UNIVERSIDAD DE GRANADA

UNDERGRADUATE THESIS

Consistency in Propositional Logic

Author:
Pedro Bonilla Nadal

Supervisor:
Dr. Serafín Moral Callejón

A thesis submitted in fulfillment of the requirements
for the degrees of Computer Engineering and Mathematics

in the

Department of Computer Science and Artificial Inteligence

April 10, 2020

Contents

1 Theoretical Introduction **1**
1.1 Definitions and first concepts. 1

2 Satisfiability by Combinatorics **7**
2.1 Lovász Local Lemma 8
2.1.1 First definitions 9
2.1.2 Statement of the Lovász Local Lemma 10
2.1.3 Nonconstructive proof of 2.1.1 11
2.1.4 Constructive proof of 2.1.1.2 12

3 Resolution Algorithms **15**
3.1 Special Cases Solvable in Polynomial Time 15
3.1.1 Unit Propagation 16
3.1.2 2CNF 16
3.1.3 Horn Formulas 16
3.2 Backtracking and DPLL Algorithms 17
3.2.1 Backtracking 17
3.2.2 Davis-Putman-Logemann-Loveland(DPLL) algorithm 18
3.2.3 Monien-Speckenmeyer(MS) Algorithm 19

4 Reductions **21**
4.1 Proofs 21
4.1.1 Hamiltonian Cycle 21

Chapter 1

Theoretical Introduction

1.1 Definitions and first concepts.

In this section Boolean formulas will be introduced. We first start with the basic building blocks, which collectively form what is called the alphabet. Namely,

- Symbols x, y, z for Boolean variables.
- Symbols p, q, r for Boolean metavariables, that is, a variable that refer to a boolean variable or a negated (see below) boolean variable.
- Values 0 and 1, referring to false and true respectively. The set $\{0, 1\}$ will be named as $\mathbb{B}$.
- Boolean Operators:
  - unary: $\neg$
  - binary: $\wedge, \vee, \rightarrow, \oplus, \leftrightarrow$

  We will consider $\wedge$ of greater priority than $\vee$. These operator are defined by theirs truth table:

$\neg$	0	1
	1	0

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

$\rightarrow$	0	1
0	1	1
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

$\leftrightarrow$	0	1
0	1	0
1	0	1

Definition 1.1.1. A Boolean formula is defined inductively:

- The constants 0 and 1 are formulas.
- Every variable is a formula.
- If F is a formula, then $\neg F$ is a formula.
- The concatenation with a symbol of two formulas is a formula to.

Examples of formulas are $x \vee y$ or $x_1 \wedge x_2 \vee (x_4 \vee \neg x_3 \wedge (x_5 \rightarrow x_6) \vee 0)$.

Definition 1.1.2. Given a set A it has an associated homonym problem that consists on, given an arbitrary element e check if $e \in A$.

Definition 1.1.3. An assignment is a function from the set of Boolean formulas to the set of Boolean formulas, on which some variables $\{x_1, ..., x_n\}$ are replaced by predefined constants $\{a_1, ..., a_n\}$ respectively. If none of the variables altered by an assignment α are present on the formula F then $\alpha(F) = F$. We denote as $Var(\alpha)$ the set of those variables that receive a value from α. Analogously, $Var(F)$ will denote the variables present on a formula F.

Now we will clarify the necessity of a meta variable. When we talk about a given formula F, should we like to talk about a variable $x \in Var(F)$ talking also as how appear in that formula (negated or not) we should use a meta variable.

One can then *apply* an assignment α to a formula F, denoting it by $F\alpha = \alpha(F)$. To describe an assignment we will use a set that pairs each variable to it value, i.e. $\alpha = \{x_1 \to 1, ..., x_n \to 0\}$. For example given an assignment $\alpha_0 = \{x_1 \to 1, x_2 \to 1, x_3 \to 0\}$ and $F_0 = x_1 \to (x_2 \wedge x_4)$ then $F_0\alpha_0 = 1 \to (1 \wedge x_4) = x_4$.

Definition 1.1.4. An assignment is said to *satisfies* a formula F if $F\alpha = 1$ and in the case $F\alpha = 0$ it is said to *falsifies* the statement.

Definition 1.1.5. A formula F is called *satisfiable* if $\exists \alpha : F\alpha = 1$. Otherwise it is called *unsatisfiable*. The set of all satisfiable formulas is denoted as SAT. The problem SAT is the associated problem. An assignment α that satisfies F is called a model an is denoted as $\alpha \models F$.

A formula F such that for every α assignment happens that $F\alpha = 1$ is a tautology. Given two formulas G, F it is said that G follows from F if $G \to F$ is a tautology.

Definition 1.1.6. A formula F is said to be in conjunctive normal form if is written as:

$$F = C_1 \wedge ... \wedge C_n$$

Where $C_i = (u_{1,i} \vee ... \vee u_{m_i,i})$ and $u_{i,j}$ are literals, that is, variables or negated variables. The set of all formulas in conjunctive normal form is called *CNF*.

A literal u would be a pure literal if there is no $\neg u$ in F

A formula in *CNF* could be seen as a collection of clauses. The associated problem with *CNF* is straightforward on $O(n)$. The problem that we will investigate is whether a arbitrary formula F have a *SAT-equivalent CNF* formula. Equivalently a clause could be seen as a set of literals. The set of all formulas in conjunctive normal form where $|C_i| = N$ $i \in 1, ..., n$ is called *NCNF*. The intersection of these set with the *SAT* set are called *CNF-SAT* y *NCNF-SAT*. If the context is clear enough the problems will be called *CNF* and *NCNF*

We could define an equal relationship on the set of formulas. Let F, G be formulas. Then $F = G$ if it happens that for each α an assignment such that $F\alpha = 1$ then $G\alpha = 1$ and $G\alpha = 1$ then $F\alpha = 1$

Proposition 1.1.1. The given equal relationship is a equivalence relationship.

