 **end for**
 $K = Next(Cola)$
**end while**
**return**
```

We show now, the soundness and completeness of our algorithm.

(Soundness) If Adv outputs Unsat then effectively $(K \cup \varphi)$ is unsatisfiable step1, because any feasible assignment S cannot be extended with value for the variable $x$ without falsifying $(K \cup \varphi)$. Similarly, when Adv outputs SAT then effectively $(K \cup \varphi)$ is satisfiable because the step 2 has built a model for $(K \cup \varphi)$. While the steps 3 and 4 fix the correct logical value of a variable $x \in v(K)$ and the procedure continue the main loop but with one less variable.

(Completeness) Our procedure always stops for any input $(K \cup \varphi)$, where $K$ is a 2-CNF and $\varphi$ a 3-CNF. The first two steps of Adv halt our procedure. On the other hand, the steps 3 and 4 determine a unique value for a variable in $v(K)$. Otherwise, the step 5 allows to modify $K$ or $\varphi$ since $v(\varphi) \subseteq v(K)$ and always is possible to find a consistent $T(x)$ (while $K$ keeps satisfiable) to be extended by resolution on $\varphi$. Thus, at most in $n = |v(K)|$ iterations, adv determines the satisfiability of $(K \cup \varphi)$.

It is clear that a set of changes over a satisfiable *KB K* in 2-CNF could change $K$ into a general CNF, in which case, $K$ will turn into a general CNF $K'$, $K \subset K'$, where the SAT problem on $K'$ is a classic NP-complete problem. One relevant question, in our opinion, is to establish the frontiers between P and NP by adding 3-clauses $\varphi$ to an initial tractable 2-CNF $K$, and taking advantages of the structures formed by any efficient algorithm solving SAT($K$), in order to determine SAT($K \wedge \varphi$). Assuming an initial formula $K$, and a new formula $\varphi$ to be added, our proposal allows us to determine some tractables cases for 2-ISAT.

i. If $K$ and $\varphi$ are 2-CNF's then $(K \wedge \varphi)$ is a 2-CNF. In this case, 2-ISAT is solvable in linear-time by applying a well known algorithm by Tarjan [18].
ii. For monotone formulas, ISAT keeps only satisfiable formulas. If each variable maintains a unique sign in

both formulas $K$ and $\varphi$, then $(K \wedge \varphi)$ is always satisfiable.

iii. If $\varphi$ consists of a single clause and we have the transitive closures of $K$, we only have to review which closures comes inconsistent for containing the falsifying assignments of $\varphi$. This process can be done in linear time on the number of literals of $K$.

## VI. Conclusions

Let $K$ be a two conjunctive normal form and φ a three conjunctive normal form, both formulas are defined over the same set of variables. We consider the computational complexity of determining SAT($K \wedge \varphi$) that is considered as a base case of the incremental satisfiability problem (2-ISAT). We show that this problem is NP-complete even if the number of occurrences of each variable in $\varphi$ is one. Also, we propose a method to review SAT($K \wedge \varphi$). Our proposal is adequate to solve the 2-ISAT problem, and our method allows to recognize tractable instances of 2-ISAT.

## References

[1] M. Mouhoub and S. Sadaoui, "Systematic versus non systematic methods for solving incremental satisifiability," *Int. J. on A.I. Tools*, vol. 16, no. 1, pp. 543-551, 2007.

[2] S. Wieringa, "Incremental satisfiability solving and its applications," Ph.D. thesis, Department of Computer Science and Engineering, Alto University, 2014.

[3] G. Cabodi, L. Lavagno, M. Murciano, A. Kondratyev, and Y. Watanabe, "Speeding-up heuristic allocation, scheduling and binding with SAT-based abstraction/refinement techniques," *ACM Trans. Design Autom. Electr. Syst.*, vol. 15, no. 2, 2010.

