Engineering an Efficient PB-XOR Solver

Jiong Yang
School of Computing, National University of Singapore

Kuldeep S. Meel
School of Computing, National University of Singapore

Abstract

Despite the NP-completeness of Boolean satisfiability, modern SAT solvers are routinely able to handle large practical instances, and consequently have found wide ranging applications. The primary workhorse behind the success of SAT solvers is the widely acclaimed Conflict Driven Clause Learning (CDCL) paradigm, which was originally proposed in the context of Boolean formulas in CNF. The wide ranging applications of SAT solvers have highlighted that for several domains, CNF is not a natural representation and the reliance of modern SAT solvers on resolution proof system limit their ability to efficiently solve several families of constraints. Consequently, the past decade has witnessed the design of solvers with native support for constraints such as Pseudo-Boolean (PB) and CNF-XOR.

The primary contribution of our work is an efficient solver engineered for PB-XOR formulas, i.e., formulas consisting of a conjunction of PB and XOR constraints. We first observe that a simple adaption of CNF-XOR architecture does not provide an improvement over baseline; our analysis highlights the need for careful engineering of the order or propagations. To this end, we propose three different tactics, all of which achieve significant performance improvements over the baseline. Our work is motivated by applications arising from binarized neural network verification where the verification of properties such as robustness, fairness, trojan attacks can be reduced to model counting queries; the state of the art model counters reduce counting to polynomially many SAT queries over the original formula conjuncted with randomly generated XOR constraints. To this end, we augment ApproxMC augmented with LinPB and we call the resulting counter as ApproxMCPB. In an extensive empirical comparison over 1076 benchmarks, we observe that ApproxMCPB can solve 912 instances while the baseline version of ApproxMC4 (augmented with CryptoMiniSat) can solve only 802 instances.

2012 ACM Subject Classification Theory of computation; Computing methodologies → Artificial intelligence

Keywords and phrases PB-XOR Solving, Pseudo-Boolean, XOR, Gauss Jordan Elimination, SAT-Solving, Model Counting

Supplementary Material The open source tools are available at `https://github.com/meelgroup/linpb` (LinPB) and `https://github.com/meelgroup/approxmcpb` (ApproxMCPB).

Funding This work was supported in part by National Research Foundation Singapore under its NRF Fellowship Programme [NRF-NRFFAI1-2019-0004] and AI Singapore Programme [AISG-RP-2018-005], and NUS ODPRT Grant [R-252-000-685-13].The computational work for this article was fully performed on resources of the National Supercomputing Centre, Singapore (`https://www.nscc.sg`)

Acknowledgements We are grateful to the anonymous reviewer for pointing out a subtle bug in the presentation of Algorithm 1. We are thankful to Priyanka Golia and Yang Suwei for their detailed feedback on the early drafts of the paper.

1 Introduction

Given a Boolean formula F, the problem of satisfiability (SAT) is to determine whether there is an assignment σ to the set of variables such that F evaluates to True. The celebrated work of Cook and Levin (independently) established the NP-completeness of SAT [6, 21]

and thereby establishing SAT at the core of the fundamental question of whether P=NP? From the practical perspective, the past three decades have been witness to unprecedented performance improvements in SAT solvers, which largely owes to the Conflict Driven Clause Learning (CDCL) paradigm, owing to seminal work of Marques-Silva and Sakallah [22], which has seen been combined with careful software engineering along with rigorous theoretical advances. Quoting Knuth: "The story of satisfiability is a tale of the triumph of software engineering blended with rich doses of beautiful mathematics."

From a theoretical perspective, the breakthrough performance improvements of SAT solvers can be cast as surprising given the reliance of CDCL solvers on the resolution as a proof system. Resolution can be characterized as a weak proof system with strong lower bounds for simple formulas such as those based on Pigeon Hole Principle [15, 38]. The weakness of resolution as a proof system is well known to the SAT community, and consequently there have been efforts since the early 2000s in the design of solvers that can perform reasoning more powerful than resolution [10, 3, 33, 20, 12].

The CDCL solver's reliance on resolution contributed to the rise of Conjunctive Normal Form (CNF) to be the input representation for modern SAT solvers. While Tseitin encoding provides an efficient method to convert an arbitrary Boolean formula into CNF with only a linear overhead [37], such an encoding deprives the solver of the natural representation of the problem. Several problems arising from practice can be naturally represented constraints using XORs and Pseudo Boolean (PB), which provided an impetus to the design of solvers with native support for such representations. It is worth remarking that for representations such as XORs and PBs, proof systems such as Gaussian Elimination and cutting planes [7] are known to be exponentially more powerful than resolution.

To summarize, the weakness of resolution and availability of instances arising from practice with natural representation in forms other than CNF have led to the design of solvers such as CryptoMiniSat [36] and RoundingSat [12] with native support for XORs and PB constraints respectively. While the design of CryptoMiniSat was originally motivated by applications in cryptanalysis, its availability served as a bedrock to the development of approximate model counting techniques over the past decade [14, 4, 13, 5, 25, 24, 1]. The current state-of-the-art approximate model counter is ApproxMC [4], which is in its fourth version [34]. ApproxMC takes in a CNF formula and then relies on hashing-based techniques to reduce counting to polynomially many SAT queries over the formulas represented as a conjunction of the original CNF formula and randomly generated XOR constraints. The past three years have witnessed the power of tight integration of CryptoMiniSat and ApproxMC [35, 34].

Akin to applications relying on SAT queries, for several applications of counting, CNF is not the natural representation. Of particular interest to us are applications arising from verification of neural network [27]. Baluta et al. proposed the framework of quantitative verification, called NPAQ, which reduces the verification of properties such as robustness, fairness, trojan attacks over Binarized Neural Networks (BNNs) to counting queries [2]. It is worth observing that the natural representation of BNNs is a conjunction of PB constraints and the counting framework of ApproxMC introduces randomly generated XOR constraints; therefore, each of the underlying SAT queries can be represented as a conjunction of PB and XOR constraints. The current implementation of ApproxMC is built on top of CryptoMiniSat due to its native support of XORs and therefore, NPAQ employs CNF encoding of PB constraints into CNF. While NPAQ was shown to scale to large instances, the scalability remains a major challenge. In this context, one wonders whether it is possible to address the scalability challenge of hashing-based approximate model counting when the instances have their natural representation in PB via the design of an efficient model counter

that has native support for both PB and XOR constraints.

Given the availability of the state-of-the-art cutting plane proof system-based PB solver, RoundingSat [12], a straightforward first step would be to integrate the easily portable Gauss-Jordan elimination module in CryptoMiniSat [36] into RoundingSat. Our initial foray, surprisingly, yielded little to no significant improvement in comparison to the current approach of invoking CryptoMiniSat over PB constraints encoded into CNF. We denote this approach by Lazy-GJE.

The primary contribution of this work is an efficient satisfiability solver, called LinPB, for PB-XOR formulas. Our design of LinPB is based on our identification of the key performance bottleneck in the aforementioned approach: the presence of *redundant* propagation. In LinPB, we propose three novel strategies for propagation: Shared-Watches, Eager-GJE, and Mixed-Watches. To evaluate the empirical effectiveness of our proposed techniques, we integrate LinPB with the ApproxMC algorithm; we call the resulting tool ApproxMCPB. We perform an empirical comparison of ApproxMCPB vis-a-vis ApproxMC4 tool and other state-of-the-art counters on over 1076 benchmarks arising from binarized neural network verification for diverse properties [2]. Our empirical comparison shows that while ApproxMC can solve only 802 instances, ApproxMCPB can solve 912 instances, thereby achieving a gain of over 100 instances. Furthermore, the PAR-2 score for ApproxMC is 3305 seconds while the PAR-2 score for ApproxMCPB is 1822 seconds, thereby achieving an almost 50% decrease in PAR-2 score. Among different strategies, we observe that usage of Lazy-GJE leads to ApproxMCPB solving 804 instances while usage of Shared-Watches, Eager-GJE, and Mixed-Watches leads to solving 892, 892, and 912 instances.

The rest of the paper is organized as follows: We discuss notations and preliminaries in Section 2 and introduce the background of PB and XOR solving in Section 3. In Section 4, We focus on core technical contributions for PB-XOR solving. We then present an extensive experimental evaluation in Section 5 and finally conclude in Section 6.

