MajorSat: A SAT Solver to Majority Logic

Yu-Min Chou
Computer Science
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.
gogomyjet@yahoo.com.tw

Yung-Chih Chen
Computer Science & Engineering
Yuan Ze University
Taoyuan, Taiwan, R.O.C.
ycchen.cse@saturn.yzu.edu.tw

Chun-Yao Wang, Ching-Yi Huang
Computer Science
National Tsing Hua University
Hsinchu, Taiwan, R.O.C.
wcyao@cs.nthu.edu.tw, s9862516@m98.nthu.edu.tw

Abstract—A majority function can be represented as sum-of-product (SOP) form or product-of-sum (POS) form. However, a Boolean expression including majority functions could be more compact compared to SOP or POS forms. Hence, majority logic provides a new viewpoint for manipulating the Boolean logic. Recently, majority logic attracts more attentions than before and some synthesis algorithms and axiomatic system for majority logic have been proposed. On the other hand, solvers for satisfiability (SAT) problem have a tremendous progress in the past decades. The format of instances for the SAT solvers is the Conjunctive Normal Form (CNF). For the instances that are not expressed as CNF, we have to transform them into CNF before running the SAT-solving process. However, for the instances including majority functions, this transformation might be not scalable and time-consuming due to the exponential growth in the number of clauses in the resultant CNF. As a result, this paper presents a new SAT solver—MajorSat, which is for solving a SAT instance containing majority functions without any transformation. Some techniques for speeding up the solver are also proposed. Besides, we also propose a transformation method that can generate the characteristic function of a majority logic gate. The experimental results show that the MajorSat solver can efficiently solve random instances containing majority functions that CNF SAT solvers, like MiniSat or Lingeling, cannot.

I. Introduction

The majority function is a concise way to represent a Boolean expression. A Boolean expression sometimes can be more compact by containing majority functions in it rather than being expressed as sum-of-product (SOP) form or product-of-sum (POS) form [1]. Recently, majority logic attracts more attentions than before and some synthesis algorithms and axiomatic system for majority logic have been proposed [1][2][14]. In [14], the SOP expression is transformed into majority logic for reducing hardware cost in quantum cellular automata devices. In [2], a synthesis method on majority-inverter graphs was developed to optimize the area, depth, and power of logic circuits. In [1], a two-level majority representation is presented for expressing Boolean functions.

On the other hand, in the past decades, the conjunctive-normal-form (CNF) solvers [3][4][6][7][16][17][18] for the Boolean satisfiability (SAT) problem have had a remarkable achievement and have been widely used in the domains of synthesis and verification of logic circuit. Despite the fact that these traditional solvers are quite efficient and mature, they are not applicable to the instances that are not represented as CNF. That is, we have to transform the non-CNF instances into CNF ones before running the SAT-solving process. However, for the instances containing majority functions, this transformation might be not scalable and time-consuming due to the exponential growth in the number of clauses of the resultant CNF.

To address this issue, we propose a new solver, MajorSat, which can directly solve instances with majority functions instead of converting majority functions into clauses and feeding to the CNF SAT solvers. The conciseness of majority functions in representation also alleviates memory consumption required to store a large number of clauses in the implementation.

The proposed MajorSat consists of three parts, and they are conflict analysis, conflict-driven learning, and variable decision order heuristic. First, the conflict analysis quickly detects conflicts within or among the majority functions if they exist. Then the conflict-driven learning process accelerates the solver by effectively pruning the search space. Finally, a variable decision order heuristic selects the next variable to be evaluated based on the current state of solving.

We also propose a transformation method to generate the characteristic function of a majority logic gate in the format of majority function. As a result, the SAT problem in majority logic networks can be solved accordingly.

We conducted two experiments in this work. In the first experiment, we would like to show the correctness of the proposed solver. We randomly selected 3-SAT benchmarks from SATLIB [19] and transformed them into the conjunction of majority functions by adding $n-1$ constant 1 into a clause of size n. Then we applied the MajorSat to solve these benchmarks and compared the results. The experimental results show that the MajorSat obtained the correct results.

In the second experiment, we generated random benchmarks in the format of the conjunction of majority functions for evaluating the efficiency of MajorSat due to no existing benchmarks in the format of conjunction of majority functions. The experimental results show that the MajorSat solved all the benchmarks, while CNF SAT solvers, MiniSat and Lingeling, run out of time limit, 1000 seconds.

