GANAK: A Scalable Probabilistic Exact Model Counter*

Shubham Sharma[1], **Subhajit Roy**[1], **Mate Soos**[2] and **Kuldeep S. Meel**[2]

[1]Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, India
[2]School of Computing, National University of Singapore

Abstract

Given a Boolean formula F, the problem of model counting, also referred to as #SAT, seeks to compute the number of solutions of F. Model counting is a fundamental problem with a wide variety of applications ranging from planning, quantified information flow to probabilistic reasoning and the like. The modern #SAT solvers tend to be either based on static decomposition, dynamic decomposition, or a hybrid of the two. Despite dynamic decomposition based #SAT solvers sharing much of their architecture with SAT solvers, the core design and heuristics of dynamic decomposition-based #SAT solvers has remained constant for over a decade. In this paper, we revisit the architecture of the state-of-the-art dynamic decomposition-based #SAT tool, sharpSAT, and demonstrate that by introducing a new notion of probabilistic component caching and the usage of universal hashing for exact model counting along with the development of several new heuristics can lead to significant performance improvement over state-of-the-art model-counters. In particular, we develop GANAK, a new scalable probabilistic exact model counter that outperforms state-of-the-art exact and approximate model counters sharpSAT and ApproxMC3 respectively, both in terms of PAR-2 score and the number of instances solved. Furthermore, in our experiments, the model count returned by GANAK was equal to the exact model count for all the benchmarks. Finally, we observe that recently proposed preprocessing techniques for model counting benefit exact model counters while hurting the performance of approximate model counters.

1 Introduction

Given a Boolean formula F, the problem of propositional model counting, also referred to as #SAT, seeks to compute the number of solutions of F. Model counting is a fundamental problem with a wide variety of applications ranging from quantified information flow, network reliability, planning, probabilistic reasoning, and the like [Roth, 1996; Bacchus *et al.*, 2003; Domshlak and Hoffmann, 2007; Gomes *et al.*, 2007; Meel *et al.*, 2016; Dueñas-Osorio *et al.*, 2017; Biondi *et al.*, 2018]. For example, given a graph G such that each of its edges fails with some probability and two nodes, s and t, the problem of computing probability of existence of a path from s to t can be reduced to that of propositional model counting [Dueñas-Osorio *et al.*, 2017].

In his seminal paper, Valiant showed that #SAT is #P-complete, where #P is the set of counting problems associated with NP decision problems [Valiant, 1979]. Theoretical investigations of #P have led to the discovery of deep connections in complexity theory, and there is strong evidence for its hardness. In particular, Toda proved that every problem in the polynomial hierarchy could be solved by just one call to a #P oracle; more formally, $\mathsf{PH} \subseteq \mathsf{P}^{\#\mathsf{P}}$ [Toda, 1989].

The earliest efforts to #SAT focused on extending the Davis-Putnam-Loveland-Longemann (DPLL) procedure [Davis and Putnam, 1960] by incrementally computing the number of solutions and adding appropriate multiplicative factors after a partial solution was found [Birnbaum and Lozinskii, 1999]. Subsequently, Relsat focused on partitioning the formula into components with a disjoint set of variables. In a significant breakthrough, Sang et al. pioneered the idea of component caching combined with Conflict Driven Clause Learning (CDCL) architecture in their exact counter Cachet [Sang *et al.*, 2004; Sang *et al.*, 2005b]. Thurley [Thurley, 2006] improved upon Cachet's component caching scheme along with tighter engineering integration and developed sharpSAT. Several knowledge compilation-based counters, often a hybrid of static and dynamic decomposition, have been proposed over the past few years along with novel techniques for preprocessing [Lagniez and Marquis, 2014; Lagniez *et al.*, 2016; Lagniez and Marquis, 2017].

Despite significant progress in model counting over the years, the core components of the architecture of dynamic decomposition based techniques have remained constant. Furthermore, SAT solving have witnessed significant improvements over the past decade owing to the development of new heuristics [Marques-Silva and Sakallah, 1999; Moskewicz *et al.*, 2001; Eén and Sörensson, 2004]. Moreover, recent years have witnessed the rise of approximate model counters owing to the combination of hashing-based frameworks and use of

*The open source tool along with benchmarks is available at https://github.com/meelgroup/ganak

independent support [Stockmeyer, 1983; Gomes *et al.*, 2007; Chakraborty *et al.*, 2013; Chakraborty *et al.*, 2016; Soos and Meel, 2019]. In this context, we revisit the architecture of the state-of-the-art exact model counter, sharpSAT, and seek to redesign the architecture and augment the existing techniques with new heuristics.

The primary contribution of this paper is a novel architecture, called GANAK[1], that deviates significantly from sharpSAT as follows:

1. We investigate the usage of universal hash functions for exact model counting. To this end, we design, to the best of our knowledge, the first probabilistic component cache scheme and the first probabilistic exact model counter. In particular, GANAK takes in a formula F and a confidence parameter δ as input and returns `count` such that `count` is the number of solutions of F with confidence at least $1-\delta$. Note that probabilistic exact model counting is almost as hard as exact model counting and significantly hard compared to probabilistic approximate model counting [Chakraborty *et al.*, 2019].
2. We propose new branching heuristic that seek to achieve the best of both worlds: perform branching on variables so as to maximize cache hits and perform branching on variables that lead to conflict as soon as possible.

Moreover, we propose new heuristics: *phase selection heuristic*, *independent support*, *exponentially decaying randomness* and *learn and start over*, and perform extensive experiments to study the effect of these heuristics, in isolation and in combination. Finally, we use our experience from the above study to build GANAK that inherits current advancements in SAT solving and model counting, improves upon them and contributes new ideas, thereby outperforming state-of-the-art model counters. In particular, GANAK outperforms state-of-the-art exact and approximate model counters sharpSAT and ApproxMC3 respectively, both in terms of PAR-2[2] score and the number of instances solved. Moreover, in our experiments, the model count returned by GANAK was equal to the exact model count for all the benchmarks.