2 Notations and Preliminaries

Let $X = \{x_1, x_2, \dots x_n\}$ be the set of Boolean variable. A literal is a variable or its negation. A clause is a disjunction of literals.

For a Boolean formula φ, we use $\mathrm{Vars}(\varphi)$ to denote the set of variables involved in φ. If an assignment σ of truth values to all the variables in $\mathrm{Vars}(\varphi)$ makes formula φ evaluate to True, it's called a *solution* or *witness* of φ. We use $\mathsf{sol}(\varphi)$ to denote the set of all witnesses of φ. Given a set of variables $\mathcal{P} \subseteq \mathrm{Vars}(\varphi)$, we denote the projection of R_F on $\mathcal{P}$ by $\mathsf{sol}(\varphi)_{\downarrow \mathcal{P}}$.

In the context of *propositional model counting*, we aim to compute the number of solutions, i.e. $|\mathsf{sol}(\varphi)|$, for a given Boolean formula φ. A *probably approximately correct* (PAC) counter denotes a probabilistic algorithm $\mathrm{ApproxCount}(\cdot, \cdot, \cdot)$ that takes as inputs a formula φ, a tolerance $\epsilon > 0$ and a confidence $1 - \delta \in (0, 1]$, and returns a count c with (ϵ, δ)-gurantees, i.e., $Pr[|\mathsf{sol}(\varphi)|/(1+\epsilon) \leq c \leq (1+\epsilon)|\mathsf{sol}(\varphi)] \geq 1 - \delta$. Similarly, projected model counting is to compute $|\mathsf{sol}(\varphi)_{\downarrow \mathcal{P}}|$ instead of $|\mathsf{sol}(\varphi)|$ for a given sampling set $\mathcal{P} \subseteq Vars(F)$.

A (linear) pseudo-Boolean (PB)-constraint is represented as $\Sigma_{i \in S} w_i x_i \geq k$ where, $S \subseteq [n]$, $w_i, k \in \mathbb{Z}$. An XOR-constraint is represented as $\oplus_{i \in S} x_i = b$ for $S \subseteq [n]$ and $b \in \{0, 1\}$ where $\oplus$ represents XOR operator. A formula is in CNF if it can be represented as conjunction of clauses. Similarly, a formula is PB form (resp. XOR form) if it can be represented as conjunction of PB (resp. XOR) constraints. Furthermore, a formula is in PB-XOR (resp. CNF-XOR) form if if can be represented as $\phi \wedge \psi$ where ϕ is a formula in PB (resp. CNF) form and ψ is a formula in XOR form.

PB Encoding of XOR

Our work focuses on the efficient handling of PB-XOR formulas. A simple baseline approach would be to express XOR constraints as PB constraints, and in this context, one wonders whether there is an efficient method to encode PB constraints. We now state a well-known encoding of XOR into PB constraints via the introduction of additional auxiliary variables.

▶ **Observation 1** (folklore). *Given a XOR constraint:* $\bigoplus_{i=1}^{i=n} x_i = b$, *and* $\oplus$ *denotes exclusive disjunction operation, we introduce auxiliary Boolean variables* $\{t_i\}, i = 1, 2, ..., \lfloor \log_2(n) \rfloor$. *Then, the XOR constraint is logically equivalent to the following pseudo-Boolean constraint:*

$$\Sigma_{i=1}^{n} x_i - \Sigma_{i=1}^{\lfloor \log_2(n) \rfloor} 2^i \cdot t_i = b \tag{1}$$

Applying the encoding in Definition 1, we achieve a one-to-one mapping between XOR constraint and its PB encoding.

3 Background

In order to put our contributions in context, we provide a brief discussion about the workings of the current state-of-the-art implementations of Gauss-Jordan elimination procedures in modern SAT solvers such as CryptoMiniSat [36].

3.1 Gauss-Jordan Elimination

Gauss-Jordan Elimination (GJE) is an efficient algorithm for solving systems of linear equations. Since XOR constraints are considered as linear equations modulo two, Gauss-Jordan Elimination (GJE) can be used to solve systems of XOR constraints. CryptoMiniSat [36] was the first SAT solver with deep integration of Gauss-Jordan Elimination into CDCL framework. Later, Han and Jiang proposed a new framework [16] building on Simplex-like techniques that performs Gauss-Jordan elimination, i.e., using reduced row echelon form instead of row echelon form. They used a two-watched variable scheme to detect propagations and conflicts in XOR constraints. Meel and Soos integrated Han and Jiang's framework into their proposed architecture BIRD that sought to take advantage of both in-processing techniques and GJE. Recently, Soos, Gocht, and Meel [34] achieved acceleration in XOR unit propagation via exploiting bit-level parallelism offered in modern CPUs. In particular, they employed bit-packed integers to represent XOR rows in a matrix and apply bitwise operations, such as and, inverse, hamming weight, to quickly detect propagations and conflicts in XORs.

Lazy Reason Clause Generation

During XOR propagation, a reason clause will be generated to be used in future conflict analysis. However, during profiling the runtime of SAT solver, the generation process is quite time-consuming if the size of the XOR constraint involves thousands of variables. Furthermore, a large portion of reason clauses are never used during conflict analysis as not all assigned variables will be involved in the conflict as we apply the 1UIP policy. To reduce the overhead from the generation of useless reason clauses, Soos, Gocht, and Meel proposed [34] a lazy generation method, which was based on the observation that once a literal is propagated by XOR propagation, the row of the XOR will preserve the propagated state until backtracking to the previous level. Therefore, the lazy method keeps an index of the row and the propagating literal but does not compute the reason clause eagerly. Whenever a reason clause is requested by conflict analysis, the reason clause is computed from the recorded row.

3.2 Conflict-Driven Pseudo-Boolean Solving

The past two decades have witnessed a rich array of techniques proposed in the context of PB solving (MiniSat+ [11], Open-WBO [23, 18], NaPS [30], Sat4j [20], PRS [10], Galena [3], Pueblo [33], RoundingSat [12]). Given the space considerations, we refer the reader to [28] for a detailed discussion on PB solvers and we will focus on providing a brief overview of the underlying PB solver, RoundingSat, in our work. RoundingSat employs a Conflict-Driven framework similar to the conflict-driven clause learning (CDCL) framework in CNF solving. The framework primarily extends conflict analysis and unit propagation from CNF to pseudo-Boolean constraints. RoundingSat employs cutting-planes based generalized resolution [7, 17, 9, 12] to resolve two PB constraints, which is exponentially stronger than resolution from a theoretical standpoint. In contrast to the two-watched literal scheme for CNF solving, RoundingSat employs a three-tiered approach where clauses, cardinality constraints, and general PB constraints were handled with different watched propagation techniques [3, 32, 20, 12, 8].

4 LinPB: An Efficient PB-XOR Solver

We now turn to the primary technical contribution of this work, our solver, LinPB, for PB-XOR formulas. As mentioned in Section 1, our first step (Section) was to lift the Lazy-GJE module inside CryptoMiniSat to RoundingSat. Observing that such a process did not yield any dividends compared to the baseline, we sought to investigate the key performance bottlenecks and accordingly propose three strategies: Shared-Watches, Eager-GJE, and Mixed-Watches, which seek to optimize the interaction between PB and XOR constraints. Since we keep the internal components of PB and GJE intact, our discussion in the rest of the section will focus on the interactions between the two components. In the rest of the section, we will use the term *PB propagation* to refer to propagations due to PB constraints and *XOR propagations* to refer to unit propagations due to XOR constraints via GJE.

4.1 Lazy Gauss-Jordan Elimination

We present the high-level overview of Lazy-GJE in Algorithm 1. We assume that the formula ϕ corresponds to PB constraints while the formula ψ corresponds to XOR constraints. Following CryptoMiniSat, we keep separate propagation indices for PB (q_ϕ) and XOR constraints (q_ψ). Trail ν represents the current assignment queue. The while loop at lines 3–7 performs PB propagation until we detect a conflict at line 6 or go through all assignments in ν. The while loop at lines 8–12 executes the similar procedure for XOR propagation. If neither PB nor XOR propagation detects a conflict, the unit propagation returns NULL at line 13. We refer to Algorithm 1 as a lazy method because GJE is invoked lazily, i.e., it is invoked only after all the unit propagations from PB constraints are processed.

