


# Towards Building A Scalable Bit-vector Model Counter

Arijit Shaw ✉ 🏠 

Chennai Mathematical Institute, India

IAI, TCG-CREST, Kolkata, India

Kuldeep S. Meel ✉ 🏠 

National University of Singapore

---

## Abstract

---

Satisfiability Modulo Theory (SMT) solvers have transformed the field of automated reasoning owing to their established efficiency in handling problems arising from diverse domains. Given the significant progress achieved by SMT solvers over the past two decades, it is natural to ask how we can aim to solve beyond satisfiability. In this context, we take note of the significant progress achieved in the development of propositional model counting over the past decade. The progress in propositional model counting has also inspired new applications, and consequently, it is natural to explore the design of scalable counters in the context of SMT. As a first step, we focus on the problem of model counting on the quantifier-free fragment of the theory of bit-vector arithmetic. Propositionalization (aka bitblasting) is often used for checking satisfiability of bitvector formulas. In this work, we create a portfolio-based bitvector model counter, **SharpSMT**, that harnesses the power of the recent progress made in propositionalization, off-the-shelf model counters and preprocessing tools.

The main contribution of our paper involves the design of a portfolio-based bitvector counter, creation of an application benchmark set for testing, and experimentation with varying components of CNF-counters, propositionalization techniques, and preprocessors; ultimately resulting in a system that solves four times the number of benchmarks solved by current state-of-the-art methods. Our empirical analysis highlights the need for careful tight integration of propositionalization, preprocessing, and CNF-counters in building such a system.

**2012 ACM Subject Classification** Theory of computation → Automated reasoning

**Keywords and phrases** Satisfiability Modulo Theories, Model Counting

**Supplementary Material** We provide the source code and all experimental data:

*Software (Source Code)*: <https://github.com/meelgroup/sharpSMT>

*Dataset (Benchmark, Experimental Data)*: <https://tinyurl.com/sat-23-sm>

## 1 Introduction

The paradigm of Satisfiability modulo Theory (SMT) solving has been core to the advances in hardware and software verification over the past two decades. Consequently, the community has developed several scalable state-of-the-art SMT solvers over the past two decades. For many problem instances, satisfiability does not suffice, and one is often interested in computations over the solution set. One such problem is counting the number of solutions of a formula when the set of solutions is finite. As a starting point, we focus on the case when the underlying formula is expressed in *quantifier-free bit-vector arithmetic* or, *QFBV* (referred to as *bit-vectors* in simpler terms hereafter). Our motivation for bit-vectors stems from it being one of the first theories to be investigated in the context of SMT solving and also recent works that have demonstrated applications of counting for bit-vector formulas in domains such as cryptography [1] and software verification [14, 26].

The challenge of counting in the bit-vector model can be addressed through two methods: (i) conducting reasoning over bit-vectors, or (ii) reducing the problem to CNF. While the

former approach has been extensively studied in recent years through the utilization of lifting techniques for word-level constraints [2], the latter has not been thoroughly assessed. Given the scalability of CNF-counters, we investigate the design of a counting tool for bit-vector formulas by relying on the propositionalization (aka bit-blasting) followed by the usage of preprocessors and CNF-counters. Given the existence of several propositionalization schemes, preprocessors and CNF-counters; it is natural to wonder what would be the ideal combination for these three components. In this paper, we focus on the question : *given a bit-vector formula, what is the best possible way to count the number of solutions of the formula?*

The primary contribution of this paper is a rigorous evaluation process to understand the efficacy of these three components in counting bit-vector formulas. In particular,

1. Our work introduces **SharpSMT**, a portfolio bit-vector model counter that combines off-the-shelf CNF-counters with propositionalization and preprocessing tools. In its optimal configuration, **SharpSMT** is capable of solving 655 benchmarks, whereas the current state-of-the-art, **SMTApproxMC**, can only solve 148. Additionally, **SharpSMT** has exact model counting capabilities, where it solves 412 instances in the best configuration.
2. Since there is no standardized set of benchmark suites for bit-vectors, we have curated a collection of 679 benchmarks from various practical domains like cryptography and software verification. This collection was used to achieve a detailed empirical analysis to find the best combination of components in **SharpSMT**.
3. Detailed experiments on these set of benchmarks reveal : (i) the best performing configuration for **SharpSMT** is (TseitinEnc-ArjunImplicit-ApproxMC). (ii) If one is interested in exact counts, then the best performing configuration of **SharpSMT** is (TseitinEnc-NoPreproc-SharpSAT-TD).

