Artificial Intelligence Research and Development
I. Sanz et al. (Eds.)

doi:10.3233/FAIA230707

On the Density of States of Boolean Formulas [1]

Carlos ANSÓTEGUI [a] María Luisa BONET [b] Jordi LEVY [c]
[a] *Universitat de Lleida (DIEI, UdL), Lleida, Spain*
[b] *Universitat Politècnica de Catalunya (CS, UPC), Barcelona, Spain*
[c] *Artificial Intelligence Research Institute (IIIA, CSIC), Bellaterra, Spain*
ORCiD ID: Carlos Ansótegui https://orcid.org/0000-0001-7727-2766, María Luisa Bonet https://orcid.org/0000-0003-1646-7177, Jordi Levy https://orcid.org/0000-0001-5883-5746

Abstract. Given a CNF formula with m clauses, and an integer k, where $0 \leq k \leq m$, a *density of states* procedure will count the number of truth assignments that falsify exactly k clauses of the formula. We present the first approach to compute the density of states of Boolean formulas exactly. This is the first non-trivial result on this known hard problem. There are previous approaches for computing approximately the density of states of Boolean Formulas [1,2], where the authors point out they are not aware of any complete solver that is able to compute the exact density of states. The present work is the first step to fill this gap.

The idea is, given a formula φ and a parameter c, construct a formula φ_c such that, the number of models of φ_c is the number of assignments that falsify exactly c clauses of φ. Then, a #SAT solver is used as a black box to count the models of φ_c. This approach can be also used to compute approximately the density of states by using an approximate #SAT solver. Finally, the method can be extended trivially to deal with Weighted Boolean formulas.

Keywords. SAT, #SAT, Model Counting, Density of States

1. Introduction

Given a Boolean formula in CNF with n variables and m clauses, we can calculate the number of truth assignments that falsify exactly k clauses, where k is any number between 0 and m. Therefore, given the set of 2^n possible truth assignments, we can partition this set into $m+1$ parts. Computing the size of all subsets in the partition is the problem known as *density of states*, a term borrowed from statistical physics.

The problem of the density of states is a generalization of *model counting* for Boolean formulas (abbreviated #SAT). A model counting procedure outputs the number of assignments that satisfy a given CNF formula. There are several procedures, that either count the number of models exactly (CDP [3],Relsat [4], Cachet [5], sharpSAT [6] and c2d [7]), or simply approximate the number [8,9,10,11,12]. See [13] for a complete

[1]This research is part of the projects PROOFS (PID2019-109137GB-C21) and PROOFS BEYOND (PID2022-138506NB-C21) funded by the AEI and MCIN.

presentation on model counting (exact or approximate) for SAT. #SAT is a harder problem than assessing satisfiability (SAT), and in fact, it is #P-Complete. As a consequence, the software solving this problem is orders of magnitude slower than a SAT solver. There are a number of applications of model counting in areas like planning or probabilistic reasoning. For instance, performing Bayesian inference can be translated into model counting [14,15,16].

The density of states is an even more ambitious problem. As we said, for any number k (between 0 and the total number of clauses), we want to know how many assignments falsify exactly k clauses. For $k = 0$, a procedure solving the density of states will give us the number of satisfying assignments. On the other hand, a density of states procedure can also be used to find the minimal k such that the number of assignments falsifying k clauses is not zero, hence solving MaxSAT. Therefore, solving the density of states subsumes both #SAT and MaxSAT.

As a generalization of model counting, density of states has applications in areas like planning and probabilistic reasoning. Also, a detailed study of the search space of all truth assignments can provide insight into the development of new algorithms (for SAT or MaxSAT), as well as a better understanding of current approaches. For instance, [17] study local search dynamics using density of states. They observe that independently of the starting point, local search executions eventually stagnate around particular cost (number of falsified clauses) intervals. These intervals correspond to costs where the formula has many assignments falsifying that number of clauses.

There isn't much previous work developed around density of states. In fact, we are the first to propose a way to compute density of states exactly, and the method can be easily adapted to an approximate method. The rest of the literature presents methods to approximate density of states [17,1,2] using different sampling schemes. These estimation methods can be faster than our exact computation method, but many times it is hard to asses the accuracy of the estimates. Therefore our method can also be used to evaluate the performance of the estimates produced by other algorithms.

The key idea behind our work is the following. Given a CNF formula φ and a parameter c, we obtain a formula φ_c such that the number of assignments that satisfy φ_c is the same as the number of assignments that falsify exactly c clauses of φ. After that, we use a model counter on φ_c to calculate (or estimate) the number of satisfying assignments. The method can be extended trivially to deal with weighted Boolean formulas. Therefore to state the method in full generality we will describe it using weighted Boolean formulas.

This method requires encoding cardinality or pseudo-Boolean constraints as CNF formulas, possibly introducing additional variables, with the property that once we instantiate the original variables, the remaining formula has either 0 or 1 models. The well-known Tseitin encoding of a circuit fulfills this property. However, several state-of-the-art encodings, mostly for efficiency issues, do not satisfy it. We revisit state-of-the-art encodings of cardinality constraints and we present them guaranteeing that once we instantiate the original variables, the remaining formula has either 0 or 1 models. For efficiency reasons, our encodings should also have the property of arc-consistency. We are not the first in studying encodings with these two properties. To be able to do model counting for Integer Linear Programming, [18] study encodings with the same properties.

We conduct an experimental investigation on a selection of industrial Partial MaxSAT instances. The #SAT solvers we use are: the exact model counter sharpSAT [6], and the approximate model counters satss [8] and sampleCount [19].

The paper proceeds as follows. After some preliminary definitions in Section 2, we describe a simple method for computing the density of states based on successive calls to a #SAT solver in Section 3. This method requires encoding cardinality and pseudo-Boolean constraints as CNF formulas without introducing redundant models. In Section 4, we explain how this can be done for several state-of-the-art encodings. In Section 5, we present some experimental results on industrial instances used in the MaxSAT evaluation. Finally, in Section 6, we show how to compute a good approximation of the density of states function by interpolation.