The rest of the paper is organized as follows: We introduce notations and preliminaries in Section 2, discuss related work in Section 3, present GANAK in Section 4 and perform the theoretical analysis of GANAK in Section 5. We describe the experimental methodology and discuss results in Section 6 and then we finally conclude in Section 7.

2 Notations and Preliminaries

Let F be a boolean formula in conjunctive normal form (CNF), and let $\mathsf{Vars}(F)$ and $\mathsf{Cl}(F)$ be the set of variables and clauses appearing in F. The set $\mathsf{Vars}(F)$ is called the support of F. An assignment σ of truth values to the variables in $\mathsf{Vars}(F)$ is called a satisfying assignment or witness of F if it makes F evaluate to true.

[1] GANAK (गणक in Sanskrit) refers to a device that counts.

[2] PAR-2 scheme, that is, penalized average runtime, used in SAT-2017 Competition [SAT, 2017], assigns a runtime of two times the time limit (instead of a "not solved" status) for each benchmark not solved by a solver.

Model counting. Given a formula F, we denote the set of all witnesses of F as R_F; we refer to the cardinality of R_F as the *model count* of F (denoted as $|R_F|$). Given a set of variables $S \subseteq \mathsf{Vars}(F)$ and a witness σ, we write $\sigma_{\downarrow S}$ for the projection of σ on S. We denote the set of all projections of R_F on S as $R_{F\downarrow S}$; we refer to the cardinality of $R_{F\downarrow S}$ as the *projected model count* of F on S (denoted as $|R_{F\downarrow S}|$). Given F, the problem of *exact model counting* is to compute $|R_F|$. Given F and $\delta \in (0,1]$, *probabilistic exact model counting* estimates `count` and guarantees that $\mathsf{Pr}\big[|R_F| = \texttt{count}\big] \geq 1-\delta$. A recent study of different relaxations of model counting shows that probabilistic exact model counting is almost as hard as exact model counting [Chakraborty *et al.*, 2019]. Given F, a tolerance $\epsilon > 0$ and a confidence $1-\delta \in (0,1]$, *approximate model counting* estimates `count` and guarantees that $\mathsf{Pr}\big[|R_F|/(1+\epsilon) \leq \texttt{count} \leq (1+\epsilon)|R_F|\big] \geq 1-\delta$. The above definitions also apply for projected model counting ($|R_{F\downarrow S}|$).

Component caching. Let $var_{id}(F)$ and $cl_{id}(F)$ be the corresponding sets of indices of $\mathsf{Vars}(F)$ and $\mathsf{Cl}(F)$. A CNF formula and components can be encoded as strings, omitting satisfied clauses and assigned literals. As given in [Thurley, 2006], let standard encoding (STD) be the encoding such that the literals in a clause are presented by their respective labels and clauses are separated by '0'. Let, for example $F = (\overline{x_3} \vee \overline{x_5} \vee x_6) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_6}) \wedge (x_2 \vee x_3 \vee x_6)$ then its STD encoding will be (-3, -5, 6, 0, -1, 4, -6, 0, 2, 3, 6). Let hybrid encoding (HC) be the encoding such that a component is encoded by writing the increasing order of the set of indices of associated variables and clauses into a set of strings. For F, $var_{id}(F) = \{1,2,3,4,5,6\}$ and $cl_{id}(F) = \{1,2,3\}$ so HC encoding of F will be $(1,2,3,4,5,6,1,2,3)$. We will use $\mathsf{HC}(\cdot)$ and $\mathsf{STD}(\cdot)$ to denote the Hybrid and Standard encodings of a component.

Hash functions. For positive integers n and m, let $H(n,m)$ denote the family of hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$. We use $h \xleftarrow{R} H(n,m)$ to denote the probability space obtained by choosing a hash function h uniformly at random from $H(n,m)$. We say that $H(n,m)$ is an universal family if for distinct $y_1, y_2 \in \{0,1\}^n$, we have $\mathsf{Pr}[h(y_1) = h(y_2) : h \xleftarrow{R} H(n,m)] \leq \frac{1}{2^m}$. Similarly, $H(n,m)$ is $\varepsilon-$universal if for distinct $y_1, y_2 \in \{0,1\}^n$, we have $\mathsf{Pr}[h(y_1) = h(y_2) : h \xleftarrow{R} H(n,m)] \leq \varepsilon$. In this paper, we use a special class of $\varepsilon-$universal hash family, called `CLHash`, owing to its efficient implementation in C++ for modern hardware [Lemire and Kaser, 2016]. Originally, `CLHash` was proposed for $m = 64$ with $\varepsilon = \frac{2.004}{2^{64}}$ but one can extend this for arbitrary m (as long as m is a multiple of 64) by defining a new hash family as a concatenation of multiple hash functions from `CLHash`. We denote the generalized family as $H_{cl}(n,m)$, which provides the following formal guarantee: for distinct $y_1, y_2 \in \{0,1\}^n$, we have

$$\mathsf{Pr}\left[h(y_1) = h(y_2) : h \xleftarrow{R} H_{cl}(n,m)\right] \leq \frac{2.004^{\frac{m}{64}}}{2^m} \qquad (1)$$

3 Related Work

Complexity-theoretic studies on propositional model counting were initiated by Valiant, who showed that the problem is #P-complete [Valiant, 1979]. Birnbaum and Lozinskii [Birnbaum and Lozinskii, 1999] introduced CDP which incrementally counted the number of solutions by introducing the multiplication factors for each partial solution found, eventually covering the entire solution space. Bayardo and Pehoushek [Bayardo and Pehoushek, 2000] presented Relsat in which they recursively identified the connected components within a dynamically specified constraint graph, each subproblem corresponding to a component is, then, solved recursively. The solution counts of each connected components are finally multiplied together to obtain the solution count for the given formula. Sang et al. [Sang *et al.*, 2004; Sang *et al.*, 2005a] used component caching, clause learning and other new heuristics suitable for #SAT in their counter Cachet. Thurley [Thurley, 2006] introduced new cache management schema in sharpSAT in which a new approach of encoding components is used to improve cache utilization.