The rest of the paper is organized as follows: We introduced the preliminaries and related work in Section 2. In Section 3, we present an overview of our framework, **SharpSMT**. We briefed our experimental methodology and results in Section 4. Finally, we conclude in Section 5.

## 2 Background

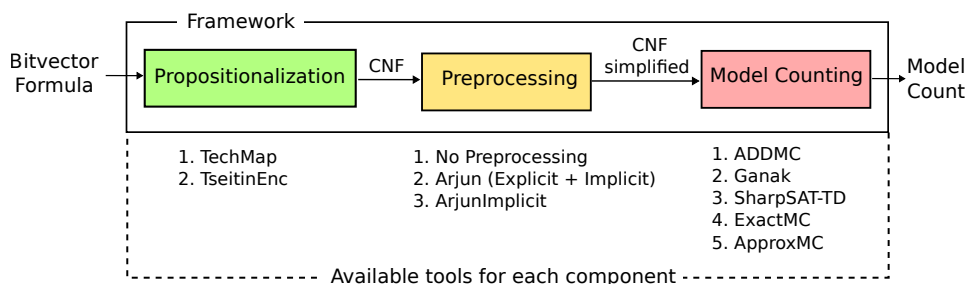
A word (or bit-vector) is an array of bits. The size of the array is called the width of the word. We consider here fixed-width words whose width is a constant. It is easy to see that a word of width  $k$  can be used to represent elements of a set of size  $2^k$ . Let  $X$  be the set of word-level variables and let  $F$  be a formula in the theory of *quantifier free bit-vectors*. A *model* or *solution* of  $F$  is an assignment of word-level constants to variables in  $X$  such that  $F$  evaluates to True. Let  $\text{sol}(F)$  denote the set of models of  $F$ . The problem of counting is to compute  $|\text{sol}(F)|$ . For many applications of model counting, small differences in the count may not be important, and it may be sufficient to provide rough estimates as long as the method is fast and the user has reasonable confidence in its accuracy. An approximate model counter takes in a formula  $F$ , tolerance parameter  $\varepsilon$ , confidence parameter  $\delta$  and returns  $c$  such that  $\Pr \left[ \frac{|\text{Sol}(F)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\text{sol}(F)| \right] \geq 1 - \delta$

**Related Work.** The problem of enumerating the solutions to theory constraints was initially addressed by Martin [22]. Afterwards, the success of propositional model counters, particularly approximate model counters, prompted efforts to extend the techniques to word-level constraints. Chistikov et al. [5] used bit-blasting to extend the hashing-based model counting technique from the highly successful approximate CNF model counter, **ApproxMC** [3], to

word-level benchmarks. Chakraborty et al. [2] designed `SMTApproxMC` by lifting the hash functions for word-level constraints. Kim et al. [16, 17] designed a system for statistical estimation to refine a probabilistic estimate of the model count continuously, but their approach lacks  $(\epsilon, \delta)$ -guarantees. Ge et al. developed a probabilistic polynomial-time model counting algorithm [10] and demonstrated its effectiveness on bit-vector problems. Ge et al. also designed a series of algorithms, including `INTCOUNT` [8], `VolCE` [12], `PolyVest` [9, 13], and `Vol2Lat` [11], for either counting or approximating the number of models for the SMT theory of linear arithmetic.