As mentioned earlier, we observed that augmenting RoundingSat with Lazy-GJE did not lead to performance improvements over the baseline, CryptoMiniSat (wherein PB constraints are encoded into CNF). Upon further investigation, we observed a considerable performance drop with the increase in the number of XOR constraints. A plausible primary reason for the behavior is that the Lazy-GJE delays conflict detection arising from XOR constraints, and the delay may lead to many redundant PB (unit) propagations. We illustrate such a scenario via an example.

▶ **Example 2. Hard instance for Lazy-GJE.** $\bigwedge_{i\in[1..n]}(x_i+\neg x_{i+1} \geq 1)\wedge\bigwedge_{j\in[0..\lfloor\frac{n}{3}\rfloor]}(x_{3j+1}\oplus x_{3j+2}\oplus x_{3j+3}=1), n \gg 1$.

Algorithm 1 Lazy Gauss Jordan Elimination

```
1  Function propagationLazyGJE()
       Data: PB constraints φ, XOR constraints ψ,
       trail ν, PB propagation index q_φ, XOR propagation index q_ψ
2      while q_φ < size(ν) or q_ψ < size(ν) do
3          while q_φ < size(ν) do
4              l_φ ← ν[q_φ]
5              q_φ ← q_φ + 1
6              if propagatePB(φ, l_φ) == conflict then
7                  return conflict
8          while q_ψ < size(ν) do
9              l_ψ ← ν[q_ψ]
10             q_ψ ← q_ψ + 1
11             if propagateXOR(ψ, l_ψ) == conflict then
12                 return conflict
13     return NULL
```

Suppose we select decision variables sequentially from x_1 to x_n and prefer negative polarity for the decision literal. Table 1 shows the procedure for a PB-XOR solver employing Lazy-GJE to solve Example 2. At level 1, the solver performs $O(n)$ PB propagations and produces $O(n)$ assignments. Then, the XOR propagation immediately detects a conflict, which, however, only involves the decision variable and first two variables implied at current level, while the rest of PB propagations are irrelevant to the conflict. The redundant PB propagations are reproduced every time the solver reaches level 1. In summary, the usage of Lazy-GJE leads to LinPB processing $O(n^2)$ redundant PB propagations. The scenario described above is reminiscent of the motivation of chronological backtracking [26], in which a solver may reassign many variables that are irrelevant to the conflict after non-chronological backtracking.

4.2 Eager-GJE

Table 1 demonstrates that lazy invocation of GJE may lead the solver to perform many redundant PB propagations in PB-XOR solving. Furthermore, Gauss Jordan Elimination is sound and complete, i.e., all unit propagations and conflicts implied by the given set of XORs would be discovered by a GJE-based decision procedure. Therefore, a natural reaction would be to invoke GJE in an eager fashion.

Algorithm 2 presents the propagation routine for Eager-GJE. Like Lazy-GJE, we use independent indexes for PB and XOR to track trail. Lines 2–6 perform PB propagation for literal l_ϕ. After each PB propagation, lines 7–11 go through all assignments in ν to detect all possible propagations and conflicts in XOR constraints via (incremental) GJE. We denote Algorithm 2 as an eager method because of the aggressive invocation of XOR propagations.

Our empirical evaluation indicates that while Eager-GJE is able to provide a remedy for some of the weaknesses of Lazy-GJE, the overhead due to GJE limits the scalability.

Level	Decision	PB propagation	XOR propagation	Conflict analysis
0	NIL	NIL	NIL	jump to level 1
1	$\neg x_1$	$\neg x_2, \neg x_3, ...\neg x_{n+1}$	conflict at $x_1 \oplus x_2 \oplus x_3 = 1$	conflict constraint $x_1 + x_2 + x_3 \geq 1$ resolve with $x_2 + \neg x_3 \geq 1$ and $x_1 + \neg x_2 \geq 1$ learn $x_1 \geq 1$ backtrack to level 0
0	NIL	x_1	NIL	jump to level 1
1	$\neg x_2$	$\neg x_3, \neg x_4, ...\neg x_{n+1}$	conflict at $x_4 \oplus x_5 \oplus x_6 = 1$	conflict constraint $x_4 + x_5 + x_6 \geq 1$ resolve with $x_5 + \neg x_6 \geq 1$ and $x_4 + \neg x_5 \geq 1$ learn $x_4 \geq 1$ backtrack to level 0
0	NIL	x_4, x_3, x_2	NIL	jump to level 1
repeat k = 2, 3, ... $\lfloor \frac{3}{n} \rfloor$				
1	$\neg x_{3k-1}$	$\neg x_{3k}, \neg x_{3k+1}...\neg x_{n+1}$	conflict at $x_{3k+1} \oplus x_{3k+2} \oplus x_{3k+3} = 1$	conflict constraint $x_{3k+1} + x_{3k+2} + x_{3k+3} \geq 1$ resolve with $x_{3k+2} + \neg x_{3k+3} \geq 1$ and $x_{3k+1} + \neg x_{3k+2} \geq 1$ learn $x_{3k+1} \geq 1$ backtrack to level 0
0	NIL	$x_{3k+1}, x_{3k}, x_{3k-1}$	NIL	jump to level 1

Table 1 Procedure to solve Example 2 by Lazy-GJE. Column Level denotes the current decision level. Column Decision presents the decision literal at the current level. Column PB-propagation and XOR-propagation show the inferred assignments or the detected conflict by propagations. The last column specifies the conflict constraint, resolvents, and learned constraints in conflict analysis. Finally, jump to the next level if no conflict; otherwise, backtrack.

4.3 Shared-Watches

We now seek to take the middle road: we want to avoid both lazy and eager invocation of GJE. Our approach is to intermingle the PB and XOR propagations. Our proposed scheme, called Shared-Watches, is presented in Algorithm 3. Unlike the separate indexes for PB and XOR to trace propagation in Lazy-GJE, we use a shared index q for both constraints. At line 5 and 7, we detect PB and XOR propagation synchronously for each assignment l and terminate the unit propagation immediately if any of them detects a conflict.

We apply Shared-Watches to Example 2 and hold the same assumption that we select decision variables sequentially from x_1 to x_n and prefer negative polarity for each decision literal. Table 2 presents a shared-watches solver to solve Example 2. Every time at level 1, after a constant number (≤ 4) of PB propagations, the solver timely detects the conflict in XOR propagation. The fast detection of the conflict saves runtime from useless propagations, and then the solving time complexity is reduced to $O(n)$.

4.4 Mixed Watches

Our empirical analysis indicates that the key performance bottleneck for Shared-Watches and Eager-GJE is the computationally expensive (incremental) GJE that is invoked by propagateXOR. In order to reduce the overhead from XOR propagation and meantime *watch* XOR constraints timely, we propose Mixed-Watches. Mixed-Watches aims to *learn* partial PB constraints of interest implied by XOR constraints and add them to PB constraints. PB watches can detect partial XOR propagations and conflicts implied by equivalent PB constraints without access to XOR watches. In other words, Mixed-Watches reduce the invocation of XOR propagation but maintain the ability to *watch* XORs in a timely fashion. It is worth remarking that learning all PB constraints implied by a XOR constraint would either necessitate the addition of a large number of auxiliary variables or storage of exponentially many (in the size of XORs) PB constraints. Therefore, the quality of learned PB constraints from XOR is of significant importance.

