Exploring Toda's Theorem as a QBF Solver

Dror Fried
The Open University of Israel, Ra'anana, Israel

Etay Segal
The Open University of Israel, Ra'anana, Israel

Gad E. Yaron [1]
The Open University of Israel, Ra'anana, Israel
The Hebrew University of Jerusalem, Jerusalem, Israel

Abstract

Toda's Theorem is a fundamental result in computational complexity theory, whose proof is based on a reduction from a QBF problem with a constant number of quantifiers to a model counting problem. The recent progress in model counting tools raises the question of whether this reduction, henceforth called Toda's reduction, can be utilized to construct a practical QBF solver. This question follows a line of research that revisits theoretical results from an algorithmic aspect, thus brings new theoretical and engineering challenges. For Toda's reduction these challenges arise mainly because the reduction is purely theoretical and based on ideas that are entirely orthogonal to the search-space approach used by current QBF solvers. In this work, we address this question by transforming Toda's reduction into a concrete probabilistic QBF solver that uses model counting as an oracle. A naive implementation is hopeless due to a massive formula blow-up. Therefore we next identify three main factors that drive the blow-up. While we present solutions that overcome some of the factors, we also show that one of them, the union bound factor, largely overlooked in the literature, is in fact dominant and in some cases unavoidable. We then show how for some cases, even this factor can be avoided, and report our preliminary results on a prototype implementation.

1 Introduction

One of the most fundamental results in computational complexity is Toda's Theorem [13] which shows that the polynomial hierarchy is contained in $P^{\#SAT}$. Namely, for any constant $d > 0$, the satisfiability of every quantified Boolean formula (QBF) F with d quantifier alternations can be decided by a polynomial-time algorithm (polynomial in the size of F), that makes an oracle call to a model counter solver. Toda's theorem is constructive in nature and could conceptually serve as a basis for a QBF solver since many QBF instances arising in practice have only a small number of alternations. Nevertheless, Toda's theorem has never been proposed or analyzed as a concrete algorithm for QBF solving. One reason may be its theoretical nature, which like many other complexity theoretic results, maintains an elegant clarity over an algorithmic description. A more practical reason may be the severe blow-up in the size of the resulting formula given to the model counter.

Recently, however, the importance of bridging the gap between theory and practice has gained acknowledgment, with more and more works that explore algorithmic aspects of pure theoretical problems (e.g. [3, 2]). Moreover, since the Toda-based approach is orthogonal to current search-based QBF solvers, a successful Toda-based algorithm for solving QBF could both provide new theoretical insights, and perhaps solve instances in which other solvers struggle. Consequently, in this work we revisit Toda's Theorem from an algorithmic perspective as a QBF solver. We focus on the first part of Toda's theorem, which shows that PH is contained in $\mathrm{BPP}^{\oplus \mathrm{P}}$. From this part, which we call *Toda's reduction*, we construct

* Corresponding author

and implement a probabilistic algorithm that transforms a given QBF formula F with d alternations, into a quantifier-free Boolean formula F_0, such that with high probability, F is *true* if and only if F_0 has an *odd* number of satisfying assignments. Then, once F_0 is established, a model counter is used to determine the parity of F_0. Although probabilistic, such an efficient QBF solver has a potential to shed some light in cases in which other QBF solvers struggle. As expected, our naive implementation is hopeless: Even a simple QBF formula with only three quantifiers, five variables, and ten clauses, results in a quantifier free formula with over 2 million clauses for the model counter to solve.

The reason behind this blow-up is the huge number of repetitions in every step of the reduction that ends in a quantifier free Boolean formula that is enormously larger than the input formula. The Valiant-Vazirani hash function, which lies in the heart of the reduction, seems to be the obvious reason for these repetitions. Our analysis points, however, to not one, but three factors, that play an important role in this blow-up. The first is the allocation of the error probabilities along the steps. The second factor is the success-probability that Valiant-Vazirani hash function provide. The third and final factor stems from union bound considerations. We present methods to battle the first two factors. Then we show that in some cases, the third union bound factor, and the blow-up that follows, is unavoidable. On the other hand, we describe a class of formulas, for which this union bound factor is small. Our preliminary results show that our solutions improve the naive implementation by several orders of magnitude, and that our resulting tool TodaQBF can solve QBF benchmarks from the QBFEVAL competition. Nevertheless, our results so far still fall short of any competitive QBF solver. We end up by discussing the further steps that are required in order to assess whether a QBF solver based on Toda's algorithm is an effective direction to pursue.