The main contributions of this work are three-fold:

1. This is a SAT solver for solving the instances containing

This work is supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 103-2221-E-007-125-MY3, MOST 103-2221-E-155-069, NSC 100-2628-E-007-031-MY3, NSC 101-2221-E-155-077, NSC 101-2628-E-007-005, NSC 102-2221-E-007-140-MY3, and NSC 102-2221-E-155-087.

majority functions directly.

2. Some properties about majority functions for analyzing conflicts are presented.
3. An implication method and a variable decision order heuristic are presented.

The rest of this paper is organized as follows. Section II gives the background of the majority logic and the satisfiability problem. Section III presents the proposed MajorSat solver. Section IV presents the transformation method for majority logic gates. Section V shows the experimental results. Section VI concludes this work.

II. Preliminaries

In this section, we give the background of the majority logic and SAT problem.

A. Majority function

A literal is a variable a or its negation $\bar{a}$. A clause is a disjunction (OR) of literals. A majority function is a function with an odd number of literals and constants 0, 1 as the inputs, and its logic value is evaluated as 1 if and only if more than half of inputs are 1. A majority function is denoted as M() and the number of inputs in a majority function represents its size.

Example 1: A majority function M(a, $\bar{b}$, 1) of size 3 is evaluated to logic value 1 if and only if 2 or 3 inputs of a, $\bar{b}$ and 1 are 1.

B. Satisfiability

The Boolean SAT problem is to determine if there exists an assignment to the variables so that the Boolean formula is 1. Transforming a majority function into a CNF requires an exponential number of operations with respect to the size of a majority function.

Example 2: A majority function M(a, b, c, d, e) of size 5 is logically equivalent to a CNF $(a \vee b \vee c) \wedge (a \vee b \vee d) \wedge (a \vee b \vee e) \wedge (a \vee c \vee d) \wedge (a \vee c \vee e) \wedge (a \vee d \vee e) \wedge (b \vee c \vee d) \wedge (b \vee c \vee e) \wedge (b \vee d \vee e) \wedge (c \vee d \vee e)$. This CNF contains $\binom{n}{\lceil n/2 \rceil}$ clauses, where n is the size of the majority function.

According to Example 2, we realize that a majority function can represent a CNF with a lot of clauses and thus reduces the complexity of space and time required in representation.

Definition 1: A majority expression, denoted as ME, is a conjunction of majority functions.

Example 3: M(a, b, $\bar{c}$) $\wedge$ M($\bar{d}$, $\bar{e}$, 0, f, g) is an ME.

III. MajorSat

This section presents the proposed MajorSat solver to majority logic. Its overall flow is shown in Fig. 1. The conflict analysis is performed first for quickly detecting conflicts within or among the majority functions if they exist. Then the conflict-driven learning with a variable decision order heuristic accelerates the solver by effectively pruning the search space.

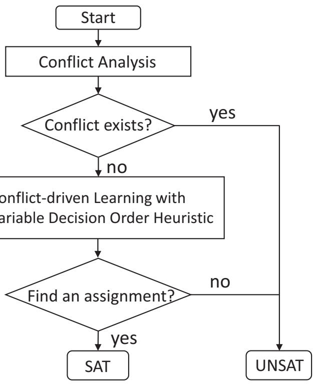

Fig. 1. The overall flow of MajorSat.

A. Conflict Analysis

This subsection presents several properties for analyzing conflicts among majority functions in an ME.

Unlike a clause in CNF, the size of a majority function within an ME can be reduced without altering the result of the ME. Property 1 states this observation.

Property 1: A majority function with size n can be simplified to a majority function with size $(n-2)$ by removing two inputs that are either the same variable but with different phases, or 0 and 1, while preserving the same result.

Example 4: M(a, a, b, c, $\bar{a}$, 0, 1) can be reduced to M(a, b, c, 0, 1) since a and $\bar{a}$ are opposite to each other in their phases. M(a, b, c, 0, 1) can be further reduced to M(a, b, c).

Conflicts could exist among majority functions in an ME. Property 2 states an observation about this.

Property 2: Given an ME, it is UNSAT if it satisfies the following conditions simultaneously.
(1) There exists a majority function of size n with an input $x \in \{a, \bar{a}, 0, 1\}$ appearing more than $\lfloor n/2 \rfloor$ times.
(2) There exists another majority function of size m with an input $y \in \{a, \bar{a}, 0, 1\}$ appearing more than $\lfloor m/2 \rfloor$ times.
(3) $x = \bar{y}$.