Proof. All three properties follows from the equivalent properties on the constants. □

We could define a partial order relation between the formulas. Let F, G be formulas. Then $F \leq G$ if it for each α an assignment such that $F\alpha = 1$ then $G\alpha = 1$.

Proposition 1.1.2. The given equal relationship is an equivalence relationship.

Proof. As we then could see each class of equivalent as the set of assignment that satisfies all of the clauses, this property arises from the order given by the inclusion on sets.

□

Lemma 1.1.1. *For every* SAT *formula there is an associated circuit.*

Proof. Every operator can be seen as a gate and every variable as an input.

□

Theorem 1.1.2 (Tseitin [8]). *There is a* 3-CNF *formula on each equivalent class. Moreover, given an element F there is a equivalent formula G in* 3-CNF *which could be done in polynomial time.*

Proof. We will show that for every circuit with n inputs and m binary gates there is a formula in *3-CNF* that could be constructed in polynomial time in n and m. Then, given a formula we will work with it considering it associated circuit.

We will construct the formula considering variables $x_1, ..., x_n$ that will represents the inputs and $y_1, ..., y_n$ that will represents the output of each gate.

$$G = (y_1) \wedge \bigwedge_{i=1}^{m} (y_i \leftrightarrow f_i(z_{i,1}, z_{i,2}))$$

Where f_i represents the formula associated to the i-gate, $z_{i,1}, z_{i,2}$ each of the two inputs of the i-gate, whether they are x_- or y_- variables. This formula is not *3-CNF* yet, but for each configuration being f_i a Boolean operator there would be a *3-CNF* equivalent.

- $z \leftrightarrow (x \vee y) = \neg(z \vee x \vee y) \vee (z \wedge (x \vee y)) = \neg(z \vee x \vee y) \vee (z \wedge x) \vee (z \wedge y) = = (\neg z \wedge \neg x \wedge \neg y) \vee (z \wedge x) \vee (z \wedge y) =(\neg z \vee (z \wedge x) \vee (z \wedge y)) \wedge (\neg x \vee (z \wedge x) \vee (z \wedge y)) \wedge (\neg y \vee (z \wedge x) \vee (z \wedge y)) = (\neg z \vee x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee z)$

- $z \leftrightarrow (x \wedge y) = \neg(z \vee (x \wedge y)) \vee (z \wedge (x \wedge y)) = (z \wedge x \wedge y) \vee (\neg z \wedge \neg x \wedge \neg y) = ((z \vee (\neg z \wedge \neg x \wedge \neg y)) \wedge (x \vee (\neg z \wedge \neg x \wedge \neg y)) \wedge (y \vee (\neg z \wedge \neg x \wedge \neg y))) = (\neg x \vee z) \wedge (\neg y \vee z) \wedge (\neg z \vee x) \wedge (\neg y \vee x) \wedge (\neg z \vee y) \wedge (\neg x \vee y)$

- $z \leftrightarrow (x \leftrightarrow y) = \neg(z \vee (x \leftrightarrow y)) \vee (z \wedge (x \leftrightarrow y) = \neg(z \vee (\neg x \wedge \neg y) \vee (x \wedge y)) \vee (z \wedge (\neg x \wedge \neg y) \vee (x \wedge y))) = (\neg z \wedge \neg(\neg x \wedge \neg y) \wedge \neg(x \wedge y)) \vee (z \wedge (\neg x \wedge \neg y) \vee (x \wedge y))) = (\neg z \wedge (x \vee y) \wedge (\neg x \vee \neg y)) \vee (z \wedge (\neg x \wedge \neg y) \vee (x \wedge y))) = z \vee (\neg x \wedge \neg y) = (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg z \vee y) \wedge (y \vee z \vee x) \wedge (y \vee \neg y \vee x) \wedge (\neg z \vee z \vee x) \wedge (\neg z \vee \neg y \vee x)$

- $z \leftrightarrow (x \oplus y) = z \leftrightarrow (\neg x \leftrightarrow y)$

In the last item we use the third one.

□

This result is important because, now we could be able to talk only about *3-CNF* formulas. The fact that they are reachable on polynomial time is important because it means it could be done efficiently. Should this be impossible it will not be of much relevance in practice, as we yearn to solve this problem as efficient as possible (in fact, as polynomial as possible). This results implies that if we know how to solve *3-CNF* then we will be able to solve 'full'SAT problems.

Definition 1.1.7. An assignment is called autark for a formula $F \in$*CNF* if for every clause $C \in F$ it happens that if $Var(C) \cap Var(\alpha) \neq \emptyset$ then $C\alpha = 1$, in other words it satisfies all clauses that it 'touches'.

The use of this definition is self-evident, as it would simplifies the problem of resolving a *CNF* clause. The strategy would be simple as obvious: try to make every clause positive. These assignment will give simplifications of the problem, and enabling a good method for these search will be useful.

Should it happen that we got an algorithm for autarks clauses, and iterating it, we could find a solution of any given formula. Finding a polynomial algorithm that find whether it exists any non-empty autark formula and provide it, we could be able of proving that NP = P, as we could solve SAT applying this algorithm iteratively. Anyway, trying to find simple autark assignment, i.e. assignment with not many variables,i s a good praxis.

Proposition 1.1.3. We could reduce the SAT-CNF problem to the Autark-Finding problem.

Proof. Suppose that an algorithm such that if it exists any autark it return one of them, and end with an error code otherwise is given.

Given a formula F, if there is not an autark then there is no solution for the SAT problem. If it find an Autark-assignment αthen we apply the same algorithm to $\alpha(F)$. Also, as it happens that $|Var(\alpha(F))| < |Var(F)|$ so we would only apply the algorithm finitely many times. Also, F will be solvable if, and only if, $F\alpha$ is solvable.

Moreover, as checking if an assignment is autark is linear on the number of clauses, then it made the autark-finding problem NP-Complete(NP-C further on). $\square$

Proposition 1.1.4. Given $F \to G$ a tautology, there exists a formula I such that $Var(I) = Var(F) \cap Var(G)$ and both $F \to I$ and $I \to G$ are tautologies. It is not known an polynomial algorithm to solve this problem.