Techniques based on BDDs and their variants [Minato, 1993], d-DNNF representation [Darwiche, 2004; Muise *et al.*, 2012; Lagniez and Marquis, 2017] and treewidth of graph representation of formula [Samer and Szeider, 2007] have also been used to compute exact model counts. Recently hashing based techniques have been used to perform approximate model counting [Stockmeyer, 1983; Gomes *et al.*, 2007; Chakraborty *et al.*, 2013; Chakraborty *et al.*, 2016; Soos and Meel, 2019]. The core idea of hashing-based frameworks is to employ 2-universal hash functions to partition the solution space into roughly equal small cells, wherein a cell is called small if the number of the solutions in the cell is less than or equal to an appropriately computed threshold, denoted by `thresh`; a SAT solver is then employed to check if a cell is small by enumerating solutions until either there are no more solutions or `thresh` + 1 solutions have been enumerated. Then the model count is approximated by using the number of cells and number of solutions found in a cell.

4 GANAK

GANAK inherits the strength of a state-of-the-art model counter, sharpSAT, and is equipped with the following new algorithmic advances:

1. PCC: Probabilistic component caching
2. CSVSADS: New variable branching heuristic
3. PC: New phase selection heuristic
4. IS: Independent support heuristic
5. EDR: Exponentially decaying randomness heuristic
6. LSO: Learn and start over heuristic

We first describe the core algorithm and then delve into a detailed discussion on each of the above heuristics.

4.1 Core Algorithm

GANAK (Algorithm 1) takes a CNF formula F and a confidence parameter $\delta \in (0, 1]$ to return the estimate of $|R_F|$ with confidence at least $1 - \delta$. GANAK uses ϵ-universal hash family, $H_{cl}(n, m)$, with $n = \mathsf{Vars}(F)$ and $m = 64$ (line 2). It, then, invokes the subroutine Counter that takes the formula F and a confidence parameter δ, returning the estimate of $|R_F|$ if the range of the hash family is sufficient to establish the probabilistic bounds; otherwise, it returns -1. The algorithm loops on Counter, doubling the range of the hash family in every iteration, till Counter returns successfully (lines 4–7).

The subroutine Counter (Algorithm 2) is a recursive procedure that attempts to compute an estimate of $|R_F|$, assuming an access to a hash family $H_{cl}(n, m)$. The key idea behind the probabilistic component cache is to use a hash family of sufficient range such that the probability of a collision in the entire execution of the subroutine, Counter, is at most δ. To this end, Counter uses the subroutine NumComponents() to compute the total number of components cached so far. Counter returns -1 if the number of components is too large for the probability of collision to be at most δ (lines 3–4).

Subsequently, in line 5, Counter chooses a literal l from the set of unassigned literals to branch on using a variable branching heuristic and phase selection heuristic, and then, counts the number of solutions for $F_{|l}$ and $F_{|\neg l}$, where $F_{|l}$ and $F_{|\neg l}$ are residual formulas obtained by assigning l to `true` and `false` respectively. Each subproblem, $F_{|l}$ and $F_{|\neg l}$ make use of component caching to achieve runtime effi-

Algorithm 1 GANAK(F, δ)

```
1: m ← 64
2: SetHash(Vars(F), m, 2)
3: count ← Counter(F, δ)
4: while count = −1 do
5:     m ← 2 × m
6:     SetHash(Vars(F), m, 2)
7:     count ← Counter(F, δ)
8: return count
```

Algorithm 2 Counter(F, δ)

```
1: m ← GetHashRange()
2: t ← NumComponents()
3: if 2 log2(t) > log2(δ) + m(1 − log2(2.004)/64) then
4:     return −1
5: l ← DecideLiteral(F)
6: for lit ← {l, ¬l} do
7:     F_|lit ← UnitPropagation(F, lit)
8:     if F_|lit contains an empty clause then
9:         count[lit] ← 0                          ▷ CDCL is done
10:    else
11:        count[lit] ← 1
12:        Comps ← DisjointComponents(F_|lit)
13:        for C ← Comps do
14:            hash ← h(HC(C))
15:            count ← GetCache(hash)
16:            if count = NOT FOUND then
17:                count ← Counter(C, δ)
18:            count[lit] = count[lit] × count
19:            if count = 0 then
20:                break
21: hash ← h(HC(F))
22: CacheStore(hash, count[l] + count[¬l])
23: return count[l] + count[¬l]
```

ciency. In particular, a subproblem is broken into components that are disjoint sets of clauses such that no two components share a variable. Then, each component is solved independently (lines 13–20) and the model count of a subproblem ($F_{|l}$ or $F_{|\neg l}$) is calculated as a product of model counts of each of its components. For each component, first the cache is examined to see if this component was encountered and solved earlier in the search (line 15), in which case the count is simply fetched from the cache; otherwise, the component is recursively solved. The count of a component is achieved via a running product of count of each of its disjoint components (line 18). If any component C has a model count of 0, then the model count of subproblem will be 0, in which case Counter skips the remaining components of that subproblem (lines 19–20). Finally, the model count of F which is the sum of model counts of F_l and $F_{\neg l}$ is returned after storing it into the cache (lines 21–23).

4.2 Probabilistic Component Caching

sharpSAT uses the HC schema [Thurley, 2006] to encode the component in the string to reduce the space over previous proposals [Sang *et al.*, 2004]. However, this encoding still consumes significant memory to store the codes.

GANAK employs *Probabilistic Component Caching (*PCC*)* to further reduce the space required to store the codes, thereby increasing the effective cache capacity with a small (bounded) probability of returning incorrect model counts. GANAK calculates an m-bit hash of HC encoding of a component using the family $H_{cl}(n, m)$ and store this hash with the corresponding model-count of that component in the cache.