Example 5: An ME: M(b, b, b, c, d) $\wedge$ M(d, e, f, $\bar{b}$, $\bar{b}$, $\bar{b}$, $\bar{b}$) is UNSAT since the input b appears $3 > \lfloor 5/2 \rfloor$ times in the first majority function, the input $\bar{b}$ appears $4 > \lfloor 7/2 \rfloor$ times in the second majority function, and $b = \bar{\bar{b}}$.

Definition 2: Given a majority function s, the minimum number of variables required to be assigned such that s is 1 is denoted as $MinV_s$.

Example 6: For a majority function s: M(a, a, b, c, $\bar{d}$), $MinV_s$ is 2 since it requires at least two variables to be assigned to make $s = 1$. The choices of these variables are $(a, b) = (1, 1)$, $(a, c) = (1, 1)$, or $(a, d) = (1, 0)$.

When two majority functions in an ME have the same variable set, there is a property for detecting UNSAT of the ME as follows.

Property 3: Given two majority functions s and t, the ME $=$ $s \wedge t$ is UNSAT if both of the following conditions hold.
(1) s and t have the same variable set with size w, and the phase of each variable in s is opposite to the phase of each variable in t.
(2) Both $MinV_s$ and $MinV_t > \lfloor w/2 \rfloor$.

Example 7: A trivial case is that M(a, b, c) $\wedge$ M($\bar{a}$, $\bar{b}$, $\bar{c}$) is UNSAT. Given an ME $= s \wedge t =$ M(a, b, $\bar{c}$) $\wedge$ M($\bar{a}$, $\bar{b}$, $\bar{b}$, $\bar{b}$, c, c, c), the size w of the variable set $\{a, b, c\}$ is 3. This ME is UNSAT since both $MinV_s = 2$ and $MinV_t = 2$ are larger than $\lfloor w/2 \rfloor = 1$.

In addition to the above properties, we can also extract conflicts from an ME through the implications among the related literals in majority functions. By analyzing implications existing in majority functions of an ME, we can build an implication graph such that the unsatisfiability of the original ME could be earlier determined by the implication graph. That is, if there exists a conflict among the literals in the implication graph, we can realize that the original ME is UNSAT. Note that all the majority functions of size larger than or equal to 3, except for tautology and contradiction, bring implications. The rule of implications for a general majority function of size n is stated as follows.

Property 4: Given a majority function of size n, resolving a literal a (assigning 0 to the literal a) that occurs k times in the function implies all the other literals that occur greater than or equal to $\lceil n/2 \rceil$ - k times for satisfying the majority function.

Example 8: Given a majority function M(a, a, b, b, $\bar{c}$, $\bar{c}$, d) of size 7, resolving (assigning 0 to) the literal a implies both literals b and $\bar{c}$ to 1 for satisfying the majority function. This is because the literal a occurs k = 2 times, and both literals b and $\bar{c}$ occur two times, which is equal to $\lceil 7/2 \rceil$ - 2 = 2. However, d cannot be implied by $\bar{a}$ due to dissatisfaction of Property 4.

Next, let us introduce how to build an implication graph from the implication relationship obtained by resolving the majority functions. In an implication graph, a node represents a literal, and a directed edge between nodes represents the implication between them. Given two variables a and b, if assigning $a = 0$ implies $b = 1$, denoted as $\bar{a} \rightarrow b$, a directed edge from node $\bar{a}$ to node b is formed in the implication graph. Note that $\bar{a} \rightarrow (b \wedge c)$ is equivalent to $(\bar{a} \rightarrow b) \wedge (\bar{a} \rightarrow c)$. After extracting all the implications based on Property 4, a complete implication graph can be built. Then we start to find the strongly-connected components (SCCs) [11] in the graph. If there exists an SCC that contains a variable with opposite phases, we can assert that the original ME is UNSAT.

Example 9: Given an ME: M(a, b, c) $\wedge$ M($\bar{b}$, $\bar{c}$, d) $\wedge$ M($\bar{a}$, $\bar{c}$, $\bar{d}$). According to Property 4, we can extract the implications as follows:
M(a, b, c): $\bar{a} \rightarrow (b \wedge c)$, $\bar{b} \rightarrow (a \wedge c)$, $\bar{c} \rightarrow (a \wedge b)$.
M($\bar{b}$, $\bar{c}$, d): $b \rightarrow (\bar{c} \wedge d)$, $c \rightarrow (\bar{b} \wedge d)$, $\bar{d} \rightarrow (\bar{b} \wedge \bar{c})$.
M($\bar{a}$, $\bar{c}$, $\bar{d}$): $a \rightarrow (\bar{c} \wedge \bar{d})$, $c \rightarrow (\bar{a} \wedge \bar{d})$, $d \rightarrow (\bar{a} \wedge \bar{c})$.
The implication graph can be built from these implications as shown in Fig. 2. Then we identify the implication graph itself is an SCC that contains literals a and $\bar{a}$ as shown in Fig. 3. Thus, the ME is UNSAT.