Proof. Let $\{x_1, ..., x_k\} = Var(F) \cup Var(G)$ then we will make I by defining its truth table the following way: Given an assignment α:

$$I\alpha = \begin{cases} 1 & \text{if } \alpha \text{ could be extended to an assignment that } \textit{satisfies } F, \\ 0 & \text{if } \alpha \text{ could be extended to an assignment that } \textit{nullifies } G, \\ * & \text{otherwise.} \end{cases}$$

Where * mean that it could be either 0 or 1. This is well defined because if for an arbitrary happens that $G\alpha = 0$ then $F\alpha = 0$.

For every β an assignment such that $Var(\beta) = Var(F) \cup Var(G)$ then if $\beta(F) = 1$ then $\beta(I) = 1$ so $F \to I$ is a tautology. Similarly it can not happens that $I\beta = 1$ and

$G\beta = 0$, because the second it will imply that $I\beta = 0$.

For the last part we will refer to the paper on the topic by: TODO

$\square$

Chapter 2

Satisfiability by Combinatorics

To get an intuition about the way that unsolvable clauses are, we gonna state some simple result about combinatorics and resolution. This will give the reader an idea of how these formulas should be.

Firstly, it is easy to break a big clause on some smaller ones, adding one another on this fashion: Suppose we got two positive integers n, m such that $m < n$ a clause $x_1 \vee x_2 \vee ... \vee x_n$ we could split it into two parts $x_1 \vee x_2 \vee ... \vee x_{m-1} \vee y, \neg y \vee x_m \vee ... \vee x_n$. Also given the same clause with a given length n we could enlarge it one variable adding $x_1 \vee ... \vee x_n \vee y$ and $x_1 \vee ... \vee x_n \vee \neg y$. Note that to enlarge a clause from a length m to a length $n > m$ we would generate 2^{n-m} clauses.

Proposition 2.0.1. Let F be a CNF formula which has exactly k literals, if $|F| < 2^k$ then F is satisfiable.

Proof. Let $n = Var(F)$, it happens that $n > k$. For each clause $C \in F$ there are 2^{n-k} assignment that falsify F, so in total there could be strictly less than $2^k \cdot 2^{n-k} = 2^n$. Therefore it exists an assignment that assign all variables and not falsifies the formula F. $\square$

Proposition 2.0.2. Let $F = \{C_1, ..., C_n\}$ be a CNF formula. If $\sum_{j=1}^{m} 2^{-|C_j|} < 1$, then F is satisfiable.

Proof. Enlarging clauses the way it is explained to the maximum length k ans applying the previous result. $\square$

Following this idea we could define the weight of a clause $C \in F$ as

$$\omega(C) = 2^{-|C|}$$

being this the probability that a uniform-random assignment violates this clause.

Corollary 2.0.0.1. *For a formula in CNF, if the sum of the weights of the clauses is less than one then the formula is satisfiable.*

Proof. For this task, we will give a probabilistic algorithm, only to prove that it will end with a big probability. Probabilistic (and heuristics) approaches to the problem would prove later on to be really useful. Let F be a CNF formula regard as a clause set. $\square$

Definition 2.0.1. Let F be a CNF formula. It is said to be minimally unsatisfiable if:

- F is unsatisfiable.
- $F \backslash \{C\}$ is satisfiable $\forall C \in F$.

The to following prove would be shown as done in [6]. For that resolution we will need the well known Hall marriage theorem[2]:

Theorem 2.0.1 (Hall marriage graph version)**.** *Let G be a finite bipartite graph with finite sets of vertex X, Y. There is a matching edge cover(a cover such that every vertex only participate in one edge) of X if and only if $|W| \leq |N_G(W)|$.*

Lemma 2.0.2. *Let F be a CNF formula. If for every subset G of F it holds that $|G| \leq |Var(G)|$, then F is satisfiable.*

Proof. We will Associate F a bipartite graph and with U, V be the two set of vertex: U consists on the set of clauses and V on the set of variables. By the marriage theorem every clause can be associated to a variable. Therefore we could make an assignment that take every variable associated to a clause to the value that the clause requires. □

This idea or neighbourhood in clause is important and curious. It defines a relation between clauses and give clauses resolution some nice graph-tools to work with.

Proposition 2.0.3. Minimally unsatisfiable, then $|F| > Var(F)$.

Proof. Since F is unsatisfiable, there must be a subset G such that is the maximal that satisfy $|G| > Var(G)$. If $G = F$ them the theorem is proved.

Otherwise, be $H \subset F \backslash G$ an arbitrary subset. If $|H| > |Var(H)(G)|$ then $|G \cup H| > |Var(G \cup H)|$ and G would not be maximal. Therefore F satisfy the condition of the lemma and is satisfiable using an assignment that does not use any variable $x \in Var(G)$. As G is minimally unsatisfiable G is satisfiable by and assignment β. We could them define an assignment:

$$\gamma(x) = \begin{cases} \beta(x) & \text{if } x \in Var(G) \\ \alpha(x) & \text{otherwise.} \end{cases}$$

this assignment would satisfy F against the hypothesis. □

2.1 Lovász Local Lemma

We continue to prove an interesting lemma on the theoretical analysis of satisfiability problem: the Lovász Local Lemma (LLL). This lemma was first proven on 1972 by Erdös and Lovász while they were studying 3-coloration of hypergraphs. Then it was Moser which understood the relationship between this result an constraint satisfaction problem. The SAT could be regard as the simplest of these problems.

This section is going to be based on the works of Moser, Tardos, Lovász and Erdös as a result. As it will be shown LLL is applicable to set sufficient condition for satisfiability. We will explain the lemma for theoretical purposes and prove the most general version, and give a constructive algorithm to solve a less general statement of the problem. The principal source of bibliography for the whole section would be Moser PhD. Thesis.

The main contribution of Moser's works to this problem is finding an efficient algorithm to find what assignment satisfies the formula, should happen that F is proved satisfiable by the previous theorem. Previously only probabilistic approaches had been successful.