4.3 Variable Branching Heuristic

The performance of CDCL-based SAT and #SAT solvers is largely dependent on variable branching heuristics. Modern SAT and #SAT solvers employ the popular VSADS score, that combines the merits of *Dynamic Largest Combined Sum* (DLCS) and *Variable State Independent Decaying Sum* (VSIDS) [Sang *et al.*, 2005a]; VSADS acts more like DLCS when there are fewer conflicts and more like VSIDS when there are more conflicts.

The enhanced cache capacity that we are able to buy out of PCC makes it imperative to incorporate the cache state of the solver as a guide for branching decisions in order to get better cache utilization. Therefore, we design a new heuristic, *Cache State and Variable State Aware Decaying Sum* (CSVSADS), that combines the cache state of the solver with the power of VSADS. Our heuristic is based on the following observation: whenever we branch on a variable, v_i, the subsequently generated components under that decision cannot contain v_i. CSVSADS attempts to improve the cache hit-rate by discouraging branching on variables whose components are recently added to the cache.

We define two parameters to guide our branching heuristic:

- **CacheScore**: The CacheScore prioritizes variables whose components were not recently added to the cache. Whenever a cache-hit or cache-store is encountered, the CacheScore (initialized to zero) of all the variables that occur in that component is decremented. Further, the CacheScore of all the variables is periodically incremented by a constant factor.
- **heparam**: This establishes a *heuristic equivalence* parameter for VSADS, allowing us to select variables that excel on both of their CacheScore and VSADS score. heparam shrinks dynamically as the cache gets filled (and vice-versa) via a user-defined parameter α, i.e. heparam $= \alpha \times \%$CacheFull, where %CacheFull is the currently occupied component cache with respect to the maximum allowable component cache size (in percentage). GANAK uses $\alpha = 0.1$ (selected empirically).

CSVSADS sorts the variables by their VSADS score and selects a variable that has the highest CacheScore among the variables that lie in the top heparam percent.

4.4 Phase Selection Heuristic

Modern #SAT solvers have successfully used the DLIS scheme for phase selection [Sang *et al.*, 2004; Thurley, 2006]. For each variable v, DLIS maintains the number of occurrences of positive ($|v|$) and negative ($|\neg v|$) literals in the formula; DLIS performs phase selection in accordance to the frequency of the positive or negative literals ($\max(|v|, |\neg v|)$).

Our heuristic attempts to use randomization to *weaken* the DLIS heuristic in situations where DLIS is not overwhelmingly assertive about the polarity of a variable. We maintain a cache, PolarityCache (similar to [Pipatsrisawat and Darwiche, 2007]), that records the variable polarities which have been assigned values in the past. Any time the solver branches on a variable, we select a phase as follows:

- Given a variable v_i, if the DLIS score in favor of one of the polarities (v_i or $\neg v_i$) is not overwhelmingly high[3], and v_i is present in the PolarityCache, we reduce our "trust" on the DLIS score by selecting a phase uniformly at random from {DLIS(v_i), `true`, `false`}; DLIS(v_i) returns the polarity from DLIS for the variable v_i.
- Otherwise, we use the default DLIS heuristic.

4.5 Independent Support

An *independent support*, $\mathcal{I} \subseteq \mathsf{Vars}(F)$, is a subset of the support such that, for $\sigma_1, \sigma_2 \in R_F$, if $\sigma_{1\downarrow\mathcal{I}} = \sigma_{2\downarrow\mathcal{I}}$ then $\sigma_1 = \sigma_2$ [Chakraborty *et al.*, 2014]. In other words, the truth values of variables in $\mathcal{I}$ uniquely determine the truth value of every variable in $\mathsf{Vars}(F) \setminus \mathcal{I}$ in every satisfying assignment. Note that $|R_F| = |R_{F\downarrow\mathcal{I}}|$. Now, if the variables in the Independent Support ($\mathcal{I}$) are selected as the decision variables, the residual formula has only two possibilities: either the formula is satisfiable (SAT) with only one satisfying assignment, or the formula is unsatisfiable (UNSAT).

GANAK leverages $\mathcal{I}$ in the following manner: if the residual formula is SAT, we set a model count to one at the last decision variable in the independent support; otherwise, we set the model count to zero. We, then, simply backtrack to the appropriate decision level.

[3] we consider a value *overwhelmingly high* if the frequency of the positive literal of v_i exceeds that of the negative literal by a factor of two (or vice versa).

If the size of the independent support is small, we need fewer decisions to reach the residual formula. Hence, it is desirable to apply the above algorithm on the *minimum* independent support. As computing the minimum independent support is overly expensive, we use the MIS [Ivrii *et al.*, 2015] algorithm for computing the *minimal* independent support. We apply the following techniques to further bring down the cost:

- We apply the above algorithm only on *hard* instances. We consider an instance hard if it encounters less than 500 conflicts while taking 5 million decisions;
- Since MIS is an *anytime* algorithm, i.e it always returns a sound independent support ($\mathcal{I}$) for a given time budget, we ran MIS with a time out of 100 seconds (determined empirically).

We found that independent support is an extremely strong heuristic (see section 6).

4.6 Exponentially Decaying Randomness

Randomization is quite popular in SAT solvers in being able to find easily searchable portions of the search space [Gomes *et al.*, 1998]. Previous studies [Sang *et al.*, 2005a] have shown that the aggressive usage of randomization is not suitable for #SAT as the entire solution space has to be searched. The question of whether randomization can be helpful during initial search process, however, was still unresolved. To investigate further, we design a new heuristic, called EDR, that exponentially decays randomness as the solver progresses through the problem. Our heuristic introduces an *exponentially increasing heuristic equivalence parameter* eiheparam, such that: $\mathsf{eiheparam} = 100 - 10e^{(-0.0001 \times \#Decisions)}$. EDR sorts the variables by their VSADS score and selects a variable randomly from the top eiheparam percentage. To our mild surprise, we found that EDR severely degrades runtime performance and perhaps, one can argue that no form of randomization is suitable for #SAT (see section 6).

