# EE570 Term Paper

**Gowtham Kaki**
gkaki@cs.purdue.edu

**Abstract**

This paper is a review of three papers from recent artificial intelligence research literature that are connected by the common theme of knowledge representation. The first paper (Li C, Zhu Z, Manya F, Simon L, Minimum Satisfiability and its Applications) defines the problem of minimum satisfiability (MinSAT) for propositional logic and constructs an algorithm based on informed bounded search of state space. Second paper (Pulina L, Tacchella A, A Structural Approach to Reasoning with Quantified Boolean Formulas) attempts to construct an efficient decision procedure for quantified boolean formulas by alternating between search and clause resolution. The last paper (Huang M, Shi X, Jin F, Zhu X, Using First-Order Logic to Compress Sentences) is an interesting application of logics based on first-order logic to compress English sentences. For evaluation we implemented the MinSatz algorithm described in first paper utilizing the rewrite system discussed in the class. While we found few inconsistencies in the algorithm as it was described in the paper, our emperical results conclude that MinSatz is efficient when compared to straightforward extensions of DPLL for MinSAT problem.

## 1 Introduction

Propositional and predicate (first-order) logics are highly expressive and widely accepted formalisms for knowledge representation. While the former forms the basis for various program logics (eg: Hoare logic) used in specifying computer programs, expressivity provided by the later is sufficient to rigorously define arithmetic (eg: Peano arithmetic). It follows that proving theorems in formalisms like Peano arithmetic reduces to deciding the validity of formulas in propositional and predicate logic. Does there exist an algorithm (called a decision procedure) that can make such decisions for propositional and predicate logics? While the answer for propositional logic is yes, it is no for first-order logic. Nevertheless, it is possible to construct a decision procedures for various subsets of predicate logic which are strictly weaker than full first-order logic. Conversely, it is also possible to construct decision procedures for propositional logic strengthened with more constraints. The rest of this paper reviews three papers. While the first two attempt to construct decision procedures for strengthened and weakened propositional and predicate logic respectively, the last paper is an interesting application of first-order logic in natural language processing and serves as a testimony to the expressivity of the logic.

## 2 Minimum Satisfiability Problem

### Introduction

Given a formula in propositional logic, satisfiability problem (SAT) asks whether there exists an assignment to literals such that the formula evaluates to true. SAT problem is one among the first problems to be proven NP-Complete. Nevertheless, there exist many decision procedures, which make use of linear time conversion of formula to conjunctive normal form (CNF), clause resolution, and other such heuristics, to effeciently solve SAT. Since many NP-hard problems of practical significance can be reduced to SAT problem, an efficient decision procedure for SAT is imperative for effeciently solving those problems.

MinSAT is an extension of SAT problem, where a formula (in CNF form) contains two types of clauses - hard and soft. The aim is to find an assignment to literals that satisfies all the hard clauses and minimizes the number of satisfied soft clauses. SAT is a special case of MinSAT where there are only hard clauses. The solution for MinSAT problem might result in a satisfying assignment that is completely different from that of its corresponding SAT problem. For instance, for the CNF formula:

$$(x1 \ \vee \ x2) \ \wedge \ (x2 \ \vee \ x3)$$

Assigning *true* to all literals is a solution to SAT, whereas, together with soft clauses $x1 \wedge x2 \wedge x3$, solution to MinSAT assigns *false* to $x1$ and $x3$, and *true* to $x2$ since this assignment results in minimum number of satisfied soft clauses after satisfying all hard clauses.

Weighted Partial MinSAT problem is MinSAT problem where each soft clause is associated with certain weight. The objective is to minimize the sum of weights of satisfied soft clauses. The paper (Li C, Zhu Z, Manya F, Simon L, Minimum Satisfiability and its Applications) proposes a branch-and-bound algorithm, called *MinSatz*, for Weighted Partial MinSAT equipped with a novel bounding technique, and reports on an emperical investigation. In the next sub-section, we cover some preliminaries and introduce the MinSatz algorithm as presented in the paper. The following sub-section

is an original critique of the algorithm, and the final subsection discusses our experience in implementing the MinSATZ algorithm.

