Good Learning and Implicit Model Enumeration

A. Morgado and J. Marques-Silva
IST/INESC-ID, Technical University of Lisbon, Portugal
{ajrm,jpms}@sat.inesc-id.pt

Abstract

A large number of practical applications rely on effective algorithms for propositional model enumeration and counting. Examples include knowledge compilation, model checking and hybrid solvers. Besides practical applications, the problem of counting propositional models is of key relevancy in computational complexity. In recent years a number of algorithms have been proposed for propositional model enumeration. This paper surveys algorithms for model enumeration, and proposes optimizations to existing algorithms, namely through the learning and simplification of goods*. Moreover, the paper also addresses open topics in model counting related with* good *learning. Experimental results indicate that the proposed techniques are effective for model enumeration.*

1. Introduction

The enumeration of propositional models has a number of significant practical applications, including knowledge compilation, model checking and hybrid solvers (see [13] for a detailed survey and list of references). In recent years a number of algorithms have been proposed, for model counting [3, 2, 1, 16, 4], but also for model enumeration [12, 9, 4, 15, 8]. Interestingly, the algorithms for model counting and model enumeration have been proposed in different contexts, and so utilize fairly different techniques. For model counting, existing techniques include connected components [2], conflict clause learning [2, 16] and component caching [1, 16]. In contrast, algorithms for model enumeration (most often in the context of model checking) have mostly utilized techniques based on clause learning, namely conflict clause (or *nogood*) learning and *blocking clause* learning. Blocking clauses can be viewed as an instantiation of *good learning*, a technique first suggested (but not implemented or detailed) by Bayardo and Pehoushek in [2]. It is also interesting to observe that model enumeration algorithms are often allowed to produce repeated models, whereas model counting algorithms must necessarily avoid repeated models.

This paper reviews algorithms for model counting and for model enumeration, addresses limitations of current model enumeration algorithms, and proposes new algorithms for model enumeration. The new algorithms improve the identification of *blocking clauses*. In addition, the paper shows how to utilize *blocking clauses* (or *goods*) in model counting algorithms.

The paper is organized as follows. The next section introduces basic notation and concepts. Afterwards, Section 3 addresses clause learning, and its utilizations in model enumeration and counting algorithms. Section 4 proposes techniques for simplifying satisfying assignments, with direct application in model enumeration and counting. Section 5 presents experimental results and Section 6 concludes the paper.

2. Preliminaries

A Conjunctive Normal Form (CNF) formula φ is defined over a finite set of variables $X = \{x_1, x_2, \ldots, x_n\}$. A CNF formula φ consists of a conjunction of clauses, where each clause ω is a disjunction of literals, and where each literal l is either a variable x_i or its complement $\neg x_i$. Where appropriate a CNF formula φ can be viewed as a set of clauses and each clause ω can be viewed as a set of literals.

An assignment A is a function from X to $\{0, u, 1\}$, where u represents an *unspecified* (or don't care) value, with $0 \leq u \leq 1$. An assignment A is said to be *complete* if $\forall_{x_i \in X}\ A(x_i) \in \{0, 1\}$; otherwise it is *partial*. Where appropriate an assignment A can be viewed as a set of pairs (x_i, v_i), $A = \{(x_1, v_1), \ldots, (x_n, v_n)\}$, where $v_i = A(x_i) \in \{0, u, 1\}$ denotes the value assigned to $x_i \in X$. In general, $A = A_U \cup A_S$, where A_U denotes the unspecified variable assignments, of the form (x_i, u), and A_S denotes the specified variable assignments, of the form (x_i, v_i), $v_i \in \{0, 1\}$. Moreover, $A_S = A_D \cup A_I$, where A_D denotes the variable assignments declared as decision assignments by the SAT solver, and A_I denotes the variable assignments implied by unit propagation.