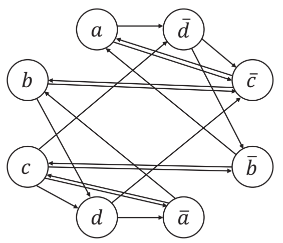

Fig. 2. The implication graph containing all the implications extracted from the original ME in Example 9.

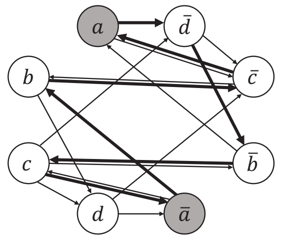

Fig. 3. Literals a and $\bar{a}$ are identified in the same SCC, which causes a conflict in ME of Example 9.

B. Conflict-driven Learning

If the conflict analysis cannot detect the dissatisfaction of an ME, we proceed to solve the ME by searching the input space. The SAT-solving process is to determine a truth value of each variable until every majority function in an ME is satisfied. The SAT-solving process will return SAT if a satisfying assignment of variables is found, or UNSAT after the entire search space is traversed without finding a satisfying assignment.

During the searching, the conflict-driven learning technique can record the reasons of conflicts before traversing the space completely. It helps in avoiding assigning variables leading to a conflict branch repeatedly, and accelerates the solving speed of a solver. The reasons of conflicts are usually recorded in the format of clauses and added to the original problem.

In CNF SAT solvers, the conflict-driven learning procedure is invoked when there is a logic implication derived from the unit propagation [13]. The unit propagation states that the only unassigned literal in a clause will be implied to 1 if all the other literals have been assigned as 0. However, for majority functions, this implication method is quite different from the unit propagation used in CNF SAT solvers. Specifically, in the MajorSat, we have to cope with all the possible implication conditions during the SAT-solving process. In the following property, we propose a new implication method — Majority Propagation, which uses a similar concept of Property 4, to analyze the implications.

Property 5 (Majority Propagation)**:** Given a majority function s of size n with k inputs that have been assigned as 0, any

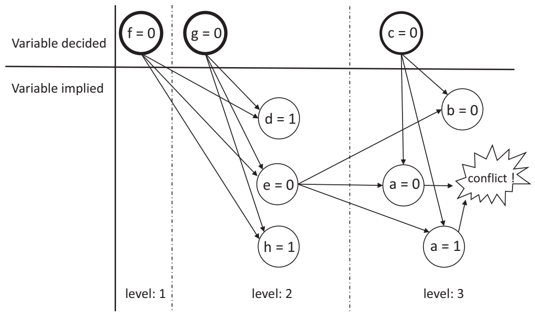

Fig. 4. The implication process of Example 12.

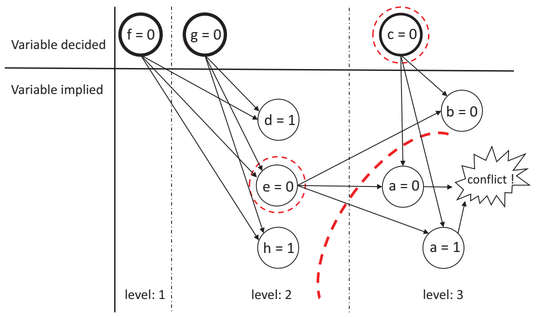

Fig. 5. The learned clause $(e \vee c)$ is generated from the implication process.

unassigned literal in s that occurs equal to or more than $\lceil n/2 \rceil - k$ times will be implied to 1 for satisfying s.

Example 10: In a majority function M(a, a, a, b, c, e, e, f, g) of size 9, if the literals e and g have been assigned as 0, k is equal to 3. The literal a will be implied to 1 since a occurs three times, which is greater than $\lceil 9/2 \rceil - 3 = 2$.

Example 11: In a majority function M(a, a, a, b, b, b, c), if the literal c is assigned as 0, k is equal to 1. The literals a and b are both implied to 1 since they both occur three times, which is equal to $\lceil 7/2 \rceil - 1 = 3$.

For each variable x in an ME, its decision level is recorded for conflict-driven learning. The decision level of a variable x is the level of search tree at which the variable is either decided or implied. The smaller value of decision level means the variable is decided or implied earlier.

We use Example 12 to demonstrate how a conflict happens during the SAT-solving procedure on an ME.

