TPLP **24** (4): 586–605, 2025.

doi:10.1017/S147106842400036X

On Lower Bounding Minimal Model Count

MOHIMENUL KABIR
School of Computing, National University of Singapore, Singapore
(*e-mail:* mahibuet045@gmail.com)

KULDEEP S MEEL
Department of Computer Science, University of Toronto, Canada
(*e-mail:* meel@cs.toronto.edu)

submitted 18 August 2024; accepted 13 September 2024

Abstract

Minimal models of a Boolean formula play a pivotal role in various reasoning tasks. While previous research has primarily focused on qualitative analysis over minimal models; our study concentrates on the quantitative aspect, specifically counting of minimal models. Exact counting of minimal models is strictly harder than #P, prompting our investigation into establishing a lower bound for their quantity, which is often useful in related applications. In this paper, we introduce two novel techniques for counting minimal models, leveraging the expressive power of answer set programming: the first technique employs methods from knowledge compilation, while the second one draws on recent advancements in hashing-based approximate model counting. Through empirical evaluations, we demonstrate that our methods significantly improve the lower bound estimates of the number of minimal models, surpassing the performance of existing minimal model reasoning systems in terms of runtime.

KEYWORDS: minimal model, propositional circumscription, model counting, ASP

1 Introduction

Given a propositional formula F, a model $\sigma \models F$ is *minimal* if $\forall \sigma' \subset \sigma$, it holds that $\sigma' \not\models F$ (Angiulli *et al.* 2014). Minimal model reasoning is fundamental to several tasks in artificial intelligence, including circumscription (McCarthy 1980; Lifschitz 1985), default logic (Reiter 1980), diagnosis (De Kleer *et al.* 1992), and deductive databases under the generalized closed-world assumption (Minker 1982). Although not new, minimal model reasoning has been the subject of several studies (Eiter and Gottlob 1993; Ben-Eliyahu and Dechter 1996; Kirousis and Kolaitis 2003; Ben-Eliyahu 2005), covering tasks such as *minimal model finding* (finding a single minimal model), *minimal model checking* (deciding whether a model is minimal), and *minimal model entailment and membership* (deciding whether a literal belongs to all minimal models or some minimal models, respectively) (Ben-Eliyahu and Palopoli 1997).

Complexity analysis has established that minimal model reasoning is intractable, tractable only for specific subclasses of CNF (Conjunctive Normal Form) formulas. Typically, finding one minimal model for positive CNF formulas[1] is in $\mathsf{P}^{\mathsf{NP}[\mathcal{O}(\log n)]}$-hard (Cadoli 1992b). Additionally, checking whether a model is minimal is co-NP-complete (Cadoli 1992a), whereas queries related to entailment and membership are positioned at the second level of the polynomial hierarchy (Eiter and Gottlob 1993).

This study delves into a nuanced reasoning task on minimal models, extending beyond the simplistic binary version of decision-based queries. Our focus shifts towards quantitative reasoning with respect to minimal models. Specifically, we aim to count the number of minimal models for a given propositional formula. While enumerating a single minimal model is insufficient in many applications, counting the number of minimal models provides a useful metric for related measures (Hunter *et al.* 2008; Thimm 2016). Apart from specific structures of Boolean formulas, exact minimal model counting is #co-NP-complete (Kirousis and Kolaitis 2003), established through *subtractive reductions*.

Although minimal models can theoretically be counted by iteratively employing minimal model finding oracles, this approach is practical only for a relatively small number of minimal models and becomes impractical as their number increases. Advanced model counting techniques have scaled to a vast number of models through sophisticated *knowledge compilation* methods (Darwiche 2004; Thurley 2006), which involve transforming an input formula into a specific representation that enables efficient model counting based on the size of this new representation. However, applying knowledge compilation to minimal model counting presents unique challenges, which is elaborated in Section 4. Beyond knowledge compilation, approximate model counting has emerged as a successful strategy for estimating the number of models with probabilistic guarantees (Chakraborty *et al.* 2013). In particular, the *hashing-based* technique, which partitions the search space into *smaller, roughly equal* partitions using randomly generated XOR constraints (Gomes *et al.* 2021), has attracted significant attention. The model count can be estimated by enumerating the models within one randomly partition (Chakraborty *et al.* 2013).

Our empirical study reveals that both approaches to minimal model counting face scalability issues in practical scenarios. Furthermore, knowing a lower bound of the model count is still useful in many applications and is often computed in the model counting literature (Gomes *et al.* 2007). Some applications require the enumeration of all minimal models (Jannach *et al.* 2016; Bozzano *et al.* 2022), but complete enumeration becomes infeasible for a large number of minimal models. Here, the lower bound of the number of minimal models provides a useful criterion for assessing the feasibility of enumerating all minimal models. Knowing this lower bound of the model count is often beneficial to estimate the size of the search space, which enables more specific targeting within the search space (Fichte *et al.* 2022). Consequently, our research shifts focus from counting all minimal models to determining a lower bound for their number.

The primary contribution of this paper is the development of methods to estimate a lower bound for the number of minimal models of a given propositional formula. This is achieved by integrating knowledge compilation and hashing-based techniques with

[1] A CNF formula is positive if each clause has at least one positive literal. Every positive CNF formula has at least one minimal model.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

minimal model reasoning, thus facilitating the estimation of lower bounds. At the core, the proposed methods conceptualize minimal models of a formula as *answer sets* of an ASP program; Answer Set Programming (ASP) is a declarative programing paradigm for knowledge representation and reasoning (Marek and Truszczyński 1999). Additionally, our proposed methods depend on the efficiency of well-engineered ASP systems. Our approach utilizing knowledge compilation effectively counts the number of minimal models or provides a lower bound. Besides, our hashing-based method offers a lower bound with a probabilistic guarantee. We apply our minimal model counting method to the domain of itemset mining, showcasing its utility. The effectiveness of our proposed methods has been empirically validated on datasets from model counting competitions and itemset mining. To assess the performance of our proposed methods, we introduce a new metric that considers both the quality of the lower bound and the computational time; our methods achieve the best score compared to existing minimal model reasoning systems.

The paper is organized as follows: Section 2 presents the background knowledge necessary to understand the main contributions of the paper; Section 4 outlines our proposed techniques for estimating the lower bound on the number of minimal models; Section 5 demonstrates the experimental evaluation of our proposed techniques; and Section 6 concludes our work with some indications of future research directions.

2 Preliminaries

Before going to the technical description, we present some background about propositional satisfiability, answer set programing, itemset mining from data mining, and a relationship between minimal models and minimal generations in transaction records.

2.1 Propositional satisfiability

In propositional satisfiability, we define the domain $\{0,1\}$, which is equivalently $\{\mathsf{false},\mathsf{true}\}$ and a *propositional variable* or *atom* v takes a value from the domain. A *literal* ℓ is either a variable v (positive literal) or its negation $\neg v$ (negative literal). A *clause* C is a *disjunction* of literals, denoted as $C = \bigvee_i \ell_i$. A Boolean formula F, in *Conjunctive Normal Form (CNF)*, is a *conjunction* of clauses, represented as $F = \bigwedge_j C_j$. We use the notation $\mathsf{Var}(F)$ to denote the set of variables within F.