Given an assignment A, the value of a CNF formula φ,

$\varphi|_A$ is defined as follows. The value of a literal $l|_A$ is given by $A(x_i)$ if $l = x_i$ and is given by $1 - A(x_i)$ if $l = \neg x_i$. The value of a clause ω, $\omega|_A$, is given by $\max_{l \in \omega} l|_A$. A clause $\omega \in \varphi$ is said to be *satisfied* if $\omega|_A = 1$ and *unsatisfied* if $\omega|_A = 0$; otherwise it is said to be *undecided*. The value of a CNF formula φ, $\varphi|_A$, is given by $\min_{\omega \in \varphi} \omega|_A$. A formula φ is said to be *satisfied* if $\varphi|_A = 1$ and *unsatisfied* if if $\varphi|_A = 0$; otherwise it is said to be *undecided*. An assignment A such that $\varphi|_A = 1$ is said to be a *satisfying assignment*.

An assignment A' is said to be *covered* by an assignment A if for every $x_i \in X$, with $A(x_i) \in \{0, 1\}$, we have $A'(x_i) \in \{0, 1\} \land A(x_i) = A'(x_i)$. Hence, A' is covered by A provided A' contains at least the specified assignments of A. Two assignments A and A' are said to *intersect* if and only if $\forall_{x_i \in X} A(x_i) = A'(x_i) \lor A(x_i) = u \lor A'(x_i) = u$. Finally, a *model* of a CNF formula φ is a complete assignment A such that $\varphi|_A = 1$.

2.1. Boolean Satisfiability Solvers

The evolution of Boolean Satisfiability (SAT) solvers over the last decade [11, 14] has motivated the widespread use of SAT solvers in practical applications. For real-world instances the most effective SAT solvers are based on backtrack search [5] and share a number of key techniques including unit propagation, clause learning, efficient and lazy data structures, adaptive branching heuristics, and search restarts (see [13] for a survey and list of references).

2.2. Model Counting

The most effective model counting algorithms are based on the Davis-Logemann-Loveland (DLL) procedure [5]. Other algorithms have been proposed (see [10] and [13] for surveys), but are in general impractical. The first DLL-based algorithm for model counting was proposed by Birnbaum and Lozinskii [3]. This algorithm corresponds to an implementation of the standard DLL algorithm, but the algorithm is modified to count identified models. Bayardo and Pehoushek [2] improved the algorithm of [3] by exploiting the existence of connected components in CNF formulas and by the utilization of a modern SAT solver. More recently, Bacchus et al [1]. and Sang et al. [16] proposed the utilization of the techniques used in [2], namely connected components and clause learning, as well as component caching.

2.3. Model Enumeration

Besides the work on model counting, a number of authors have addressed model enumeration techniques [12, 9, 15, 8]. Whereas in algorithms for model counting the accuracy of counting is paramount, in algorithms for model enumeration the objective is to identify a small number of partial assignments that cover all models of the original formula, even though some models may be covered by more than one partial assignment. Observe that practical model enumeration algorithms consist of enumerating partial assignments, which *implicitly* represent sets of models, and such that all model are covered by the enumerated partial assignments. Consequently, most recent work on model enumeration has focused on techniques for reducing the size of computed satisfying partial assignments [12, 9, 15, 8] (see [13] for a survey). Observe that most model enumeration algorithms assume a Boolean circuit, and often the enumerated partial assignments are expressed in terms of primary inputs and state variables [12, 9, 15, 8]. Moreover, the process of blocking the enumeration of satisfying partial assignments that intersect A' usually involves creating a new clause. Finally observe that the most effective techniques used in model counting, namely connected components [2] and component caching [16], are *not* readily applicable to model enumeration. The utilization of connected components allows multiplying the number of solutions in different components. For model enumeration, one must actually *enumerate* the products of the models for the different components. The same holds true for component caching. As a result, the remainder of the paper addresses techniques that can be readily used in model enumeration.

3. Clause Learning

As indicated earlier in Section 2, algorithms for model counting and for model enumeration have used clause learning extensively. Algorithms for model counting have solely used clause learning from conflicts, i.e. *conflict clause learning* [2, 1, 16], whereas algorithms for model enumeration have used both clause learning from conflicts and from identified satisfying partial assignments, i.e. *blocking clause learning* [12, 9, 8].