Example 12: Given an ME with three majority functions: M(a, a, a, a, b, c, e, 1, 1) ∧ M($\bar{a}$, $\bar{b}$, c, e, 1) ∧ M(d, $\bar{e}$, f, g, h). Assume that the variables f and g are decided as 0 at the decision level of 1 and 2, respectively. Then $d = 1$, $e = 0$, and $h = 1$ can be implied from the last majority function as shown in the level 2 of Fig. 4. After deciding $c = 0$, we can have $a = 0$, $b = 0$ from the second majority function. Also, we have $a = 1$ from the first majority function. As a result, we find that $a = 1$, and $a = 0$ are both implied from the implication process. Therefore, a conflict is detected. The complete implication process is shown in Fig. 4.

To speed up the SAT-solving process, we can add a learned clause from the conflict to the original ME. In Example 12, since we have known that the assignment of $f = 0$, $g = 0$ and $c = 0$ causes a conflict, we can add a clause $(f \vee g \vee c)$ to the ME intuitively, which means that one of f, g, and c will be assigned as 1 at least in the future for satisfying the original ME. However, the learned clause $(f \vee g \vee c)$ could be further minimized. Next, we use a learning method in [15] to obtain a smaller learned clause.

In the implication process of assigning variables, a node represents a variable which has been decided or implied at a certain decision level. When a conflict happens between two nodes, e.g., $v = 0$ and $v = 1$, we can identify a set s of nodes that are in the fanin cones of both conflict nodes. We then remove the nodes from s which are dominated by the other nodes in s. A node y is dominated by a node x if and only if all paths from node y to the fanout of node x must pass through node x. In Example 12, as shown in Fig. 5, the set s of nodes in the fanin cones of both conflict nodes $a = 0$ and $a = 1$ is $\{f = 0, g = 0, e = 0, c = 0\}$. Since $f = 0$ and $g = 0$ are dominated by $e = 0$, they are removed from s, and s becomes $\{e = 0, c = 0\}$. Then the learned clause is formulated by disjuncting the negation of nodes in s, and it is $(e \vee c)$. The resultant expression becomes M(a, a, a, a, b, c, e, 1, 1) ∧ M($\bar{a}$, $\bar{b}$, c, e, 1) ∧ M(d, $\bar{e}$, f, g, h) ∧ $(e \vee c)$.

Since the learned clause is added to the original ME during the conflict-driven learning process, the expression to be dealt with is not an ME. This scenario extends the format that the proposed MajorSat can solve. That is, the input format of MajorSat is generalized as the conjunction of majority functions and clauses.

C. Variable Decision Order Heuristic

In modern CNF SAT solvers, variable decision heuristics [8][10] significantly influence the efficiency of solving process. The order of variables is computed based on the appearance times of variables among all clauses. A good order of variable decision increases the probability of reaching a satisfiable branch in the searching process. However, for an ME, the scenario that a literal occurs multiple times within a single majority function makes the variable decision order more complex. Thus, for the MajorSat, we also propose a variable decision order method, which is different from that of CNF SAT solvers, to steer the solving direction to a satisfiable branch.

Definition 3: $threshold_m$ and $weight_m(x)$ are defined as $\lceil n/2 \rceil$ and the appearance time of the variable x in a majority function m, respectively, where n is the size of majority function m.

Since the input format of MajorSat is extended to the conjunction of majority functions and clauses, we denote the score function of a variable x in a Majority function m as $scoreM_m(x)$, and that in a Clause c as $scoreC_c(x)$.

Definition 4: Given a majority function m and a variable x, $scoreM_m(x)$ is 0 if x is absent in m; otherwise, $scoreM_m(x)$ is defined as

$$1 - \frac{threshold_m - weight_m(x)}{size\ of\ m}$$

In Definition 4, when the appearance time of a variable in a majority function m is more, the higher value $scoreM_m(x)$ will be. When the $scoreM_m(x)$ is higher, the probability that the decision of x leading to the satisfaction of m is higher.

Definition 5: Given a clause c, and a variable x, $scoreC_c(x)$ is 1 if x is in c; otherwise, $scoreC_c(x)$ is 0.

Definition 5 implies that satisfying a variable in a clause can lead to the satisfaction of the clause.

The score for a variable x, denoted as $score(x)$, is used to steer the branching direction in the searching process. It is computed by the summation of score functions of the majority functions and the clauses as follows:

$$score(x) = \sum_{m \in M} scoreM_m(x) + \sum_{c \in C} scoreC_c(x)$$

where M and C denote majority functions and clauses. The $score(x)$ value reflects the influence of assigning the variable x to satisfy the majority functions and clauses. This $score(x)$ value increases when the appearance time of the variable x in the majority functions increases or the variable x exists in the clauses. Therefore, we determine the variable decision order by the value of $score(x)$ in a descending order.