Algorithm 2 Eager-GJE

```
1  Function propagationEagerGJE()
       Data: pseudo-Boolean constraints φ, XOR constraints ψ,
       trail ν, propagation index q_φ, XOR propagation index q_ψ
2      while q_φ < size(ν) do
3          l_φ ← ν[q_φ] ;
4          q_φ ← q_φ + 1 ;
5          if propagatePB(φ, l_φ) == conflict then
6              return conflict
7          while q_ψ < size(ν) do
8              l_ψ ← ν[q_ψ] ;
9              q_ψ ← q_ψ + 1 ;
10             if propagateXOR(ψ, l_ψ) == conflict then
11                 return conflict
12     return NULL
```

Algorithm 3 Shared-Watches

```
1  Function propagationSharedWatches()
       Data: pseudo-Boolean constraints φ, XOR constraints ψ, trail ν, propagation
             index q
2      while q < size(ν) do
3          l ← ν[q]
4          q ← q + 1
5          if propagatePB(φ, l) == conflict then
6              return conflict
7          if propagateXOR(ψ, l) == conflict then
8              return conflict
9      return NULL
```

We propose to learn both conflict and reason constraints used by conflict analysis (CA-reason) from XOR constraints since the conflict and propagation play an essential role in CDCL, and these constraints are likely to be triggered again in the future. Algorithm 4 presents the pseudocode for conflict analysis in PB-XOR solving with Mixed-Watches. Lines 2–3 add the conflict constraint (C_{confl}) to learned PB constraints if C_{confl} is detected from XOR propagation. Lines 4–11 perform conflict analysis. We retrieve the last assignment l from trail ν at line 5. If l in the conflict constraint, we fetch the reason constraint (C_{reason}) at Line 7. If the reason constraint is generated from a XOR constraint, we add C_{reason} to learned PB constraints at lines 8–9. The conflict constraint resolve with the reason constraint at line 10. The last assignment is removed from the trail nu at line 11, and then the next iteration starts. Finally, the function returns the constraint after analysis at line 12. In Section 5, we use a portfolio method to empirically show that learning both conflict and CA-reason constraints is the best learning heuristic, and Mixed-Watches cooperates well with Lazy-GJE.

Level	Decision	PB propagation	XOR propagation	Conflict analysis
0	NIL	NIL	NIL	jump to level 1
1	$\neg x_1$	$\neg x_2, \neg x_3$	conflict at $x_1 \oplus x_2 \oplus x_3 = 1$	conflict constraint $x_1 + x_2 + x_3 \geq 1$ resolve with $x_2 + \neg x_3 \geq 1$ and $x_1 + \neg x_2 \geq 1$ learn $x_1 \geq 1$ backtrack to level 0
0	NIL	x_1	NIL	jump to level 1
1	$\neg x_2$	$\neg x_3, \neg x_4, \neg x_5, \neg x_6$	conflict at $x_4 \oplus x_5 \oplus x_6 = 1$	conflict constraint $x_4 + x_5 + x_6 \geq 1$ resolve with $x_5 + \neg x_6 \geq 1$ and $x_4 + \neg x_5 \geq 1$ learn $x_4 \geq 1$ backtrack to level 0
0	NIL	x_4, x_3, x_2	NIL	jump to level 1
repeat k = 2, 3, ... $\lfloor \frac{3}{n} \rfloor$				
1	$\neg x_{3k-1}$	$\neg x_{3k}, \neg x_{3k+1}, \neg x_{3k+2}, \neg x_{3k+3}$	conflict at $x_{3k+1} \oplus x_{3k+2} \oplus x_{3k+3} = 1$	conflict constraint $x_{3k+1} + x_{3k+2} + x_{3k+3} \geq 1$ resolve with $x_{3k+2} + \neg x_{3k+3} \geq 1$ and $x_{3k+1} + \neg x_{3k+2} \geq 1$ learn $x_{3k+1} \geq 1$ backtrack to level 0
0	NIL	$x_{3k+1}, x_{3k}, x_{3k-1}$	NIL	jump to level 1

Table 2 Procedure to solve Example 2 by Shared-Watches. Column Level denotes the current decision level. Column Decision presents the decision literal at the current level. Column PB-propagation and XOR-propagation show the inferred assignments or the detected conflict by propagations. The last column specifies the conflict constraint, resolvents, and learned constraints in conflict analysis. Finally, jump to the next level if no conflict; otherwise, backtrack.

5 Experimental Evaluation

We equipped the state-of-the-art pseudo-Boolean solver RoundingSat[12] with proposed PB-XOR tactics and called the resulting solver LinPB. To showcase the impact of LinPB, we integrated LinPB into the state-of-the-art hashing-based counting technique ApproxMC, implementing the first pseudo-Boolean model counter, ApproxMCPB. We conducted an extensive study on 1076 benchmarks[1] arising from quantitative verification of binarized neural networks with respect to different properties such as robustness, trojan attack, and fairness. These benchmarks represent a wide range of security applications where quality and runtime performance of counters are key determining factors [2]. To evaluate the performance of ApproxMCPB, we performed a comparison with state-of-the-art CNF projected counting techniques ApproxMC4 [34], Ganak[31], GPMC[29] and projMC[19]. We used CNF encoding as described in [2], and equivalent pseudo-Boolean encoding[2] for ApproxMCPB. We developed the PB counter employing PB encoding of XOR constraints as another baseline.[3]

Experiments were conducted on a high-performance computer cluster, each node consisting of 2xE5-2690v3 CPUs with 2x12 real cores and 96GB of RAM. We set the time limit as 5000 seconds and the memory limit as 4GB for each counter per benchmark. Keeping in line with the prior work, we set the confidence factor $\delta = 0.2$ and tolerance factor $\epsilon = 0.8$ by default for approximate counters. We used the number of solved benchmarks and PAR-2 score to evaluate the performance. The PAR-2 score represents the average running time on benchmarks with a doubling-time penalty on timeout benchmarks.

The objective of our experimental evaluation is to analyze the performance of ApproxMCPB both in terms of runtime and approximation quality. In particular, we sought to answer the following questions:

RQ 1 How does the performance of GJE tactics for ApproxMCPB?

[1] The benchmarks are available at `https://teobaluta.github.io/NPAQ/#benchmarks`.
[2] Refer to Appendix A for PB encoding.
[3] The baseline solved nearly 200 fewer benchmarks than ApproxMCPB and thereby of no interest to us.

Algorithm 4 Mixed-Watches

```
1  Function conflictAnalysisMixedWatches(C_confl, ν)
       Data: conflict constraint C_confl, trail ν
2      if C_confl is from XOR propagation then
3          AddConstraintToPB(C_confl)
4      while C_confl is not asserting do
5          l ← getLast(ν)
6          if ¬l in C_confl then
7              C_reason ← getReason(l)
8              if C_reason is from XOR propagation then
9                  AddConstraintToPB(C_reason)
10             C_confl ← resolve(C_confl, C_reason)
11         ν ← removeLast(ν)
12     return C_confl
```

RQ 2 How does the runtime performance of ApproxMCPB compare with ApproxMC4 and other state-of-the-art projected counting techniques?

RQ 3 How far are the counts computed by ApproxMCPB from the exact counts?

In summary, the usage of Lazy-GJE leads to ApproxMCPB solving 804 instances while usage of Shared-Watches, Eager-GJE, and Mixed-Watches allows ApproxMCPB solve 892, 892 and 912 instances respectively. While the state-of-the-art tool, ApproxMC4, can only solve 802 instances, ApproxMCPB can solve 912 instances, an increment of 110 instances. Furthermore, the PAR-2 score for ApproxMC4 is 3305 seconds while PAR-2 score for ApproxMCPB is 1822 seconds, thereby achieving almost 50% decrease in PAR-2 score. Moreover, the speedup of ApproxMCPB to ApproxMC4 is independent of the number of solutions. In terms of approximation quality, the average observed tolerance is 0.037, far better than the theoretical guarantee of 0.8.

5.1 Performance of GJE tactics

This section evaluates the performance of ApproxMCPB augmented with different PB-XOR tactics: Lazy-GJE, Eager-GJE, Shared-Watches, and Mixed-Watches[4]. Table 3 summarizes the results. ApproxMCPB augmented with Lazy-GJE solved only 804 of 1076 benchmarks while ApproxMCPB augmented with Shared-Watches, Eager-GJE, and Mixed-Watches solved 892, 892, and 912 instances respectively, thereby achieving a gain of over 100 instances. Furthermore, the PAR-2 score for the usage of Lazy-GJE is 2755 seconds while the PAR-2 score for the usage of Shared-Watches, Eager-GJE, and Mixed-Watches is 2017, 2042, 1822 seconds respectively, thereby achieving a decrease of over 700 seconds. Observe that the usage of Mixed-Watches leads to ApproxMCPB solving 20 more instances than the other tactics.