3.1 Conflict and Blocking Clauses

Conflict clause learning has been used in model counting and in model enumeration essentially as it is implemented in modern SAT solvers [11, 14]. Conflict clauses prevent visiting parts of the search space that exhibit conflicting conditions equivalent to parts already visited, and are particularly effective for algorithms that must (implicitly) visit the complete search space.

Blocking clauses correspond to clauses that prevent satisfying assignments from being repeated [12, 9, 8]. In its simplest form a blocking clause consists of the negation of the current search path. Hence, if the search path corresponds to the decision assignments $\{(x_{i_1}, v_{i_1}), \ldots (x_{i_k}, v_{i_k})\}$, then the blocking clause is defined by $(x_{i_1} \neq v_{i_1} \lor \ldots \lor x_{i_k} \neq$

v_{i_k}), where $x_{i_r} \neq v_{i_r}$ corresponds to x_{i_r} if $v_{i_r} = 1$ and to $\neg x_{i_r}$ if $v_{i_r} = 0$. Observe that blocking clause learning corresponds to *good learning* (whose utilization was suggested in [2]).

3.2 Using Blocking Clauses

This section details the conditions for the utilization of blocking clauses in algorithms for model counting and model enumeration. Observe that blocking clauses have not been used in model counting algorithms. In model enumeration algorithms the utilization of blocking clauses has assumed an underlying Boolean circuit [12, 9, 8]. In contrast, our formulation is CNF-based, and can be integrated both in model counting and in model enumeration algorithms.

Let $\varphi_T = \varphi \cup \varphi_C \cup \varphi_B$ denote a CNF formula consisting of the original formula φ, a set of clauses learned from conflicts φ_C and a set of clauses learned from satisfying partial assignments φ_B. When a new satisfying partial assignment is identified, all clauses in φ_T are satisfied. If our objective is to minimize the size of the computed satisfying partial assignment, then we can just consider assignments used for satisfying the clauses in φ. However, if the computed satisfying partial assignment is simplified by considering only φ, then we must develop techniques for preventing duplicate counting of complete assignments from different satisfying partial assignments. For example, if $A_1 = \{x_1 = 1, x_2 = 0, x_3 = u, x_4 = u\}$ is one satisfying partial assignment and $A_2 = \{x_1 = u, x_2 = 0, x_3 = 1, x_4 = u\}$ is another satisfying partial assignment, then both partial assignments contain the complete assignment $A_3 = \{x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0\}$. It is straightforward to conclude that the complete elimination of redundancy from a set of satisfying partial assignments requires exponential time on the number of satisfying partial assignments.

Another, more practical, approach consists of considering decision assignments used for satisfying clauses in $\varphi_T - \varphi_C = \varphi \cup \varphi_B$. Since we require blocking clauses to be satisfied, then each newly computed satisfying partial assignment will *not* cover models included in previously computed satisfying partial assignments. For the example above, if the first satisfying partial assignment is A_1, then the blocking clause learned from A_1 is $\omega_B = (\neg x_1 \vee x_2)$. The existence of ω_B prevents A_2 from being a satisfying partial assignment for $\varphi \cup \varphi_B$. An acceptable satisfying partial assignment is $A_2' = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = u\}$, which does not intersect A_1.

Observe that the condition for selecting partial assignments that satisfy $\varphi \cup \varphi_B$ applies both to model enumeration, where enumerated partial satisfying assignments will not cover models covered by previously enumerated satisfying partial assignments, and to model counting, where the number of assignments associated with each satisfying partial assignment needs not be corrected given previously computed satisfying partial assignments. Hence, the creation of blocking clauses from partial assignments that satisfy $\varphi \cup \varphi_B$ provides a simple mechanism for implementing *good learning* [2]. Indeed, with learning of blocking clauses, *no* new satisfying partial assignment will intersect a previous satisfying partial assignment. Thus, each partial assignment represents a *disjoint* set of models, and models cannot be multiply counted. Moreover, observe that counting models given a set of disjoint partial assignments is straightforward: a partial assignment with m unassigned variables represents 2^m models. Finally, note that blocking clauses derived from partial satisfying assignments for $\varphi \cup \varphi_B$ can be integrated with existing techniques used in model counting algorithms, including connected components [2] and component caching [1, 16].