2. Preliminaries

We consider an infinite countable set of *Boolean variables* $\mathscr{X}$. A *literal* l is either a variable $x_i \in \mathscr{X}$ or its negation $\overline{x_i}$. A *clause* C is a finite set of literals, denoted as $C = l_1 \vee \cdots \vee l_r$, or as $\square$ for the empty clause. A *(CNF) formula* φ is a finite set of clauses.

A *weighted clause* is a pair (C,w), where C is a clause and w is a natural number or infinity, indicating the penalty for falsifying the clause C. A clause is *hard* if the corresponding weight is infinity, otherwise, the clause is *soft*. A *weighted formula* is a multiset of weighted clauses $\varphi = \{(C_1,w_1),\ldots,(C_m,w_m),(C_{m+1},\infty),\ldots,(C_{m+m'},\infty)\}$ where the first m clauses are soft and the last m' clauses are hard.

The set of variables occurring in a (weighted) formula φ is noted as $\mathrm{var}(\varphi)$.

A *(total) truth assignment* for a formula φ is a function $I : \mathrm{var}(\varphi) \to \{0,1\}$, that can be extended to literals, clauses, and formulas, in the standard way. This definition of total assignment can be extended to partial assignments $I : V \to \{0,1\}$. Then, $I(\varphi)$ is another formula with variables in $\mathrm{var}(\varphi) \setminus V$.

For a weighted formula $\varphi = \{(C_1,w_1),\ldots,(C_m,w_m),(C_{m+1},\infty),\ldots,(C_{m+m'},\infty)\}$ and truth assignment $I : \mathrm{var}(\varphi) \to \{0,1\}$, we define the *cost* as $\mathrm{cost}(\varphi,I) = \sum_{i=1}^{m} w_i\,(1 - I(C_i))$, if I satisfies all hard clauses, i.e. $I(\{C_{m+1},\ldots,C_{m+m'}\}) = 1$. Otherwise, we say that the formula is unsatisfiable, and $\mathrm{cost}(\varphi,I) = \infty$.

We define the *optimal cost* of φ as $\mathrm{cost}(\varphi) = \min\{\mathrm{cost}(\varphi,I) \mid I : \mathrm{var}(\varphi) \to \{0,1\}\}$ and an *optimal assignment* of φ as any assignment I such that $\mathrm{cost}(\varphi) = \mathrm{cost}(\varphi,I) \neq \infty$.

We define *MaxSAT* as the problem of finding a truth assignment that maximizes the number of satisfied clauses of a CNF formula. We define *Weighted Partial MaxSAT* as the problem of finding a truth assignment that satisfies all the hard clauses and minimizes the cost of a weighted formula, i.e. finding an optimal assignment. Similarly, we define *MinSAT* as the problem of finding a truth assignment that minimizes the number of satisfied clauses, and *Weighted Partial MinSAT* as the problem of finding a truth assignment I that satisfies all the hard clauses and maximizes the number of falsified clauses.

A *cardinality constraint* is a restriction of the form $\sum_{i=1}^{m} x_i \leq k$, where $k \in \mathbb{Z}$ and variables x_i may get values in $x_i \in \{0,1\}$. A *pseudo-Boolean constraint* is a restriction of the form $\sum_{i=1}^{m} w_i x_i \leq k$, where $w_i, k \in \mathbb{Z}$ and variables x_i also may get values in $x_i \in \{0,1\}$.

3. Density of States via Model Counting

In this section, we describe our basic approach to exactly compute the density of states of a Boolean formula. For the sake of clarity, we will first describe how to solve the

Weighted Partial MaxSAT problem. The density of states approach will be based on these ideas.

Given a weighted formula $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$, first, we check that all hard clauses are satisfiable. Then, we generate the formula $\varphi_k = \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\} \cup CNF\left(\sum_{i=1}^{m} w_i b_i \leq k\right)$, for any possible value of $k \in \{0..\sum_{i=1}^{m} w_i\}$ and find the smallest value of k such that φ_k is satisfiable. This solves the Weighted Partial MaxSAT problem, using the fact:

$$\text{cost}(\varphi) = \min\{k \mid \varphi_k \text{ is SAT and } \varphi_{k-1} \text{ is UNSAT}\}.$$

The search for such k can be done using a dichotomy search and calling a SAT solver. A more efficient way would be to use modern SAT-based state-of-the-art MaxSAT solvers [20].

Now we are ready to state the more general problem. Counting the number of models with a cost equal to a given k can be defined as computing the following function.

Definition 1 *Given a weighted formula* $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ *and a value* $c \in \mathbb{N} \cup \{\infty\}$ *we define*

$$\text{dens}(\varphi, c) = \left|\{I : \text{var}(\varphi) \to \{0,1\} \mid \text{cost}(\varphi, I) = c\}\right|$$

Computing the density of states consists in obtaining $\text{dens}(\varphi, c)$, for any possible value of $c \in \{0, \ldots, \sum_{i=1}^{m} w_i\}$. In what follows we will describe how we can obtain each value $\text{dens}(\varphi, c)$.

First, we generate the formula

$$\varphi_c = CNF\left(\{C_1 \leftrightarrow \overline{b}_1, \ldots, C_m \leftrightarrow \overline{b}_m, C_{m+1}, \ldots, C_{m+m'}, \sum_{i=1}^{m} w_i b_i = c\}\right) \tag{1}$$

where given a formula ψ, $CNF(\psi)$ is a translation of ψ into Conjunctive Normal Form. Here we need to take into account that such translations usually use additional variables, and they will have to fulfill certain properties.