An assignment τ over X is a function $\tau : X \to \{0,1\}$, where $X \subseteq \mathsf{Var}(F)$. For an atom $v \in X$, we define $\tau(\neg v) = 1 - \tau(v)$. The assignment τ over $\mathsf{Var}(F)$ is a *model* of F if τ evaluates F to be true. Given $X \subseteq \mathsf{Var}(F)$ and an assignment τ, we use the notation $\tau_{\downarrow X}$ to denote the *projection* of τ onto variable set $X \subseteq \mathsf{Var}(F)$. Given a CNF formula F (as a set of clauses) and an assignment $\tau : X \to \{0,1\}$, where $X \subseteq \mathsf{Var}(F)$, the *unit propagation* of τ on F, denoted $F|_\tau$, is recursively defined as follows:

$$F|_\tau = \begin{cases} 1 & \text{if } F \equiv 1 \\ F'|_\tau & \text{if } \exists C \in F \text{ s.t. } F' = F \setminus \{C\}, \ell \in C \text{ and } \tau(\ell) = 1 \\ F'|_\tau \cup \{C'\} & \text{if } \exists C \in F \text{ s.t. } F' = F \setminus \{C\}, \ell \in C, C' = C \setminus \{\ell\} \\ & \quad \text{and } (\tau(\ell) = 0 \text{ or } \{\neg\ell\} \in F) \end{cases}$$

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

We often consider an assignment τ as a set of literals it assigns and $\mathsf{Var}(\tau)$ denotes the set of variables assigned by τ. For two assignments τ_1 and τ_2, τ_1 satisfies τ_2, denoted as $\tau_1 \models \tau_2$, if $\tau_{1\downarrow \mathsf{Var}(\tau_2)} = \tau_2$. Otherwise, τ_1 does not satisfy τ_2, denoted as $\tau_1 \not\models \tau_2$.

An XOR constraint over $\mathsf{Var}(F)$ is a Boolean "XOR" ($\oplus$) applied to the variables $\mathsf{Var}(F)$. A random XOR constraint over variables $\{x_1, \ldots, x_k\}$ is expressed as $a_1 \cdot x_1 \oplus \ldots a_k \cdot x_k \oplus b$, where all a_i and b follow the *Bernoulli* distribution with a probability of 1/2. An XOR constraint $x_{i_1} \oplus \ldots x_{i_k} \oplus 1$ (or $x_{i_1} \oplus \ldots x_{i_k} \oplus 0$ resp.) is evaluated as true if an even (or odd resp.) number of variables from $\{x_{i_1}, \ldots, x_{i_k}\}$ are assigned to true.

To define minimal models of a propositional formula F, we introduce an *ordering operator* over models. For two given models τ_1 and τ_2, τ_1 is considered *smaller* than τ_2, denoted as $\tau_1 \leq \tau_2$, if and only if for each $x \in \mathsf{Var}(F)$, $\tau_1(x) \leq \tau_2(x)$. We define τ_1 as *strictly smaller* than τ_2, denoted as $\tau_1 < \tau_2$, if $\tau_1 \leq \tau_2$ and $\tau_1 \neq \tau_2$. A model τ is a *minimal model* of F if and only if τ is a model of F and no model of F is strictly smaller than τ. We use the notation $\mathsf{MinModels}(F)$ to denote minimal models of F and for a set $X \subseteq \mathsf{Var}(F)$, $\mathsf{MinModels}(F)_{\downarrow X}$ denotes the minimal models of F projected onto the variable set X. The minimal model counting problem seeks to determine the cardinality of $\mathsf{MinModels}(F)$, denoted $|\mathsf{MinModels}(F)|$.

In this paper, we sometimes represent minimal models by listing the variables assigned as true. For example, suppose $\mathsf{Var}(F) = \{a, b, c\}$ and under minimal model $\tau = \{a, b\}$, $\tau(a) = \tau(b) = \mathsf{true}$ and $\tau(c) = \mathsf{false}$. The notation $\neg\tau$ denotes the negation of assignment τ; in fact, $\neg\tau$ is a clause or disjunction of literals (e.g., when $\tau = \{a, b\}$, $\neg\tau = \neg a \vee \neg b$). Throughout the paper, we use the notations τ and σ to denote an arbitrary assignment and a minimal model of F, respectively. For each model $\sigma \in \mathsf{MinModels}(F)$, each of the variables assigned to true is *justified*; more specifically, for every literal $\ell \in \sigma$, there exists a clause $c \in F$ such that $\sigma \setminus \{\ell\} \not\models c$. Otherwise, $\sigma \setminus \{\ell\}$ (smaller than σ) is a model of F.

2.2 Answer set programming

An *answer set program* P consists of a set of rules, each rule is structured as follows:

$$\text{Rule } r\text{: } a_1 \vee \ldots a_k \leftarrow b_1, \ldots, b_m, \mathsf{not}\ c_1, \ldots, \mathsf{not}\ c_n \tag{1}$$

where, $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are propositional variables or atoms, and k, m, n are non-negative integers. The notations $\mathsf{Rules}(P)$ and $\mathsf{atoms}(P)$ denote the rules and atoms within the program P. In rule r, the operator "not" denotes *default negation* (Clark 1978). For each rule r (Eq. (1)), we adopt the following notations: the atom set $\{a_1, \ldots, a_k\}$ constitutes the *head* of r, denoted by $\mathsf{Head}(r)$, the set $\{b_1, \ldots, b_m\}$ is referred to as the *positive body atoms* of r, denoted by $\mathsf{Body}(r)^+$, and the set $\{c_1, \ldots, c_n\}$ is referred to as the *negative body atoms* of r, denoted by $\mathsf{Body}(r)^-$. A rule r called a *constraint* when $\mathsf{Head}(r) = \emptyset$. A program P is called a *disjunctive logic program* if $\exists r \in \mathsf{Rules}(P)$ such that $|\mathsf{Head}(r)| \geq 2$ (Ben-Eliyahu and Dechter 1994).

In ASP, an interpretation M over $\mathsf{atoms}(P)$ specifies which atoms are assigned true; that is, an atom a is true under M if and only if $a \in M$ (or false when $a \notin M$ resp.). An interpretation M satisfies a rule r, denoted by $M \models r$, if and only if $(\mathsf{Head}(r) \cup \mathsf{Body}(r)^-) \cap M \neq \emptyset$ or $\mathsf{Body}(r)^+ \setminus M \neq \emptyset$. An interpretation M is a *model* of P, denoted by $M \models P$, when $\forall_{r \in \mathsf{Rules}(P)} M \models r$. The *Gelfond-Lifschitz reduct* of a program

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

P, with respect to an interpretation M, is defined as $P^M = \{\mathsf{Head}(r) \leftarrow \mathsf{Body}(r)^+ | r \in \mathsf{Rules}(P), \mathsf{Body}(r)^- \cap M = \emptyset\}$ (Gelfond and Lifschitz 1991). An interpretation M is an *answer set* of P if $M \models P$ and no $M' \subset M$ exists such that $M' \models P^M$. We denote the answer sets of program P using the notation $\mathsf{AS}(P)$.

2.3 From minimal models to answer sets

Consider a Boolean formula, $F = \bigwedge_i C_i$, where each clause is of the form: $C_i = \ell_0 \vee \ldots \ell_k \vee \neg\ell_{k+1} \vee \ldots \neg\ell_m$. We can transform each clause C_i into a rule r of the form: $\ell_0 \vee \ldots \vee \ell_k \leftarrow \ell_{k+1}, \ldots, \ell_m$. Given a formula F, let us denote this transformation by the notation $\mathcal{DLP}(F)$. Each minimal model of F corresponds uniquely to an answer set of $\mathcal{DLP}(F)$.

2.4 Approximate lower bound

We denote the probability of an event e using the notation $\mathsf{Pr}[e]$. For a Boolean formula F, let c represents a lower bound estimate for the number of minimal models of F. We assert that c is a lower bound for the number of minimal models with a *confidence* δ, when $\mathsf{Pr}[c \leq |\mathsf{MinModels}(F)|] \geq 1 - \delta$.

2.5 Minimal generator in Itemset mining

We define transactions over a finite set of items, denoted by $\mathcal{I}$. A *transaction* t_i is an ordered pair of (i, I_i), where i is the unique identifier of the transaction and $I_i \subseteq \mathcal{I}$ represents the set of items involved in the transaction. A transaction database is a collection of transactions, where each uniquely identified by the identifier i, corresponding to the transaction t_i. A transaction (i, I_i) supports an itemset $J \subseteq \mathcal{I}$ if $J \subseteq I_i$. The *cover* of an itemset J within a database D, denoted as $C(J, D)$, is defined as: $\mathcal{C}(J, D) = \{i | (i, I_i) \in D \text{ and } J \subseteq I_i\}$. Given an itemset I and transaction database D, the itemset I is a *minimal generator* of D if, for every itemset J where $J \subset I$, it holds that $\mathcal{C}(I, D) \subset \mathcal{C}(J, D)$.