Example 13: Given an expression F containing majority functions and clauses: M(a, a, a, b, b, $\bar{c}$, d) $\wedge$ M($\bar{b}$, $\bar{c}$, $\bar{d}$) $\wedge$ ($\bar{a} \vee b \vee c$), the score calculation of each variable in F is shown as follows:

$score(a) = (1 - (4-3)/7) + 0 + 1 = 13/7 = 39/21$
$score(b) = (1 - (4-2)/7) + (1 - (2-1)/3) + 1 = 50/21$
$score(c) = (1 - (4-1)/7) + (1 - (2-1)/3) + 1 = 47/21$
$score(d) = (1 - (4-1)/7) + (1 - (2-1)/3) + 0 = 26/21$

since $score(b) > score(c) > score(a) > score(d)$, the variable decision order is determined as $b > c > a > d$.

The scores of some variables will be updated when the conflict happens. That is, after the conflict happens, the scores of variables that appear on the paths from conflict nodes to decision nodes are added by 1. Then, the variable decision order is recomputed based on the newly updated scores.

Example 14: In Example 12, after assigning $f = 0$, $g = 0$, and $c = 0$, the conflict happens in the variable a. According to the implication process in Fig. 4, the paths from conflict nodes, (a = 0, a = 1), to decision nodes, (c = 0, f = 0, g = 0) include nodes of a = 0, a = 1, c = 0, e = 0, f = 0 and g = 0. Therefore, $score(a)$, $score(c)$, $score(e)$, $score(f)$, and $score(g)$ are added by 1.

The reason of this score updating is as follows: When a conflict happens, increasing the scores of variables that appear on the paths from the conflict nodes to the decision nodes can increase the possibility of these variables to be chosen next. Choosing these variables could likely lead the search to unsatisfiable branches, which helps in learning more conflict clauses.

IV. Majority Gate Transformation

In circuit verifications, CNF is widely used to generates the characteristic function of a circuit. Given a Boolean network, its CNF is achieved by applying Tseitin transformation [12]. However, if there is a Boolean network composed of many majority gates, it would be time-consuming to compute the CNF of that network through Tseitin transformation due to a large number of resultant AND, OR gates as shown in Example 2. Therefore, we propose the Majority transformation that generates the characteristic function of a majority gate. As a result, the satisfiability of a Boolean network containing majority gates can be obtained by first encoding the network into an ME and then solving the ME by the MajorSat solver. Property 6 states this transformation rule.

Property 6 (Majority Transformation)**:** Given a majority gate f = M(x_1, x_2, ..., x_{n-1}, x_n) of size n, its characteristic function can be expressed as M(x_1, x_2, ..., x_{n-1}, x_n, $\bar{f}^n$, 1^n) $\wedge$ M($\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_{n-1}$, $\bar{x}_n$, f^n, 1^n), where v^n means that the variable v appears n times in the majority function.

Proof: For a majority gate f = M(x_1, x_2, ..., x_{n-1}, x_n), its characteristic function is ($f \rightarrow$ M(x_1, x_2, ..., x_{n-1}, x_n)) $\wedge$ ($\bar{f} \rightarrow \overline{\text{M}}$($x_1$, x_2, ..., x_{n-1}, x_n)) = ($f \rightarrow$ M(x_1, x_2, ..., x_{n-1}, x_n)) $\wedge$ ($\bar{f} \rightarrow$ M($\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_{n-1}$, $\bar{x}_n$)). By the rule of inference, this formula can be rewritten as ($\bar{f} \vee$ M(x_1, x_2, ..., x_{n-1}, x_n)) $\wedge$ ($f \vee$ M($\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_{n-1}$, $\bar{x}_n$)). For the first clause ($\bar{f} \vee$ M(x_1, x_2, ..., x_{n-1}, x_n)), we can transform it to an equivalent form M(x_1, x_2, ..., x_{n-1}, x_n, $\bar{f}^n$, 1^n). This is because when $\bar{f} = 1$, both forms are 1s; when more than half of x_1, x_2, ..., x_n are 1s, both forms are 1s; otherwise, both forms are 0s. Similarly, the second clause ($f \vee$ M($\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_{n-1}$, $\bar{x}_n$)) can be transformed into a majority function M($\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_{n-1}$, $\bar{x}_n$, f^n, 1^n) by using the same method. Therefore, the characteristic function of the gate f = M(x_1, x_2, ..., x_{n-1}, x_n) can be expressed as M(x_1, x_2, ..., x_{n-1}, x_n, $\bar{f}^n$, 1^n) $\wedge$ M($\bar{x}_1$, $\bar{x}_2$, ..., $\bar{x}_{n-1}$, $\bar{x}_n$, f^n, 1^n).