4. Simplifying Partial Assignments

Given the results of the previous section, we know that when a satisfying partial assignment is identified, we only need to consider the decision assignments necessary for satisfying $\varphi \cup \varphi_B$. In this section we address techniques for reducing the number of decision assignments that need be associated with a computed satisfying partial assignment. As a result we survey techniques recently proposed for the simplification of satisfying partial assignments [12, 15], describe limitations of these techniques, and propose improvements.

4.1. Simple Variable Lifting

Variable lifting denotes a number of techniques used for the elimination of assignments that can be declared redundant [12, 15]. In this section we detail a simple variable lifting procedure, essential for modern SAT solvers. Observe that most modern SAT solvers need to assign *all* variables before declaring the partial assignment to be satisfying. This condition results from the organization of modern SAT solvers, and the utilization of lazy data structures, where it is difficult to guarantee that the state of every clause is always known. Consequently, a simple variable lifting technique consists of eliminating from the satisfying partial assignment A all variable assignments that do *not* satisfy any clause and do *not* imply any other variable assignment. As described in [12, 15] other lifting techniques exist, but require significant computational overhead.

4.2. Set Covering Model

Another approach for simplifying computed satisfying partial assignments consists of formulating and solving a

set covering model. Observe that in practice there are efficient algorithms for computing approximations for the set covering problem [6]. Our model follows the one suggested in [15].

Let A be a satisfying assignment for φ. Our objective is to minimize the number of specified assignments in A, obtaining a new assignment A'. For each $x_i \in X$, such that $A(x_i) \in \{0,1\}$, define a *selector* variable s_i, that denotes whether the assignment on x_i is selected, i.e. $s_i = 1$ if and only if the assignment $A(x_i)$ is included in A'.

Observe that since implied variable assignments are required for satisfying at least one clause (i.e. its *antecedent* [11]), the optimization model can be restricted to the set of decision variable assignments A_D [15]. Let $\Delta(A)$ denote the clauses of φ that are exclusively satisfied by the decision variable assignments of A. For each clause $\omega \in \Delta(A)$, let $\Sigma(A,\omega)$ denote the selector variables associated with the variable assignments that satisfy ω. In addition, let $\Upsilon(A)$ denote the set of selector variables corresponding to the decision variable assignments that are included in the set of 0-valued literals associated with the antecedents of all implied variable assignments. As a result, the conditions that must be satisfied by the selector variables are the following:

$$\Psi_1 \land \bigwedge_{s\in\Upsilon(A)} (s) \triangleq \bigwedge_{\omega\in\Delta(A)} \left(\bigvee_{s\in\Sigma(A,\omega)} s \right) \bigwedge_{s\in\Upsilon(A)} (s) \quad (1)$$

Finally, the objective is to minimize the number of decision variable assignments specified in A'. Let $\Theta(A)$ denote the set of selector variables associated with decision variable assignments. Then the cost function is to minimize $\sum_{s\in\Theta(A)} s$. A simple observation, often applied in practice, is that for every clause ω such that $|\Sigma(A,\omega)| = 1$, then the assignment that satisfies ω must be included in A'.

4.3. Binate Covering Model

Unfortunately, the set covering model described in the previous section does not yield the smallest partial assignment A', which satisfies the CNF formula φ, and which covers the partial assignment A computed by the SAT solver. Consider the CNF formula $\varphi = (x_1 \lor x_4) \land (x_2 \lor x_5) \land (x_3 \lor x_6) \land (x_2 \lor x_3 \lor x_4)$, and the partial assignment $A_1 = \{x_1 = 0, x_2 = 0, x_3 = 0\}$. Clearly, the set covering model proposed in Section 4.2 will not further minimize A_1, corresponding to three specified assignments. However, it is clear from the example that the assignment $A_2 = \{x_2 = 0, x_3 = 0\}$ also satisfies φ, and implies the same variable assignments.

Given the previous example, we first formalize the notion of partial assignment minimization and then propose a binate covering formulation[1] for identifying a minimum size partial assignment given a satisfying partial assignment computed by a SAT solver.