Second, we use a SAT model counter to obtain the number of models φ_c has. The idea behind these two steps is that:

$$\text{dens}(\varphi, c) = \#SAT(\varphi_c), \qquad \text{for any } c \in \mathbb{N} \tag{2}$$

To justify the correctness of equation (2), we need to see that any interpretation I, such that $I(\varphi) = c$, can be extended to a unique interpretation of the b_i variables, and can also be extended to a unique interpretation of the additional variables of the CNF encoding that satisfies φ_c; also, we need to see that any interpretation that satisfies φ_c, can be restricted to an interpretation of the original variables of φ, say I', such that $I'(\varphi) = c$. The second part of the statement is clearly true, but to ensure the first part, we need a CNF encoding of the pseudo-Boolean constraints that satisfies the *counting safety* property. The notion of counting safe encodings was first defined in [18] in the context of counting models in integer domains. They propose to encode pseudo-Boolean and integer linear programs into SAT, and then to use #SAT solvers, as an effective method for counting models in these domains. Their *counting safety* definition stated that the pseudo-Boolean

constraint and the Boolean encoding had to have the same number of models. Here we strengthen the definition.

Definition 2 *We say that a CNF encoding of a pseudo-Boolean (or cardinality) constraint*[2] $CNF(\sum_{i=1}^{m} w_i x_i \leq k)$ *is* **counting safe** *if, for any assignment* $I : \{x_1, \ldots, x_m\} \to \{0,1\}$*, the formula* $I(CNF(\sum_{i=1}^{m} w_i x_i \leq k))$ *has either* **zero** *models (when* $\sum_{i=1}^{m} w_i I(x_i) \not\leq k$*) or just* **one** *model (when* $\sum_{i=1}^{m} w_i I(x_i) \leq k$*).*

Notice that this property is not trivial since state-of-the-art CNF encodings of pseudo-Boolean constraints usually contain additional auxiliary variables. In order to make encodings more efficient to calculate, the counting safety property is dropped in many of them. Therefore, in many encodings, the property that is preserved is that if under an assignment the constraint is true, the encoding is satisfiable, and vice versa. This is true about the encodings of [21,22,23,22,24,25]. There are a few encodings that preserve counting safety (see [18] for details on two of them).

In order to obtain an efficient encoding we are also interested in other properties like having few new clauses, few additional variables, and also that, simple consequences of a partial assignment applied to a pseudo-Boolean constraint can be obtained by performing unit propagation to the CNF encoding. This last property is called *arc-consistency*. In the case of pseudo-Boolean (and cardinality) constraints, the property can be defined as follows.

Definition 3 *We say that a CNF encoding* φ *of a pseudo-Boolean (or cardinality) constraint* $CNF(\sum_{i=1}^{m} w_i x_i \leq k)$ **preserves arc-consistency** *if, for any partial assignment* $I : V \to \{1\}$*, where* $V \subseteq \{x_1, \ldots, x_n\}$*, and variable instantiation* $I' : \{x_i\} \to 1$*, if* $I(\sum_{i=1}^{m} w_i x_i \leq k)$ *is satisfiable and* $I'(I(\sum_{i=1}^{m} w_i x_i \leq k))$ *is unsatisfiable, then* $\bar{x}_i$ *may be inferred from* $I(\varphi)$ *by unit propagation.*

In section 4, we will talk about encodings of cardinality and pseudo-Boolean constraints that have both properties: safe counting and arc-consistency. One of these encodings will be used in our implementation.

In what follows, we propose various improvements to gain efficiency. In our implementation, we solve the MaxSAT and MinSAT problems for the formula φ. This gives us two values say k_1 and k_2, respectively, such that $\text{dens}(\varphi, c) = 0$ for $c < k_1$ and for $c > k_2$. So, we restrict the computation of the density to $c \in \{k_1, \ldots, k_2\}$.

As we will see, the time needed by an exact model counter on a formula φ_c increases with the number of models. Therefore, we use an approximate model counter when the number of models of φ_c is big.

It is also possible that for a number c, $k_1 \leq c \leq k_2$, that the number of models of φ_c is equal to 0. For instance, if φ is a formula with all even weights, then $\text{dens}(\varphi, c) = 0$ for c odd. Therefore, it is desirable to check the satisfiability of φ_c in that range $k_1 \leq c \leq k_2$, using a state-of-the-art SAT solver, before calling a model counter. This is especially important if we use an approximated model counter, that may have problems when the number of models is very small.

[2]Here we only consider less-equal constraints, with the understanding that the less, greater, greater-equal and equal constraints can easily be expressed in terms of least-equal.

4. Encodings of Cardinality and Pseudo-Boolean Constraints

There are several (standard) ways to encode a cardinality constraint $\sum_{i=1}^{m} x_i \leq k$ or a pseudo-Boolean constraint $\sum_{i=1}^{m} w_i x_i \leq k$ as a CNF formula preserving arc-consistency [21,22,23,24,25]. In [18], it is proved that the encoding in [26], based on binary adders, and the [24] encoding, based on BDDs, both satisfy the counting safety property. The first one, however, does not preserve arc-consistency.

In the next subsections, we review two more encodings for cardinality constraints, one based on sequential counters, and another based on sorting networks. We also analyze how to translate BDDs into CNF formulas in order to ensure the counting safety property.

4.1. Sequential Counters

A simple approach to encoding a sequential counter is to use a variable s_i^j that gets the value true when $x_1 + \ldots + x_i = j$, and a pair of clauses of the form $s_i^j \wedge \bar{x}_{i+1} \rightarrow s_{i+1}^j$ and $s_i^j \wedge x_{i+1} \rightarrow s_{i+1}^{j+1}$ that define the value of s_i^j in terms of the values of variables with smaller indexes. Then, the encoding of the cardinality is given by the following set of clauses.

$$
\begin{array}{ll}
\bar{s}_i^j \vee x_{i+1} \vee s_{i+1}^j & \text{for } j = 0...k,\ i = j...n+j-k-2 \\
\bar{s}_i^j \vee \bar{x}_{i+1} \vee s_{i+1}^{j+1} & \text{for } j = 0...k,\ i = j...n+j-k-1 \\
s_0^0 & \\
\bar{s}_i^{k+1} & \text{for all } i = k+1, \ldots, n
\end{array}
$$

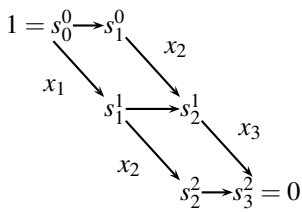
This encoding has the counting safety property, but not arc-consistency.