4.7 Learn and Start Over

Modern SAT solvers use random restarts [Gomes *et al.*, 1998] aggressively in search of a *good* variable ordering that can quickly lead to a satisfiable assignment. It is, however, not clear if restarts can be beneficial for model counting as #SAT solvers have to traverse the complete solution space; moreover, the dismal performance of EDR further seems to discourage randomization, and thus the very idea of restarts. Surprisingly, we found that restarts indeed help #SAT in many instances. Our new technique, *Learn and Start Over* (LSO), adapts random restarts for #SAT as follows: firstly, instead of invoking restarts multiple times, we invoke it only once after the first 5000 decisions. Secondly, we learn from the previous invocation by maintaining all the scores obtained in the previous run—VSADS score, CacheScore, and the complete state of the component cache and the PolarityCache to explore different and better ordering of decision variables.

5 Theoretical Analysis

Theorem 1. *Given a propositional formula F, confidence parameter $\delta \in (0, 1]$ and a family of hash functions $H_{cl}(n, m)$ suppose* GANAK(F, δ) *returns* `count` *then* $\mathsf{Pr}\left[|R_F| = \texttt{count}\right] \geq 1 - \delta$.

Proof. Let $C = \{c_1, \cdots, c_t\}$ be the set of total number of components generated in the run of GANAK(F, δ), and $t = |C|$. We define a relation $\prec$ such that for $c_i, c_j \in C, c_i \prec c_j$ if c_i is cached before c_j in a run of GANAK(F, δ). Let us consider pairs, $\hat{C} = \{(c_i, c_j) \mid c_i, c_j \in C, c_i \prec c_j\}$. Now, given $(c_i, c_j) \in \hat{C}$ and $h \xleftarrow{R} H_{cl}(n, m)$, let A_{ij} represent a *collision* between two components c_i and c_j under $h(\mathsf{HC}(\cdot))$, i.e. $c_i \neq c_j$ and $h(\mathsf{HC}(c_i)) = h(\mathsf{HC}(c_j))$. Let A be the event that there is at least one *collision* in a run of GANAK i.e. $A = \bigcup_{(c_i,c_j)\in\hat{C}} A_{ij}$. Hence, $\bar{A}$ represents the event such that the `count` returned by GANAK(F, δ) is equal to $|R_F|$.

$$\text{Now,} \quad \mathsf{Pr}[A] = \mathsf{Pr}[\cup_{(c_i,c_j)\in\hat{C}} A_{ij}] \leq \sum_{(c_i,c_j)\in\hat{C}} \mathsf{Pr}[A_{ij}]$$

$$\text{Hence,} \quad \mathsf{Pr}[A] \leq \binom{t}{2} \frac{2.004^{\frac{m}{64}}}{2^m} \qquad \text{(by Equation 1)}$$

$$\text{So,} \quad \mathsf{Pr}[\bar{A}] \geq 1 - \frac{t^2 2.004^{\frac{m}{64}}}{2^m}$$

From Algorithm 1 and 2, $2\log_2(t) - m\left(1 - \frac{\log_2(2.004)}{64}\right) \leq \log_2(\delta)$. So, $\log_2\left(\frac{t^2 2.004^{\frac{m}{64}}}{2^m}\right) \leq \log_2(\delta) \implies \frac{t^2 2.004^{\frac{m}{64}}}{2^m} \leq \delta$. Therefore, $1 - \frac{t^2 2.004^{\frac{m}{64}}}{2^m} \geq 1 - \delta \implies \mathsf{Pr}[\bar{A}] \geq 1 - \delta$ □

6 Evaluation

We evaluate the runtime performance of GANAK on 2031 publicly available benchmarks arising from a wide range of applications of model counting such as probabilistic reasoning, plan recognition, DQMR networks, ISCAS89 combinatorial circuits, quantified information flow, logistics, and the like as have been previously employed in studies on model counting [Brglez *et al.*, 1989; Sang *et al.*, 2005b; Chakraborty *et al.*, 2016; Lagniez and Marquis, 2017]. All our experiments were conducted on a high-performance computer cluster, with each node consists of E5-2690 v3 CPU with 24 cores and 96GB of RAM. We used 24 cores per node with memory limit set to 4GB per core and all individual instances for each tool were executed on a single core. For our experiments, we set $\delta = 0.05$ (for GANAK), a maximum component cache size of 2GB (for sharpSAT and GANAK) and a timeout of 5000 seconds (for all the tools). All other parameters are set to their default values.

We attempt to answer the following research questions:

1. What is the impact of different configurations of our heuristics?
2. How does GANAK perform with respect to the state-of-the-art exact and approximate model counters for model counting ($|R_F|$) and projected model counting ($|R_{F\downarrow S}|$)?
3. Empirically, in what percentage of the benchmarks, the model-count returned by GANAK deviates from the exact model-count?

Our experiments demonstrate that GANAK performs best when all the heuristics (except EDR) are enabled. GANAK outperforms state-of-the-art exact (sharpSAT) and approximate (ApproxMC3) model counters, both in terms of PAR-2 score and the number of instances solved. Finally, in our experiments, the model count returned by GANAK was equal to the exact model count *for all the benchmarks*.

6.1 Impact of Configurations of Heuristics

In order to better understand the performance of GANAK, we investigate the impact of different configurations of heuristics. Our preliminary experiments demonstrated that the usage of EDR severely downgrades the performance of GANAK, and therefore, we eliminated EDR from further analysis. We performed exhaustive experiments by turning each of the remaining heuristics—PCC, CSVSADS, PC, IS and LSO—on or off independently on 1650 benchmarks.