[4] N. Eén and K. Sörensson, "An extensible SAT-solver," in *Proc. of 6th International Conference on Theory and Applications of Satisfiability Testing (SAT), Santa Margherita Ligure*, vol. 2919, Enrico Giunchiglia and Armando Tacchella, Ed. Italy: LNCS, 2003, pp. 502-518.

[5] G. De Ita, R. Marcial-Romero, and J. A. Hernández, "The incremental Satisfiability problem for a two conjunctive normal form," *Electronic Notes in Theoretical Computer Science*, vol. 328, pp. 31-45, 2016.

[6] J. Stacho, "3-Colouring AT-free graphs in polynomial time, algorithms and computation," *ISAAC 2010*, pp. 144-155, 2010.

[7] V. V. Lozin and D. S. Malyshev, "Vertex coloring of graphs with few obstructions," *Discrete Applied Mathematics*, vol. 216, pp. 273-280, 2017, Elsevier.

[8] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, 2nd ed. North Holland, 2004.

[9] D. Johnson, "The NP-completeness column: An ongoing guide," *Jour. of Algorithms*, vol. 6, pp. 434-451, 1985.

[10] S. Arora, E. Chlamtac, and M. Charikar, "New approximation guarantee for chromatic number," *Proceedings STOC*, 2006.

[11] O. V. Borodin, A. N. Glebov, A. Raspaud, and M. R. Salavatipour, "Planar graphs without cycles of length from 4 to 7 are 3-colorable," *Journal of Combinatorial Theory*, Series B 93, pp. 303-311, 2005.

[12] H. Grötzsch and Z. Wiss, "Ein dreifarbensatz fur dreikreisfreie netze auf der kugel," Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 8, pp. 109-120, 1959.

[13] Z. Dvorák, D. Král, and R. Thomas, "Three-coloring triangle-free graphs on surfaces," in *Proc. of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, NY, 2009, pp. 120-129.

[14] F. V. Fomin and D. Kratsch, "Exact exponential algorithms," *Texts in Theoretical Computer Science, an EATCS Series*, Springer, 2010.

[15] C. Hoang, M. Kaminski, V. Lozin, J. Sawada, and X. Shu, "Deciding k colorability of P5-free graphs in polynomial time," *Algorithmica*, vol. 57, pp. 74-81, 2010.

[16] G. B. Mertzios and P. G. Spirakis. (2012). Algorithms and almost tight results for 3-colorability of small diameter graphs. [Online]. Available: arxiv.org/pdf/1202.4665v2.pdf

[17] J. Gimbel and C. Thomassen, "Coloring graphs with fixed genus and girth," *Trans. Amer. Math. Soc.*, vol. 349, pp.4555-4564, 1997.

[18] B. Aspvall, M. R. Plas, and R. E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas," *Information Processing Letters*, vol. 8, no. 3, pp. 121-123, 1979.

[19] D. Gusfield, and L. Pitt, "A bounded approximation for the minimum cost 2-sat problem," *Algorithmica*, vol. 8, pp. 103–117, 1992.

**Cristina López R.** is a researcher-professor at DAIS - Academic Division in informatic and systems, in Juarez Autonomous University of Tabasco, México. Her areas of interest are intelligent systems and satisfiability methods. She is currently a Ph.D student in the language & knowledge engineering (LKE) at the Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, México.

**Guillermo De Ita L.** obtained his PhD in engineering in the CINVESTAV of the IPN, Mexico. He has worked for more than 10 years as a developer and consultant in geographical information databases systems for different enterprises in Mexico. He has done research stances at the University of Chicago, Texas, INAOEP Puebla, and the INRIA institute in Lille France. Currently, he is researcher-professor at the Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, México

**Pedro Bello L.** is a researcher-professor at the Faculty of Computer Science, BUAP. His areas of interest are graph theory and belief revision. He developed a system of simulation as a master degree thesis for the Pierre Auger Project. He is currently a PhD student in the Language & Knowledge Engineering (LKE) at the Faculty of Computer Sciences in Benemérita Universidad Autónoma de Puebla, Puebla, México.