2 Preliminaries

A Boolean formula $F(\vec{x})$ over a set of variables $\vec{x} = \{x_1, \ldots, x_d\}$ is a logical expression that evaluates to either *true* or *false* . We denote by $|F|$ the *size* of F, which is the length of the string representing F. A *solution* (also called a *satisfying assignment* or a *model*) of F is an assignment for F that evaluates F to be *true*. The number of solutions for F is denoted by $\#F$. A quantified Boolean formula (QBF) (in a Prenex-Normal Form) is a formula of the form: $F \equiv Q_1\vec{x}_1 \ldots Q_d\vec{x}_d\, \hat{F}(\vec{x}_1, \ldots, \vec{x}_d)$. Each Q_i is either an existential quantifier ($\exists$) or a universal quantifier ($\forall$), and $\hat{F}$ is a quantifier-free Boolean formula. The string of quantifiers $Q_1\vec{x}_1 \ldots Q_d\vec{x}_d$ is called the *prefix* of F. The set of variables $\vec{x}_i$ is called the *block* of Q_i.

We next define several Boolean operators. We first define the $+$ operator used throughout the paper. For Boolean formulas $F(\vec{x})$ and $G(\vec{y})$, where $\vec{x}, \vec{y}$ are not necessarily disjoint, the term $F(\vec{x}) + G(\vec{y})$ denotes the formula $[F(\vec{x}) \wedge x_{new}] \vee [\neg x_{new} \wedge G(\vec{y})]$ where x_{new} is a fresh variable. Note that $\#[F(\vec{x}) + G(\vec{y})] = \#F(\vec{x}) + \#G(\vec{y})$. Then the term $F(\vec{x}) + 1(\vec{x})$ where $\vec{x} = (x_1, \ldots, x_m)$ denotes $[F(\vec{x}) \wedge x_{new}] \vee [\neg x_{new} \wedge x_1 \wedge \ldots \wedge x_m]$. Specifically the formula $x_1 \wedge \cdots \wedge x_m$ has exactly one satisfying assignment, hence $\#[F(\vec{x}) + 1(\vec{x})] = \#F(\vec{x}) + 1$. Then, we define the *parity quantifier* $\bigoplus$ as follows: $\bigoplus\vec{x}F(\vec{x}) = \mathit{true}$ if $\#F(\vec{x}) \bmod 2 = 1$ and *false* otherwise.

Finally, we discuss hash functions. A key component in Toda's reduction is the use of hash functions to probabilistically sieve an odd number of solutions. The classical approach uses the Valiant-Vazirani Theorem [14], which isolates a single solution with probability at least $1/(8n)$. Given a randomly chosen hash function h, we use the term "success probability" to refer to the probability that $F \wedge h$ has an odd number of satisfying assignments.

3 Toda's Reduction: From Theory to Algorithm

In order to convert Toda's reduction to an algorithm, we first need to uncover the fine details of the reduction. In a high level, the reduction converts a given QBF formula F to a parity-quantified formula $F_0 = \bigoplus G$, followed by checking (by using e.g. a model counter) the parity of the number of solutions for G. The promise of the reduction is that with high probability, F is *true* if and only if F_0 is *true* or in other words if the number of solutions for G is *odd*.

Toda's Reduction Step by Step

Building upon the Valiant-Vazirani technique of probabilistically isolating a single solution, Toda's reduction presents a systematic method for transforming any quantified Boolean formula into a formula that uses only a single parity quantifier ($\bigoplus$).