Figure 1a shows a heat-map on PAR-2 scores for each configuration of heuristics; a lower PAR-2 score, i.e. a tilt towards the red end of the spectrum, exhibits favorable configurations. Figure 1b represent the comparison between GANAK versus the baseline configuration that has all the heuristics turned off: for each configuration, the numerator indicates the number of benchmarks on which the respective configuration performs better than the baseline (vice-versa for the denominator); we only report results on the benchmarks that take at least 1 second to run and that deviate by at least 10% by runtime across the respective configurations.

Figure 1 illustrate that GANAK works best when all the heuristics are turned on. Further analysis reveals that IS is the strongest heuristic; PCC, i.e. enabling probabilistic component caching, comes a close second. Interestingly, PCC starts performing well only when IS is turned on, showing a strong synergy between these two heuristics. We believe that the reason behind this is that IS restricts the decision variables to a small, well-chosen set, thereby helping the solver achieve better utilization for the cached components.

The spectrum shift on the top-right eight cells illustrates that CSVSADS starts improving the performance of the solver in presence of PCC and IS; as CSVSADS aims to improve cache utilization, this trend is understandable. Finally, the shift in the spectrum towards the red end as we move top and right, exhibits the effectiveness of PC and LSO. Though LSO does not seem to have much effect on the PAR-2 score, Figure 1b demonstrates that switching on LSO increases the number of instances on which GANAK performs better than the baseline.

6.2 Comparison with Other Tools

Figure 2 show cactus plots comparing the performance of GANAK with the state-of-the-art exact (sharpSAT) and approximate (ApproxMC3) model counters on a total of 2031 benchmarks. We present the number of instances on x-axis and the time taken on y-axis; a point (x, y) implies that a solver took less than or equal to y seconds to perform model counting on x benchmarks.

Model Counting Over All Variables ($|R_F|$)

For exact model counting, we use two preprocessing tools, pmc equipped with $\#eq$ [Lagniez and Marquis, 2014] and B+E [Lagniez *et al.*, 2016]. In both cases, GANAK is able to perform model counting on more benchmarks (1500 and 1587 with pmc($\#eq$) and B+E, respectively) than any of the state-of-the-art tools: (1385, 1519) were solved by sharpSAT and (1141, 1165) by ApproxMC3[4]) within a timeout of 5000s. Moreover, GANAK achieves the lowest PAR-2 scores of ($0.61\times$, $0.52\times$) of ApproxMC3 and ($0.83\times$, $0.88\times$) of sharpSAT. Table 1 compares GANAK with the competing tools for the best performing setting (i.e., preprocessed with B+E) on a subset[5] of our benchmarks. Columns 2 (3) provide the number of variables (clauses) while the subsequent columns represent the time taken by the respective tools. Figure 2a and Table 1 clearly demonstrates that GANAK comprehensively outperforms both sharpSAT (on which it is built upon) and ApproxMC3.

Benchmark	Vars	Clauses	sharpSAT	ApproxMC3	GANAK
diagStencilClean	378131	2110471	23.58	17.39	23.58
06A-2	3857	15028	0.75	28.38	0.27
50-10-1-q	460	720	1129.78	318.34	118.64
4_11.pm_5steps	80376	42689	36.26	TO	0.96
90-42-1-q	8652	13776	250.53	TO	126.55
orchain220	2433	3562	1321.57	TO	74.99
75-18-6-q	1548	2448	TO	TO	154.55
orchain211	2929	4259	TO	TO	4311.17

Table 1: Runtime comparison of competing model counters in seconds for model counting over all variables when benchmarks are pre-processed with B+E.

Projected Model Counting ($|R_{F\downarrow S}|$)

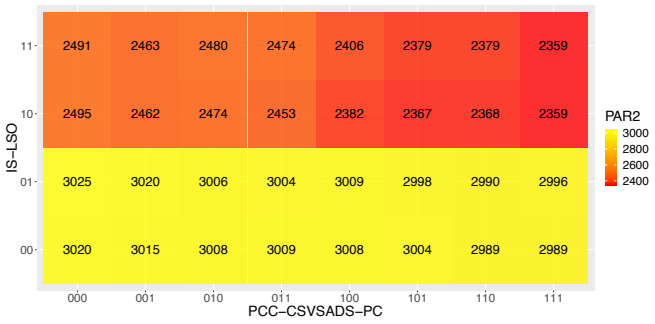
We perform projected model counting on our benchmarks with $S \subseteq \mathsf{Vars}(F)$, where S is provided in the DIMACS files preceded by "`c ind`". We implement projected model counting in a manner similar to that of IS, where the decisions are taken only on variables from S. We disable our IS heuristic when we perform projected model counting on S. As sharpSAT did not support projected model counting, we added this support to enable comparison. For pre-processing, we used pmc equipped with eq that preserves the equivalence across the formulas.

Figure 2b compares the competing model counters on the benchmarks without and with pmc(eq) pre-processing. Again, GANAK is able to perform model counting on more benchmarks (1290 and 1411 without and with pmc(eq), respectively) than sharpSAT (1137, 1302) and ApproxMC3 (1151, 1124) within a timeout of 5000s. Moreover, GANAK achieves the lowest PAR-2 scores of ($0.83\times$, $0.68\times$) of ApproxMC3 and ($0.84\times$, $0.86\times$) of sharpSAT. Table 2 compares GANAK with the competing tools for projected model counting on the best performing setting (i.e., preprocessed with pmc(eq)) on a subset of our benchmarks.

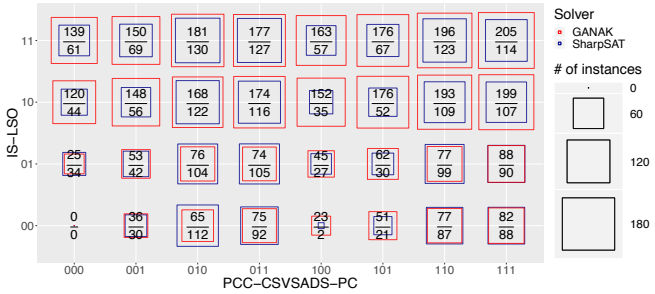
The preprocessing enabled by pmc(eq) seems to favor CDCL based model counters than hashing-based model counters; both sharpSAT and GANAK solve more instances with pmc(eq), whereas the performance of ApproxMC3, in fact,

[4]Since ApproxMC3 requires a sampling set, for a fair comparison, we use the independent support computed by MIS (ran with a timeout of 100s) as the sampling set.