### 3 Approach

We developed `SharpSMT`, a model counting tool for the quantifier-free fragment of the theory of bitvectors. Our tool estimates the size of the solution set for a given bitvector formula. To achieve this goal, `SharpSMT` employs a portfolio-based approach that involves propositionalization and preprocessing before applying an off-the-shelf model counter. Our approach of reducing the problem to CNF enables our tool `SharpSMT` to take advantage of ongoing improvements in propositional model counting. As shown in Figure 1, `SharpSMT` comprises three primary components: (i) a tool for propositionalizing SMT2 formulas into CNFs, (ii) a preprocessor for simplifying CNFs, and (iii) an off-the-shelf model counter. In the following part of this section, we briefly describe the task and available tools for each component. More description and rationale about selecting the current tools are given in the ??.



■ **Figure 1** Overview of the framework and available tools in `SharpSMT`.

#### Phase 1 : Propositionalization

The most commonly used method for converting a bitvector formula into a propositional Boolean formula involves representing the formula as an And-Inverter Graph (AIG) circuit, which is subsequently converted into conjunctive normal form (CNF). There are multiple techniques available for converting an AIG to CNF, with the two most commonly used ones being: (i) `TseitinEnc` : the standard Tseitin encoding [27] and (ii) `TechMap` : technology mapping based logic synthesis as proposed by Een et al. [7]. The `TseitinEnc` method provides a straightforward encoding for the AIG, whereas the `TechMap` method performs various optimizations on the circuit to produce a minimized CNF. From the perspective of satisfiability, studies have revealed that the performance of the encoding scheme is heavily reliant on the benchmark set being investigated, with `TseitinEnc` exhibiting superior results in some benchmark sets and `TechMap` being more effective in others [15]. In `SharpSMT`, we experimented with both the method.

## Phase 2 : Preprocessing

Preprocessing is a technique commonly employed in satisfiability and model counting to simplify the original problem instance. In practical scenarios, the problem instances are typically rooted in circuits within a specific domain. These circuits are comprised of gates, and the variables correspond to either input or output variables, with the output variables being determined by the input variables. Preprocessing techniques have been developed to effectively handle the *input-output bipartition* property, as discussed in several studies [19, 20, 25]. At SharpSMT, we utilize Arjun [25] in two modes. The first mode employs an explicit gate detection algorithm, while the second mode disables this feature. When the explicit gate detection is disabled, we refer to the preprocessing technique as ArjunImplicit. However, when the feature is enabled, we simply refer to the technique as Arjun.

## Phase 3 : Model Counting

The third phase of SharpSMT comprises of a propositional model counter that finally determines the number of solutions. We investigated various approaches to model counting, including the paradigms of : (i) Search-based counters (ii) Compilation-based counters (iii) Hashing-based approximate counters. In recent years, there has been a proliferation of model counters, and the annual model counting competitions have demonstrated that various model counters have distinct strengths and weaknesses for different problem types. In the context of the current research, we examine the applicability of five model counters within the framework of SharpSMT : two search-based counters SharpSAT-TD [18], Ganak [23]; two compilation-based counters - ADDMC [6], and ExactMC [21]; and a hashing based approximate counter ApproxMC [3, 4, 24]. By leveraging these diverse model counters, we aim to enhance the computational efficiency of our investigations.

Thus, SharpSMT consists of three distinct *components*. We call particular combination of these three components as a *configuration* of SharpSMT. In the following part of the paper, we use the notation (A-B-C) to denote a configuration of SharpSMT, where A represents the propositionalization technique, B represents the preprocessing technique, and C represents the CNF-counter utilized in that particular configuration of SharpSMT.