## MinSatz

A literal is a propositional variable or a negated propositional variable. A clause is a disjunction of literals. A weighted clause is a pair $(c, w)$, where $c$ is a clause and $w$, its weight, is a natural number or infinity. A clause is hard if its weight is infinity; otherwise it is soft. A Weighted Partial MinSAT (MaxSAT) instance is a multiset of weighted clauses $\phi = \{(h_1, \infty), ..., (h_k, \infty), (c_1, w_1), ..., (c_m, w_m)\}$, where the rst k clauses are hard and the last m clauses are soft. The Weighted Partial MinSAT problem for instance $\phi$ consists in finding an assignment with minimum cost satisfies all hard clauses. MinSAT problem is Weighted Partial MinSAT problem where weights of all soft clauses is 1. Since solution for MinSAT can be trivially extended to include weights, the paper as well as this review focuses on the exact MinSAT problem. In the following discussion, definitions for *clique*, *maximum clique* and *clique partition* for an undirected graph follow the usual definitions and reader is referred to the paper for formal definitions.

MinSatz can be seen as an extension to the famed Davis-Putnam-Logemann-Loveland (DPLL) algorithm for propositional CNF-SAT solving. Algorithm 1 shows the pseudocode for DPLL. DPLL is, essentially, a backtracking search

**Data**: $\phi$ : Set of CNF clauses
$\Gamma$ : Current assigment to variables
**Result**: A (possibly empty) model for the formula.
($\phi$,$\Gamma$) := unitPropagate ($\phi$,$\Gamma$);
**if** $\phi$ *contains empty clause* **then**
  return {}
**end**
**if** $\phi$ *is empty* **then**
  return $\Gamma$
**end**
v := selectVariable ($\phi$);
**return** (DPLL ($\phi \cup$ {v},$\Gamma$) $\vee$ DPLL ($\phi \cup$ {$\neg$v},$\Gamma$))

**Algorithm 1:** DPLL ($\phi$,$\Gamma$)

algorithm aided by a procedure called unit propagation. A unit clause is a CNF clause that contains a single literal ($l$). Unit propagation relies on the fact that there is only one way to satisfy a unit clause. For each unit clause in the set of clauses, unit propagation creates a new assignment necessary to satisfy that clause and propagates that assignment in the following way. Clauses containing literal $l$ are removed from the set of clauses since they are already satisfied. On the other hand, every clause containg literal $\bar{l}$ is replaced with a clause with literal $\bar{l}$ removed.

MinSatz extends DPLL with lower bound on the cost of satisfied soft constraints. The algorithm, as described in the paper, is given below: As with DPLL, MinSatz starts with unit propagation, but restricts it to hard clauses as soft clauses need not be necessarily satisfied. MinSatz returns if hard unit propagation resulted in contradiction or no more

**Data**: $\phi$ : Set of hard and soft CNF clauses
LB : Lower bound (Lowest cost incurred so far)
**Result**: Minimum cost of satisfied soft constraints
($\phi$) := hardUnitPropagate ($\phi$);
**if** $\phi$ *contains empty clause* **then**
  return -1
**end**
**if** $\phi$ *is empty* $\vee$ $\phi$ *only contains empy soft clauses* **then**
  return n-empty($\phi$)
**end**
UB := n-empty($\phi$) + overestimation($\phi$);
**if** *UB* $\leq$ *LB* **then**
  **return** LB
**end**
v := selectVariable ($\phi$);
LB := MinSatz ($\phi \cup$ {v},LB);
LB := MinSatz ($\phi \cup$ {$\neg$v},LB);
**return** LB

**Algorithm 2:** MinSatz ($\phi$,LB)

empty clauses left. Otherwise, it computes an upper bound of the maximum number of soft clauses that are falsified (UB) if the current partial assignment is extended to a complete one. This number UB is then compared with the number of clauses falsified in the best assignment found so far (LB). If UB $\leq$ LB, a better solution cannot be found from the current node, and MinSatz backtracks. Otherwise, it proceeds similar to DPLL, but while updating the lower bound with the best solution found so far.