The probabilistic method is a useful method to prove the existence of objects with an specific property. The philosophy beneath this type of demonstration is the following: in order to prove the existence of an object we do not need to give the said object, instead, we could just consider a random object in the space that we consider an prove that the probability is strictly positive. Then we can deduce that an object with that property exists (if it did not probability would be 0). It is not necessary to provide the exact value, bounding it by a constant greater that 0 would be enough.

This technique was pioneered by Paul Erdös. The LLL takes part because is an useful tool to prove lower bounds for probabilities, allowing us to provide the result.

This section will follow this order:

- Present the notation and general expression for the LLL.
- Use the result to prove an interesting property on satisfiability on CNF.
- Prove the general result with the probabilistic result.
- Provide the more concise CNF-result with a constructive algorithm.

2.1.1 First definitions

We will work here with a very specific type of formulas. Let us call a formula F is in k-CNF if it is in CNF and $\forall C \in F, |C| = k$.

Definition 2.1.1. Let C be a clause in F, the neighborhood of C, denoted as $\Gamma_F(C)$ as

$$\Gamma_F(C) = \{D \in F : D \neq C, Var(C) \cap Var(D) \neq \emptyset\}$$

Analogously, the inclusive neighborhood $\Gamma_F^+(C) = \Gamma(C) \cup \{C\}$.

Further on Γ and Γ^+ will respectively denote inclusive or exclusive neighborhood on CNF formulas or graphs

Definition 2.1.2. Two clauses are *conflicting* if there is a variable that is required to be true in one of then and to be false in the other. The graph G_F^* such that there is an edge between C and D iff they *conflict* in some variable.

Definition 2.1.3. Let Ω be a probability space and let $\mathcal{A} = \{A_1, ..., A_m\}$ be arbitrary events in this space. We say that a graph G on the vertex set $\mathcal{A}$ is a *lopsidependency graph* for $\mathcal{A}$ is more likely in the conditional space defined by intersecting the complement of any subset of its non-neighbors. In others words:

$$P\left(A \,\middle|\, \bigcap_{B \in S} \overline{B}\right) \leq P(A) \qquad \forall A \in \mathcal{A},\ \forall S \subset \mathcal{A} \backslash \Gamma_G^+(A)$$

If, instead of requiring the event to be more likely, we require it to be independent (i.e. to be equal in probability) the graph is called *dependency graph*.

2.1.2 Statement of the Lovász Local Lemma

Theorem 2.1.1 (Lovász Local Lema). *Let Ω be a probability space and let $\mathcal{A} = \{A_1, ..., A_m\}$ be arbitrary events in this space. Let G be a lopsidependency graph for $\mathcal{A}$. If there exists a mapping $\mu : \mathcal{A} \to (0,1)$ such that*

$$\forall A \in \mathcal{A} : P(A) \leq \mu(A) \prod_{B \in \Gamma_G(A)} (1 - \mu(B))$$

then $P\left(\bigcap_{A \in \mathcal{A}} \overline{A}\right) > 0$.

By considering the random experiment of drawing an assignment uniformly, with the event corresponding to violating the different clauses we could reformulate this result. The weight of each clause is the probability of violating each clause. Therefore, we can state a SAT-focused result.

Corollary 2.1.1.1 (Lovász Local Lema for SAT). *Let F be a CNF formula. If there exists a mapping $\mu : F \to (0,1)$ that associates a number with each clause in the formula such that*

$$\forall A \in \mathcal{A} : \omega(A) \leq \mu(A) \prod_{B \in \Gamma^*_G(A)} (1 - \mu(B))$$

then F is satisfiable.

Proof. To prove the result it would only be necessary to show that Γ^* is the lopsidependency graph for this experiment. Given $C \in F$ and $\mathcal{D} \subset F \backslash \Gamma_{G^*_F}(D)$ (i.e. no $D \in \mathcal{D}$ conflict with C). We want to check the probability of a random assignment falsifying C given that it satisfies all of the clauses in $\mathcal{D}$, and prove that it is at most $2^{-|C|}$.

Let α be an assignment such that it satisfies $\mathcal{D}$ and violates C. We could generate new assignment from α changing any value on $Var(C)$, and they still will satisfy $\mathcal{D}$ (as there are no conflict) so the probability is still at most 2^{-k}.

□

The result that we will prove in a constructive way will be slightly more strict, imposing the condition not only in Γ^* but in Γ^+

Corollary 2.1.1.2 (Constructive Lovász Local Lema for SAT). *Let F be a CNF formula. If there exists a mapping $\mu : F \to (0,1)$ that associates a number with each clause in the formula such that*

$$\forall A \in \mathcal{A} : \omega(A) \leq \mu(A) \prod_{B \in \Gamma_G(A)} (1 - \mu(B))$$

then F is satisfiable.

In order to get a result easier to check. If $k \leq 2$ the k-SAT problem is polynomial solvable so we will not be interested on such formulas.

Corollary 2.1.1.3. *Let F be a k-CNF with $k > 2$ formula such that $\forall C \in F$ and $|\Gamma_F(C)| \leq 2^k/e - 1$ then F is satisfiable.*

Proof. We will try to use 2.1.1.2. We will define such $\mu : F \to (0,1)$, $\mu(C) = e \cdot 2^{-k}$. Let $C_0 \in F$ be an arbitrary clause.

$$2^{-k} = \omega(C) \leq \mu(C) \prod_{B \in \Gamma_F(C)} (1 - \mu(B)) = e2^{-k}(1 - e2^{-k})^{|\Gamma_F(C)|}$$

With the hypothesis

$$2^{-k} \leq e2^{-k}(1 - e2^{-k})^{2^k/e-1}$$
$$1 \leq e(1 - e2^{-k})^{2^k/e-1}$$

Being famous that the convergence of the sequence $\{(1 - e2^{-k})^{2^k/e-1}\}_k$ to $1/e$ is monotonically decreasing.