In order to have arc-consistency, we can implement a monotonic variant of this Boolean function. Sinz [21] proposes the use of variables s_i^j that are true if $x_1 + \ldots + x_i \geq j$, and the following set of clauses.

$$
\begin{array}{ll}
\bar{s}_i^j \vee s_{i+1}^j & \text{for } j = 0...k+1,\ i = j...n+j-k-2 \\
\bar{s}_i^j \vee \bar{x}_{i+1} \vee s_{i+1}^{j+1} & \text{for } j = 0...k,\ i = j...n+j-k-1 \\
s_0^0 & \\
\bar{s}_n^{k+1} &
\end{array}
$$

This formula can be simplified by unit propagation, leading to a formula with $k(n-1)$ auxiliary variables and size $\mathcal{O}(nk)$. The encoding is arc-consistent, but does not have the counting safety property, as the following counter-example shows.

Example 4 For the cardinality constraint $x_1 + x_2 + x_3 \leq 1$, the sequential counter implementation described in [21] generates the following set of clauses, where literals in red may be removed by unit propagation.



$$
\begin{array}{ll}
\bar{s}_0^0 \vee \bar{x}_1 \vee s_1^1 & s_0^0 \\
\bar{s}_1^0 \vee \bar{x}_2 \vee s_2^1 & \bar{s}_0^0 \vee s_1^0 \\
\bar{s}_1^1 \vee \bar{x}_2 \vee s_2^2 & \bar{s}_1^1 \vee s_2^1 \\
\bar{s}_2^1 \vee \bar{x}_3 \vee s_3^2 & \bar{s}_2^2 \vee s_3^2 \\
\bar{s}_3^2 &
\end{array}
$$

For $x_1 = x_2 = x_3 = 0$ the formula has three models. In all them $s_0^0 = s_1^0 = 1$ and $s_2^2 = s_3^2 = 0$, but $\langle s_1^1, s_2^1 \rangle \in \{\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 1,1 \rangle\}$.

The next lemma presents an encoding that preserves arc-consistency and has the counting safety property. Like in [21], the variable s_i^j is true when $x_1 + \ldots + x_i \geq j$.

Lemma 5 *For any cardinality constraint $x_1 + \ldots + x_n \leq k$, the following set of clauses*

$$
\begin{array}{cl}
\bar{s}_i^j \vee s_{i+1}^j & \text{for } j = 0 \ldots k+1 \text{ and } i = j \ldots n+j-k-2 \\
s_i^j \vee x_{i+1} \vee \bar{s}_{i+1}^j & \text{for } j = 0 \ldots k+1 \text{ and } i = j \ldots n+j-k-2 \\
\bar{s}_i^j \vee \bar{x}_{i+1} \vee s_{i+1}^{j+1} & \text{for } j = 0 \ldots k \text{ and } i = j \ldots n+j-k-1 \\
s_i^j \vee \bar{x}_{i+1} \vee \bar{s}_{i+1}^{j+1} & \text{for } j = 0 \ldots k \text{ and } i = j \ldots n+j-k-1 \\
\bar{s}_i^i \vee x_i & \text{for } i = 1 \ldots k \\
s_0^0 & \\
\bar{s}_n^{k+1} &
\end{array}
$$

defines an equivalent formula with both the arc-consistency and the counting safety properties.

PROOF: All clauses are sound, in the sense that, any interpretation of the x_i variables satisfying the cardinality constraint can be extended to an interpretation of the s_i^j variables satisfying the formula.

The formula is an extension of the formula proposed in [21] without additional variables. Therefore, this formula is also arc-consistent because it has at least the same inference power.

Now, we will prove that it has the counting safety property, i.e. that there exists at most one model once we have instantiated the x_i variables. The proof is by induction. The value of s_0^0 is 1 in all models. For the induction step, we prove that the values of $s_{i+1}^0, \ldots, s_{i+1}^{r+1}$, for $r = \min\{i,k\}$, are determined by the values of $s_i^0, \ldots, s_i^r$, for $r = \min\{i, k+1\}$. Notice that the value of s_i^0 is 1, and the value of s_i^{k+1} is 0, in all models. There are two cases. If $x_{i+1} = 0$, then the clauses $\bar{s}_i^j \vee s_{i+1}^j$ and $s_i^j \vee x_{i+1} \vee \bar{s}_{i+1}^j$ ensure that s_i^j and s_{i+1}^j have the same value, for $j = 0, \ldots, \min\{i, k+1\}$. When $i \leq k$, the value of s_{i+1}^{i+1} is determined by the clause $\bar{s}_{i+1}^{i+1} \vee x_{i+1}$ that sets it to 0. If $x_{i+1} = 1$, then the clauses $\bar{s}_i^j \vee \bar{x}_{i+1} \vee s_{i+1}^{j+1}$ and $s_i^j \vee \bar{x}_{i+1} \vee \bar{s}_{i+1}^{j+1}$ ensure that s_{i+1}^{j+1} and s_i^j have the same value, for $j = 0, \ldots, \min\{i,k\}$. Notice that s_{i+1}^0 is always 1 and the value of s_{i+1}^{k+1}, when $i \geq k$, is 0. ■

4.2. Sorting and Cardinality Networks

Sorting networks can be used to encode cardinality constraints [22,23]. A sorting network is a circuit $\langle y_1, \ldots, y_n \rangle = SORT(\langle x_1, \ldots, x_n \rangle)$ that given a sequence, returns a sorted permutation of the input such that $y_i \geq y_j$, for all $1 \leq i < j \leq n$. A merging network is a circuit $\langle z_1, \ldots, z_{2n} \rangle = MERGE(\langle x_1, \ldots, x_n \rangle, \langle y_1, \ldots, y_n \rangle)$ that given two sorted sequences, returns a sorted permutation of the concatenation of both sequences.