Figure 1 presents the cactus plot of the performance of different tactics. We present the number of solved benchmarks on the x-axis and the time taken on the y-axis. A point (x, y) represents that x benchmarks can be solved within y seconds for the particular tactic. We

[4] See Appendix B for the optimal configuration of Mixed-Watches

Total	Lazy-GJE	Eager-GJE	Shared-Watches	Mixed-Watches
1076 (PAR-2)	804 (2755)	892 (2042)	892 (2017)	**912 (1822)**

Table 3 The number of solved benchmarks for ApproxMCPB configured with different Gauss Jordan Elimination tactics. PAR-2 score is in parentheses. Mixed-Watches applies the heuristic of learning both conflict and CA-reason constraints based on Lazy-GJE. Time out after 5000s.

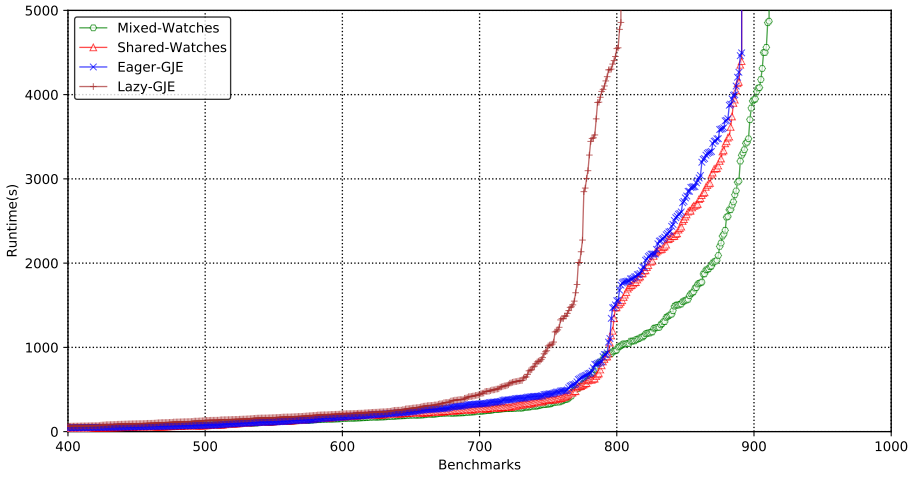

Figure 1 Runtime for ApproxMCPB of different GJE tactics on 1076 BNN benchmarks. The x-axis represents the number of solved benchmarks, while the y-axis shows the counting time. A point (x, y) represents that x benchmarks can be solved within y seconds. The number of solved benchmarks sorts counters in descending order in the legend.

observed that all the curves almost converge to an overlapped curve before the 300-second runtime threshold, which means the usage of different tactics makes ApproxMCPB solve a similar number of benchmarks within a runtime threshold less than 300 seconds. Then, Lazy-GJE begins to diverge and leads to ApproxMCPB solving fewer benchmarks than other tactics with the same runtime threshold. Eager-GJE and Shared-Watches diverge together at around 1000-second threshold, and Shared-Watches slightly outperforms Eager-GJE after diversion. The observation reveals that the usage of Mixed-Watches always leads to ApproxMCPB solving no fewer benchmarks than other tactics no matter what runtime threshold is used. Similarly, the usage of Shared-Watches always produces a no worse result than Eager-GJE and Lazy-GJE. In summary, Eager-GJE, Shared-Watches, and Mixed-Watches successively extend the reach of ApproxMCPB.

Runtime breakdown To analyze the time consumption of main procedures, we profile the runtime breakdown for conflict analysis, PB propagation, XOR propagation, and others. For each procedure, we sum the runtime over solved benchmarks and compute the proportion in total runtime. Then, we calculate the PAR-2 score and proportionally break it down into the four procedures. Figure 2 presents the breakdown of PAR-2 score. The x-axis shows the main procedures in PB-XOR solving, while the y-axis presents the PAR-2 score proportionally taken by each procedure. Colors represent different GJE tactics. We observed that Lazy-GJE spends much more time on conflict analysis and PB propagation than other tactics, while Eager-GJE and Shared-Watches spend more time on XOR propagation and other procedures. Particularly, Mixed-Watches spends relatively less time on all procedures among four tactics. The observation reveals that the usage of Eager-GJE and Shared-Watches indeed incurs more

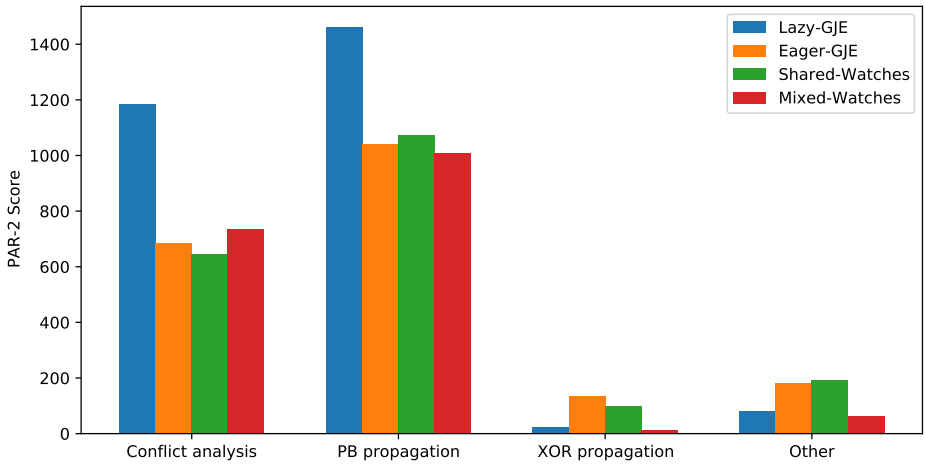

Figure 2 Breakdown of PAR-2 score. We proportionally break down the PAR-2 score into four procedures according to the runtime taken by each procedure. The x-axis shows the four main procedures in PB-XOR solving, while the y-axis presents the PAR-2 score proportionally taken by each procedure.

	Exact		Probabilistic Exact	Approximate	
Total	GPMC	projMC	Ganak	ApproxMC4	ApproxMCPB
1076 (PAR-2)	511 (5713)	430 (6584)	1 (9991)	802 (3305)	**912 (1822)**

Table 4 Number of solved benchmarks for ApproxMCPB vs. state-of-the-art projected model counting techniques. PAR-2 score is in the parentheses. ApproxMCPB uses the best configuration of Mixed-Watches. Time out after 5000s.

overhead from XOR-propagation and other procedures. The usage of Mixed-Watches leads to ApproxMCPB achieving a similar efficiency in conflict analysis and PB-propagation as Eager-GJE and Shared-Watches while maintaining a small overhead from XOR-propagation and other procedures, thereby emerging as the most efficient tactic.

5.2 Performance vs. State-of-the-art Projected Counting Techniques

Since the design of LinPB was motivated by model counting applications, we present an empirical comparison of ApproxMCPB vis-a-vis other state-of-the-art counting techniques. For all the results in this section, we equip ApproxMCPB with Mixed-Watches tactic. Table 4 summarizes the results. We observed that state-of-the-art techniques can solve at most 802 of 1076 instances while ApproxMCPB can solve 912 instances, thereby achieving a gain of over 100 instances. The PAR-2 score for state-of-the-art techniques is at least 3305 seconds while the PAR-2 score for ApproxMCPB is 1822 seconds, thereby achieving an almost 50% decrease in PAR-2 score. Particularly, the exact counting techniques can solve only 511 instances, roughly half of ApproxMCPB. Therefore, ApproxMCPB significantly outperforms state-of-the-art projected counting techniques. Figure 3 presents the number of solved benchmarks in terms of the runtime threshold for all counters. The righter the curve is, the more benchmarks the counter can solve within a runtime threshold. We observed that ApproxMCPB can always solve more instances than other techniques given any runtime threshold.