Let A be a satisfying assignment for a CNF formula φ, with $A = A_D \cup A_I$, where A_D denotes the decision variable assignments, and A_I denotes the implied variable assignments. A minimum satisfying assignment given the assignment A, $\mu(A)$, is an assignment that covers A and A_I, such that $\varphi|_{\mu(A)} = 1$, and such that any other assignment A' that covers $\mu(A)$ is such that $\varphi|_{A'} \neq 1$.

Given an assignment A, a minimum satisfying assignment is the least specified assignment, in terms of decision variable assignments, that covers A, but satisfies φ. Observe that our definition requires the set of implied variable assignments A_I to be kept constant.

Let the satisfying partial assignment be A, and let the specified variable assignments be A_S, with $A_S \subseteq A$. We concentrate on implied variable assignments with an antecedent clause having at least one 0-value literal corresponding to a decision variable, and represent these implied variable assignments by A_I^D. For each implied variable assignment $(x_i, v_i) \in A_I^D$, let $\Gamma(A, x_i)$ denote the set of clauses in φ which explain the implied value $x_i = v_i$, and such that each clause has at least one 0-value literal corresponding to a decision variable (i.e. these are clauses for which all literals besides the literal in x_i are assigned value 0, and such that at least one literal corresponds to a decision variable). For each clause $\omega \in \Gamma(A, x_i)$ let $\gamma(A,\omega)$ denote the set of decision literals of clause ω assigned value 0.

As before, with each variable assignment $(x_k, v_k) \in A_S$ we associate a *selector* variable s_k that denotes whether the assignment is to be included in the simplified assignment A'. Given $\gamma(A,\omega)$, with $\omega \in \Gamma(A, x_i)$, let $\Phi(A,\omega)$ denote the selector variables associated with the literals in $\gamma(A,\omega)$.

Since at least one of the clauses in $\Gamma(A, x_i)$ must imply the value assignment on x_i, we can specify a set of constraints the selector variables must satisfy:

$$\Psi_2 \triangleq \bigwedge_{(x_i,v_i)\in A_I^D} \left(\bigvee_{\omega\in\Gamma(A,x_i)} \left(\bigwedge_{s\in\Phi(A,\omega)} (s) \right) \right) \quad (2)$$

Besides these constraints, we need to include the original set covering conditions, associated with satisfying clauses that are satisfied with decision variable assignments, to obtain the complete set of constraints the selector variables must satisfy:

$$\Psi_3 \triangleq \Psi_1 \land \Psi_2 \quad (3)$$

As for the set covering model of the previous section, let $\Theta(A)$ denote the set of selector variables associated with decision variable assignments. Then the cost function is to

[1] The binate covering problem is a variant of pseudo-Boolean optimization where each constraint is a propositional clause [7].

minimize $\sum_{s\in\Theta(A)} s$. Observe that the constraints (3) are not in CNF format. However, it is straightforward to map (3) into CNF format with the inclusion of additional variables. As a result the binate covering problem formulation results.

The main difficulty with the binate covering model is that it cannot be approximated, unless P = NP [6]. In the next section we propose a set covering model, that builds on the binate covering model described above, and for which approximation algorithms can still be used. Finally, and to our best knowledge, this section provides the first example illustrating the inadequacy of the set covering model for finding the minimum satisfying assignment of a CNF formula that covers an initial satisfying partial assignment A.

4.4. A Variant of the Set Covering Model

One of the drawbacks of the model described in Section 4.2 results from the structure of practical instances of SAT. Practical instances of SAT contain a large number of binary clauses, that often account for more than 80% of the total number of clauses. Moreover, binary and ternary clauses can often represent more than 90% of the total number of clauses. The large number of binary clauses motivates that most decision assignments, when they imply variable assignments, will do so most often due to binary clauses. Hence, these decision assignments will be declared essential and reduce the ability of the set covering model of Section 4.2 for simplifying the satisfying partial assignment computed by the SAT solver.