We can implement a sorting network using the recurrence

$$SORT(\langle x_1,\ldots,x_{2n}\rangle) = MERGE(SORT(\langle x_1,\ldots,x_n\rangle), SORT(\langle x_{n+1},\ldots,x_{2n}\rangle))$$

We can also implement a merge-network

$$\langle z_1,\ldots,z_{4n}\rangle = MERGE(\langle x_1,\ldots,x_{2n}\rangle, \langle y_1,\ldots,y_{2n}\rangle)$$

using two merge networks (one for the even position and the other one for the odd positions)

$$\langle t_1,\ldots,t_{2n}\rangle = MERGE(\langle x_1,x_3,\ldots,x_{2n-1}\rangle, \langle y_1,y_3,\ldots,y_{2n-1}\rangle)$$
$$\langle t'_1,\ldots,t'_{2n}\rangle = MERGE(\langle x_2,x_4,\ldots,x_{2n}\rangle, \langle y_2,y_4,\ldots,y_{2n}\rangle)$$

and adding the following gates

$$z_1 = t_1$$
$$z_3 = AND(t_2,t'_1) \ldots z_{4n-1} = AND(t_{2n},t'_{2n-1})$$
$$z_2 = OR(t_2,t'_1) \quad \ldots z_{4n-2} = OR(t_{2n},t'_{2n-1})$$
$$z_{4n} = t'_{2n}$$

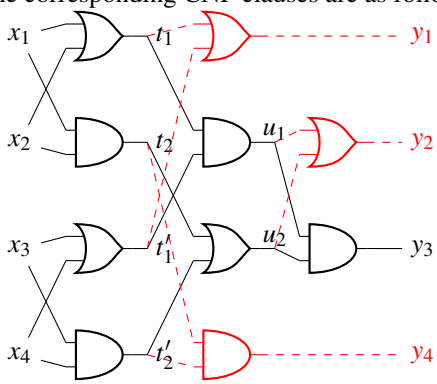
These sort and merge circuits have size $\mathcal{O}(n\log^2 n)$ and $\mathcal{O}(n\log n)$, respectively.

We can implement a circuit that checks $x_1+\ldots+x_n \le k$, implementing $\langle y_1,\ldots,y_n\rangle = SORT(\langle x_1,\ldots,x_n\rangle)$ and checking if $y_{k+1} = 0$. Then, by encoding gates as clauses, we get a CNF encoding of the cardinality constraint of size $\mathcal{O}(n\log^2 n)$. This encoding also preserves arc-consistency [22].

Notice that we only check one of the outputs of the circuit. Therefore, we can remove all gates that are only needed to compute the value of the other outputs y_i for $i \neq k+1$. By iteratively removing these useless gates, we obtain an encoding of size $\mathcal{O}(n\log^2 k)$ [23]. This encoding is (asymptotically) better than the encoding based on sequential counters (where the size was $\mathcal{O}(nk)$). However, this is not significant when k is small.

Gates are encoded as clauses that only ensure that the output is bigger or equal to the circuit output. More precisely, gates $y = AND(x_1,x_2)$ are encoded as $x_1 \wedge x_2 \to y$, and gates $y = OR(x_1,x_2)$ as $\{x_1 \to y,\ x_2 \to y\}$. This makes the resulting formula not have the counting safety property.

Example 6 Consider the cardinality constraint $x_1+x_2+x_3+x_4 \le 2$. The circuit encoding the cardinality constraint is drawn below, where gates in red are unused gates (removed in [23]), and the corresponding CNF clauses are as follows.



$x_1 \to t_1$
$x_2 \to t_1$
$x_1 \wedge x_2 \to t_2$
$x_3 \to t'_1$
$x_4 \to t'_1$
$x_3 \wedge x_4 \to t'_2$
$t_1 \wedge t'_1 \to u_1$
$t_2 \to u_2$
$t'_2 \to u_2$
$u_1 \wedge u_2 \to y_3$
$\bar{y}_3$

For $x_1 = 1$ and $x_2 = x_3 = x_4 = 0$ (satisfying the constraint), we have six models:

t_1	t_2	t'_1	t'_2	u_1	u_2	y_3
1	0	0	0	0	0	0
1	0	0	0	0	1	0
1	0	0	0	1	0	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0

The problem in the previous example is that the Boolean function implemented with the clauses is not the function implemented in the circuit, but a *monotone* variation. Notice that when $x_1 = x_2 = 1$, both the gate $y = AND(x_1, x_2)$ and the clause $x_1 \wedge x_2 \rightarrow y$ force $y = 1$. However, when $x_1 = 0$ and $x_2 = 1$, the gate $y = AND(x_1, x_2)$ forces $y = 0$, but the clause $x_1 \wedge x_2 \rightarrow y$ allows y to be 0 or 1. We can construct a counting safety formula by adding clauses ensuring that the gates and the clauses have the same behavior.

Lemma 7 *If we encode sorting networks, adding gates* $y = AND(x_1, x_2)$ *the clauses* $\{x_1 \wedge x_2 \rightarrow y,\ y \rightarrow x_1,\ y \rightarrow x_2\}$, *and for gates* $y = OR(x_1, x_2)$ *the clauses* $\{x_1 \rightarrow y,\ x_2 \rightarrow y,\ y \rightarrow x_1 \vee x_2\}$, *then the resulting formula has the counting safety property and the encoding preserves arc-consistency.*

PROOF: The formula is an extension of the formula proposed in [22,23], hence arc-consistent. The additional clauses force the formula to behave exactly like the circuit, where all intermediate points have a determined value. ■

4.3. Reduced Ordered BDDs

Reduced Ordered BDDs are a canonical representation of Boolean functions [27], hence of pseudo-Boolean constraints. These BDDs may also be translated into arc-consistent CNF formulas [22]. Although ROBBDs may be exponentially large [24,25], we have obtained good (practical) results using them in our MaxSAT solvers [28,29,20,30].