We start with a given formula $F_d = Q_1\vec{x}_1, \cdots Q_d\vec{x}_d\hat{F}(\vec{x}_1, \cdots, \vec{x}_d)$, and $0 < \epsilon < 1/2$. First, all universal quantifiers ($\forall\vec{x}$) are replaced with negated existential quantifiers ($\neg\exists\vec{x}\neg$), resulting in a prefix of the form $\neg\exists\vec{x}_1\neg\exists\vec{x}_2\neg\cdots\neg\exists\vec{x}_d\neg\hat{F}$. The process then proceeds inductively from step d down to step 1, where F_0 is the resulting formula. Suppose at step i we have: $F_i = \neg\exists\vec{x}_1\neg\cdots\neg\exists\vec{x}_i\bigoplus\vec{y}_i G_i(\vec{x}_1, \cdots, \vec{x}_i, \vec{y}_i)$, where G_i is quantifier free and $\vec{y}_i$ contains variables from previous steps. Denote the *outer variables* $(\vec{x}_1, \ldots, \vec{x}_{i-1})$ by $\vec{x}$, with $s = |\vec{x}|$. To eliminate $\exists\vec{x}_i$, we apply a hash function h over $\vec{x}_i$. Valiant-Vazirani ensures that for each assignment $\vec{\sigma}$ to $\vec{x}$: if the inner formula has no solutions then the hashed formula has none; if, however, solutions exist, then with probability at least p (where $p \geq 1/(8n)$ for Valiant-Vazirani), the hashed formula has an odd number of solutions.

To amplify this probability, we repeat k times with independent hash functions, we then combine the repetitions and take care of the $\neg$ that is adjacent to that quantifier using properties of the parity-quantifier (for more information about these properties see the full proof at [1]). All in all we end with:

$$F''_i \equiv \vec{Q}\vec{x}\bigoplus_{\vec{x}_i^1,\vec{y}_i^1,\ldots,\vec{x}_i^k,\vec{y}_i^k}\Big[\big[(G_i(\vec{x},\vec{x}_i^1,\vec{y}_i^1)\wedge h_1(\vec{x}_i^1)) + 1(\vec{x}_i^1,\vec{y}_i^1)\big] \wedge\cdots\wedge \big[(G_i(\vec{x},\vec{x}_i^k,\vec{y}_i^k)\wedge h_k(\vec{x}_i^k)) + 1(\vec{x}_i^k,\vec{y}_i^k)\big]\Big] \tag{1}$$

Note that each repetition j uses fresh copies $\vec{x}_i^j, \vec{y}_i^j$ of the inner variables, while the outer variables $\vec{x}$ remain shared across all copies.

The promise that we get at the end of each step is as follows: for every given $\epsilon_i > 0$, there is a number k_i of repetitions, polynomial in the size of F_i, such that if F_i is *true* then F_{i-1} is *true* with probability of at least $1 - \epsilon_i$, and if F_i is *false* then F_{i-1} is *false* with probability of at least $1 - \epsilon_i$. To achieve the overall error bound ϵ across all of the steps, the error of each step ϵ_i has to be bounded. The standard approach, following [1], uses a *geometric allocation*: setting $\epsilon_i = \epsilon/2^i$ for each step i. By a union bound over the d steps, $\Pr(\text{any error}) \leq \sum_{i=1}^{d}\epsilon_i = \epsilon\sum_{i=1}^{d}1/2^i < \epsilon$. The overall result is then:

▶ **Theorem 1** (Toda's Reduction [1]). *Let F be a QBF formula with d alternations, and $0 < \epsilon < 1/2$. Then we have the following: If F is true, then with probability at least $1 - \epsilon$, F_0 is true. If F is false, then with probability at least $1 - \epsilon$, F_0 is false.*

Toda's Reduction based Algorithm

Following the description of Toda's reduction in the previous section, we next present Algorithm 1 that implements the reduction. The algorithm proceeds as follows. We are given

a QBF formula F, with the number of alternations d, and a probability measure $\epsilon > 0$. First, in Line 2, we compute an *inner-error* probability value ϵ_i for each quantifier level i. This allocation of error probability at every step ensures that the overall algorithm maintains the desired error bound ϵ. Lines 3-5 and 10-12 corresponds to the first part where we converted the formula to be of the form $\neg\exists\vec{x}_1 \cdots \neg\exists\vec{x}_d\neg\hat{F}$, which naturally is formed a bit differently depending of whether the inner and/or outer quantifier is $\forall$ or $\exists$. The core of the algorithm is in Lines 7-9, where we iterate through all quantifiers from innermost to outermost. We start in Step d and decrease to Step 1. For each quantifier we apply the **Amplify** function, which applies a hash function and amplifies using repetitions as defined in the reduction. Finally, in Line 13 we return F_0, that contains only parity quantifiers, having eliminated all existential and universal quantifiers from the original formula.