## 4 Experimental Evaluation

The primary objective behind the development of SharpSMT was to identify an optimal approach for performing bitvector counting. To assess the efficacy and accuracy of SharpSMT under various configurations, we utilized an experimental setup that involved 30 different configurations of SharpSMT. The evaluation procedure was conducted using the following setup. We curate a substantial collection of benchmarks that pertain to the problem of model counting and naturally encode into bitvector formulas. The benchmarks are sources from quantitative software reliability estimation [26], robust reachability [14], cryptography [1], as well as prior literature on bitvector counting [16]. We refer to these benchmark classes as *counterssharp*, *robust*, *delphinium* and *smc* respectively in the following part. The total number of benchmarks used in our evaluation amounts to 679. We conducted all our experiments on a high-performance computer cluster, with each node consisting of E5-2690 v3 CPU. We set the memory limit to 24 GB for all configurations, and we ran each solver instance on a single core. To adhere to the standard timeout used in model counting competitions, we set the timeout for all experiments to 3600 seconds. We compare our performance with prior

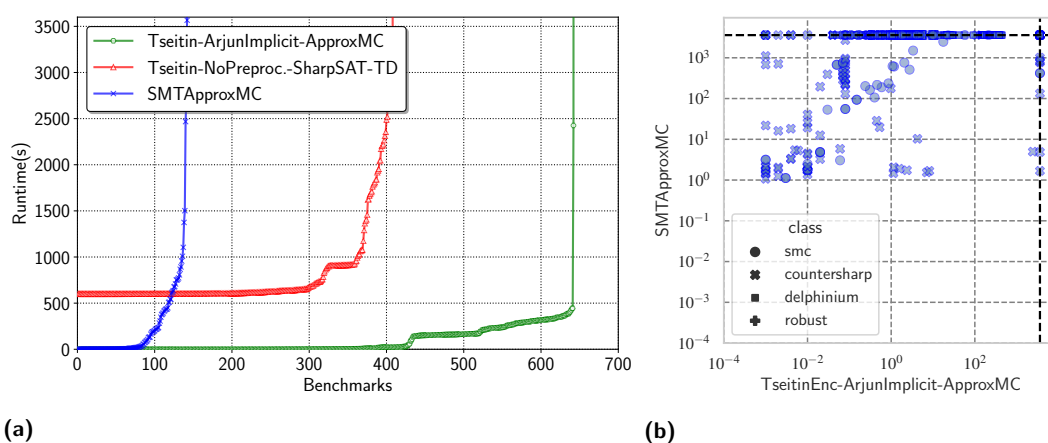
state-of-the-art bitvector model counter, SMTApproxMC. In particular, we sought to answer to the following questions:

- RQ1.** How does the performance of SharpSMT compare to SMTApproxMC?
- RQ2.** How do different components, including propositionalization, preprocessing, and model counting, affect the performance of SharpSMT?
- RQ3.** What is the variation in the counter’s performance results across benchmark sets?

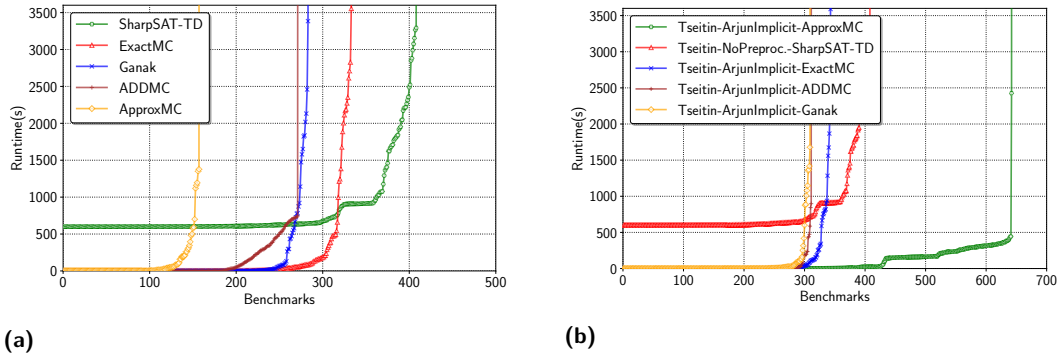
**Summary of Results.** When configured optimally, SharpSMT has the ability to solve over four times the amount of instances than SMTApproxMC. Out of a total of 679 instances, SharpSMT is able to solve 655, whereas SMTApproxMC is only able to solve 148. Furthermore, even when SharpSMT produces an exact model count, it can still solve a significantly greater number of instances than SMTApproxMC, specifically 419 instances. We found that the performance of SharpSMT is greatly impacted by the selection of the propositionalization method, preprocessing techniques, and counting tools used. The counter’s effectiveness is contingent on the combination of its components, and no individual component setting has been shown to be superior across the board.