Example 15: Given a majority gate m: f = M(a, b, c), the characteristic function of m is M(a, b, c, $\bar{f}$, $\bar{f}$, $\bar{f}$, 1, 1, 1) $\wedge$ M($\bar{a}$, $\bar{b}$, $\bar{c}$, f, f, f, 1, 1, 1).

V. Experimental Results

We implemented the proposed MajorSat in C++ language. We conducted two experiments on an Intel Xeon® E5530 2.40GHz CentOS 4.6 platform with 64GB memory. The first one shows the correctness of our solver through the CNF benchmarks of random 3-SAT in SATLIB [19]. In the second experiment, we demonstrate the solving ability of MajorSat on different scales of randomly generated benchmarks in ME format. We randomly generated these benchmarks for the experiment due to no existing benchmark in the format of ME. Furthermore, these benchmarks will be solved by MiniSat [16] and Lingeling [18] after been converted into CNF for comparing the efficiency among MajorSat and these CNF solvers.

Table I summarizes the experimental results of the first experiment. Columns 1-3 list the information about the CNF benchmarks including name, the number of variables, and the

TABLE I
THE EXPERIMENTAL RESULTS ON SATLIB BENCHMARKS.

				Solving Result	
Benchmarks	|variable|	|clause|	Golden Result	CNF	ME
uf20-91	20	91	SAT	SAT	SAT
uf50-218	50	218	SAT	SAT	SAT
uf75-325	75	325	SAT	SAT	SAT
uf100-430	100	430	SAT	SAT	SAT
uf125-538	125	538	SAT	SAT	SAT
uf150-645	150	645	SAT	SAT	SAT
uuf50-218	50	218	UNSAT	UNSAT	UNSAT
uuf75-325	75	325	UNSAT	UNSAT	UNSAT
uuf100-430	100	430	UNSAT	UNSAT	UNSAT
uuf125-538	125	538	UNSAT	UNSAT	UNSAT
uuf150-645	150	645	UNSAT	UNSAT	UNSAT

number of clauses. Column 4 lists the satisfiability of the benchmark. Columns 5-6 list the result of MajorSat in the two formats: CNF, or ME. The CNF benchmarks were converted to ME by converting each clause into a majority function as follows: A clause of size n is converted to a majority function by adding $n - 1$ constant 1s in it. For example, $(a \vee b \vee c)$ was converted to M(a, b, c, 1, 1). The experimental results show that the results of MajorSat in these two formats match the golden results of the benchmarks.

In the second experiment, the scale of randomly generated benchmarks is expressed as (the number of variables_the number of majority functions_the size of each majority function). We generated benchmarks (75_75_17∼29), (100_100_11∼21), and (125_125_11∼15) for the MajorSat. Then we converted them into CNF for MiniSat and Lingeling solvers.

Table II summarizes the experimental results of the second experiment. Column 1 lists the benchmarks. Column 2 lists the solving time of MajorSat measured in second. Column 3 lists the time for conversion from MEs to CNFs. Columns 4-5 list the time for MiniSat in solving CNFs and the total time measured in second. Columns 6-7 list the corresponding results for Lingeling.

According to Table II, we can see that the conversion from an ME to the corresponding CNF is time-consuming. For the benchmarks 75_75_21∼29, this conversion time even exceeds the solving time of MajorSat. The total time required in MajorSat is less than that in MiniSat and Lingeling for all the benchmarks, which demonstrates a solver targeting at majority functions is important.

In summary, MajorSat provides efficient solving performance for majority logic. An exponential growth in the number of clauses when converting an ME to its corresponding CNF increases the solving difficulty for CNF solvers.

VI. CONCLUSION

In this paper, we propose a new SAT solver for solving majority logic. Several properties about majority functions are also investigated to increase the efficiency of MajorSat. The experimental results show that MajorSat is more efficient in solving majority expressions than CNF solvers.

TABLE II
THE COMPARISON OF EXPERIMENTAL RESULTS OF MAJORSAT, MINISAT, AND LINGELING ON RANDOMLY GENERATED MES AND THE CORRESPONDING CNFS.