To compute UB, MinSatz uses a novel method to overestimate the number of satisfiable soft clauses. To capture the intuition, consider the two soft clauses containg literals $l$ and $\bar{l}$. It it easy to see that both cannot be simultaneously unsatisfiable. Now, consider the case where there are two unit soft clauses with literals $l_1$ and $l_2$, respectively, and a hard clause $\{l_1,l_2\}$. Again, both soft clauses cannot be simultaneously unsatisfiable as that results in unsatisfiability of a hard clause. MinSatz constructs a graph G that contains a vertex for every soft clause, and an edge between two vertices if corresponding soft clauses cannot be simultaneously unsatisfiable (as described above). Observe that number of cliques in the clique partition of G gives an upper bound on number of soft clauses that can be rendered unsatisfiable if the current assignment is completed.

## Critique of MinSatz

A closer look at MinSatz reveals some cases where the algorithm seems to deviate from the expected behaviour:

**Case when all soft clauses are empty** Consider the example where CNF formula consists of following hard part:

$(x1 \vee x2) \wedge (\neg x1 \vee x2) \wedge (x1 \vee \neg x2) \wedge (\neg x1 \vee \neg x2)$

Assume that soft part contains a single clause $\neg x1$. Clearly, the hard part of the formula is unsatisfiable, therefore MinSatz is expected to return -1 (signaling error) or 0 (denoting that no soft clauses can be satisfied). However, MinSatz returns 1. This anamoly arises as MinSatz does not confirm

if the current assignment can be completed to satisfiability once it finds that all soft clauses are empty (line number 5 of the algorithm). For the current example, in the recursive call where $x1$ is set to *true*, the sole soft constraint is unsatisfied and MinSatz immediately returns 1. Further recursive calls where $x1$ is set to *false* have 1 as the lower bound, which cannot be exceeded. As a result, MinSatz concludes that maximum number of unsatisfiable soft clauses is 1.

Notwithstanding this unexpected behaviour on CNF formulas, there is a well-defined sub-class of CNF formulas where approach taken by MinSatz can be proven correct, albeit in a non-trivial way. Such class is the class of CNF formulas with Horn clauses. A Horn clause is a clause that contains atmost one positive literal (a non-negated variable). A well-known result states that a Horn CNF forumla without a unit clause is always satisfiable. Since hard unit propagation results in a formula that does not contain unit clauses, if MinSatz is invoked on Horn CNF formula, it is always the case that there exists a satisfying assignment that completes the current assignment. It is therefore sound to return immediately, without checking the satisfiability of hard part, if there are no non-empty soft clauses.

### Case when first recursive call encounters unsatisfiablility

Consider a point in the execution of MinSatz where lower bound found so far is a positive number (k). Assume that first recursive call made (at line 13) determines that the formula is unsatisfiable. It, therefore, returns -1, which becomes new lower bound for second recursive call (line number 14). Unfortunately, the information of correct lower bound is now lost. It is therefore quite conceivable that second recursive call finds the formula satisfiable and returns a number (k') less than k as result, which is definitely not optimal as previous recursive calls have already found a better solution (k $>$ k').

### Case when multiple satisfying assigments possible

Consider the example where CNF formula consists of following hard part:

$$(x1 \vee x2) \wedge (\neg x1 \vee x3)$$

Let soft part of the formula contain three unit clauses - $\neg x1$, $\neg x2$, and $\neg x3$. Consider the case when $selectVariable$ selects variable $x1$ and it is set to true in the subsequent recursive call. As described previously, unit propagation in the the recursive call immediately removes both hard clauses (since both are satisfied) and returns a formula with empty hard part. Since there are no more hard clauses, MinSatz returns 1 as only one soft constraint is unsatisfied under current assignment of $\{x1 \mapsto true\}$. This is clearly sub-optimal as $x2$ and $x3$ can be interpreted in a way so as to make rest of the soft constraints unsatisfiable as well. Therefore, correct answer in this case should be 3.

### MinSatzEE570

We have corrected the anamolies described in previous section and constructed MinSatzEE570. Algorithm 3 describes MinSatzEE570. MinSatzEE570 uses a modified unit propagation procedure (hardUnitPropagateEE570) that forces MinSatz to generate interpretation for *every* variable in hard part regardless of its relevance to satisfiability of hard clauses. When propagating a unit clause containing literal $l$, hardUnitPropagateEE570 behaves like hardUnitPropagate for clauses containing $\bar{l}$. However, for a clause ($C$) containing $l$ along with $k$ other literals ($l_1$,$l_2$,...,$l_k$), hardUnitPropagateEE570 removes $C$, but adds k clauses of form $l_i \vee \bar{l_i}$, where $1 \leq i \leq k$. The newly added clauses are tautologies and do not contrain the satisfiability of formula. However, they preserve the variables that would have been lost otherwise, letting MinSatz to assign interpretation that leads to least cost assignment.