A BDD may be encoded in CNF [22] by introducing an auxiliary variable y_i for every decision variable x_i. Then, if the false and true child of x_i are respectively x_j and x_k, we add the following clauses:

$$\bar{x}_i \wedge \bar{y}_j \rightarrow \bar{y}_i \qquad x_i \wedge \bar{y}_k \rightarrow \bar{y}_i \qquad \bar{y}_j \wedge \bar{y}_k \rightarrow \bar{y}_i$$
$$\bar{x}_i \wedge y_j \rightarrow y_i \qquad x_i \wedge y_k \rightarrow y_i \qquad y_j \wedge y_k \rightarrow y_i$$

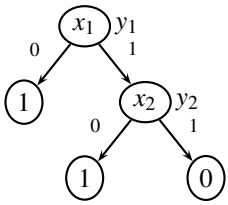
It is trivial to prove that this CNF encoding has the counting safety property, because auxiliary variables in leaves are always true or false, and the values of other auxiliary variables y_i are determined by the values of their child's auxiliary variables y_j and y_k (once we have instantiated the decision variable x_i).

In [25] it is proposed an alternative translation from monotonic BDDs to CNF that only uses two clauses per node and that is also arc-consistent. In this case, we also have an auxiliary variable per node, and the two clauses are as follows:

$$\overline{y}_j \rightarrow \overline{y}_i$$
$$x_i \wedge \overline{y}_k \rightarrow \overline{y}_i$$

In this case, the resulting formula does not have the counting safety property, as the following example shows.

Example 8

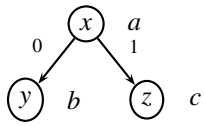


The monotonic BDD on the left, resulting from translating the pseudo-Boolean constraint $2x_1 + x_2 \leq 2$, is translated as the following CNF by [25]

$$\varphi = \{y_1,\ x_1 \wedge \overline{y}_2 \rightarrow \overline{y}_1,\ x_2 \rightarrow \overline{y}_2\}$$

For the interpretation $I(\langle x_1, x_2\rangle) = \langle 0,0\rangle$ the formula $I(\varphi)$ has two models $\langle y_1, y_2\rangle \in \{\langle 1,0\rangle,\ \langle 1,1\rangle\}$.

Lemma 9 *If we translate a pseudo-Boolean constraint into a BDD, and then, for every BDD node of the following form we add the clauses on the right,*



$$\overline{b} \rightarrow \overline{a}$$
$$x \wedge \overline{c} \rightarrow \overline{a}$$
$$\overline{x} \wedge b \rightarrow a$$
$$c \rightarrow a$$

then the resulting formula is arc-consistent and has the counting safety property.

PROOF: The formula is an extension of the one proposed in [25], where we add two new clauses for each node, hence arc-consistent. To prove that the formula has the counting safety property, assume that the decision variables in the BDD are all assigned. We will prove by induction that all auxiliary variables get a unique determined value. By induction hypothesis we have that values of decision variables (the x in the picture) and values of auxiliary variables in the sons (b and c) are determined. Now, we will prove that they determine the value of the auxiliary variable in the father (a). To prove this, notice that by construction the BDD is monotonic and $c \rightarrow b$ is always satisfied, thus the case $c = 1$ and $b = 0$ does not need to be considered. For the rest of cases, the following table gives the unique value for a allowed by the four clauses:

x	b	c	a
0	0	0	0
0	1	0	1
0	1	1	1
1	0	0	0
1	1	0	0
1	1	1	1

■

5. Experimental Results

We have conducted an experimental investigation to explore the density of states of industrial formulas and test our method. We have run our experiments on machines with 2.27GHz CPU and 1G RAM.

We have selected 10 instances from the Partial MaxSAT industrial category in the MaxSAT evaluation 2011. We have also added two pigeonhole formulas PHP_9^{10} and PHP_5^{10} where "each pigeon has to go to one hole" is a soft clause, and "two pigeons cannot go to the same hole" is a hard clause.

As we have said, our approach is parametric on any #SAT solver. Here, we used the exact #SAT solver sharpSAT [6], and the approximate #SAT solvers satss [8] and sampleCount [19]. We will refer to the resulting solvers as DOS(sharpSAT), DOS(satss) and DOS(sampleCount). We encode cardinality constraints as cardinality networks as it is described in Subsection 4.2. Initially, we compute the optimal cost of the MaxSAT (k_1) and MinSAT problems (k_2) as described in Section 3. We will only show the results for the interval $c \in [k_1, k_2]$, since for the rest of the values of c, the number of truth assignments violating c clauses is 0. In the figures, we show the number of truth assignments that violate exactly $c \in [k_1, k_2]$ soft clauses in the Partial MaxSAT formula. We do not consider models falsifying the hard clauses.

The timeout for the DOS(sharpSAT) solver is a global limit for the whole instance. We use 20 hours. Therefore, we compute the points following the sequence $c = k_1, k_2, k_1+1, k_2-1, k_1+2, \ldots$. When the cutoff is applied, we obtain an incomplete DOS where middle points (the costliest to compute) are missing. Approximate #SAT solvers may run forever, trying to improve the accuracy of the solution. In this case, we have to provide a timeout for each value c. We use 30 minutes.

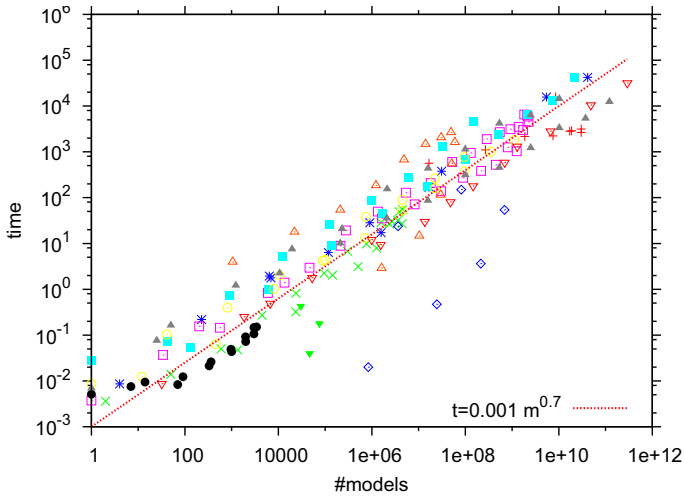


Figure 1. Time needed by sharpSAT on φ_c as a function of the number of models of φ_c for distinct industrial MaxSAT formulas φ (each family of formulas in a distinct color).