	MajorSat		MiniSat		Lingeling	
Benchmarks	$t_{sol}(s)$	$t_{conv}(s)$	$t_{sol}(s)$	$total(s)$	$t_{sol}(s)$	$total(s)$
75_75_17	2.37	1.37	> 1000	> 1000	> 1000	> 1000
75_75_19	9.51	5.53	> 1000	> 1000	> 1000	> 1000
75_75_21	12.38	22.80	> 1000	> 1000	> 1000	> 1000
75_75_23	20.16	97.58	> 1000	> 1000	> 1000	> 1000
75_75_25	42.37	410.07	> 1000	> 1000	> 1000	> 1000
75_75_27	118.14	> 1000	–	> 1000	–	> 1000
75_75_29	158.01	> 1000	–	> 1000	–	> 1000
100_100_11	0.15	0.04	3.55	3.59	10.70	10.74
100_100_13	2.81	0.14	475.10	475.24	404.40	404.54
100_100_15	12.94	0.43	> 1000	> 1000	> 1000	> 1000
100_100_17	59.59	2.10	> 1000	> 1000	> 1000	> 1000
100_100_19	140.05	8.10	> 1000	> 1000	> 1000	> 1000
100_100_21	894.18	30.36	> 1000	> 1000	> 1000	> 1000
125_125_11	2.88	0.04	895.80	895.84	237.60	237.64
125_125_13	11.08	0.17	> 1000	> 1000	> 1000	> 1000
125_125_15	152.87	0.59	> 1000	> 1000	> 1000	> 1000

REFERENCES

[1] L. Amarú, P.-E. Gaillardon, and G. De. Micheli, “Majority Logic Representation and Satisfiability,” *in Proc. International Workshop on Logic & Synthesis*, 2014.

[2] L. Amarú, P.-E. Gaillardon, and G. De. Micheli, “Majority-Inverter Graph: A Novel Data-Structure and Algorithms for Efficient Logic Optimization,” *in Proc. Design Automation Conference*, pp. 1-6, 2014.

[3] G. Audemard and L. Simon, “Predicting Learnt Clauses Quality in Modern SAT Solvers,” *in Proc. International Joint Conference on Artificial Intelligence*, pp. 399-404, 2009.

[4] A. Biere, “Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013,” *SAT Competition*, 2013.

[5] M. Davis and H. Putnam, “A Computing Procedure for Quantification Theory,” *Journal of the ACM*, 7:201-215, July 1960.

[6] N. Een and N. Sorensson, “An Extensible SAT-Solver,” *in Proc. International Conference on Theory and Applications of Satisfiability Testing*, pp 502-518, 2003.

[7] N. Een and N. Sorensson, “MiniSat v1.13 — A SAT Solver with Conflict-Clause Minimization,” *SAT Competition*, 2005.

[8] E. Goldberg and Y. Novikov. BerkMin, “A Fast and Robust SAT Solver,” *in Proc. Design, Automation and Test in Europe*, pp. 142-149, 2002.

[9] J. P. Marques-Silva and K. A. Sakallah, “GRASP: A Search Algorithm for Propositional Satisfiability,” *IEEE Transaction on Computers*, vol. 48, pp. 506-521, 1999.

[10] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Engineering an Efficient SAT Solver,” *in Proc. Design Automation Conference*, pp. 530-535, June 2001.

[11] R. E. Tarjan, “Depth-first search and linear graph algorithms,” *SIAM Journal on Computing*, pp. 146-160, 1972.

[12] G. Tseitin, “On the complexity of derivation in propositional calculus,” *Studies in Constr. Math. and Math. Logic*, 1968.

[13] R. Zabih and D. A. McAllester, “A Rearrangement Search Strategy for Determining Propositional Satisfiability,” *in Proc. National Conference on Artificial Intelligence*, pp. 155-160, 1988.

[14] R. Zhang, K. Walus, W. Wang, and G.A. Jullien, “A Method of Majority Logic Reduction for Quantum Cellular Automata,” *IEEE Transaction on Nanotechnology*, vol. 3, no. 4, pp. 443-450, Dec. 2004.

[15] L. Zhang, C.F Madigan, M.H Moskewicz, and S. Malik, “Efficient Conflict Driven Learning in a Boolean Satisfiability Solver,” *in Proc. International Conference on Computer Aided Design*, pp. 279-285, 2001.

[16] http://minisat.se/

[17] http://www.labri.fr/perso/lsimon/glucose/

[18] http://fmv.jku.at/lingeling/

[19] http://www.cs.ubc.ca/∼hoos/SATLIB/benchm.html