□

2.1.3 Nonconstructive proof of 2.1.1

We explain the way Erdös, Lovász and Spencer originally proved the Lemma. This material is from [1] and [7]. The write-up presented here will resemble the one done by [5].

Thorough the proof we will use repeatedly the definition of conditional probability, i.e. for any events $\{E_i\}_{i \in 1,\ldots,r}$,

$$P\left(\bigcap_{i=1}^{r} E_1\right) = \prod_{i=1}^{r} P\left(E_i \middle| \bigcap_{j=1}^{i-1} E_j\right)$$

Further on this subsection we will consider Ω to be a probability space and $\mathcal{A} = \{A_1, ..., A_m\}$ to be arbitrary events in this space, G to be the lopsidependency graph, and $\mu : \mathcal{A} \to (0, 1)$ with such that the conditions of the theorem are satisfied. We first prove an auxiliary lemma.

Lemma 2.1.2. *Let $A_0 \in \mathcal{A}$ and $\mathcal{H} \subset \mathcal{A}$. then*

$$P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) \leq \mu(A)$$

Proof. The proof is by induction on the size of $|\mathcal{H}|$. The case $H = \emptyset$ follows from the hypothesis easily:

$$P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) = P(A) \leq^{1.} \mu(A) \prod_{B \in \Gamma_G^*(A)} (1 - \mu(B)) \leq^{2.} \mu(A)$$

Where 1. uses the hypothesis and 2. uses that $0 < \mu(B) < 1$. Now we suppose that $|\mathcal{H}| = n$ and that the claim is true for all $\mathcal{H}'$ such that $|\mathcal{H}'| < n$. We distinguish two cases. The induction hypothesis will not be necessary for the first of them

- When $\mathcal{H} \cap \Gamma_G^*(A) = \emptyset$ then $P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) = 0 \leq P(A)$ by definition of Γ_G^* and $P(A) \leq \mu(A)$ by definition of μ.

- Otherwise we have $A \notin \mathcal{H}$ and $\mathcal{H} \cap \Gamma_G^*(A) \neq \emptyset$. Then we can define to sets $\mathcal{H}_A = \mathcal{H} \cap \Gamma_G^*(A) = \{H_1, ..., H_k\}$ and $\mathcal{H}_0 = \mathcal{H} \backslash \mathcal{H}_A$.

$$P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) = \frac{P\left(A \cap \left(\bigcap_{B \in \mathcal{H}_A} \overline{B}\right) \middle| \bigcap_{B \in \mathcal{H}_0} \overline{B}\right)}{P\left(\bigcap_{B \in \mathcal{H}_A} \overline{B} \middle| \bigcap_{B \in \mathcal{H}_0} \overline{B}\right)}$$

We will bound numerator and denominator. For the numerator:

$$P\left(A \cap \left(\bigcap_{B \in \mathcal{H}_A} \overline{B}\right) \middle| \bigcap_{B \in \mathcal{H}_0} \overline{B}\right) \leq P\left(A \middle| \bigcap_{B \in \mathcal{H}_0} \overline{B}\right) \leq P(A)$$

Where the second inequality is given by the definition of lopsidependency graph. On the other hand, for the denominator, we can define $\mathcal{H}_i := \{H_i, ..., H_k\} \cup \mathcal{H}_0$.

$$\begin{aligned} P\left(\bigcap_{B \in \mathcal{H}_A} \overline{B} \middle| \bigcap_{B \in \mathcal{H}_0} \overline{B}\right) &= \prod_{i=1}^{k} P\left(\overline{B_i} \middle| \bigcap_{B \in \mathcal{H}_i} \overline{B}\right) \\ &\geq^{3.} \prod_{i=1}^{k} (1 - \mu(H_i)) \geq^{4.} \prod_{B \in \Gamma_G^*(A)} (1 - \mu(B)) \end{aligned}$$

Where in 3. the induction hypothesis is used, and in 4. is considering that $H_i \in \Gamma_G^*(A)$ Considering now both parts:

$$P\left(A \middle| \bigcap_{B \in \mathcal{H}} \overline{B}\right) \leq \frac{P(A)}{\prod_{B \in \Gamma_G^*(A)} (1 - \mu(B))} \leq \mu(A)$$

Where the last inequality uses the hypothesis on μ.

□

proof of the theorem 2.1.1.

$$P\left(\bigcap_{A \in \mathcal{A}} \overline{A}\right) = \prod_{i=1}^{m} P\left(\overline{A_i} \middle| \bigcap_{j=1}^{i-1} \overline{A_j}\right) \geq^{5.} \prod_{i=1}^{m} (1 - \mu(A_i))$$

Where in 5. is used 2.1.2 and since $\mu : \mathcal{A} \to (0,1)$ then $P\left(\bigcap_{A \in \mathcal{A}} \overline{A}\right) > 0$.

□

2.1.4 Constructive proof of 2.1.1.2

Moser[5] proves that it exists an algorithm such that it give an assignment satisfying the SAT formula, should it happen that the formula satisfies 2.1.1.1 conditions. This is no a big deal, as a backtrack would be also capable of providing the solution, given that we know its existence. Not so trivial is that it would run in $O(|F|)$. We will show the version of the algorithm shown in [6].