Dependence on #Solutions We now analyze how the speedup achieved by ApproxM-

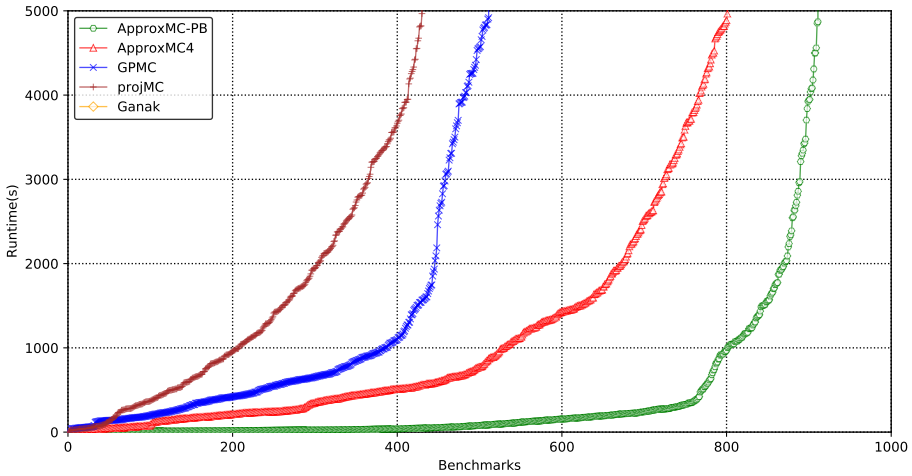

Figure 3 Runtime for ApproxMCPB vs. state-of-the-art projected model counters. The x-axis represents the number of solved benchmarks, while the y-axis shows the counting time. A point (x, y) represents that x benchmarks can be solved within y seconds. Ganak can solve only one instance and therefore fails to be plotted. The number of solved benchmarks sorts counters in descending order in the legend. Time out 5000s.

CPB varies with the #Solutions. To this end, Figure 4 presents the speedup of the ApproxMCPB to ApproxMC4 on the y-axis with the #Solutions. We selected benchmarks solved by ApproxMCPB or ApproxMC4. Each point represents a benchmark. The x-axis presents the number of solutions of the benchmark in the log2 scale[5], while the y-axis represents the speedup, i.e., the counting-time[6] ratio of ApproxMC4 to the ApproxMCPB $\frac{T_{CNF}}{T_{PB}}$ on the benchmark. The horizontal gray line highlights the boundary of speedup $y = 1$.

We observed that almost all points are above the horizontal line, indicating ApproxMCPB outperforms ApproxMC4 on most instances. Even though most instances can be both solved, ApproxMCPB can solve over 100 instances beyond the reach of ApproxMC4 while ApproxMC4 only solved 7 instances beyond the reach of ApproxMCPB. Furthermore, the speedup of ApproxMCPB to ApproxMC4 randomly falls into the interval $[10^0, 10^2]$ on almost all benchmarks. Therefore, ApproxMCPB is able to achieve consistent speedup over the entire spectrum of #Solutions.

5.3 Correctness

To evaluate the approximation quality, we compare the counts computed by approximate model counters with counts returned by exact model counters. Figure 5 shows the model counts computed by ApproxMCPB, and the bounds obtained by scaling the exact counts with the tolerance factor ($\epsilon = 0.8$). We selected benchmarks solved by at least one exact counter. All exact counts from the same benchmark are equal. The exact count sorts benchmarks in ascending order on the x-axis, while the y-axis represents the model count. We observed that for all the benchmarks, ApproxMCPB computed counts within the tolerance. Furthermore, for each instance, the observed tolerance (ϵ_{obs} was calculated

[5] Given that the estimation is $a * 2^b$, we denote the log value as $b + log_2(a + 1)$ to avoid invalid $log_2(0)$.
[6] Drawing from the definition of PAR-2 score, we double the runtime if the benchmark is unsolved within the time limit.

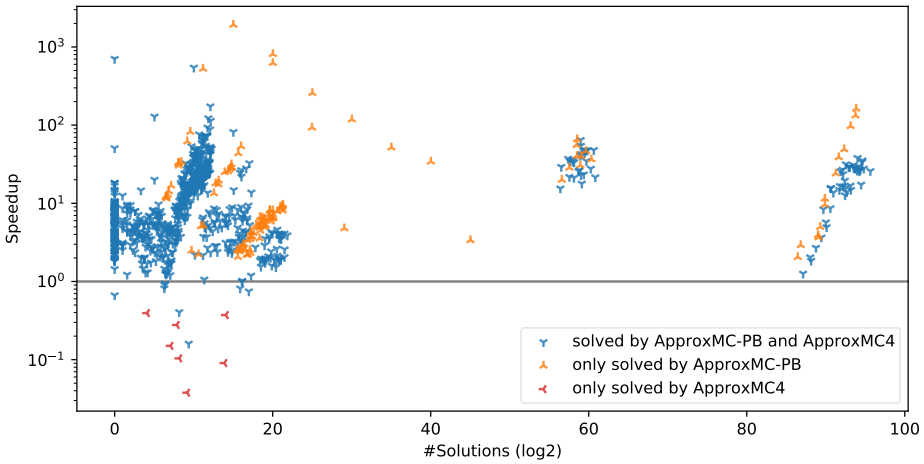

Figure 4 Speedup of ApproxMCPB to ApproxMC4 in terms of the number of solutions of benchmarks. Speedup represents counting-time ratio of ApproxMC4 to ApproxMCPB. #Solutions is in the log2 scale. The horizontal gray line denotes the boundary of speedup $y = 1$. The runtime is doubled if unsolved within the time limit (5000s).

as $max(\frac{|sol(F)|}{AprxCount} - 1, \frac{AprxCount}{|sol(F)|} - 1)$, where $AprxCount$ is the estimate by ApproxMCPB. We observed that the arithmetic mean of ϵ_{obs} across all benchmarks is 0.037 - far better than the theoretical guarantee of 0.8.

Furthermore, we observed that the estimates of ApproxMCPB always match that of ApproxMC4. Recall that the hashing-based approximate counting technique is to employ randomly generated XOR constraints to partition the solution space into roughly equal small cells and count the number of solutions in a randomly picked cell to estimate the total number of solutions. We used the same random seed for ApproxMCPB and ApproxMC4. Hence, both counters always generated the same set of XOR constraints and counted the same cell to estimate the #Solutions.

6 Conclusion and Discussion

In this paper, we focused on the design of LinPB, a solver with native support for PB-XOR formulas. The need for LinPB was motivated by the recent surge of interest in verification of (binarized) neural networks wherein the quantitative verification queries were shown to reduce to model counting. Binarized neural networks can be naturally represented as PB constraints while hashing-based techniques reduce counting to polynomially many SAT queries wherein the original formula is conjuncted with random XOR constraints.

We observed that a straightforward adaptation of the Lazy-GJE approach does not yield performance improvements. Our empirical investigations highlighted the importance of the interaction of PB and XOR propagations. To this end, we designed three propagation strategies: Eager-GJE, Shared-Watches, and Mixed-Watches. We demonstrate the effectiveness of LinPB by augmenting it with the state-of-the-art hashing-based algorithm, ApproxMC; we call the resulting counter, ApproxMCPB. Our empirical evaluation demonstrates ApproxMCPB is able to solve 110 more benchmarks than the baseline approach with a decrease of PAR-2 score by 1483.

The runtime performance of LinPB opens up several interesting directions of future work. We sketch out two directions of particular interest. First, it is worth observing that,

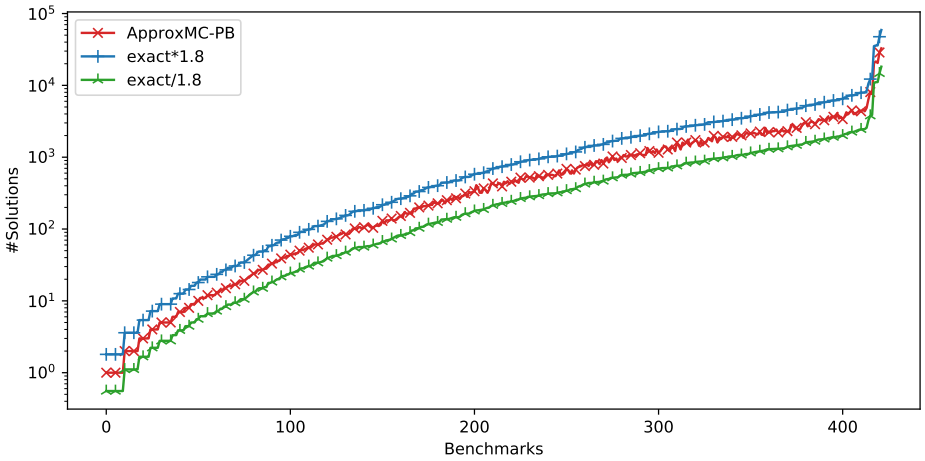

Figure 5 Plot showing counts obtained by ApproxMCPB vis-a-vis exact counts.

unlike modern CNF solvers, the PB solvers are still in the nascent phase, and consequently lack intricate efficient preprocessing techniques. Therefore, in our design of LinPB, we did not adapt the BIRD architecture [35], which was designed to efficiently transform XOR constraints between clauses and native representation, aiming to utilize the powerful inprocessing technique of CNF. The development of efficient inprocessing for PB constraints would invite extending LinPB with a BIRD-eseque architecture. Secondly, the significant performance improvements of Mixed-Watches over Eager-GJE and Shared-Watches leads us to speculate that adoption of these strategies in the context of CNF-XOR solving would also lead to performance improvements.