[5]Detailed data of all benchmarks with all the settings is available at https://github.com/meelgroup/ganak.

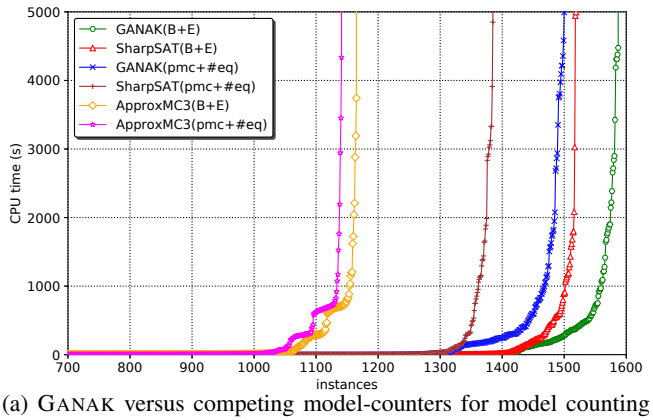


(a) Heatmap on the PAR-2 scores achieved on different configurations on the heuristics. (Best viewed in color)

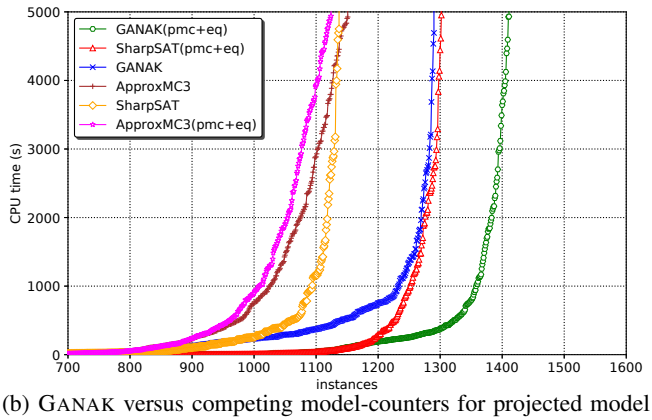


(b) Number of instances on which a configuration performs better than baseline (all heuristics turned off). (Best viewed in color)

Figure 1: Impact of different configurations of heuristics. The 0 (1) bits indicate off (on) condition of a particular heuristic.



(a) GANAK versus competing model-counters for model counting on all variables



(b) GANAK versus competing model-counters for projected model counting

Figure 2: Cactus plots comparing the performance of different model counters. (Best viewed in color)

Benchmark	Vars	Clauses	sharpSAT	ApproxMC3	GANAK
s713	509	1056	3836.79	2.18	3693.49
case120	284	618	11.11	2.60	11.06
pm-4-steps	98746	12006	10.01	11.53	9.93
orchain42	1385	1943	73.8	TO	0.11
orchain168	1661	2333	4144.27	TO	0.10
75-15-8-q	1065	777	4952.46	TO	180.25
50-14-4-q	924	880	TO	TO	198.0
75-17-4-q	1377	985	TO	TO	202.45

Table 2: Runtime comparison of competing model counters in seconds for projected model counting when benchmarks are preprocessed with pmc(eq).

deteriorates. Furthermore, the existence of benchmarks where ApproxMC3 outperforms GANAK indicates the potential for hybrid approaches and calls for deeper investigation for demystifying the performance of counting techniques.

7 Conclusion

GANAK demonstrates that #SAT solvers can significantly benefit from probabilistic component caches, especially when ably supported by heuristics like IS, CSVSADS, PC and LSO. In particular, we are able to outperform exact (sharpSAT) and approximate (ApproxMC3) model counters both in terms of PAR-2 score and the number of instances solved. This is a significant contribution as we are able to solve many more of the *hardest* instances for a #P-complete problem. We believe that the heuristics proposed in this paper will also benefit exhaustive DPLL and CDCL based knowledge compilation frameworks and related tools (like c2d [Darwiche, 2004], DSHARP [Muise *et al.*, 2012], D4 [Lagniez and Marquis, 2017], KUS [Sharma *et al.*, 2018] and WAPS [Gupta *et al.*, 2019]); we intend to investigate this in the future.

Acknowledgements

This research is supported in part by the National Research Foundation Singapore under its AI Singapore Programme (Award Number: [AISG-RP-2018-005]) and the NUS ODPRT Grant [R-252-000-685-133]. The computational work for this article was performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg/). Part of this research work was done during a visit of the first author at NUS, Singapore. We are thankful to Nutanix Software India Pvt. Ltd. (https://www.nutanix.in/) for supporting the conference registration and travel of the first author.

References

[Bacchus *et al.*, 2003] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. Algorithms and complexity results for #SAT and Bayesian inference. In *Proc. of FOCS*, pages 340–351, 2003.

[Bayardo and Pehoushek, 2000] Roberto J. Bayardo, Jr. and Joseph Daniel Pehoushek. Counting Models Using Connected Components. In *Proc. of AAAI*, pages 157–162, 2000.

[Biondi *et al.*, 2018] Fabrizio Biondi, Michael Enescu, Annelie Heuser, Axel Legay, Kuldeep S. Meel, and Jean Quilbeuf. Scalable approximation of quantitative information flow in programs. In *Proc. of VMCAI*, pages 71–93, 2018.

[Birnbaum and Lozinskii, 1999] Elazar Birnbaum and Eliezer L. Lozinskii. The Good Old Davis-Putnam Procedure Helps Counting Models. *J. Artif. Int. Res.*, pages 457–477, 1999.