#### 4.1 Comparison with prior state-of-the-art

We experimented with various configurations for SharpSMT and compared its performance with SMTApproxMC, the current state-of-the-art bitvector model counter. The results showed that with the configuration of (TseitinEnc-ArjunImplicit-ApproxMC), SharpSMT solved 655 instances, whereas SMTApproxMC could solve only 148 instances. Additionally, when SharpSMT provided an exact model count with a configuration of (TseitinEnc-NoPreproc-SharpSAT-TD), it managed to solve 419 instances, which is significantly more than what SMTApproxMC could handle. The cactus plot in Figure 2a illustrates the comparison between SMTApproxMC and the most effective approximate and exact counting setups of SharpSMT, presenting the number of instances on the  $x$ -axis and the time taken on the  $y$ -axis. Each line in the cactus plot corresponds to the performance of a bitvector model counter in a specific configuration. Moreover, the scatter plot in Figure 2b compares the optimal configuration of



■ **Figure 2 Comparison to prior state-of-the-art.** (a) Cactus plot illustrating the performance comparison of SMTApproxMC and SharpSMT in the best approximate and exact counting settings. (b) Scatter plot comparing performances of SMTApproxMC and SharpSMT in its best configuration.



■ **Figure 3 Comparison of Impact of CNF counters.** (a) Cactus plot showing the performance of SharpSMT with different model counters with TseitinEnc and no preprocessing. (b) Cactus plot showing the performance of SharpSMT with each of the model counters combined with the best possible propositionalization and preprocessing techniques.

SharpSMT with SMTApproxMC. The  $x$ -axis shows the best configuration for SharpSMT, while the  $y$ -axis shows SMTApproxMC. Each point  $(x, y)$  represents an instance that SharpSMT and SMTApproxMC solved in  $x$  and  $y$  seconds, respectively. The scatter plot indicates that SharpSMT can solve most instances that SMTApproxMC solves, but in significantly less time.

## 4.2 Impact of Different Components

**Impact of CNF Counters** The cactus plots in Figure 3 illustrate the performance variations of different model counters under two scenarios: (i) Figure 3a shows the results when the same propositionalization techniques and no preprocessing are applied; and (ii) Figure 3b shows the results when each model counter is combined with the best possible propositionalization and preprocessing techniques. To summarize:

- When SharpSMT uses no preprocessing, the highest performance is shown by using the model counter SharpSAT-TD, by solving 412 out of 679 benchmarks. The second best performance is shown by the model counter ExactMC, which solved 337 benchmarks. Conversely, the performance while employing other model counters such as ApproxMC, ADDMC, and Ganak was poor, with each solving less than 200 benchmarks.
- The performance of the model counters remained almost consistent when propositionalization system were varied between TseitinEnc and TechMap.
- However, significant variations in performance were observed when certain preprocessing methods were used. Specifically, performance of ApproxMC gets a huge improvement in this case. The results of these variations are presented in a later section.

**Impact of Propositionalization Techniques** The bar graphs in Figure 4 compare different propositionalization techniques. In these graphs, the  $y$ -axis displays the number of instances solved, with each bar representing the number of instances solved in a particular configuration. To evaluate the impact of different propositionalization techniques, we use two bars for each model counter, with each bar representing the performance of the counter when a specific propositionalization technique is employed. In the bar graph of Figure 4a, we compare propositionalization, when no preprocessing is used. We compare the impact of propositionalization in a similar bar plot while ArjunImplicit is used for preprocessing in Figure 4b. In summary we find the following from the graphs:

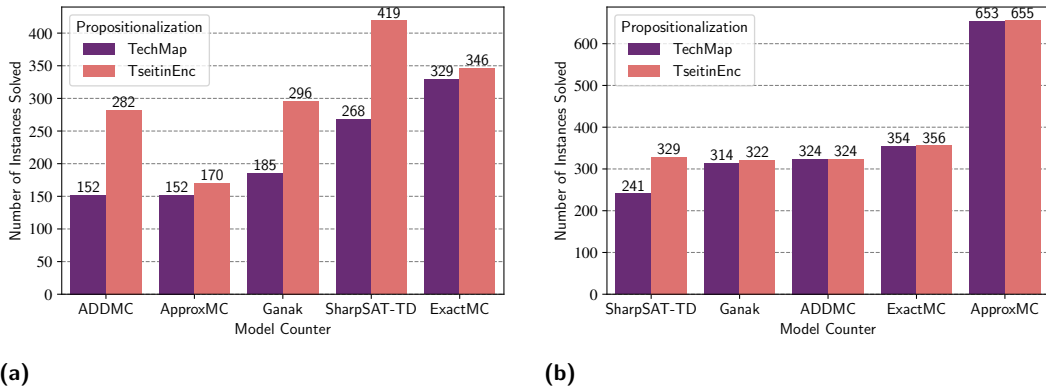


Figure 4 Comparison of propositionalization techniques. (a) Bar graph showing the impact of propositionalization with no preprocessing. (b) Bar graph showing the impact of propositionalization with ArjunImplicit preprocessing.

- According to the results presented in Figure 4a, using TseitinEnc as a propositionalization technique instead of TechMap can improve the performance of SharpSMT when no preprocessing is applied. This improvement is observed across all CNF-counters.
- On the other hand, Figure 4b reveals that if ArjunImplicit is employed as a preprocessing technique, the performance of SharpSMT remains relatively unchanged when the propositionalization technique is switched.

**Impact of Preprocessing** Figure 5 compares preprocessing techniques. In bar graphs of Figure 5a, we compare the impact of preprocessing while TseitinEnc was used as propositionalization techniques. The scatter plot in Figure 5b, we show the impact of varying the preprocessing technique between Arjun and ArjunImplicit, while ApproxMC is the CNF-counter. To summarize the results of preprocessing:

- The preprocessors had very little significant positive impact on the performance of exact

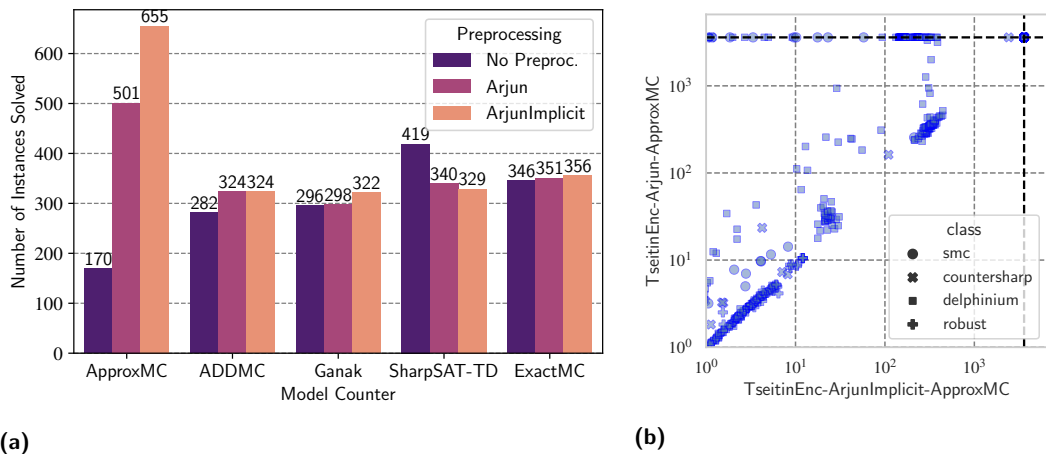


Figure 5 Comparison of preprocessing techniques. (a) Bar graph showing the impact of preprocessing after TseitinEnc. and (b) Cactus plot comparing performance of SharpSMT with two different preprocessing.

model counting tools, and in a few cases, they even showed a slight negative impact.