Next we propose a modified set covering model, which addresses the simplification of satisfying partial assignments when a large number of binary clauses exists. The model resembles the binate covering model but, for the rearrangement of antecedents, we solely consider variables that have a binary clause ω as the antecedent, for which the 0-value literal corresponds to a decision variable, and so $|\Phi(A,\omega)| = 1$. As a result, the number of selector variables per clause becomes one, because $|\Phi(A,\omega)| = 1$, and (3) reduces to the following set covering model:

$$\Psi_1 \land \bigwedge_{s\in\Upsilon^{NB}(A)} (s) \bigwedge_{(x_i,v_i)\in A_I^{D,B}} \left(\bigvee_{\omega\in\Gamma(A,x_i), s\in\Phi(A,\omega)} s \right) \tag{4}$$

where $\Upsilon^{NB}(A)$ corresponds to $\Upsilon(A)$, used in the previous sections, but denotes the selector variables associated with decision variable assignments that imply variable assignments through a ternary or larger clause (i.e. the non-binary clauses, since implied assignments due to unit clauses do not depend on decision assignments). Moreover, $A_I^{D,B}$ denotes the implied variable assignments that have a binary clause as the antecedent and such that the 0-value literal corresponds to a decision assignment. Observe that we now have a set covering formulation, which can be approximated in polynomial time.

As an example, let us consider the CNF formula $\varphi = (x_1 \lor x_4) \land (x_2 \lor x_5) \land (x_3 \lor x_4) \land (x_3 \lor x_6)$, and the partial assignment $A_1 = \{x_1 = 0, x_2 = 0, x_3 = 0\}$. In this case, the basic set covering model of Section 4.2 will be unable to reduce A_1. However, the variant of the set covering model (4) yields the constraints $(s_1 \lor s_3) \land (s_2) \land (s_3)$, which can be satisfied with $s_2 = s_3 = 1$, yielding the new satisfying partial assignment $A_2 = \{x_2 = 0, x_3 = 0\}$.

5. Results

This section presents experimental results for three model simplification techniques: variable lifting, the set covering model and the variant of the set covering model. The classes of instances considered, from SATLIB[2] are relatively easy for modern SAT solvers, but challenging enough for model enumeration purposes. Moreover, the model enumeration algorithm has been implemented on top of SAT solver CQuest, a zChaff-like SAT solver [14]. For the results presented in this section, all experiments have been run under Linux RH 9, on a Pentium 1.7 GHz machine, with 1 GByte of RAM. The CPU time limit was set to 600 seconds.

The results are shown in Table 1, for each of the simplification techniques considered, respectively variable lifting **(A)**, the set covering model **(B)**, and the variant of the set covering model **(C)**. In each table, the following figures are shown. `Alg` denotes which algorithm is assumed. `Class` denotes the class of instances considered. `#I` denotes the number of instances in each class. `#C` denotes the total number of partial assignments that need to be enumerated for covering all models, for all instances in the class for which the algorithm terminates in less than the allowed CPU time. `#M` denotes the total number of models for each class of instances, for which the algorithms terminates in less than the allowed CPU time. `R` denotes (`#C/#M` in percentage, representing the average number of partial assignments needed for each identified model. `T` denotes the CPU time for running all instances, for which the algorithms terminates in less than the allowed CPU time. `#A` denotes the number of instances aborted, i.e. for which the algorithm was unable to terminate in the allowed CPU time.

From the results several conclusions can be drawn. First, the utilization of simplification techniques yields in general much smaller numbers of partial assignments when compared with the total number of models. Moreover, the proposed simplification techniques can be very effective, albeit in general requiring additional overhead, which causes a few instances to take more than the allowed CPU time limit. The results also suggest that the variant of the set covering

[2] Available from http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html.