At first sight it is not clear if it terminates. If F verify 2.1.1.1 it is proved that if would end after running Repair at most $O(\sum_{C \in F} \frac{\mu(C)}{1-\mu(C)})$

s

Algorithm 1 Moser's Algorithm

$C_1, ..., C_m \leftarrow$ Clauses in F to satisfy, globally accessible
$\alpha \leftarrow$ assignment on $Var(F)$

procedure REPAIR(α, C)
    for $v \in Var(C)$ **do**
        $\alpha(v) = \text{random} \in \{0, 1\}$
    for j := 1 to m **do**
        if $(Var(C_j) \cap Var(C) \neq \emptyset) \wedge (C_j\alpha = 0)$ **then**
            Repair(C_j)

Randomly choose an initial assignment α
for j := 1 to m **do**
    if $\alpha(C_j) = 0$ **then**
        Repair(C_j)

Chapter 3

Resolution Algorithms

This chapter is fundamental as it attack the main problem of SAT: solving it. Onward we will see how it could be solved, and develop applied techniques. There are a lot of approach to this problem and they differ on it way to attack it. We have to realise that three thing are important to judge a algorithm:

- The simplicity: following Occam's razor, between to solution that do not appear to be better or worse, one should choose the easiest one. This solution are far more comprehensible and tend to be more variable and adaptable for our problem. We should not despise an easy solution to a complex problem only because far more difficult approach give slightly better results.
- The complexity: and by that I mean it algorithmic ('Big O') complexity. It is important to get good running times in all cases and have a analysis of the worst cases scenario that the algorithm could have.
- The efficiency: Some algorithms will have the same complexity as the most simple ones, but will use some plans to be able to solve the most part of the problems fast (even in polynomial time). There are some cases that would make this algorithms be pretty slow, but more often than not a trade-off is convenient.

The first section will talk about special cases, in order to continue to general ones. In general thorough the chapter we will follow the book [6].

3.1 Special Cases Solvable in Polynomial Time

In this section we will discuss some cases of the sat problem solvable in P. These cases are of interest because polynomial is no achievable in all cases. These algorithms will have incredible property and will excel in all property that were just described. Nonetheless, they only work with a subset of all possible formulas. They should be use whenever possible as no general polynomial time is believed to exists, nor it is proved its non-existence.

Definition 3.1.1. Let F be a formula. A subset $V \subset Var(F)$ is called a backdoor if $F\alpha \in \mathrm{P}$ for every assignment α that maps all V.

A goal for a SAT-solver could be to find a backdoor of minimum size. DPLL would try to search for a backdoor, using heuristics in order not to explore all subsets (only achievable if such backdoor exists).

3.1.1 Unit Propagation

Unit propagation is a simple concept that is worth standing out because it would be commonplace. Given a CNF formula F if there is a clause with only one element that should be assigned accordingly to the clause, otherwise F is unsatisfiable. This lead to the unit propagation concept. Whenever we have a unitary clause $\{p\}$ we should *resolve* it and start working with $F[p = 1]$ being $[p = 1]$ the assignment that maps the value of the metavariable p to 1, which could possibly imply mapping a variable to 0.

Also, the unit propagation might result on a recursive problem, as other unit clauses could appear.

3.1.2 2CNF

It is already know that 3CNF is equivalent to SAT. This is not known for 2CNF and is believed to be false.

Proposition 3.1.1. 2CNF is in P

Proof. To prove that 2CNF is in P, an algorithm polynomial on the number of clauses will be given. Let $F \in$ 2CNF. Without loosing of generality, we will consider that there are no clauses in F $\{u, u\}$ or $\{u, \neg u\}$ as the first one should be handle with unit propagation and the second one is a tautology. Therefore each clause is $(u \vee v)$ with $var(u) \neq var(v)$, which could be seen as $(\neg u \to v)(vu)$.

We would consider a step to be as follow: we choose a variable $x \in Var(F)$ and set it to 0. Them a chain of implication would arise, which might end on conflict. If no conflict arise, then is an autark assignment, so repeat the process. Otherwise set it to 1 and proceed. If conflict arise, then F is unsatisfiable. If no conflict arise, then is an autark assignment, so repeat the process.

Each step is of polynomial time over the number of clauses. Also there would be at most as many steps as variables, therefore we have a polynomial algorithm.

□

3.1.3 Horn Formulas

In this subsection we will analyze Horn formulas. They named after Alfred Horn, who defined them on his work[3]. They are of special interest as is HORNSAT is P-complete.

Definition 3.1.2. Let F be a formula in CNF. It is said to be a horn formula if for every $C \in F$ there is at most one non-negated literal. HORN will be the set of all horn formulas.

HORNSAT will be the intersection of HORN and SAT problems. Nonetheless, given the easiness of checking whether a formula is in HORN, it would usually consider as the problem that check the satisfiability of a horn formula.

Proposition 3.1.2. HORNSAT is in P.

Proof. Given a formula it could have a clause with only one non-negated literal or not have it. If it does not have a clause like this, set all the variables to 0 and is solved. Otherwise, unit-propagate the unary clause and repeat the process, as it would necessary be done to solve the problem. If a contradiction is raised, them the formula is not satisfiable. □

Now we will discuss a simple generalization of Horn formulas: the renamable Horn Formulas. These formulas would allow to give some use to the otherwise not really useful horn definition. They would also add a condition that can be checked efficiently.

Definition 3.1.3. Let F be a CNF formula. F is called renamable Horn if there is a subset U of the variables $Var(F)$, so that $F[x = \neg x | x \in U]$ is a Horn formula. That set would be called a renaming.

Definition 3.1.4. Let F be a CNF formula. Then a 2CNF formula F^* is defined as:

$$F^* = \{(u \vee v) | u, v \text{are literals in the same clause} K \in F\}$$

Theorem 3.1.1. *The CNF formula F is renamable Horn if and only if the associated F∗ formula is satisfiable. Moreover, if satisfying assignment α for F^* exists then it encodes a renaming U in the sense that $x \in U \iff \alpha(x) = 1$.*

Proof. Let F be renamable Horn a U be a renaming. We consider the assignment α that map to 1 all variable in U and map to 0 otherwise. Let $\{u \vee v\} \in F^*$ after the renaming. There should be at least one negative variable so if every variable is set to 0, F^* is satisfiable.

The other direction is analogous: let α be an assignment that satisfy F^*. Them there is no to literals in the same clause set to 0. Defining $U = x \in Var(F) | \alpha(x) = 1$ there is no two positives variables in a clause. □

These mean that if a renaming exists, if could be obtained efficiently, and them solve efficiently with the HORNSAT algorithm

3.2 Backtracking and DPLL Algorithms

In this section we will talk about algorithms that explore the space of possible assignments in order to find one that satisfy a give formula, or otherwise prove its non-existence. Onward whenever a formula is given, it would be a CNF formula.