2.6 Encoding minimal generators as minimal models

Given a transaction database D, we encode a Boolean formula $\mathsf{MG}(D)$ such that minimal models of $\mathsf{MG}(D)$ correspond one-to-one with the minimal generators of D. This encoding introduces two types of variables: (i) for each item $a \in \mathcal{I}$, we introduce a variable p_a to denote that a is present in a minimal generator (ii) for each transaction t_i, we introduce a variable q_i to denote the presence of the itemset in the transaction t_i. Given a transaction database $D = \{t_i | i = 1, \ldots n\}$, consisting of the union of transactions t_i, consider the following Boolean formula:

$$\mathsf{MG}(D) = \bigwedge_{i=1}^{n} (\neg q_i \rightarrow \bigvee_{a \in \mathcal{I} \setminus I_i} p_a) \tag{2}$$

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Lemma 1.
Given a transaction database D, σ is a minimal model of $\mathsf{MG}(D)$ if and only if the corresponding itemset $I_\sigma = \{a | p_a \in \sigma\}$ is a minimal generator of D.

The encoding of $\mathsf{MG}(D)$ bears similarities to the encoding detailed in (Jabbour *et al.* 2017; Salhi 2019). However, our encoding achieves compactness by incorporating a one-sided implication, which enhances the efficiency of the representation.

3 Related Works

Given its significance in numerous reasoning tasks, minimal model reasoning has garnered considerable attention from the scientific community.

Minimal models of a Boolean formula F can be computed using an iterative approach with a SAT solver (Li *et al.* 2021). The fundamental principle is as follows: for any model $\alpha \in \mathsf{MinModels}(F)$, no model of F can exist that is strictly smaller than α; thus, $F \wedge \neg\alpha$ yields no model. Conversely, if $F \wedge \neg\alpha$ returns a model, it is strictly smaller than α.

Minimal models can be efficiently determined using *unsatisfiable core-based* MaxSAT algorithms (Alviano 2017). This technique leverages the unsatisfiable core analysis commonly used in MaxSAT solvers and operates within an incremental solver to enumerate minimal models sorted by their size. In parallel, another line of research focuses on the enumeration of minimal models by applying cardinality constraints to calculate models of bounded size (Liffiton and Sakallah 2008; Faber *et al.* 2016). Notably, Faber *et al.* 2016 employed an algorithm that utilized an external solver for the enumeration of cardinality-minimal models of a given formula. Upon finding a minimal model, a blocking clause is integrated into the input formula, ensuring that these models are not revisited by the external solver.

There exists a close relationship between minimal models of propositional formula and answer sets of ASP program (Ben-Eliyahu and Dechter 1994). Beyond solving disjunctive logic programs (ref. Section 2), minimal models can also be effectively computed using specialized techniques within the context of ASP, such as domain heuristics (Gebser *et al.* 2013) and preference relations (Brewka *et al.* 2015).

Due to the intractability of minimal model finding, research has branched into exploring specific subclasses of positive CNF formulas where minimal models can be efficiently identified within polynomial time (Ben-Eliyahu and Dechter 1996; Angiulli *et al.* 2014). Notably, a Horn formula possesses a singular minimal model, which can be derived in linear time using unit propagation (Ben-Eliyahu and Dechter 1996). Ben-Eliyahu and Palopoli 1997 developed an *elimination algorithm* designed to find and verify minimal models for *head-cycle-free* formulas. Angiulli *et al.* 2022 introduced the *Generalized Elimination Algorithm* (GEA), capable of identifying minimal models across any positive formula when paired with a suitably chosen eliminating operator. The efficiency of the GEA hinges on the complexity of the specific eliminating operator used. With an appropriate eliminating operator, the GEA can determine minimal models of head-elementary-set-free CNF formulas in polynomial time. Notably, this category is a broader superclass of the head-cycle-free subclass.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Graph-theoretic properties have been effectively utilized in the reasoning about minimal models. Specifically, Angiulli *et al.* 2022 demonstrated that minimal models of positive CNF formulas can be decomposed based on the structure of their dependency graph. Furthermore, they introduced an algorithm that leverages model decomposition, utilizing the underlying dependency graph to facilitate the discovery of minimal models. This approach underscores the utility of graph-theoretic concepts in enhancing the efficiency and understanding of minimal model reasoning.

To the best of our knowledge, the literature on minimal model counting is relatively sparse. The complexity of counting minimal models for specific structures of Boolean formulas, such as Horn, dual Horn, bijunctive, and affine, has been established as #P (Durand and Hermann 2008). This complexity is notably lower than the general case complexity, which is #co-NP-complete (Kirousis and Kolaitis 2003).

4 Estimating the Number of Minimal Models

In this section, we introduce methods for determining a lower bound for the number of minimal models of a Boolean formula. We detail two specific approaches aimed at estimating this number. The first method is based on the *decomposition* of the input formula, whereas the second method utilizes a hashing-based approach of approximate model counting.

4.1 Formula decomposition and minimal model counting

Considering a Boolean formula $F = F_1 \wedge F_2$, we define the components F_1 and F_2 as *disjoint* if no variable of F is mentioned by both components F_1 and F_2 (i.e., $\mathsf{Var}(F_1) \cap \mathsf{Var}(F_2) = \emptyset$). Under this condition, the models of F can be independently derived from the models of F_1 and F_2 and the vice versa. Thus, if F_1 and F_2 are disjoint in the formula $F = F_1 \wedge F_2$, the total number of models of F is the product of the number of models of F_1 and F_2. This principle underpins the decomposition frequently applied in knowledge compilation (Lagniez and Marquis 2017).

Building on the concept of the knowledge compilation techniques, we introduce a strategy centered on formula decomposition to count minimal models. Unlike methods that count models for each disjoint component, we enumerate minimal models of F projected onto the variables of disjoint components. Our approach incorporates a level of enumeration that stops upon enumerating a specific count of minimal models, thereby providing a lower bound estimate of the total number of minimal models. Our method utilizes a straightforward "Cut" mechanism to facilitate formula decomposition.

4.1.1 Formula decomposition by "Cut"

A "cut" $\mathcal{C}$ within a formula F is identified as a subset of $\mathsf{Var}(F)$ such that for every assignment $\tau \in 2^{\mathcal{C}}$, $F|_\tau$ *effectively decomposes* into disjoint components (Lagniez and Marquis 2017). This concept is often used in context of model counting (Korhonen and Järvisalo 2021). It is important to note that models of $F|_\tau$ can be directly expanded into models of F.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

4.1.2 Challenges in knowledge compilation for counting minimal models

When it comes to counting minimal models, the straightforward application of unit propagation and the conventional decomposition approach are not viable (Kabir 2024). More specifically, simple unit propagation does not preserve minimal models. Additionally, the count of minimal models cannot be simply calculated by multiplying the counts of minimal models of its disjoint components. An example provided below demonstrates these inconsistencies.

Example 1.
Consider a formula $F = \{a \vee b \vee c, \neg a \vee \neg b \vee d, \neg a \vee \neg b \vee e\}$.

(i) With the assignment $\tau_1 = \{e\}$. Then $\{a\}$ becomes a minimal model of $F|_{\tau_1}$. However, the extended assignment $\tau_1 \cup \{a\}$ is not a minimal model of F.
(ii) Considering a cut $\mathcal{C} = \{a, b\}$ and the partial assignment $\tau_2 = \{a, b\}$, then $F|_{\tau_2}$ is decomposed into two components, each containing the unit clauses $\{d\}$ and $\{e\}$, respectively. Despite this, the combined assignment $\tau_2 \cup \{d, e\} = \{a, b, d, e\}$ is not a minimal model of F, as a strictly smaller assignment $\{a, d, e\}$ also satisfies F.