- For the approximate model counting tool **ApproxMC**, **Arjun** demonstrated a positive impact, enabling the solving of 501 instances, which is 331 more than no preprocessing.
- The impact of preprocessing technique on **ApproxMC** was even greater when **ArjunImplicit** was used. In this case **SharpSMT** could solve 655 and 653 instances respectively, in cases of **TseitinEnc** and **TechMap** propositionalization respectively.

**Optimal Configuration** Based on the findings above, the best performing settings are:

1. **Approximate Count:** **SharpSMT** with configuration of (**TseitinEnc-ArjunImplicit-ApproxMC**).
2. **Exact Count:** **SharpSMT** with configuration of (**TseitinEnc-NoPreproc-SharpSAT-TD**).

### 4.3 Benchmark-wise Variation

Performance of **SharpSMT** varied across different benchmark classes as configurations for the component changes. Our benchmark suite consists of four different classes: **delphinium**, **counterssharp**, **robust**, and **smc**. The results of the bitvector model counters show significant variation across the benchmarks. It should be noted that the previous state-of-the-art component, **SMTApproxMC**, was only able to solve instances from two benchmark categories (**smc** and **counterssharp**), and failed to solve any instances from **robust** or **delphinium**. When no preprocessing is used, the performance of different configurations varies significantly across benchmarks. For example, the **SharpSMT** performance with the **TseitinEnc-NoPreproc-ApproxMC** configuration is similar to that of **SMTApproxMC**, with most of the solved instances coming from **smc** and **counterssharp**. However, the situation changes when preprocessing techniques are employed. In the best configurations, **SharpSMT** is capable of solving nearly all instances across all benchmark sets.

## 5 Conclusion

In this paper, we propose a framework called **SharpSMT** that counts bitvector formulas using various propositionalization methods, preprocessing techniques, and CNF-counters. By leveraging **SharpSMT**, we investigate different strategies for counting bitvector formulas and achieve a significant performance improvement of four times the current state-of-the-art. Our framework provides a valuable starting point *towards building a model counter* capable of directly *reasoning over bitvectors*. **SharpSMT** identifies which methods are effective in counting bitvector instances and sets a new performance benchmark for bitvector model counters. We believe that our work lays the foundation for future research on bitvector reasoning without the need for propositionalization. Our next goal is to develop a bitvector model counter that can reason over bitvectors and solve more instances.