3.2.1 Backtracking

We will start with the approach based on the simple and well-known backtracking algorithm:

Algorithm 2 Backtrack

1: **procedure** BACKTRACKING(F)
2:     **if** $0 \in F$ **then return** 0
3:     **if** $F = 1$ **then return** 1
4:     Choose $x \in Var(F)$
5:     **if** $backtracking(F\{x = 0\})$ **then return** 1
6:     **return** $backtracking(F\{x = 0\})$

This algorithm describe a recursion with $0(2^n)$ complexity with n being the number of variables. It also lend itself to describe a pletora of approachs varying how we choose the variable x in line 4. This algorithm will be an upper bound in complexity and a lower bound in simplicity for the rest of algorithms in this section.

An easy modification could be done to improve a little it efficency in the context of k-CNFSAT. Choosing a clause of at most k variable we could choose between $2^k - 1$ satisfying assignments. The recursion equation of this algorithm will be $T(n) = (2^k - 1) * (T(n-k))$, so it would have upper bound $O(a^n)$ with $a = (2^k - 1)^{\frac{n}{k}} < 2$.

3.2.2 Davis-Putman-Logemann-Loveland(DPLL) algorithm

This algorithm is an improve of the backtracking algorithm, still really simple a prone to multiple modifications and improve.

Algorithm 3 DPLL

1: **procedure** DPLL(F)
2: **if** $0 \in F$ **then return** 0
3: **if** $F = 1$ **then return** 1
4:
5: **if** F contains a unit clause $\{p\}$ **then return** $DPLL(F\{p = 1\})$
6: **if** F contains a pure literal u **then return** $DPLL(F\{u = 1\})$
7:
8: Choose $x \in Var(F)$ with an strategy.
9: **if** $DPLL(F\{x = 0\})$ **then return** 1
10: **return** $DPLL(F\{x = 0\})$

We could see to main differences:

- The algorithm try to look for backdoors and simplifications in line 5 and 6. Altough only some of these techniques are present, and even in some implementations skip the pure literal search, its an improve. Search for autarks assigment or renames could also be a good idea.
- It use heuristics to select variables. It does not imply that they always are choosen better (and there would be cases that run worse), but tend to be better. In practice, hard heuristics approachs give excelent results. citation needed . The roles of heuristics y to reduce the branching steps. Because of this, many heuristics functions have been proposer. For the formulation of some of them we will define:

$$f_k(u) = \text{number of ocurrences of literal } u \text{ in clauses of size k}$$

$$f(u) = \text{number of ocurrences of literal } u$$

  - DLIS(dynamic largest individual sum): choose u that maximizes f. Try first $u = 1$
  - DLCS (dynamic largest clasue sum): choose u that maximizes $f(u) + f(\neg u)$. Try first whichever has largest individual sum.
  - Jeroslaw-Wang: For the one sided versión choose u such that maximizes the sum of the weights of the clauses that include the literal. For the two sided versión choose a variable instead of a literal.
  - Shortest Clause: choose the first literal from the shortest clause, as this clause is one of the clauses with the biggest weight in F.
  - VSIDS: This heuristics function is a variation of DLIS. The difference is that once a conflict is obtained and the algorithm need to back track, the weight of that literals are increased by 1.

3.2.3 Monien-Speckenmeyer(MS) Algorithm

This algorithm is a variation of the DPLL-Shortest Clause algorithm, specifying that once you choose the shortest clause, all variable you choose should be from that clause until you satisfy it, as it will continue to be the shortest given that there is no clause with repited literals as well as no clause that is a tautology. This algorithm (DPLLSC) on k-SAT generate a recursion such that $T(n) = \sum_{i=1}^{k} T(n-i)$. Under the hipotesis that MS does not has a under-exponential worst case complexity, then $T(n) = a^n$ for some $a \in (1, \infty)$. Then

$$a^k = \sum_{i=1}^{k} T(i) = \frac{1-a^k}{1-a}$$

that solved in the ecuation $a^{k+1} + 1 = 2a^k$. The difference between MS and DPLLSC is that MS include an autark assignment search in adition to the unit clause search and generalizing the pure literal search (that would be a search of autarks of size 1). When we select a clause (the shortest) we first try to generate an autark with its variables and otherwise continue the algorithm.

Algorithm 4 DPLL

procedure MS(F)
  if $0 \in F$ **then return** 0
  if $F = 1$ **then return** 1

  if F contains a unit clause $\{p\}$ **then return** $MS(F\{p = 1\})$
  if F contains a pure literal u **then return** $MS(F\{u = 1\})$
  Choose the shortest clause $C = \{u_1, ..., u_m\}$
  for $i \in \{1, ..., m\}$ **do**
    $\alpha_1 := [u_1 = 0, ..., u_{i-1} = 0, u_i = 1]$
    if α_i is autark **then return** $MS2(F\alpha_i)$
  if $MS(F\{u_1 = 1\})$ **then return** 1
  return $MS(F\{u_1 = 0\})$

Other version of the algorithm repeat the last for in the succesive calls of F (calling $MS(F\alpha_i)$). Nonetheless we consider that with an deterministic heuristic (that, for example, choose the first clause between the set of clauses with minimum size) the result is equivalent and this provide a simpler algorithm.

For the k-SAT complexity analysis we have to consider wheter or not an autark was finded. If so, $T(n) \le T(n-1)$. Otherwise we are appliying an non autark assigment that neccesarily colide with a clause that is therefor of size $k-1$ at most. Let us denote by $B(n)$ the number of recursive calls with n variables and under the hypotesis that there is a clause with at most $k-1$ variables. Is easy to check that $O^*(T(n)) = O^*(T(n))$. In these case $T(n) \le \sum_{i=1}^{k} B(n-i)$ and $B(n) \le \sum_{i=1}^{k-1} B(n-i)$. Both of these cases are worse than $T(n-1)$ so in order to study a worst case complexity we have to study the case when no autark is finded. Under the hipotesis that $B(n) = a^n$ we get $a^k + 1 = 2^{k-1}$. For $k = 3$ we obtain $\frac{1+\sqrt{5}}{2}$.