We next describe the function **Amplify**, shown in Algorithm 2. In Line 2 we calculate s, the total number of variables in the outer quantifiers. Using s, ϵ and the $\vec{x}_i$ variables, we then determine the number of repetitions k_i needed to achieve the desired error probability by using the **ComputeRepetitions** function, described in the next section. Then in Line 5, for each repetition, we AND the function with a **hash**, where the hash is the hash function from Valiant-Vazirani, as described in the Preliminaries. In line 6 we **Rename** the variables from the current and previous quantifiers, giving each repetition its own fresh copy of variables. Finally, in Line 8 we **Combine** all of the repetitions as described in the reduction above.

The Practicality of a Naive Implementation

Using the implementation as is, even for a simple 3QBF formula with 5 variables and bounding the error to only $\epsilon = 0.1$ results in a formula more than 200,000 times larger than the original. Even starting with a small formula of just 10 clauses, the transformed formula would exceed 2,000,000 clauses. Such a blow-up makes a Toda-based solver impractical even on the smallest formulas. This blow-up is addressed in the next sections.

Algorithm 1 Toda's Reduction

Input: QBF formula $\hat{F} = Q_1\vec{x}_1, \cdots Q_d\vec{x}_d F(\vec{x}_1, \cdots, \vec{x}_d)$, d alternations,error bound $\epsilon > 0$

Output: Parity-quantified formula

1 **begin**
2 $\quad\epsilon_1, \ldots, \epsilon_d \leftarrow$ `ComputeInnerError`(ϵ, d)
3 $\quad$**if** $Q_d \equiv \forall$ **then**
4 $\quad\quad F \leftarrow \neg F$
5 $\quad$**end**
6 $\quad F_d \leftarrow F$
7 $\quad$**for** $i \leftarrow d$ **to** 1 **do**
8 $\quad\quad F_{i-1}(\vec{x}_1, \cdots, \vec{x}_{i-1}, \vec{y}_{i-1}) \leftarrow$ `Amplify`$(F_i, \vec{x}_1, \cdots, \vec{x}_i, \vec{y}_i, \epsilon_i)$
9 $\quad$**end**
10 $\quad$**if** $Q_1 \equiv \exists$ **then**
11 $\quad\quad F_0 \leftarrow F_0 + 1$
12 $\quad$**end**
13 $\quad$**return** $\bigoplus\vec{y}_0\hat{F}_0(\vec{y}_0)$
14 **end**

Algorithm 2 Amplify

Input: Boolean formula $F(\vec{x}_1, \cdots, \vec{x}_i, \vec{y}_i)$, $\vec{x}_1, \cdots, \vec{x}_i, \vec{y}_i, \epsilon > 0$,
Output: Boolean Formula $\hat{F}$

```
begin
    s ← (Σ_{j=1}^{j=i-1} |x_j|)
    k_i ← ComputeRepetitions(s, |x_i|, ε)        ▷ See next section
    for j ← 1 to k_i do
        F_j = (F ∧ hash(x_i))
        Rename(F_j, x_i, y_i)
    end
    return Combine(F_1, ··· , F_{k_i})
end
```

4 Analyzing the Algorithm's Blow-Up

We now analyze the key components that determine the number of repetitions at each step, and hence the size of the resulting formula. Our analysis reveals three independent factors, each contributing multiplicatively to the blow-up. For each quantifier elimination step, we need to determine how many repetitions of the hash technique are required to achieve the desired inner-error probability. Let p be the success probability of a single hash application, and let k_i be the number of repetitions at step i.