**Data**: $\phi$ : Set of hard and soft CNF clauses
LB : Lower bound (Lowest cost incurred so far)
**Result**: Minimum cost of satisfied soft constraints
1 ($\phi$) := hardUnitPropagateEE570 ($\phi$);
2 **if** $\phi$ *contains empty clause* **then**
3 | return -1
4 **end**
5 **if** $\phi$ *is empty* **then**
6 | return n-empty($\phi$)
7 **end**
8 **if** *all soft clauses in* $\phi$ *are empty* **then**
9 | **if** *DPLL(hard($\phi$),{}) returns non-empty model* **then**
10 | | return n-empty($\phi$)
11 | **else**
12 | | return -1
13 | **end**
14 **end**
15 UB := n-empty($\phi$) + overestimation($\phi$);
16 **if** *UB* $\leq$ *LB* **then**
17 | **return** LB
18 **end**
19 v := selectVariable ($\phi$);
20 LB1 := MinSatz ($\phi \cup \{v\}$,LB);
21 LB := (LB1$\leq$0)?LB:LB1;
22 LB2 := MinSatz ($\phi \cup \{\neg v\}$,LB);
23 **if** *LB2 = -1* $\wedge$ *LB1 = -1* **then**
24 | **return**-1
25 **else**
26 | **return** (LB2$\leq$0)?LB:LB2
27 **end**

**Algorithm 3:** MinSatzEE570 ($\phi$,LB)

When all soft clauses in the formula are empty, but hard part is not yet satisfied, it should be observed that MinSAT reduces to SAT. MinSatzEE570, therefore calls DPLL on the remaining hard part and returns appropriate value depending on whether DPLL was able to find a model. Lastly, the lower bound calculation is adjusted for the fact that MinSAT returns -1

### Implementation

We have implemented DPLL and MinSatzEE570 in racket (a dialect of Scheme). A generic term rewrite system de-

scribed in the class has been used to simplify formula during unit propagation. We evaluated MinSAT by varying the schemes of over-estimation between 1. A naive scheme which assumes all soft-constraints that are not yet satisfied or unsatisfied, to be satisfiable, and 2. A more intelligent scheme based on clique construction as described in the paper. Since DIMACS graph dataset used for evaluation in the paper requires large effort to parse and encode graphs as propositional formulas, we used relatively small and contrived examples for evaluation. We observed that second scheme of over-estimation marginally outperforms first scheme for examples with high number of mutually unsatisfiable soft constraints. Both the source code and data are available online for reader's peruse [1].

## 3 Deciding Quantified Boolean Formulas

### Introduction

First-order logic extends propositional logic with universal and existential quantification over variables, along with predicate and function symbols. It is well-known that finding models for full first order formulas is an undecidable problem. The second paper (Pulina L, Tacchella A, A Structural Approach to Reasoning with Quantified Boolean Formulas) therefore considers a weak subset of first-order logic, which is propositional logic extended with quantification over boolean variables. It is a known result that formulas in such logic can be converted in linear time to a prenex-quantified conjunction of clauses. Formulas in such form are called quantified boolean formulas. The paper reprises several approaches proposed for deciding quantified boolean formulas and proposes a method (called STRUQS solver) that applies them alternatively for better results. Results of evaluating the solver over QBFEVAL08 dataset have been presented.

### Preliminaries

The definitions of literal, its complement, clause and propositional CNF formula from previous section are still considered valid. A quantified boolean formula is a formula of form $\varphi = Q_1 z_1 Q_2 z_2 ... Q_n z_n \phi$ where

- $Q_i$ is a quantifier, either existential $\exists$ or universal $\forall$. The part $Q_1 z_1 Q_2 z_2 ... Q_n z_n$ is called prefix of $\varphi$.
- $z_1, ..., z_n$ are distinct variables, and
- $\phi$, called matrix of $\varphi$, is a propositional formula.