Chapter 4

Reductions

In order to demonstrate the utility a series of reductions will be developed. This will imply a formal approach to the resolution of the problems, as well as deploying a little theoretical background to some problems when needed. Unalike other chapters this section is original work, although it generality and not being really complicated made it possible to be found on other works (maybe).

4.1 Proofs

4.1.1 Hamiltonian Cycle

By Cook theorem and the ease of checking whether a cycle is a Hamiltonian cycle, it is known that a reduction from the problem of the Hamiltonian Cycle to SAT exists. This theorem is constructive, so it effectively does give a reduction. Nonetheless, this reduction is unmanageable and in order to use SAT-solvers to improve Hamiltonian cycle resolution it would be necessary to improve it. On this subsection an alternative reduction will be proven.

Definition 4.1.1. A Hamiltonian cycle is a cycle that visit every node in a graph. The associated problem is to check, given a graph, whether whether cycle exists.

We will consider the problem of the Hamiltonian cycle of undetected graphs. Therefore an edge would have two sources instead of a source and a target as it is regarded on directed graphs. Prior to the reduction a little lemma will be proven.

Lemma 4.1.1. *Let $G = (V = \{v_1, ..., v_n\}, E = \{e_1, ..., e_m\})$ be a graph. The set $\{e_{i_1}, ..., e_{i_n}\} \subset E$ is a Hamiltonian cycle if, and only if, each vertex is the source to exactly two edges and the path $\{e_{i_1}, ..., e_{i_n}\} \subset E$ is connected.*

Proof. If each vertex is the source of an edge, then every vertex is accessible by an edge. Also, as every vertex has exactly two edges, each connected component of the graph would be a cycle. As the graph is connected there is only one of such components.

□

In order to make the reduction we will represent with Boolean clauses these two condition:

- We will start defining the variables $e_1, ..., e_n$ that will represent if the edge e_i is choose for the path. Also, if a vertex e_i has as sources v_j, v_k then the variables e_{i,v_j} and e_{i,v_k} will be also defined. The first set of formulas to consider will be:

$$e_i \iff e_{i,v_j} \iff e_{i,v_k} \qquad \forall i \in 1, .., m, \forall j, k \in 1, ..., n$$

Note than if e_j does not have as source v_j then $e_{j,v_j} \iff 0$. To ensure that each vertex is the source of exactly two edges we will define these clauses:

$$\bigwedge_{k=1}^{m} \left(\bigwedge_{i=1}^{m} \bigvee_{\substack{j=1 \\ j \neq i}}^{m} e_{j,v_k} \right)$$

In order two ensure that each vertex is source to at least two edges. Then to ensure that there would not be more than two:

$$\bigwedge_{h=1}^{m} \bigwedge_{\substack{i=1 \\ j=1 \\ k=1}}^{n} \neg e_{i,v_h} \vee \neg e_{j,v_h} \vee \neg e_{k,v_h}$$

- To prove the connectivity we will use the connectivity matrix. Henceforth all matrix will be consider as $n \times n$-sized matrix. Given $A = (a_{i,j})$ such that $a_{i,j} = 1$ if, and only if, there is a edge between v_i and v_j, otherwise $a_{i,j} = 0$. Then consider $A^k = (a^*_{i,j})$, it happens that if $(a^*_{i,j}) = 1$ then there is a path of exactly length k. Then to check the connectivity we will define $A' = \sum_{i=0}^{n} A^i$ and defining the formula:

$$(a'_{1,1} \wedge ... \wedge a'_{1,n}) \tag{4.1}$$

  Matrix product could be seen as a Boolean operation (for the purpose that we reach): Given $A = (a_{i,j})$, $B = (b_i, j)$ and $C = A \cdot B$ then

$$c_{i,j} = (a_{i,1} \wedge b_{1,j}) \vee ... \vee (a_{i,n} \wedge b_{n,j})$$

  As we do not care about the exact value of the sum in A' but only whether $a'_{i,j}$ is greater than 0 we could consider as sum the *or* operation element-wise. This prove that the expression 4.1 is a formula, a bit laborious to do by hand but quite compatible.

It simple to follow that if we could satisfy all the formulas then there would be a Hamiltonian cycle = $\{e_i \in E : e_i = 1\}$ where the second e_i is the variable and the first one is the edge. If no such cycle exists the formulas will be unsatisfiable. Further work to do would consider the implementation and resolution of the problem, and trying to express every formula in CNF.

We have resolve the problem to graph, although the same resolution is available for multigraphs (graph which could have more than one edge with the same sources), as this difference does not affect the property. The next easy results prove this statement.

Bibliography

[1] Paul Erdős and László Lovász. "Problems and results on 3-chromatic hypergraphs and some related questions". In: *Colloquia Mathematica Societatis Janos Bolyai 10. Infinite And Finite Sets, Keszthely (Hungray)*. Citeseer. 1973.

[2] Philip Hall. "On representatives of subsets". In: *Classic Papers in Combinatorics*. Springer, 2009, pp. 58–62.

[3] Alfred Horn. "On sentences which are true of direct unions of algebras". In: *The Journal of Symbolic Logic* 16.1 (1951), pp. 14–21.

[4] Robin Moser. "A constructive proof of the Lovász local lemma". In: *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 2009, pp. 343–350.

[5] Robin Moser. *Exact Algorithms for Constraint Satisfaction Problems*. Logos Verlag Berlin GmbH, 2013, pp. 17–45.

[6] Uwe Schöning and Jacobo Torán. *The Satisfiability Problem: Algorithms and Analyses*. Vol. 3. Lehmanns media, 2013.

[7] Joel Spencer. "Asymptotic lower bounds for Ramsey functions". In: *Discrete Mathematics* 20 (1977), pp. 69–76.

[8] Grigori S Tseitin. "On the complexity of derivation in propositional calculus". In: *Automation of reasoning*. Springer, 1983, pp. 466–483.