Recall that the outer variables $\vec{x}$ are not constrained by the hash function. There are 2^s possible assignments to $\vec{x}$, and for each assignment $\vec{\sigma}$, the formula $\phi(\vec{\sigma}, \vec{y})$ defines a different Boolean function over the inner variables. When we apply the hash-based sieve, we are essentially applying it to all 2^s of these inner formulas concurrently. Thus, for the reduction to succeed, we need the sieve to preserve the correct truth value for *every* such assignment simultaneously. For a single fixed assignment $\vec{\sigma}$, the probability that all k_i independent hash applications fail is at most $(1-p)^{k_i}$. However, correctness must hold across all 2^s assignments. Using a union bound over the 2^s possible assignments: $\Pr(\text{failure for any assignment}) \leq 2^s \cdot (1-p)^{k_i}$. Requiring this to be at most ϵ_i gives $(1-p)^{k_i} \leq \epsilon_i / 2^s$, from which we get:

▶ **Proposition 2.** *Setting $k_i \geq \log_{1-p}(\epsilon_i / 2^s)$ ensures that the probability for error at step i is at most ϵ_i. This decomposes as:* $k_i \geq \underbrace{\log_{1-p}(\epsilon_i)}_{\text{error term}} + \underbrace{s \cdot \log_{1-p}(1/2)}_{\text{union bound term}}$

Proposition 2 reveals three independent factors. The first factor ϵ_i is the *allowed error-rate* in each step i. The second factor p is the *hash success-probability* in a single use of a hash function to isolate an odd number of solutions (e.g. Valiant-Vazirani's hash function). The third and last factor s, is the *union bound factor* which equals the number of outer variables.

To get a concrete sense of scale, we replace p with $1/8n$, which is the success probability bound for the Valiant-Vazirani's hash function. Then, we convert this expression to natural logarithms and use the approximation $\ln(1-p) \approx -p$ (which holds since p is small). All in all we get $k_i \geq\approx 8n \ln(1/\epsilon_i) + 5.5n \cdot s$. In other words: The union bound and hash probability factors (s and p) currently affects the number of repetitions linearly while the error allocation factor (ϵ_i) affects only logarithmically. In the next section we explain how to battle these factors.

5 Addressing the Blow-Up Factors

We now present our methods for reducing each of the three factors identified in the previous section.

Addressing the Allowed Error-Rate Factor

The geometric allocation $\epsilon_i = \epsilon/2^i$ seems wasteful. To find a better alternative, we first explain *how* each type of step can produce an error. At an existential step we use hashing to try and isolate an odd number of solutions. More precisely we transform $\exists \vec{y}\ F(y)$ to $\bigoplus \vec{y}\ F(y) \wedge h(y)$. If the formula is *true* (solutions exists), the step might fail to isolate only an odd number of solutions producing a false result. But if the formula is *false* then F has no solutions at all and as a result $F \wedge h$ will always have no solutions and the reduction will always work. Thus a *false* formula never becomes *true* through an existential step. Using the same logic for a universal step, the situation is analogous but vice versa.

Let $q_\exists \in [p, 1]$ be the actual success probability at an existential step, and $q_\forall \in [p, 1]$ at a universal step. The error behavior is summarized below:

	Existential step	**Universal step**
Pr(true stays true)	$q_\exists$	1 (always)
Pr(true becomes false)	$1 - q_\exists$	0 (never)
Pr(false stays false)	1 (always)	$q_\forall$
Pr(false becomes true)	0 (never)	$1 - q_\forall$

Next, the usage of the hash functions guarantees success probability *at least* p, where the actual probability can be anywhere in $[p, 1]$. In particular, a step may work better than the promised lower bound. While this sounds harmless, it can actually *increase* the overall error. Consider a $\forall\exists$ pair: if the $\exists$ step fails, the subsequent $\forall$ step might also "fail", accidentally restoring the correct answer. This "rescue effect" depends on the $\forall$ step having a nonzero failure probability. But if the $\forall$ step works perfectly ($q_\forall = 1$), no rescue occurs.