Traditional methods such as unit propagation and formula decomposition cannot be straightforwardly applied to minimal model counting. Importantly, every atom in a minimal model must be justified. In Example 1, (i) the variable e is assigned truth values without justification, leading to an incorrect minimal model when the assignment is extended with the minimal model of $F|_{\{e\}}$. (ii) The formula is decomposed without justifying the variables a and b, resulting in incorrect minimal model when combining assignments from the other two components of $F|_{\{a,b\}}$. Therefore, for accurate minimal model counting, operations such as unit propagation and formula decomposition must be applied only to assignments that are justified. Consequently, a knowledge compiler for minimal model counting must frequently verify the justification of assignments. It is worth noting that verifying the justification of an assignment is computationally intractable.

4.1.3 Minimal model counting using justified assignment

We introduce the concept of a *justified assignment* $\tau^\star$, based on a given assignment τ. Within the minimal model semantics, any assignment of false is inherently justified. Therefore, we define justified assignment $\tau^\star$ as follows: $\tau^\star = \tau_{\downarrow\{v \in \mathsf{Var}(F) | \tau(v)=0\}}$.

By applying unit propagation of $\tau^\star$, instead of τ, every minimal model derived from $F|_{\tau^\star}$ can be seamlessly extended into a minimal model of F. While $F|_\tau$ effectively decompose into multiple disjoint components, the use of a justified assignment $\tau^\star$ does not necessarily lead to effectively $F|_{\tau^\star}$ decomposing into disjoint components.

A basic approach to counting minimal models involves enumerating all minimal models. When a formula is decomposed into multiple disjoint components, the number of minimal models can be determined by conducting a projected enumeration over these disjoint variable sets and subsequently multiplying the counts of projected minimal models. The following corollary outlines how projected enumeration can be employed across disjoint variable sets to accurately count the number of minimal models.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Algorithm 1. Proj-Enum(F, $\mathcal{C}$)

1: cnt $\leftarrow 0$, $\mathcal{B} \leftarrow \emptyset$
2: **while** $\exists \sigma \in$ MinModelswithBlocking$(F, \mathcal{B})$ **do**
3: $\tau \leftarrow \sigma_{\downarrow \mathcal{C}}$, $d \leftarrow 1$
4: **for Each** comp $\in$ Components$(F|_{\tau^\star})$ **do**
5: $d = d \times |$ProjMinModels$(F, \tau,$ Var(comp))$|$
6: cnt $\leftarrow$ cnt $+ d$
7: $\mathcal{B}$.add(τ)
8: **return** cnt

Lemma 2.
Let F be decomposed into disjoint components $F_1, \ldots, F_k$, with each component F_i having a variable set $V_i = \mathsf{Var}(F_i)$, for $i \in [1, k]$. Suppose $V = \mathsf{Var}(F) = \bigcup_i V_i$. Then, $|\mathsf{MinModels}(F)| = \prod_{i=1}^{k} |\mathsf{MinModels}(F)_{\downarrow V_i}|$.

4.1.4 Algorithm: counting minimal models by projected enumeration

We introduce an enumeration-based algorithm, called Proj-Enum, that leverages justified assignments and projected enumeration to accurately count the number of minimal models of a Boolean formula F. The algorithm takes in as input a Boolean formula F and a set of variables $\mathcal{C}$, referred to as a "cut" in our context. To understand how the algorithm works, we introduce two new concepts: MinModelswithBlocking$(F, \mathcal{B})$ and ProjMinModels(F, τ, X). MinModelswithBlocking$(F, \mathcal{B})$ finds $\sigma \in \mathsf{MinModels}(F)$ such that $\forall \tau \in \mathcal{B}, \sigma \not\models \tau$; here, $\mathcal{B}$ is a set of *blocking* clauses and each blocking clause is an assignment τ. ProjMinModels(F, τ, X) enumerates the set $\{\sigma_{\downarrow X} | \sigma \in \mathsf{MinModels}(F), \sigma \models \tau\}$, where τ serves as the *conditioning* factor and X serves as the projection set. The algorithm iteratively processes minimal models of F (Line 2), starting with an initially empty set of blocking clauses ($\mathcal{B} = \emptyset$). Upon identifying a minimal model σ, the algorithm projects σ onto $\mathcal{C}$, denoting the projected set as τ. Subsequently, Algorithm 1 enumerates all minimal models $\sigma \in \mathsf{MinModels}(F)$ that satisfy $\sigma \models \tau$.

To address the inefficiency associated with brute-force enumeration, the algorithm utilizes the concept of justified assignment and projected enumeration (Line 5). Lemma 2 establishes that the number of minimal models can be counted through multiplication. The notation Components(F) (Line 4) denotes all disjoint components of the formula F. It is important to note that if $F|_{\tau^\star}$ does not decompose into more than one component, then the projection variable set X defaults to $\mathsf{Var}(F)$, which leads to brute-force enumeration of non-projected minimal models. Finally, the algorithm adds τ to $\mathcal{B}$ (Line 7) to prevent the re-enumeration of the same minimal models.

4.1.5 Implementation details of Proj-Enum.

We implemented Proj-Enum using Python. To find minimal models using MinModelswithBlocking$(F, \mathcal{B})$, the algorithm invokes an ASP solver on $\mathcal{DLP}(F)$ (invoked

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

clingo as the underlying ASP solver). Each assignment $\tau \in \mathcal{B}$ is incorporated as a constraint (Alviano *et al.* 2022), which ensures that minimal models of F are preserved ((Kabir *et al.* 2022), see Corollary. 2). To compute $\mathsf{ProjMinModels}(F, \tau, X)$, Algorithm 1 employs an ASP solver on $\mathcal{DLP}(F)$, using X as the projection set. Additionally, it incorporates each literal from τ as a *facet* into $\mathcal{DLP}(F)$ (Alrabbaa *et al.* 2018) to ensure that the condition τ is satisfied. We noted that the function $\mathsf{ProjMinModels}(F, \tau, X)$ requires more time to enumerate all minimal models. To leverage the benefits of decomposition, we enumerate upto a specific threshold number of minimal models (set the threshold to 10^6 in our experiment) invoking $\mathsf{ProjMinModels}(F, \tau, X)$. Consequently, our prototype either accurately counts the total number of minimal models or provides a lower bound. Employing a tree decomposition technique (Hamann and Strasser 2018), we calculated a cut of the formula that effectively decompose the input formula into several components.

4.2 Hashing-based minimal model counting

The number of minimal models can be approximated using a hashing-based model counting technique, which adds constraints that restrict the search space. Specifically, this method applies *uniform and random* XOR constraints to a formula F, focusing the search on a smaller subspace (Gomes *et al.* 2021). A particular XOR-based model counter demonstrates that if t trials are conducted where s random and uniform XOR constraints are added each time, and the constrained formula of F is satisfiable in all t cases, then F has at least $2^{s-\alpha}$ models with high confidence, where α is the *precision slack* (Gomes *et al.* 2006a). Each XOR constraint incorporates variables from $\mathsf{Var}(F)$. Our approach to minimal model counting fundamentally derives from the strategy of introducing random and uniform XOR constraints to the formula. In the domain of approximate model counting and sampling, the XOR constraints consist of variables from a subset of $\mathsf{Var}(F)$, denoted as $\mathcal{X}$ within our algorithm, which is widely known as *independent support* (Chakraborty *et al.* 2016; Soos and Meel 2022).