Table 1. Experimental results

Alg	Class	#I	#C	#M	R(%)	T(sec)	#A
(A)	BMS	28	9.2e4	2.4e8	0.04	0.1e3	2
	CBS	84	2.6e5	1.1e7	2.34	0.5e3	1
	RTI	496	1.1e6	5.4e7	2.06	2.6e3	1
	UF	42	2.1e5	2.9e8	0.07	1.5e3	5
(B)	BMS	28	8.1e4	2.4e8	0.03	0.9e3	2
	CBS	84	1.5e5	7.7e6	1.93	1.8e3	4
	RTI	496	6.1e5	3.3e7	1.87	6.6e3	10
	UF	42	3.4e4	5.8e6	0.60	0.9e3	10
(C)	BMS	28	8.1e4	2.4e8	0.03	1.0e3	2
	CBS	84	1.3e5	7.0e6	1.85	1.2e3	5
	RTI	496	6.2e5	3.3e7	1.89	6.9e3	10
	UF	42	3.6e4	5.8e6	0.62	0.9e3	10
(D)	BMS	28	–	2.3e7	–	1.0e3	7
	CBS	84	–	9.4e6	–	3.6e3	1
	RTI	496	–	4.1e7	–	7.5e3	2
	UF	42	–	4.1e6	–	1.4e3	11

model is not effective for the classes of instances considered. This is in part to be expected, since the classes considered correspond to randomly generated problem instances, with little structure, and so particularly difficult for the variant of the set covering model. Besides comparing the three simplification techniques, we also ran relsat [2], which enumerate models explicitly. The results are shown in Table 1 for algorithm **(D)**. As can be observed (and when compared to our simplest model), relsat aborts more instances, and requires more than twice the CPU time for enumerating all models. For some of the instances for which relsat aborts the reason for aborting results from the very large number of models to be explicitly enumerated.

6. Conclusions and Future Work

This paper surveys algorithms for implicit model enumeration based on learning *goods* (or *blocking clauses*), and proposes new improvements to these algorithms based on techniques for simplifying *learned goods*. The objective for simplifying *learned goods* is to effectively reduce the number of partial assignments that must be enumerated for covering all models of a propositional formula. Moreover, the paper illustrates how techniques for model enumeration can be applied in model counting. This entails the utilization of *good learning* in model counting.

In addition, and even though we have focused on model enumeration, most often a requirement in several applications, including model checking and hybrid solvers, all the techniques described in this paper can be applied in model counting algorithms, and integrated with the techniques proposed in recent years [2, 1, 16].

Despite the promising preliminary results, the problem instances studied are still fairly small. This is motivated by the difficulty of enumerating all satisfying partial assignments for real-world problem instances with current SAT solver technology. In the near future, the development of additional simplification techniques, and improvements to SAT solver technology, is expected to allow enumerating models for more complex problem instances.

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and bayesian inference. In *Symp. Found. Comp. Science*, 2003.

[2] R. Bayardo Jr. and J. Pehoushek. Counting models using connected components. In *Proc. National Conference on Artificial Intelligence*, July 2000.

[3] E. Birnbaum and E. Lozinskii. The good old Davis-Putnam procedure helps counting models. *Journal of Artificial Intelligence Research*, 10:457–477, 1999.

[4] A. Darwiche. New advances in compiling CNF to decomposable negational normal form. In *Proc. National Conference on Artificial Intelligence*, 2004.

[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, July 1962.

[6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[7] G. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[8] H. Jin, H. Han, and F. Somenzi. Efficient conflict analysis for finding all satisfying assignments of a Boolean circuit. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, April 2005.

[9] B. Li, M. Hsiao, and S. Sheng. A novel SAT all-solutions solver for efficient preimage computation. In *Proc. Design and Test in Europe Conf.*, March 2004.

[10] E. Lozinskii. Computing propositional models. *Information Processing Letters*, 41:327–332, 1992.

[11] J. P. Marques-Silva and K. A. Sakallah. GRASP: A new search algorithm for satisfiability. In *Int. Conf. Computer-Aided Design*, November 1996.

[12] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *International Conference on Computer-Aided Verification*, July 2002.

[13] A. Morgado and J. P. Marques-Silva. Algorithms for propositional model counting and enumeration. Technical Report RT-004-05-CDIL, INESC-ID, February 2005.

[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering an efficient SAT solver. In *Design Automation Conference*, June 2001.

[15] K. Ravi and F. Somenzi. Minimal satisfying assignments for conjunctive normal form formulae. In *Proc. Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, April 2004.

[16] T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi. Combining component caching and clause learning for effecive model counting. In *Proc. Int. Conf. Theory and Applications of Satisfiability Testing*, May 2004.