To find the worst case, we allow an adversary to choose all actual probabilities in $[p, 1]$. Analysis shows that the adversary's optimal strategy is to entirely eliminate the "rescue effect" by setting to 1 the probabilities of all the $\exists$ steps (or all the $\forall$ steps, depending on the formula). Given the scope of this paper, we omit the technical details of this analysis. This leaves us with $l = \lceil d/2 \rceil$ independent steps that must all succeed. This leads us to:

▶ **Theorem 3** (Balanced Allocation). *Setting* $\epsilon_i = 1 - (1 - \epsilon)^{1/\lceil d/2 \rceil}$ *for all steps ensures the overall error is at most* ϵ.

Addressing the Hash Success-Probability Factor

The Valiant-Vazirani hash function provides $p = O(1/n)$, which enters as a multiplicative factor. As observed in [7, 5, 10], while the original goal of the Valiant-Vazirani hash was to isolate a single solution for Toda's reduction it suffices only to isolate an odd number of solutions. This weaker requirement allows the usage of much stronger hash functions: Naik, Regan, and Sivakumar [10] achieve success probability arbitrarily close to 1/2, and Gupta [8] reaches exactly 1/2. We present here a construction that is specializing a family introduced by Naik, Regan, and Sivakumar [10].

▶ **Definition 4** (Multilinear hash). *Let* $\mathbb{F} = GF(2^k)$ *with* $k = \lceil \log_2 n + \ell - 1 \rceil$. *Sample* $a_1, \ldots, a_n \in \mathbb{F}$ *and* $v \in \{0,1\}^k$ *uniformly at random. Define:* $h_{a_1,\ldots,a_n,v}(x_1, \ldots, x_n) = \langle \prod_{i=1}^n a_i^{x_i}, v \rangle$, *where* $\langle \cdot, \cdot \rangle$ *is the inner product modulo* 2.

▶ **Theorem 5.** *If f is satisfiable, then* $\Pr[\#[f \wedge h] \text{ is odd}] \geq 1/2 - 2^{-\ell}$.

For the proof, which uses error-correcting codes, see [10]. Then by replacing the Valiant-Vazirani hash function with the multilinear hash, and setting $\ell = 3$, we get $p \geq 7/16 \approx 0.44$—a dramatic improvement over Valiant-Vazirani's $p = O(1/n)$.

In addition, we next introduce a method to construct stronger hash functions from weaker ones, which we call *modular addition*. The construction follows a classical result on random walks on groups (see [6]): the parity of the sum of independent copies of a bounded integer random variable converges to uniform. Translating this to hash function construction works as follows. Let H be a family of hash functions with base success probability p_{base}. Given a satisfiable formula $F(\vec{x})$, for a randomly chosen $h \in H$, the number of solutions $\#[F \wedge h]$ is a random variable X. Look at: $D_l = F(\vec{x}) \wedge (h_1(\vec{x}) + h_2(\vec{x}) + \cdots + h_l(\vec{x}))$. Since $\#[F \wedge (h_1 + h_2)] = \#[F \wedge h_1] + \#[F \wedge h_2]$, the model count of D_l equals the sum of l independent copies of X, which according to the mentioned result approaches to uniform over parity. More precisely we get the following (see [6] for proof):

▶ **Theorem 6.** *The success probability for D_l is:* $p_l = \frac{1}{2}\left[1 - (1 - 2p_{\text{base}})^l\right]$.

We note that empirically, it is beneficial to pair the multilinear hash with modular addition using a smaller accuracy parameter ℓ (and hence a smaller field, resulting in a smaller circuit) while still achieving high overall success probability.

Addressing the Union Bound Factor

It is well known that the union bound is practically tight for independent events [4]. When different outer assignments yield independent inner behaviors, the sieve's outcomes on different assignments are independent, meaning this term is not a by-product of a loose analysis, but rather a structural limit of this approach. Since the union bound factor is dominant this means that currently, for formulas that hold such a behavior, using Toda based QBF solver is most likely impractical.