The level of a variable $z$ is defined as 1 + number of quantifiers succeeding the quantifier of $z$ in the prefix of the formula. The level of a literal is the level of its corresponding variable. The semantics of QBFs is defined as:

- If prefix is empty, then semantics of $\varphi$ is same as semantics of propositional formula $\phi$
- If $\varphi = \exists\, x\psi$, $\varphi$ is true iff $\varphi_x$ is true or $\varphi_{\bar{x}}$ is true.
- If $\varphi = \forall\, x\psi$, $\varphi$ is true iff both $\varphi_x$ and $\varphi_{\bar{x}}$ are true.

Few observations can be made immediately from the above semantics:

[1] https://github.com/gowthamk/ee570

- Semantics of any QBF can be represented as a tree with propositional formulae as its leaves and only $\wedge$ or $\vee$ as internal nodes. We call such a tree as semantic tree of QBF. Semantic tree can be systematically constructed by eliminating quantifiers in prefix from left to right while adding a $\wedge$ -node for universal quantifier, and $\vee$ -node for existential quantifier.
- Order of quantifiers in the prefix of a QBF matters, as changing the order results in different semantic trees. For example, the two QBFs:

$$\forall\, x\, \forall\, y\, \exists\, z.(x \vee z) \wedge (y \vee \neg z)$$
$$\forall\, x\, \exists\, z\, \forall\, y.(x \vee z) \wedge (y \vee \neg z)$$

  give rise to completely different interpretations.
- If all literals in QBF contains are universally quantified, then determining satisfiability of the QBF is equivalent to determining validity of its matrix. This follows from the fact that for QBF to be satisfiable, the matrix should evaluate to true under all possible interpretations of variables, which means that matrix should be a tautology.
- If matrix of QBF is a CNF formula, then each clause is referred to as QBF clause. If all the literals in a QBF clause universally quantified, the clause and, consequently, the QBF evaluate to false. Such a clause is called contradictory. This is because disjunction of literals is not a tautology.

### Backtracking Search

It is easy to conceive a decision procedure for QBFs based on naive backtracking search of its semantic tree. Such a procedure is $O(2^n)$ times as expensive as propositional satisfiability procedure, where n is the number of variables in QBF. Improvements on naive search, which make use of aforementioned observations, are possible. For instance, instead of unrolling the semantic tree entirely, search can backtrack if encounters a contradictory clause.

### Clause Resolution

Clause resolution is a deductive technique for propositional logic. It is captured by the following rule:

$$\frac{\Sigma \models \Phi_1 \vee ... \vee \Phi_n \vee \Psi \qquad \Sigma \models \neg\Psi \vee \Theta_1 ... \vee \Theta_m}{\Sigma \models \Phi_1 \vee ... \vee \Phi_n \vee \Theta_1 ... \vee \Theta_m}$$

Clause resolution rule for QBF clauses is similar to the above rule, except that

- The literal that occurs with different polarity in both clauses ($\Psi$ above) should be an existential literal
- $\Phi_i \neq \Theta_j$ where $0 \leq i \leq n$, and $0 \leq j \leq m$

Clause resolution for QBFs, unlike that for propositional logic, is complete (kleine et al., Resolution for Quantified Boolean formulas). It is therefore possible to construct a decision procedure for QBFs solely based on clause resolution by systemic variable elimination. In this paper, supported by emperical evidence, authors argue that interleaving clause resolution with backtracking search results in an effecient decision procedure than using only of the two methods. They

implement their decision procedure in STRUQS solver. The algorithm for STRUQS was given in figure 2 of the paper. We refrain from reproducing due to space constraints. We do, however, describe its salient features below:

- STRUQS uses optimized version of backtracking procedure (referred to as *BackJumping* or $BJ$) and variable elimination through clause resolution procedure (referred to as *VariableElimination* or $VE$.
- STRUQS uses a heuristic to decide between the two methods to use. It first checks if there is any literal ($l$) that can be eliminated through clause resolution. If there is such a literal, it calculates the product (called *diversity* or *div*) of number of clauses in which $l$ occurs and number of clauses in which $\bar{l}$ occurs. It chooses $VE$ if and only if *div* exceeds a pre-determined thereshold. Otherwise it chooses $BJ$.

### Evaluation

The paper presents experimental results of using STRUQS to decide QBFs from QBFEVAL'08 dataset. It was observed that with a *div* value of 2000, STRUQS was able to solve 39% of entire dataset in 600s of CPU time. This performance would have ranked STRUQS third among all competitors of QBFEVAL'08.