[Brglez *et al.*, 1989] Franc Brglez, David Bryan, and Krzysztof Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. of ISCAS*, pages 1929–1934, 1989.

[Chakraborty *et al.*, 2013] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A Scalable Approximate Model Counter. In *Proc. of CP*, pages 200–216, 2013.

[Chakraborty *et al.*, 2014] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Balancing scalability and uniformity in SAT witness generator. In *Proc. of DAC*, pages 1–6, 2014.

[Chakraborty *et al.*, 2016] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proc. of IJCAI*, pages 3569–3576, 2016.

[Chakraborty *et al.*, 2019] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. On the Hardness of Probabilistic Inference Relaxations. In *Proc. of AAAI*, 2019.

[Darwiche, 2004] Adnan Darwiche. New Advances in Compiling CNF to Decomposable Negation Normal Form. In *Proc. of ECAI*, pages 318–322, 2004.

[Davis and Putnam, 1960] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, pages 201–215, 1960.

[Domshlak and Hoffmann, 2007] Carmel Domshlak and Jörg Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *JAIR*, pages 565–620, 2007.

[Dueñas-Osorio *et al.*, 2017] Leonardo Dueñas-Osorio, Kuldeep S. Meel, Roger Paredes, and Moshe Y. Vardi. Counting-based reliability estimation for power-transmission grids. In *Proc. of AAAI*, 2017.

[Eén and Sörensson, 2004] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Proc. of SAT*, pages 502–518, 2004.

[Gomes *et al.*, 1998] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proc. of AAAI*, pages 431–437, 1998.

[Gomes *et al.*, 2007] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Near-Uniform sampling of combinatorial spaces using XOR constraints. In *Proc. of NIPS*, pages 481–488, 2007.

[Gupta *et al.*, 2019] Rahul Gupta, Shubham Sharma, Subhajit Roy, and Kuldeep S. Meel. WAPS: Weighted and Projected Sampling. In *Proc. of TACAS*, pages 59–76, 2019.

[Ivrii *et al.*, 2015] Alexander Ivrii, Sharad Malik, Kuldeep S. Meel, and Moshe Y. Vardi. On computing minimal independent support and its applications to sampling and counting. *Constraints*, 2015.

[Lagniez and Marquis, 2014] Jean-Marie Lagniez and Pierre Marquis. Preprocessing for propositional model counting. In *Proc of AAAI*, pages 2688–2694, 2014.

[Lagniez and Marquis, 2017] Jean-Marie Lagniez and Pierre Marquis. An Improved Decision-DNNF Compiler. In *Proc. of IJCAI*, pages 667–673, 2017.

[Lagniez *et al.*, 2016] Jean-Marie Lagniez, Emmanue Lonca, and Pierre Marquis. Improving Model Counting by Leveraging Definability. In *Proc. of IJCAI*, pages 751–757, 2016.

[Lemire and Kaser, 2016] Daniel Lemire and Owen Kaser. Faster 64-bit universal hashing using carry-less multiplications. *Journal of Cryptographic Engineering*, pages 171–185, 2016.

[Marques-Silva and Sakallah, 1999] J. P. Marques-Silva and K. A. Sakallah. GRASP: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, pages 506–521, 1999.

[Meel *et al.*, 2016] Kuldeep S. Meel, Moshe Y. Vardi, Supratik Chakraborty, Daniel J Fremont, Sanjit A Seshia, Dror Fried, Alexander Ivrii, and Sharad Malik. Constrained Sampling and Counting: Universal Hashing Meets SAT Solving. In *Proc. of Beyond NP Workshop*, 2016.

[Minato, 1993] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proc. of DAC*, pages 272–277, 1993.

[Moskewicz *et al.*, 2001] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc. of DAC*, pages 530–535, 2001.

[Muise *et al.*, 2012] Christian Muise, Sheila A. McIlraith, J. Christopher Beck, and Eric I. Hsu. Dsharp: Fast d-DNNF Compilation with sharpSAT. In *Proc. of CAI*, pages 356–361, 2012.

[Pipatsrisawat and Darwiche, 2007] Knot Pipatsrisawat and Adnan Darwiche. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *Proc. of SAT*, pages 294–299, 2007.

[Roth, 1996] Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, pages 273–302, 1996.

[Samer and Szeider, 2007] Marko Samer and Stefan Szeider. Algorithms for Propositional Model Counting. In *Proc. of LPAR*, pages 484–498, 2007.

[Sang *et al.*, 2004] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. of SAT*, 2004.

[Sang *et al.*, 2005a] Tian Sang, Paul Beame, and Henry Kautz. Heuristics for Fast Exact Model Counting. In *Proc. of SAT*, pages 226–240, 2005.

[Sang *et al.*, 2005b] Tian Sang, Paul Beame, and Henry Kautz. Performing Bayesian inference by weighted model counting. In *Prof. of AAAI*, pages 475–481, 2005.

[SAT, 2017] Proc. of SAT Competition 2017: Solver and Benchmark Descriptions. University of Helsinki, Department of Computer Science, 2017.

[Sharma *et al.*, 2018] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. Knowledge Compilation meets Uniform Sampling. In *Proc. of LPAR*, pages 620–636, 2018.

[Soos and Meel, 2019] Mate Soos and Kuldeep S. Meel. BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting. In *Proc of AAAI*, 2019.

[Stockmeyer, 1983] Larry Stockmeyer. The complexity of approximate counting. In *Proc. of STOC*, pages 118–126, 1983.

[Thurley, 2006] Marc Thurley. SharpSAT: counting models with advanced component caching and implicit BCP. In *Proc. of SAT*, pages 424–429, 2006.

[Toda, 1989] Seinosuke Toda. On the computational power of PP and (+)P. In *Proc. of FOCS*, pages 514–519, 1989.

[Valiant, 1979] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*, pages 410–421, 1979.