Algorithm 2 outlines a hashing-based algorithm, named $\mathsf{HashCount}$, for determining the lower bound of minimal models of a Boolean formula F. This algorithm takes in a Boolean formula F, an independent support $\mathcal{X}$, and a confidence parameter δ. During its execution, the algorithm generates total $|\mathcal{X}| - 1$ random and uniform XOR constraints, denoted as Q^i, where i ranges from 1 to $|\mathcal{X}| - 1$. To better explain the operation of the algorithm, we introduce a notation: $\mathsf{MinModels}(F^m)$ represents the minimal models of F satisfying first m XOR constraints, $Q^1, \ldots, Q^m$. Upon generating random and uniform XOR constraints, the algorithm finds the value of m such that $|\mathsf{MinModels}(F^m)| > 0$ (meaning that $\exists \sigma \in \mathsf{MinModels}(F), \sigma \models Q^1 \wedge \ldots \wedge Q^m$), while $|\mathsf{MinModels}(F^{m+1})| = 0$ (meaning that $\nexists \sigma \in \mathsf{MinModels}(F), \sigma \models Q^1 \wedge \ldots \wedge Q^{m+1}$) by iterating a loop (Line 6). The loop terminates either when a $\mathsf{Timeout}$ occurs or when it successfully identifies the value of m. If the $\mathsf{Timeout}$ happens, the algorithm assigns the maximum observed value of m (denoted as $\hat{\mathsf{m}}$) to $\mathsf{m}^\star$, ensuring that $|\mathsf{MinModels}(F^{\hat{\mathsf{m}}})| \geq 1$ (Line 7), which is sufficient to offer lower bounds. Finally, Algorithm 2 returns $2^{\mathsf{m}^\star - \alpha}$ as the probabilistic lower bound of $|\mathsf{MinModels}(F)|$.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Algorithm 2. HashCount$(F, \mathcal{X}, \delta)$

$\alpha \leftarrow -\log_2(\delta) + 1$
generate $|\mathcal{X}| - 1$ random constraints, namely $Q^1, \ldots, Q^{|\mathcal{X}|-1}$
hasMinModels$[0] \leftarrow 1$, hasMinModels$[|\mathcal{X}|] \leftarrow 0$, loIndex $\leftarrow 0$, hiIndex $\leftarrow |\mathcal{X}|$, $m \leftarrow 1$
$\hat{\mathsf{m}} \leftarrow \perp$, $\mathsf{m}^\star \leftarrow \perp$
for $i \leftarrow 1$ to $|\mathcal{X}| - 1$ **do** hasMinModels$[i] \leftarrow \perp$
while true **do**
  if Timeout **then** $\mathsf{m}^\star \leftarrow \hat{\mathsf{m}}$ **break**
  if $\exists \sigma \in$ MinModels(F^m) **then**
    $\hat{\mathsf{m}} \leftarrow$ Max$(\hat{\mathsf{m}}, m)$
    if hasMinModels$[m+1] = 0$ **then** $\mathsf{m}^\star \leftarrow m$ **break**
    for $i \leftarrow 1$ to m **do** hasMinModels$[i] \leftarrow 1$
    loIndex $\leftarrow m$
    if $2 \times m < |\mathcal{X}|$ **then** $m \leftarrow 2 \times m$
    else $m \leftarrow \frac{(\text{hiIndex}+m)}{2}$
  else
    if hasMinModels$[m-1] = 1$ **then** $\mathsf{m}^\star \leftarrow m - 1$ **break**
    for $i \leftarrow m$ to $|\mathcal{X}| - 1$ **do** hasMinModels$[i] \leftarrow 0$
    hiIndex $\leftarrow m$
    $m \leftarrow \frac{(\text{loIndex}+m)}{2}$
return $2^{\mathsf{m}^\star - \alpha}$

Algorithm 3. MinLB(F, δ)

if $\nexists \sigma \in$ MinModels(F) **then return** 0
if $|\text{Cut}(F)| \leq 50$ **then return** Proj-Enum$(F, \text{Cut}(F))$
else return HashCount$(F, \text{IndependentSupport}(F), \delta)$

4.2.1 Implementation details of HashCount

The effectiveness of an XOR-based model counter is dependent on the performance of a theory+XOR solver (Soos *et al.* 2020). In our approach, we begin by transforming a given formula F into a disjunctive logic program $\mathcal{DLP}(F)$ and introduce random and uniform XOR constraints into $\mathcal{DLP}(F)$ to effectively partition the minimal models of F. To verify the presence of any models in the XOR-constrained ASP program, we leverage the ASP + XOR solver capabilities provided by ApproxASP (Kabir *et al.* 2022). To compute an independent support for HashCount, we implemented a prototype inspired by Arjun (Soos and Meel 2022), which checks answer sets of a disjunctive logic program in accordance with Padoa's theorem (Padoa 1901).

4.3 Putting it all together

We designed a hybrid solver MinLB, presented in Algorithm 3, that selects either Proj-Enum or HashCount depending on the decomposability of the input formula. The core

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

principle of MinLB is that Proj-Enum effectively leverages decomposition and projected enumeration on *easily decomposable* formulas. We use the size of the cut ($|\mathsf{Cut}(F)|$) as a proxy to measure the decomposability. Thus, if $|\mathsf{Cut}(F)|$ is small, then MinLB employs Proj-Enum on F (Line 2); otherwise, it employs HashCount (Line 3).

Theorem 1.
$\Pr[\mathsf{MinLB}(F, \delta) \leq |\mathsf{MinModels}(F)|] \geq 1 - \delta$

5 Experimental Results

5.1 Benchmarks and baselines

Our benchmark set is collected from two different domains: (i) model counting benchmarks from recent competitions (Fichte *et al.* 2021) and (ii) minimal generators benchmark from itemset mining dataset CP4IM.[2] We used existing systems for minimal model reasoning as baselines. These included various approaches such as (i) repeated invocations of the SAT solver MiniSAT (Li *et al.* 2021), (ii) the application of MaxSAT techniques (Alviano 2017), (iii) domain-specific heuristics (Gebser *et al.* 2013), and (iv) solving $\mathcal{DLP}(F)$ with ASP solvers, all systems primarily count via enumeration. These systems either return the number of minimal models by enumerating all of them or a lower bound of the number of minimal models in cases where they run out of time or memory. We cannot count minimal models by #SAT-based ASP counters (Kabir *et al.* 2024) because of their incapablity of handling disjunctive logic programs. Experimentally, we observed that, for enumerating all minimal models, the technique solving disjunctive ASP programs using clingo (Gebser *et al.* 2012) surpassed the other techniques. Therefore, we have exclusively reported the performance of clingo in our experimental analysis. Additionally, we evaluated ApproxASP (Kabir *et al.* 2022), which offers (ϵ, δ)-guarantees in counting minimal models. We attempted an exact minimal model counting tool using a subtractive approach — subtracting the *non-minimal model* count from the total model count–we denote the implementation using the notation #MinModels in further analysis. In our experiment, we ran HashCount with a confidence δ value of 0.2 and ApproxASP with confidence $\delta = 0.2$ and tolerance $\epsilon = 0.8$. The prototype is available at: https://github.com/meelgroup/MinLB.

5.2 Environmental settings

All experiments were conducted on a high-performance computing cluster equipped with nodes featuring AMD EPYC 7713 CPUs, each with 128 real cores. Throughout the experiment, the runtime and memory limits were set to 5000 seconds and 16GB, respectively, for all considered tools.