We analyze the running time for DOS(sharpSAT) on each data point. We can see that time is almost proportional to the number of assignments (see Figure 1). In fact, it seems to follow a relation $t = 0.001m^{0.7}$, where t is the time in seconds and m the number of

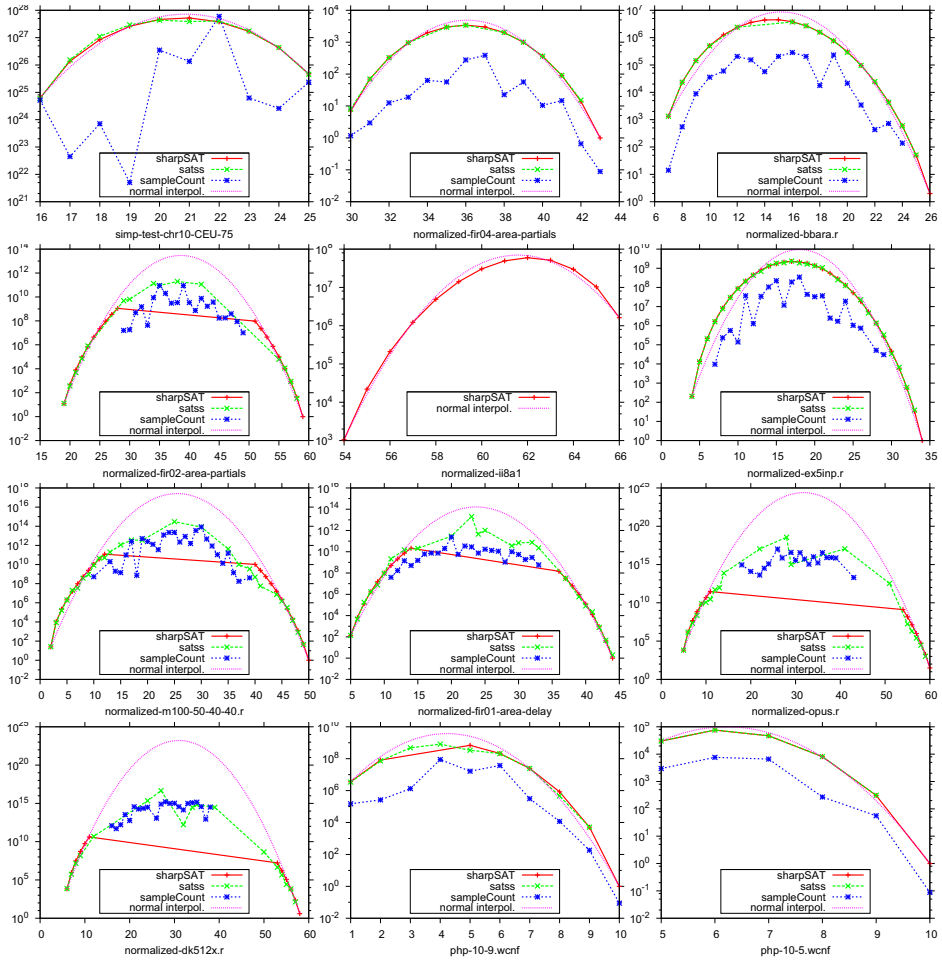


Figure 2. Density of states, $\log(\mathrm{dens}(\varphi, c))$ as a function of c, computed by DOS(sharpSAT), DOS(satss) and DOS(sampleCount) on some industrial Partial MaxSAT instances and two pigeonhole formulas. We also include a (truncated) normal distribution interpolation (parabolic interpolation in this logarithmic representation) computed from the values of $\mathrm{dens}(\varphi, k_1)$, $\mathrm{dens}(\varphi, k_2)$ and $\sum_{c=k_1}^{k_2} \mathrm{dens}(\varphi, c)$.

models. Therefore, by predicting the c's with maximum number of assignments we can also produce the hardest instances for #SAT solvers. Notice that #SAT solvers are tested on SAT instances, while with our approach we can extend the benchmarks to contain DOS queries on Partial MaxSAT instances.

Figure 2 shows the density of states $\mathrm{dens}(\varphi, c)$ computed by DOS(sharpSAT), DOS(satss) and DOS(sampleCount). As we can see, the number of truth assignments that violate c clauses follows approximately a (truncated) normal distribution, since representing the results in logarithmic scale we obtain an inverted parabolic representation. In general, approximate #SAT solvers compute approximate values smaller than the real values. Results of DOS(satss) seem more accurate than results of DOS(sampleCount). However, we have to take into account that these solvers are parametrizable. By adjusting parameters differently, we may obtain better results.

6. Approximating DOS by Interpolation

SAT-based MaxSAT solvers may be (easily) modified in order to produce a SAT formula whose models are the optimal assignments. Running a #SAT solver on this formula, we compute $\text{dens}(\varphi, k_1)$. In a similar way, we can easily compute $\text{dens}(\varphi, k_2)$. In a formula (without hard clauses), the number of possible assignments is 2^n, being n the number of variables. However, in a weighted formula, this number is smaller because we do not consider assignments violating hard clauses. We can easily compute the number of assignments satisfying hard clauses as $\sum_{c=k_1}^{k_2} \text{dens}(\varphi, c) = \#SAT(\varphi_{hard})$. These three values represent two points and the area below $\text{dens}(\varphi, c)$. They are enough to interpolate a parabola in the representation of $\log(\text{dens}(\varphi, c))$, i.e. to approximate a (truncate) normal distribution for $\text{dens}(\varphi, c)$. We represent this interpolation in Figure 2. As we can see, in our examples, and probably in most industrial MaxSAT instances, this interpolation is better than the results obtained by DOS(satss) and DOS(sampleCount). Moreover, it can be efficiently computed with one call to a #SAT solver to get $\#SAT(\varphi_{hard})$, one call to a MaxSAT solver to get k_1, and then a call to a #SAT solver to get $\text{dens}(\varphi, k_1)$, one call to a MinSAT solver to get k_2, and then a call to a #SAT solver to get $\text{dens}(\varphi, k_2)$.