References

1 Dimitris Achlioptas, Zayd S. Hammoudeh, and Panos Theodoropoulos. Fast sampling of perfectly uniform satisfying assignments. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing – SAT 2018*, pages 135–147, Cham, 2018. Springer International Publishing.

2 Teodora Baluta, Shiqi Shen, Shweta Shine, Kuldeep S. Meel, and Prateek Saxena. Quantitative verification of neural networks and its security applications. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 11 2019.

3 D. Chai and A. Kuehlmann. A fast pseudo-boolean constraint solver. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(3):305–317, 2005. doi:10.1109/TCAD.2004.842808.

4 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *Proceedings of International Conference on Constraint Programming (CP)*, pages 200–216, 9 2013.

5 Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. Improving approximate counting for probabilistic inference: From linear to logarithmic sat solver calls. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 7 2016.

6 Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA, 1971. Association for Computing Machinery. doi:10.1145/800157.805047.

7 W. Cook, C.R. Coullard, and Gy. Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987. URL: https://www.sciencedirect.com/science/article/pii/0166218X87900394, doi:https://doi.org/10.1016/0166-218X(87)90039-4.

8 Jo Devriendt. Watched propagation of - integer linear constraints. In *Proceedings of International Conference on Constraint Programming (CP)*, pages 160–176, 09 2020. doi:10.1007/978-3-030-58475-7_10.

9 Heidi Dixon, Matt Ginsberg, and Andrew Parkes. Generalizing boolean satisfiability i: Background and survey of existing work. *J. Artif. Intell. Res. (JAIR)*, 21:193–243, 02 2004. doi:10.1613/jair.1353.

10 Heidi E. Dixon and Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *Eighteenth National Conference on Artificial Intelligence*, page 635–640, USA, 2002. American Association for Artificial Intelligence.

11 N. Eén and Niklas Sörensson. Translating pseudo-boolean constraints into sat. *J. Satisf. Boolean Model. Comput.*, 2:1–26, 2006.

12 Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-boolean solving. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1291–1299. International Joint Conferences on Artificial Intelligence Organization, 7 2018. doi:10.24963/ijcai.2018/180.

13 Stefano Ermon, Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Taming the curse of dimensionality: Discrete integration by hashing and optimization. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page II–334–II–342. JMLR.org, 2013.

14 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, page 54–61. AAAI Press, 2006.

15 Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985. Third Conference on Foundations of Software Technology and Theoretical Computer Science. URL: https://www.sciencedirect.com/science/article/pii/0304397585901446, doi:https://doi.org/10.1016/0304-3975(85)90144-6.

16 Cheng-Shen Han and Jie-Hong Roland Jiang. When boolean satisfiability meets gaussian elimination in a simplex way. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification*, pages 410–426, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

17 John Hooker. Generalized resolution for 0–1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 03 1992. doi:10.1007/BF01531033.

18 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-boolean constraints. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, 2015. doi:10.1007/978-3-319-23219-5_15.

19 Jean-Marie Lagniez and Pierre Marquis. A recursive algorithm for projected model counting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1536–1543, Jul. 2019. URL: https://ojs.aaai.org/index.php/AAAI/article/view/3967, doi:10.1609/aaai.v33i01.33011536.

20 Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7:59–6, 01 2010.

21 Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3), 1973.

22 J.P. Marques-Silva and K.A. Sakallah. Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.

23 Ruben Martins, Vasco Manquinho, and Inês Lynce. Open-wbo: A modular maxsat solver,. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing – SAT 2014*, pages 438–445, Cham, 2014. Springer International Publishing.

24 Kuldeep S. Meel. *Constrained Counting and Sampling: Bridging the Gap between Theory and Practice*. PhD thesis, Rice University, 2017.

25 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 *Proceedings of Workshop on Beyond NP(BNP)*, 2016.

26 Alexander Nadel and Vadim Ryvchin. Chronological backtracking. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing – SAT 2018*, pages 111–121, Cham, 2018. Springer International Publishing.

27 Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying Properties of Binarized Deep Neural Networks. *arXiv e-prints*, page arXiv:1709.06662, September 2017. `arXiv:1709.06662`.

28 Olivier Roussel and Vasco Manquinho. Pseudo-boolean and cardinality constraints. In *Handbook of satisfiability*, pages 695–733. IOS Press, 2009.

29 Kenji Hashimoto Ryosuke Suzuki and Masahiko Sakai. Improvement of projected model-counting solver with component decomposition using sat solving in components. *JSAI Technical Report*, SIG-FPAI-103-B506:31–36, Mar. 2017. in Japanese.

30 Masahiko SAKAI and Hidetomo NABESHIMA. Construction of an robdd for a pb-constraint in band form and related techniques for pb-solvers. *IEICE Transactions on Information and Systems*, E98.D(6):1121–1127, 2015. `doi:10.1587/transinf.2014FOP0007`.

31 Shubham Sharma, Subhajit Roy, Mate Soos, and Kuldeep S. Meel. Ganak: A scalable probabilistic exact model counter. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

32 H.M. Sheini and K.A. Sakallah. Pueblo: a modern pseudo-boolean sat solver. In *Design, Automation and Test in Europe*, pages 684–685 Vol. 2, 2005. `doi:10.1109/DATE.2005.246`.

33 Hossein Sheini and Karem Sakallah. Pueblo: A hybrid pseudo-boolean sat solver. *JSAT*, 2:165–189, 03 2006. `doi:10.3233/SAT190020`.

34 Mate Soos, Stephan Gocht, and Kuldeep S. Meel. Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling. In *Proceedings of International Conference on Computer-Aided Verification (CAV)*, 7 2020.

35 Mate Soos and Kuldeep S. Meel. Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 1 2019.

36 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending sat solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 244–257, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

37 G. S. Tseitin. On the complexity of derivation in propositional calculus. *Automation of Reasoning*, pages 466–483, 1983. `doi:10.1007/978-3-642-81955-1_28`.

38 Alasdair Urquhart. Hard examples for resolution. *J. ACM*, 34(1):209–219, January 1987. `doi:10.1145/7531.8928`.

A Pseudo-Boolean Encoding of Binarized Neural Network

Conditional pseudo-Boolean constraint is a fundamental building block to binarized neural network (BNN). We introduce the pseudo-Boolean encoding of conditional pseudo-Boolean constraint in Definition 3. Then we sketch the idea to encode BNN on top of conditional pseudo-Boolean constraints.

▶ **Definition 3.** *Given a conditional pseudo-Boolean constraint* $\phi : y \rightarrow \Sigma_{i=1}^{N} w_i x_i$ *op* b*, op* $\in \{\geq, \leq\}$*,* $w_i, b \in \mathbb{Z}$*,* $x_i, y \in \{0, 1\}$*, we define as the pseudo-Boolean encoding of* ϕ*:*

$$\begin{cases} C = \Sigma_{i=1}^{i=N} |w_i| \\ \Sigma_{i=1}^{i=N} w_i x_i + (b + C)\neg y \geq b \quad \textit{if op is } \geq \\ \Sigma_{i=1}^{i=N} w_i x_i + (b - C)\neg y \leq b \quad \textit{if op is } \leq \end{cases} \tag{2}$$

The neuron, a basic component in binarized neural network can be logically represented by the constraint:

$$y \leftrightarrow \Sigma_{i=1}^{N} w_i x_i + b \geq 0 \tag{3}$$

in which $w_i \in \{+1, -1\}$. Formula (3) is equivalent to:

$$y \rightarrow \Sigma_{i=1}^{N} w_i x_i + b \geq 0 \wedge \neg y \rightarrow \Sigma_{i=1}^{N} w_i x_i + b < 0 \tag{4}$$

Finally, we encode formula (4) based on conditional pseudo-Boolean constraints introduced in Definition 3. The complete encoding is shown as follows.