5.3 Evaluation metric

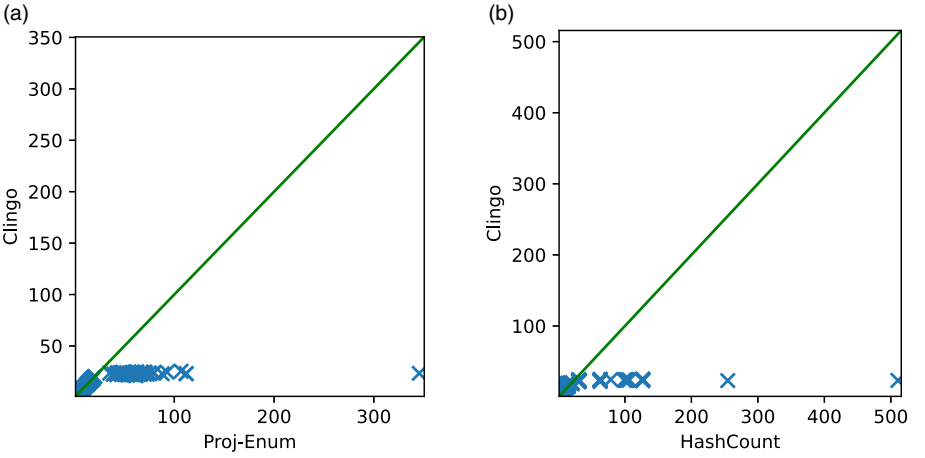
The goal of our experimental analysis is to evaluate various minimal model counting tools based on their runtime and the quality of their lower bounds. It is essential to employ a

[2] https://dtai.cs.kuleuven.be/CP4IM/datasets/

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Table 1. *The Time Quality Penalty scores of* MinLB *and other tools on model counting benchmark*

clingo	*ApproxASP*	#MinModels	MinLB (our prototype)
6491	6379	7743	5599



The lower bound returned by Proj-Enum The lower bound returned by HashCount

Fig 1. The lower bound of Proj-Enum and HashCount vis-a-vis the lower bound returned by clingo on minimal model counting benchmark. The axes are in log scale.

metric that encompasses both runtime performance and the quality of the lower bound. Consequently, following the TAP score (Agrawal *et al.* 2021; Kabir and Meel 2023), we have introduced a metric, called the *Time Quality Penalty* (TQP) score, which is defined for each tool and instance as follows:

$$\mathsf{TQP}(t, C) = \begin{cases} 2 \times \mathcal{T}, & \text{if no lower bound is returned} \\ t + \mathcal{T} \times \frac{1+\log(\mathsf{C}_{\min}+1)}{1+\log(C+1)}, & \text{otherwise} \end{cases}$$

In the metric, $\mathcal{T}$ represents the timeout for the experiment, t denotes the runtime of the tool, C is the lower bound returned by the tool, and $\mathsf{C}_{\min}$ is the minimum lower bound returned by any of the tools under consideration for the instance. The TQP score is based on the following principle: lower runtime and higher lower bound yield a better score.

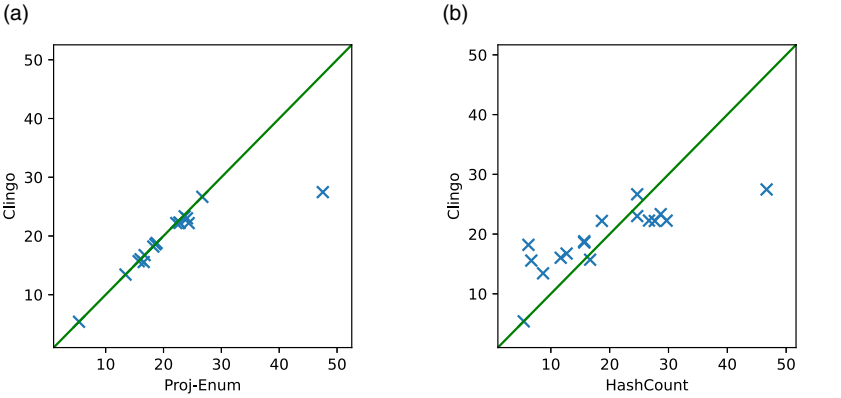
5.4 Performance on model counting competition benchmark

We present the TQP scores of MinLB alongside other tools in Table 1. This table indicates that MinLB achieves the lowest TQP scores. Among existing minimal model enumerators, clingo demonstrates the best performance in terms of TQP score. Additionally, in Figure 1, we graphically compare the lower bounds returned by Proj-Enum and HashCount

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Table 2. *The Time Quality Penalty scores of* MinLB *and other tools on minimal generator benchmark*

clingo	*ApproxASP*	#MinModels	MinLB (our prototype)
6944	5713	9705	5043



The lower bound returned by Proj-Enum The lower bound returned by HashCount

Fig 2. The lower bound returned by Proj-Enum and HashCount vis-a-vis the lower bound given by clingo on minimal generators benchmark. The axes are in log scale.

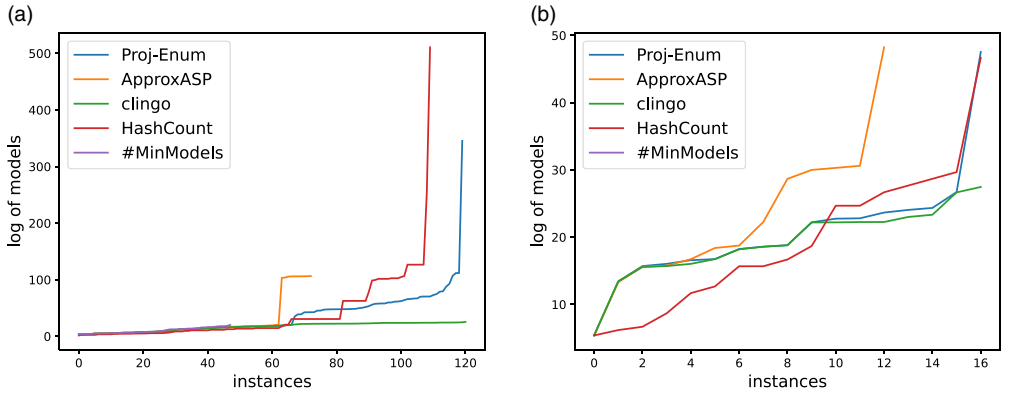
against those returned by *clingo*. Here, a point (x, y) indicates that, for an instance, the lower bounds returned by our prototypes and clingo are 2^x and 2^y, respectively. For an instance, if the corresponding point resides below the diagonal line, it indicates that Proj-Enum (HashCount, resp.) returns a better lower bound than *clingo*. These plots clearly illustrate that Proj-Enum and HashCount return better lower bounds compared to other existing minimal model enumerators.

5.5 Performance on minimal generator benchmark

Table 2 showcases the TQP scores of MinLB alongside other tools on the minimal generator benchmark. Notably, HashCount achieves the most favorable TQP scores on the benchmark. Additionally, Figure 2 graphically compares the lower bounds returned by Proj-Enum and HashCount against those computed by *clingo*.

For a visual representation of the lower bounds returned by our HashCount and Proj-Enum, we illustrate them graphically in Figure 3. In the plot, a point (x, y) signifies that a tool returns a lower bound of at most 2^y for x instances. The plot demonstrates that the lower bounds returned by Proj-Enum and HashCount surpass those of existing systems.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press



Model counting benchmark Minimal generators benchmark

Fig 3. The lower bounds returned by Proj-Enum, HashCount, and existing minimal model counting tools. The y-axis show the log of the number of models.

5.6 Another performance metric

To facilitate the comparison of lower bounds returned by two tools, we have introduced another metric for comparative analysis. If tools A and B yield lower bounds C_A and C_B respectively, their *relative quality* is defined in the following manner:

$$r_{AB} = \frac{1 + \log(C_A + 1)}{1 + \log(C_B + 1)} \tag{3}$$

If $r_{AB} > 1$, then the lower bound returned by tool A is superior to that of tool B.

5.6.1 In-depth study on Proj-Enum *and* HashCount

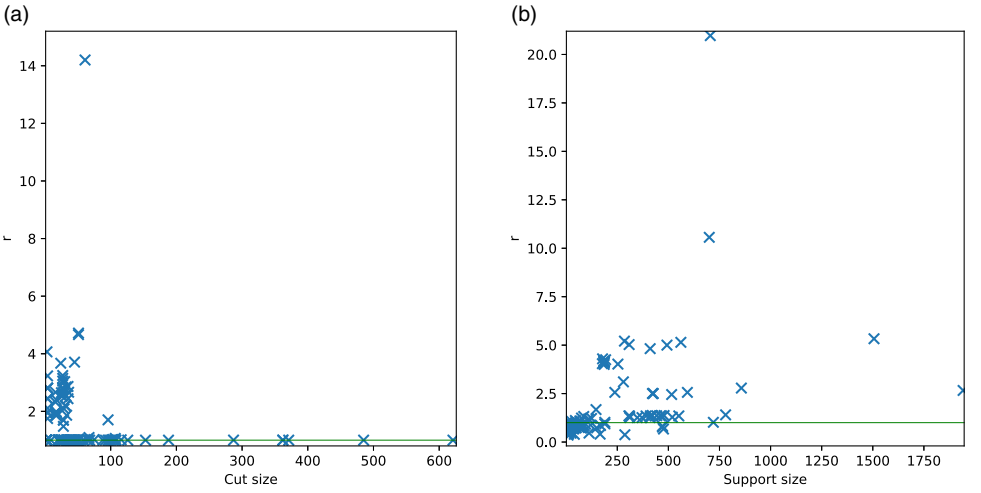
The performance of Proj-Enum and HashCount contingent upon the size of the cut and independent support, respectively. In this analysis, we explore the strengths and weaknesses of Proj-Enum and HashCount by measuring their relative quality, as defined in Equation (3), across various sizes of cuts and independent supports, respectively. This comparative analysis is visually represented in Figure 4, where clingo serves as the reference baseline.