7. Conclusions

In this pioneering work, we present the first method to compute *exactly* the density of states of a MaxSAT formula. The idea is to reduce the problem to a sequence of computations of #SAT. The use of an approximate #SAT solver results into an approximate DOS solver.

While there exists previous approximate DOS solvers [1,2] for MaxSAT, this work describes the first *exact* DOS solver. Moreover, it is also the first implementation able to manage *Partial* and *Weighted* MaxSAT formulas.

Finally, our work produces sets of test instances for #SAT solvers from over-constrained instances. Given a formula φ, the set of instances φ_c for $c \in \{k_1, \ldots, k_2\}$ (k_1 and k_2 are the MaxSAT and MinSAT values respectively), produce a set of benchmarks of different hardness for a model counter.

References

[1] Ermon S, Gomes CP, Selman B. Computing the Density of States of Boolean Formulas. In: CP'10. vol. 6308 of LNCS; 2010. p. 38-52.

[2] Ermon S, Gomes CP, Selman B. A Flat Histogram Method for Computing the Density of States of Combinatorial Problems. In: IJCAI'11; 2011. p. 2608-13.

[3] Birnbaum E, Lozinskii EL. The Good Old Davis-Putnam Procedure Helps Counting Models. Journal of Artificial Intelligence Research. 1999;10:457-77.

[4] Bayardo RJ, Pehoushek JD. Counting Models Using Connected Components. In: AAAI'00; 2000. p. 157-62.

[5] Sang T, Bacchus F, Beame P, Kautz HA, Pitassi T. Combining Component Caching and Clause Learning for Effective Model Counting. In: SAT'04; 2004. p. 20-8.

[6] Thurley M. sharpSAT - Counting Models with Advanced Component Caching and Implicit BCP. In: SAT'06. vol. 4121 of LNCS; 2006. p. 424-9.

[7] Darwiche A. New Advances in Compiling CNF into Decomposable Negation Normal Form. In: ECAI'04; 2004. p. 328-32.

[8] Gogate V, Dechter R. Approximate Counting by Sampling the Backtrack-free Search Space. In: AAAI'07; 2007. p. 198-203.
[9] Gomes CP, Sabharwal A, Selman B. Model Counting: A New Strategy for Obtaining Good Bounds. In: AAAI'06; 2006. p. 54-61.
[10] Kroc L, Sabharwal A, Selman B. Leveraging belief propagation, backtrack search, and statistics for model counting. Annals of Operations Research. 2011;184(1).
[11] Wei W, Erenrich J, Selman B. Towards Efficient Sampling: Exploiting Random Walk Strategies. In: AAAI'04; 2004. p. 670-6.
[12] Wei W, Selman B. A New Approach to Model Counting. In: SAT'05; 2005. p. 324-39.
[13] Gomes CP, Sabharwal A, Selman B. 20. In: Model Counting. vol. 185 of Frontiers in Artificial Intelligence and Applications. IOS Press; 2009. p. 633-54.
[14] Bacchus F, Dalmao S, Pitassi T. Algorithms and Complexity Results for #SAT and Bayesian Inference. In: FOCS'03; 2003. p. 340-51.
[15] Sang T, Beame P, Kautz HA. Performing Bayesian Inference by Weighted Model Counting. In: AAAI'05; 2005. p. 475-82.
[16] Darwiche A. Modeling and Reasoning with Bayesian Networks. Cambridge University Press; 2009.
[17] Belaidouni M, Hao JK. SAT, Local Search Dynamics and Density of States. In: Artificial Evolution; 2001. p. 192-204.
[18] Morgado A, Matos PJ, Manquinho VM, Marques-Silva JP. Counting Models in Integer Domains. In: SAT'06; 2006. p. 410-23.
[19] Gomes CP, Hoffmann J, Sabharwal A, Selman B. From Sampling to Model Counting. In: IJCAI'07; 2007. p. 2293-9.
[20] Ansótegui C, Bonet ML, Levy J. SAT-based MaxSAT algorithms. Artificial Intelligence. 2013;196:77-105.
[21] Sinz C. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: CP'05. vol. 3709 of LNCS; 2005. p. 827-31.
[22] Eén N, Sörensson N. Translating Pseudo-Boolean Constraints into SAT. JSAT. 2006;2(1-4):1-26.
[23] Asín R, Nieuwenhuis R, Oliveras A, Rodríguez-Carbonell E. Cardinality Networks: a theoretical and empirical study. Constraints. 2011;16(2):195-221.
[24] Bailleux O, Boufkhad Y, Roussel O. A Translation of Pseudo Boolean Constraints to SAT. JSAT. 2006;2(1-4):191-200.
[25] Abío I, Nieuwenhuis R, Oliveras A, Rodríguez-Carbonell E. BDDs for Pseudo-Boolean Constraints - Revisited. In: SAT'11; 2011. p. 61-75.
[26] Warners JP. A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. Information Processing Letters. 1998;68(2):63-9.
[27] Bryant RE. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans Computers. 1986;35(8):677-91.
[28] Ansótegui C, Bonet ML, Levy J. Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In: SAT'09; 2009. p. 427-40.
[29] Ansótegui C, Bonet ML, Levy J. A New Algorithm for Weighted Partial MaxSAT. In: AAAI'10; 2010. p. 3-8.
[30] Ansótegui C, Gabàs J, Levy J. Exploiting subproblem optimization in SAT-based MaxSAT algorithms. J Heuristics. 2016;22(1):1-53.