Let's consider the k-th block of BNN: $BLK_k : v^k \rightarrow v^{k+1}$ ($v^k, v^{k+1} \in \{1, -1\}^N$) including

1. linear layer$(w^k, b^k) : v^k \rightarrow o^{lin}$
$o_i^{lin} = \Sigma_{i=1}^{N} w_i^k v_i^k + b_i^k$
$(v^k, w^k \in \{+1, -1\}^N,\ b, o^{lin} \in \mathbb{R}^N)$

2. batch normalization layer$(\mu^k, \sigma^k, \alpha^k, \gamma^k) : o^{lin} \rightarrow o^{bn}$
$o_i^{bn} = \frac{t_i^{lin} - \mu_i^k}{\sigma_i^k} \cdot \alpha_i^k + \gamma_i^k$
$(\mu^k, \sigma^k, \alpha^k, \gamma^k, o^{lin}, o^{bn} \in \mathbb{R}^N)$

3. binarization layer: $o^{bn} \rightarrow v^{k+1}$
$v_i^{k+1} = 1 \leftrightarrow o_i^{bn} \geq 0$
$(v^{k+1} \in \{1, -1\}^N,\ o^{bn} \in \mathbb{R}^N)$

According to the encoding in "Quantitative Verification of Neural Networks and Its Security Applications" by Teo, we can get the following constraint for each neuron when $\alpha > 0$ (In following constraints $v_i^{k+1}, v_i^k \in \{0, 1\}$ because we have transferred them into boolean variables):

$$\begin{cases} v_i^{k+1} = 1 \leftrightarrow \Sigma_{i=1}^{N} w_i^k v_i^k \geq C_i'^k \\ C_i'^k = \left\lceil \frac{C_i^k + \Sigma_{i=1}^{N} w_i}{2} \right\rceil \\ C_i^k = \left\lceil -\frac{\sigma_i^k}{\alpha_i^k} \gamma_i^k + \mu_i^k - b_i^k \right\rceil \end{cases} \tag{5}$$

By Eq. 2, Eq 3, Eq. 4, we can get:

$$\begin{cases} \Sigma_{i=1}^{N} w_i^k v_i^k + \beta_i^k \neg v_i^{k+1} \geq C_i'^k \\ \beta_i^k = C_i'^k + N \\ -\Sigma_{i=1}^{N} w_i^k v_i^k + \beta_i'^k v_i^{k+1} \geq 1 - C_i'^k \\ \beta_i'^k = N + 1 - C_i'^k \end{cases} \tag{6}$$

Note that β_i^k and $\beta_i'^k$ are constants. The other two are linear encoding. Similarly we can get constraints for $\alpha < 0$:

$$\begin{cases} v_i^{k+1} = 1 \leftrightarrow \Sigma_{i=1}^{N} w_i^k v_i^k \leq C_i'^k \\ C_i'^k = \left\lfloor \frac{C_i^k + \Sigma_{i=1}^{N} w_i}{2} \right\rfloor \\ C_i^k = \left\lfloor -\frac{\sigma_i^k}{\alpha_i^k} \gamma_i^k + \mu_i^k - b_i^k \right\rfloor \\ -\Sigma_{i=1}^{N} w_i^k v_i^k + \beta_i^k \neg v_i^{k+1} \geq -C_i'^k \\ \beta_i^k = -C_i'^k + N \\ \Sigma_{i=1}^{N} w_i^k v_i^k + \beta_i'^k v_i^{k+1} \geq 1 + C_i'^k \\ \beta_i'^k = N + 1 + C_i'^k \end{cases} \tag{7}$$

Corner case when $\alpha = 0$:

$$v_i^{k+1} = 1 \leftrightarrow \gamma_i^k \geq 0 \tag{8}$$

B Configuration of Mixed-Watches

In this section, we focus on examining the integration compatibility of Mixed-Watches with Lazy-GJE, Eager-GJE, and Shared-Watches with different heuristics for Mixed-Watches. To this end, we focus on the following heuristics:

1. CA-reason: Addition of the reason constraint generated from XOR-propagation and used by conflict analysis to PB constraints.
2. All-reason: Addition of all the reason constraints generated from XOR-propagation to PB constraints.
3. Confl: Addition of the conflict constraints detected from XOR-propagation to PB constraints.

		Heuristics				
GJE tactics	No mixed[7]	CA-reason	All-reason	Confl	CA-reason ∪ Confl	All-reason ∪ Confl
Lazy-GJE	804 (2755)	909 (1834)	908 (1855)	897 (1961)	**912 (1822)**	908 (1854)
Eager-GJE	892 (2042)	881 (2172)	801 (2776)	889 (2071)	881 (2180)	801 (2777)
Shared-Watches	892 (2017)	907 (1850)	902 (1951)	909 (1839)	907 (1860)	900 (1976)

[7] The original tactic without Mixed-Watches.

Table 5 Number of solved benchmarks for Mixed-Watches integrated with other GJE tactics and applying different heuristics. PAR-2 score is in the parentheses. The heuristic means adding the corresponding constraints from XOR-propagation to PB constraints. Time out after 5000s.

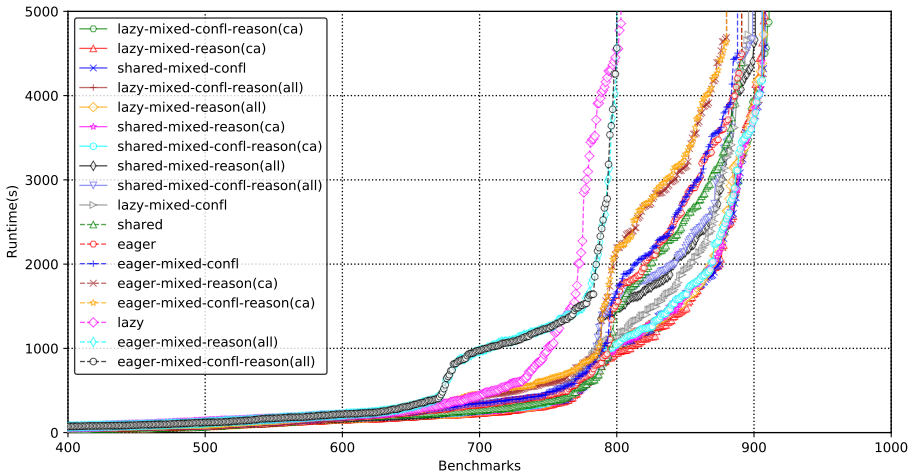

Figure 6 Runtime for Mixed-Watches integrated with other GJE tactics and applying different heuristics. The x-axis represents the number of solved benchmarks, while the y-axis shows the counting time. A point (x, y) represents that x benchmarks can be solved within the runtime threshold y. The number of solved benchmarks sorts counters in descending order in the legend. Time out after 5000s.

Table 5 summarizes the results. The first column shows the GJE tactic integrated with Mixed-Watches. The second column presents the number of solved benchmarks and PAR-2 score in the parentheses for the original tactic without Mixed-Watches, while the following columns show the result for GJE tactics integrated with different heuristics of Mixed-Watches. The third, fourth, and fifth columns refer to CA-reason, All-reason, and Confl heuristics while the last two columns refer to combination of the aforementioned heuristics.

The bold cell in Table 5 highlights that a Mixed-Watches integrated with Lazy-GJE and employing CA-reason and Confl heuristics, solved the most number of benchmarks and achieved the smallest PAR-2 score. Furthermore, we observe that Mixed-Watches improves the performance Lazy-GJE by around one hundred more solved benchmarks from 804 to 912 and improves the performance of Shared-Watches by a dozen solved benchmarks while making Eager-GJE solve fewer benchmarks than the original tactic. On the other hand, Table 5 summarizes that learning both conflict and reason constraints used by conflict analysis (CA-reason) from XOR-propagation is the best heuristic for Lazy-GJE based Mixed-Watches. All heuristics involving CA-reason constraints always solves more benchmarks than All-reason.

To provide a comprehensive picture, we present the cactus plot in Figure 6 for different combinations. The legend of the Figure has all the combinations sorted in descending order by the number of solved benchmarks.