---

**References**

---

- 1 Gabrielle Beck, Maximilian Zinkus, and Matthew Green. Automating the development of chosen ciphertext attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1821–1837, 2020.
- 2 Supratik Chakraborty, Kuldeep Meel, Rakesh Mistry, and Moshe Vardi. Approximate probabilistic inference via word-level counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.
- 3 Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. A scalable approximate model counter. In *Proc. of International Conference on Principles and Practice of Constraint Programming*, pages 200–216. Springer, 2013.
- 4 Supratik Chakraborty, Kuldeep S Meel, and Moshe Y Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic sat calls. In *IJCAI*, pages 3569–3576, 2016.
- 5 Dmitry Chistikov, Rayna Dimitrova, and Rupak Majumdar. Approximate counting in smt and value estimation for probabilistic programs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 320–334. Springer, 2015.
- 6 Jeffrey M. Dudek, Vu H.N. Phan, and Moshe Y. Vardi. ADDMC: Weighted model counting with algebraic decision diagrams. *AAAI 2020 - 34th AAAI Conference on Artificial Intelligence*, pages 1468–1476, 2020. [arXiv:1907.05000](https://arxiv.org/abs/1907.05000), [doi:10.1609/aaai.v34i02.5505](https://doi.org/10.1609/aaai.v34i02.5505).
- 7 Niklas Eén, Alan Mishchenko, and Niklas Sörensson. Applying logic synthesis for speeding up sat. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 272–286. Springer, 2007.
- 8 Cunjing Ge and Armin Biere. Decomposition strategies to count integer solutions over linear constraints. In *Proc. of IJCAI*, pages 1389–1395, 2021.
- 9 Cunjing Ge and Feifei Ma. A fast and practical method to estimate volumes of convex polytopes. In *Frontiers in Algorithmics: 9th International Workshop, FAW 2015, Guilin, China, July 3-5, 2015, Proceedings*, volume 9130, page 52. Springer, 2015.
- 10 Cunjing Ge, Feifei Ma, Tian Liu, Jian Zhang, and Xutong Ma. A new probabilistic algorithm for approximate model counting. In *International Joint Conference on Automated Reasoning*, pages 312–328. Springer, 2018.
- 11 Cunjing Ge, Feifei Ma, Xutong Ma, Fan Zhang, Pei Huang, and Jian Zhang. Approximating integer solution counting via space quantification for linear constraints. In *Proc. of IJCAI*, pages 1697–1703, 2019.
- 12 Cunjing Ge, Feifei Ma, and Jian Zhang. Volce: An efficient tool for solving #smt(la) problems. In *Proceedings of the Second Workshop on Logics for Reasoning about Preferences, Uncertainty, and Vagueness, PRUV@IJCAR 2018*, volume 2157 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018.
- 13 Cunjing Ge, Feifei Ma, Peng Zhang, and Jian Zhang. Computing and estimating the volume of the solution space of smt (la) constraints. *Theoretical Computer Science*, 743:110–129, 2018.
- 14 Guillaume Girol, Benjamin Farinier, and Sébastien Bardin. Not all bugs are created equal, but robust reachability can tell the difference. In *International Conference on Computer Aided Verification*, pages 669–693. Springer, 2021.
- 15 Susmit Jha, Rhishikesh Limaye, and Sanjit A Seshia. Beaver: Engineering an efficient smt solver for bit-vector arithmetic. In *Computer Aided Verification: 21st International Conference, CAV 2009, Grenoble, France, June 26-July 2, 2009. Proceedings 21*, pages 668–674. Springer, 2009.
- 16 Seonmo Kim and Stephen McCamant. Bit-vector model counting using statistical estimation. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 133–151. Springer, 2018.
- 17 Seonmo Kim and Stephen McCamant. Structural bit-vector model counting. In *SMT*, pages 26–36, 2020.

- 18 Tuukka Korhonen and Matti Järvisalo. Integrating tree decompositions into decision heuristics of propositional model counters. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- 19 Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Improving model counting by leveraging definability. In *Proc. of IJCAI*, pages 751–757, 2016.
- 20 Jean-Marie Lagniez, Emmanuel Lonca, and Pierre Marquis. Definability for model counting. *Artificial Intelligence*, 281:103229, 2020.
- 21 Yong Lai, Kuldeep S Meel, and Roland HC Yap. The power of literal equivalence in model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 3851–3859, 2021.
- 22 Jean-Philippe Martin. Upper and lower bounds on the number of solutions. *Technical Report MSR-TR-2007-164, Microsoft Research*, 2007.
- 23 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S Meel. Ganak: A scalable probabilistic exact model counter. In *IJCAI*, volume 19, pages 1169–1176, 2019.
- 24 Mate Soos and Kuldeep S Meel. Bird: engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1592–1599, 2019.
- 25 Mate Soos and Kuldeep S. Meel. Arjun: An efficient independent support computation technique and its applications to counting and sampling. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 11 2022.
- 26 Samuel Teuber and Alexander Weigl. Quantifying software reliability via model-counting. In Alessandro Abate and Andrea Marin, editors, *Proc. of Quantitative Evaluation of Systems*, pages 59–79, Cham, 2021. Springer International Publishing.
- 27 Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.