## 4 First-Order Logic to Compress Sentences

### Introduction

The third paper (Huang M, Shi X, Jin F, Zhu X, Using First-Order Logic to Compress Sentences) presents a novel method to compress English sentences. Sentence compression aims to produce a condensed version of the original sentence while retaining the most important information and making the compressed sentence grammatical. Such compression finds practical use in scenarios such as automatically answering questions posed by humans, where it is used to compress and extract the gist of the question in order to find best matches. The novelty of method proposed in the paper lies in the new deletion-based compression model, where sentences are compressed by deleting not only words, but other dependent words in the sentence as well. Following sections describe the process in detail. These dependencies are captured by first-order logic formulas. A Markov Logic Network (MLN) is used to make probabistic inferences.

### Markov Logic Network

A Markov Logic Network is a probabilistic deductive system that makes uncertain inference about a first-order system, when given annotated rules of that system, where rules are annotated with relative weights (probabilities). For example, for appropriately annotated propositional logic, MLN deduces a tautology with probability of 1 and contradiction with probability of 0. More formally, a Markov logic network $L$ is a set of pairs ($\varphi$, $w$), where $\varphi$ is a formula in first-order logic and $w$ is a weight for the formula, a real number.

### Rules in First-Order Logic

Consider the sentence *"I am positively sure"*. Removing the word *"positively"* from the sentence does not result in much loss of information. However, sentence loses its sense if the word *"sure"* is removed, instead. Looking at similar examples, one might conclude that adjectives are not imperative to convey the meaning and decide to compress sentences by removing adjectives. This can be captured as the following rule in logic:

$$adjective(i) \Rightarrow delete(i)$$

In the above rule, $adjective$ is a predicate which is true if word at index $i$ is an adjective. $delete(i)$ is a predicate, which, when true, prompts sentence compressor to delete the word at index $i$.

After closer examination, once might encounter a sentence, such as *"the relative velocity of object is 2 km/s"*, from which removing adjective *"relative"* changes its meaning. To compensate, the following rule can be added:

$$adjective(i) \ \wedge \ (\ \exists\, j.property(words[i, i+i], j)) \Rightarrow \neg delete(i)$$

The above rule conveys that if word at index $i$ is an adjective and there exists a word in the sentence whose property is being described by words at $i$ and $i+1$, then the adjective should not be deleted. However, premises of the later rule subsume those of previous rule, yet it arrives at a contradictory conclusion. To avoid the contradiction, one might add weights to rules with the underlying assumption that a rule with higher weight overrides one with lower.

The previous two rules make local decisions about deleting a word without concerning with deletions being performed in the rest of the sentence. Such rules alone cannot capture dependencies between deletions. For instance, consider the sentence *"He ate red meat and mangoes"*. Removing the word *"meat"* requires deleting *"red"* also, to restore the consistency. Such a dependency is capture by rule given below:

$$adjective(i) \ \wedge \ delete(i+1) \Rightarrow delete(i)$$

Such rules, which capture constraints between multiple deletes, are known as global rules.

The deletion model for sentence compression described in the paper is based on multiple such rules, all of which have either $delete$ or $\neg delete$ as their conclusion. Such rules are formulated based on constraints of English grammar and experience of processing datasets. Predicates based on parts-of-speech, such as $adjective$ can be constructed with help of NLP parsers (eg: Stanford parser) that tag words with their parts of speech. Inference in presence of rules with weights can be carried our usign Markov Logic Network (MLNs) described in previous section. MLNs can be trained to learn relative weights of rules and strength of dependency between different words using a training set. The paper uses pre-existing tools to perform the same.

**Evaluation** The Markov Logic Network created with multiple rules, some of which are described in the paper, was used for compressing a corpus of written and spoken sentences drawn from various newspaper articles and commentary. Nearly one-third of the corpus was used to train

the MLN. The compressed sentences are then compared with outputs of other compression schemes as well as human generated compression, which is considered gold standard. Overall, the scheme performed bettwen then known automated compression schemes on written corpus, coming close to human-generated compression. For the spoken corpus, however, the proposed scheme is worse than existing schemes as adjudged by human evaluators.