In the graphical representations, each point (x, y) corresponds to an instance where for the size of cut (independent support resp.) is x and the prototype Proj-Enum (HashCount resp.) achieves a relative quality of y. A relative quality exceeding 1 indicates that the lower bound returned by Proj-Enum or HashCount surpasses that of clingo. These plots reveal that Proj-Enum tends to perform well with smaller cut sizes, while HashCount demonstrates better performance across a range from small to medium sizes of independent support.

6 Conclusion

This paper introduces two innovative methods for computing a lower bound on the number of minimal models. Our first method, Proj-Enum, leverages knowledge compilation

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press



The relative quality of Proj-Enum with varying cut size.

The relative quality of HashCount with varying independent support size.

Fig 4. The relative quality of Proj-Enum and HashCount vis-a-vis different cut and independent support size, where clingo is used as the reference baseline. The horizontal line is across $r = 1$.

techniques to provide improved lower bounds for easily decomposable formulas. The second method, HashCount, utilizes recent advancements in ASP + XOR reasoning systems and demonstrates performance that varies with the size of the independent support. Our proposed methods exploit the expressive power of ASP semantics and robustness of well-engineered ASP systems. Looking forward, our research will focus on counting projected minimal models. We also plan to explore the counting of *minimal correction subsets*, which are closely related to minimal models.

Acknowledgement

This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004], Ministry of Education Singapore Tier 2 grant MOE-T2EP20121-0011, and Ministry of Education Singapore Tier 1 Grant [R-252-000-B59-114]. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg). Kabir was visiting the University of Toronto during the research.

References

Agrawal, D., Pote, Y. and Meel, K. S. 2021. Partition function estimation: A quantitative study. In *IJCAI*, 4276–4285.

Alrabbaa, C., Rudolph, S. and Schweizer, L. 2018. Faceted answer-set navigation, In *RuleML+RR*. Springer, 211–225

Alviano, M. 2017. Model enumeration in propositional circumscription via unsatisfiable core analysis. *Theory and Practice of Logic Programming* 17, 5-6, 708–725.

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Alviano, M., Dodaro, C., Fiorentino, S., Previti, A. and Ricca, F.2022. Enumeration of minimal models and MUSes in WASP. In *LPNMR*. Springer, 29–42

Angiulli, F., Ben-Eliyahu, R., Fassetti, F. and Palopoli, L. 2014. On the tractability of minimal model computation for some CNF theories. *Artificial Intelligence* 210, 56–77.

Angiulli, F., Ben-Eliyahu, R., Fassetti, F. AND Palapoli, L. 2022. Graph-based construction of minimal models. *Artificial Intelligence* 313, 103754.

Ben-Eliyahu, R. 2005. An incremental algorithm for generating all minimal models. *Artificial Intelligence* 169, 1, 1–22.

Ben-Eliyahu, R. and Dechter, R. 1994. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*. 12, 1-2, 53–87.

Ben-ELIYAHU, R. and Dechter, R. 1996. On computing minimal models. *Annals of Mathematics and Artificial Intelligence* 18, 1, 3–27.

Ben-ELIYAHU, R. and Palopoli, L. 1997. Reasoning with minimal models: Efficient algorithms and applications. *Artificial Intelligence* 96, 2, 421–449.

Bozzano, M., Cimatti, A., Griggio, A., Jonas, M. and Kimberley, G. 2022. Analysis of cyclic fault propagation via ASP. In *LPNMR*. Springer, 470–483

Brewka, G., Delgrande, J., Romero, J. and Schaub, T. 2015. Asprin: Customizing answer set preferences without a headache. In *AAAI*, vol. 29.

Cadoli, M. 1992a. The complexity of model checking for circumscriptive formulae. *Information Processing Letters* 44, 3, 113–118.

Cadoli, M. 1992b. On the complexity of model finding for nonmonotonic propositional logics, In Italian Conference on Theoretical Computer Science, 1992, 125–139.

Chakraborty, S., Meel, K. S. and Vardi, M. Y. 2013. A scalable approximate model counter. In *CP*. Springer, 200–216

Chakraborty, S., Meel, K. S. and Vardi, M. Y. 2016. Algorithmic improvements in approximate counting for probabilistic inference: from linear to logarithmic SAT calls. In *IJCAI*, 3569–3576

Clark, K. L.1978. Negation as failure. In *Logic and Data Bases*, 293–322

Darwiche, A. 2004. New advances in compiling CNF to decomposable negation normal form. In *ECAI*. Citeseer, 328–332

De kleer, J., Mackworth, A. K. and Reiter, R. 1992. Characterizing diagnoses and systems. *Artificial Intelligence* 56, 2-3, 197–222.

Durand, A. and Hermann, M. 2008. On the counting complexity of propositional circumscription. *Information Processing Letters* 106, 4, 164–170.

Eiter, T. and Gottlob, G. 1993. Propositional circumscription and extended closed-world reasoning are $\prod_2^{\mathrm{P}}$-complete. *Theoretical Computer Science* 114, 2, 231–245.

Faber, W., Vallati, M., Cerutti, F. and Giacomin, M. 2016. Solving set optimization problems by cardinality optimization with an application to argumentation.

Fichte, J. K., Gaggl, S. A. and Rusovac, D. 2022. Rushing and strolling among answer sets–navigation made easy. In *AAAI*, vol. 36, 5651–5659

Fichte, J. K., Hecher, M. and Hamiti, F. 2021. The model counting competition 2020. *ACM Journal of Experimental Algorithmics* 26, 1–26.

Gebser, M., Kaufmann, B. and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence* 187-188, 52–89.

Gebser, M., Kaufmann, B., Romero, J., Otero, R., Schaub, T. and Wanko, P. 2013. Domain-specific heuristics in answer set programming. In *AAAI*, vol. 27, 350–356.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3-4, 365–385.

Gomes, C. P., Hoffmann,J., Sabharwal, A. and Selman, B. 2007. From sampling to model counting. In *IJCAI*, vol. 2007, 2293–2299

Gomes, C. P., A., Sabharwal and Selman, B. 2006a. Model counting: A new strategy for obtaining good bounds. In *AAAI*, vol. 10, 1597538–1597548

Gomes, C. P., Sabharwal, A. and Selman, B. 2006b. Near-uniform sampling of combinatorial spaces using XOR constraints. *NIPS* 19, 670–676.

Gomes, C. P., Sabharwal, A. and Selman, B. 2021. Model counting. In *Handbook of Satisfiability*. IOS Press, 993–1014.

Hamann, M. and Strasser, B. 2018. Graph bisection with pareto optimization. *ACM Journal of Experimental Algorithmics* 23, 1–34.

Hunter, A. and Konieczny, S. 2008. Measuring inconsistency through minimal inconsistent sets. *KR* 8, 42, 358–366.

Jabbour, S., Sais, L. and Salhi, Y. 2017. Mining Top-k motifs with a SAT-based framework. *Artificial Intelligence* 244, 30–47.

Jannach, D., Schmitz, T. and Shchekotykhin, K. 2016. Parallel model-based diagnosis on multi-core computers. *Journal of Artificial Intelligence Research* 55, 835–887.

Kabir, M. 2024. Minimal model counting via knowledge compilation. arXiv preprint arXiv: 2409.10170.

Kabir, M., Chakraborty, S. and Meel, K. S. 2024.Exact ASP counting with compact encodings. In *AAAI*, vol. 38, 10571–10580

Kabir, M., Everado, F. O., Shukla, A. K., Hecher, M., Fichte, J. K. and Meel, K. S. 2022. ApproxASP–a scalable approximate answer set counter. In *AAAI*, vol. 36, 5755–5764

Kabir, M. and Meel, K. S. 2023. A fast and accurate ASP counting based network reliability estimator. In *LPAR*, 270–287

Kirousis, L. M. and Kolaitis, P. G. 2003. The complexity of minimal satisfiability problems. *Information and Computation* 187, 1, 20–39.

Korhonen, T. and Jarvisalo, M. 2021. Integrating tree decompositions into decision heuristics of propositional model counters. In *CP*, 8–1

Lagniez, J.-M. and Marquis, P. 2017. An improved decision-DNNF compiler. In *IJCAI*. vol. 17, 667–673

Li, Z., Yisong, W., Zhongtao, X. and Renyan, F. 2021. Computing propositional minimal models: MiniSAT-based approaches. *Journal of Computer Research and Development* 58, 11, 2515–2523.

Liffiton, M. H. and Sakallah, K. A. 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. *Journal of Automated Reasoning* 40, 1–33.

Lifschitz, V. 1985. Computing circumscription. In *IJCAI*. vol. 85, 121–127

Marek, V. W. and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In *The logic Programming Paradigm: A 25-Year Perspective*, 375–398

Mccarthy, J. 1980. Circumscription–a form of non-monotonic reasoning. *Artificial Intelligence* 13, 1-2, 27, 39.

Minker, J. 1982. On indefinite databases and the closed world assumption. In *CADE*. Springer, 292–308

Padoa, A. 1901. Essai d'une théorie algébrique des nombres entiers, précédé d'une introduction logique à une théorie déductive quelconque. *Bibliothèque Du Congrès International De Philosophie* 3, 309–365

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13, 1-2, 81–132.

Salhi, Y. 2019. On enumerating all the minimal models for particular CNF formula classes. In *ICAART*, vol. 2, 403–410

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

Soos, M., S., Gocht and Meel, K. S. 2020. Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In *CAV*. Springer, 463–484

Soos, M. and Meel, K. S. 2022. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In *ICCAD*, 1–9

Thimm, M. 2016. On the expressivity of inconsistency measures. *Artificial Intelligence* 234, 120–151.

Thurley, M. 2006. SharpSAT–counting models with advanced component caching and implicit BCP. In *SAT*. Springer, 424–429

Appendix

The missing proofs are available in the extended version of the paper: https://arxiv.org/abs/2407.09744.

Analysis of Proj-Enum

Lemma 3.
For Boolean formula F and a cut $\mathcal{C}$, the minimal models of F can be computed as follows: $\mathsf{MinModels}(F) = \bigcup_{\tau \in 2^{\mathcal{C}}} \mathsf{ProjMinModels}(F, \tau, \mathsf{Var}(F))$.

Proof.
Lemma 2 demonstrates that minimal models can be computed through component decompositions. By taking the union over $\tau \in 2^{\mathcal{C}}$, we iterate over all possible assignments of $\mathcal{C}$. Consequently, Proj-Enum algorithm computes all minimal models of F by conditioning over all possible assignments over $\mathcal{C}$. Therefore, the algorithm is correct. □

Analysis of HashCount

We adopt the following notation: $s^\star = \log_2 |\mathsf{MinModels}(F)|$. Each minimal model σ of F is an assignment over $\mathsf{Var}(F)$, and according to the definition of random XOR constraint (Gomes *et al.* 2006b), σ satisifies a random XOR constraint with probability of 1/2. Due to uniformity and randomness of XOR constraints, each minimal model of F satisfies m random and uniform XOR constraints with probability of $1/2^m$. In our theoretical analysis, we apply the Markov inequality: if Y is a non-negative random variable, then $\mathsf{Pr}[Y \geq a] \leq \frac{\mathbb{E}[Y]}{a}$, where $a > 0$.

Lemma 4.
For arbitrary s, $\mathsf{Pr}[|\mathsf{MinModels}(F^s) \geq 1|] \leq \frac{2^{s^\star}}{2^s}$.

Proof.
For each $\sigma \in \mathsf{MinModels}(F)$, we define a random variable $Y_\sigma \in \{0, 1\}$ and $Y_\sigma = 1$ indicates that σ satisifies the first s XOR constraints $Q^1, \ldots, Q^s$, otherwise, $Y_\sigma = 0$. The random variable Y is the summation, $Y = \sum_{\sigma \in \mathsf{MinModels}(F)} Y_\sigma$. The expected value of Y can be calculated as $\mathbb{E}(Y) = \sum_{\sigma \in \mathsf{MinModels}(F)} \mathbb{E}(Y_\sigma)$.

Due to the nature of random and uniform XOR constraints, each minimal model $\sigma \in \mathsf{MinModels}(F)$ satisfies all s XOR constraints with probability $\frac{1}{2^s}$. It follows that the expected value $\mathbb{E}(Y_\sigma) = \frac{1}{2^s}$, and the expected value of Y is $\mathbb{E}(Y) = \frac{|\mathsf{MinModels}(F)|}{2^s} = \frac{2^{s^\star}}{2^s}$. According to the Markov inequality: $\mathsf{Pr}[|\mathsf{MinModels}(F^s) \geq 1|] \leq \frac{2^{s^\star}}{2^s}$. □

Lemma 5.
Given a formula F and confidence δ, if $\mathsf{HashCount}(F, \delta)$ *returns* $2^{s-\alpha}$, *then* $\mathsf{Pr}[2^{s-\alpha} \leq |\mathsf{MinModels}(F)|] \geq 1 - \delta$

Proof.
Given a input Boolean formula F and for each $m \in [1, |\mathcal{X}| - 1]$, we denote the following two events: I_m denotes the event that Algorithm 2 invokes $\mathsf{MinModels}(F^m)$ and E_m denotes the event that $|\mathsf{MinModels}(F^m)| \geq 1$. The algorithm $\mathsf{HashCount}(F, \delta)$ returns an incorrect bound when

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press

$s - \alpha > s^\star$ and let use the notation Error to denote that $\mathsf{HashCount}(F, \delta)$ returns an incorrect bound or $\mathsf{HashCount}(F, \delta) > 2^{s^\star}$. The upper bound of $\mathsf{Pr}[\mathsf{Error}]$ can be calculated as follows:

$$\begin{aligned}
\mathsf{Pr}[\mathsf{Error}] &= \mathsf{Pr}[\mathsf{HashCount}(F, \delta) \text{returns} 2^{s-\alpha} \text{and } s > s^\star + \alpha] \\
&\leq \sum_{s > s^\star + \alpha} \mathsf{Pr}[I_s \cap E_s] \leq \sum_{s > s^\star + \alpha} \mathsf{Pr}[E_s] \\
&\leq \sum_{s > s^\star + \alpha} \mathsf{Pr}[|\mathsf{MinModels}(F^s)| \geq 1] \leq \sum_{s > s^\star + \alpha} \frac{2^{s^\star}}{2^s} \qquad \text{According to Lemma 4} \\
&\leq \frac{1}{2^\alpha} \times 2 \leq 2^{1-\alpha} \leq 2^{\log_2 \delta} \leq \delta
\end{aligned}$$

Thus, $\mathsf{Pr}[\mathsf{HashCount}(F, \delta)$ returns $2^{s-\alpha}$ and $s \leq s^\star + \alpha] \geq 1 - \delta$. □

https://doi.org/10.1017/S147106842400036X Published online by Cambridge University Press