Nonetheless, this observation holds true only when outer assignments yield independent behaviors, which may not always be the case. Recall that the 2^s factor in Proposition 2 arises because we apply a union bound over all 2^s possible assignments $\sigma \in \{0,1\}^s$ to the outer variables. If however, two outer assignments σ_1, σ_2 induce the same inner function ($\phi(\sigma_1, \vec{y}) \equiv \phi(\sigma_2, \vec{y})$ for all $\vec{y}$) then they behave identically under the hash. Note that this does not mean that $\phi(\sigma_1, \vec{y})$, $\phi(\sigma_2, \vec{y})$ have to be syntactically equivalent. As a result there is no need to account for both in the union bound. In the case mentioned above we say that σ_1 and σ_2 belong to the same *equivalence class*. If the 2^s outer assignments collapse into only C equivalence classes, then the union bound need only be taken over C. Thus, we can now use C instead of 2^s in Proposition 2. We next describe a first basic method for bounding C:

We define a *boundary outer variable* as an outer variable that appears in at least one clause alongside an inner variable. Let B denote the set of all such boundary outer variables. Because the remaining outer variables (that are not in B) appear only in purely outer clauses, they cannot impose any logical constraints on the inner variables. Therefore, any full outer assignment σ can only influence the rest of the formula through the specific values it assigns to B. Consequently, the effect of σ on the inner function is entirely decided by the Boolean state of these boundary outer variables.

▶ **Theorem 7.** *The maximum number of distinct inner functions is therefore strictly bounded by the number of possible states for these boundary outer variables:* $C \leq 2^{|B|}$.

Thus using $|B|$ instead of s in Proposition 2 allows us to have an overall smaller number of repetitions.

6 Implementation and Discussion

We have implemented the techniques described above in a prototype tool called TodaQBF, which uses the GANAK model counter [12] as a parity-counting oracle. Specifically, our implementation incorporates the new balanced error allocation (Theorem 3), the multilinear hash function with modular addition (Theorems 5 and 6), and the boundary variable method for detecting loose union bounds (Theorem 7). We also added several engineering techniques, such as preserving the formula CNF structure, that do not fit the scope of this paper. We briefly describe our results and the challenges that emerged.

Preliminary Results

We ran our experiments on the formulas from the QBFEVAL benchmarks [11] having a timeout of 30sec. All the formulas were preprocessed with the QBF preprocessor bloqqer [9]. All in all we managed to solve about 1,000 benchmarks that became a single quantifier post-bloqqer. Admittedly these can also be solved by a SAT solver. In addition we also managed to solve 24 post-bloqqer $\forall\exists$ instances out of 8,900. These formulas (post-bloqqer) have between 2 and 19 universal variables, 161–468 existential variables, and 2,778–6,144 clauses. Post Toda's reduction, the size of the formulas were up to 640,843 variables and 3,194,872 clauses. GANAK handled these in 32–2,400 seconds per instance, solving all with an accuracy far above the specified ϵ error-rate. While this is much slower than dedicated QBF solvers on the solved instances (solved in about 1 second), it is several orders of magnitude faster than the naive implementation of Toda's, which runs out of memory before even completing the formula construction.

Discussion

In this work we have explored the feasibility of having an efficient probabilistic QBF solver based on Toda's theorem, that uses a model counter as an oracle. We have outlined three factors which can affect the efficiency of such a solver. We showed that the first two can be battled, but the third one, the union bound factor, is sometimes unavoidable. This means that such a successful solver for general benchmarks would have to rely on considerable improvements in the remaining factors and the quality of the model counter.

Our next mission is therefore to try and understand how to find and exploit benchmarks which possess a low union bound factor. While in Section 5 we described one method for that, we believe that much more can be done. We are also interested in pursuing non-trivial instances that inherently possess such a low bound and are of interest to the community.

Finally we observe that although these observations make sense in theory, in practice the structure of the QBF formula plays a decisive role in the efficiency of our methods, partially because of the nature of the model counter that we use. For example, neither of the instances that we solved had a union bound that we could actually improve. On the other hand, we found some 3QBF benchmarks from QBFEVAL that resulted in a huge reduction in the union bound, however, these instances where too hard for TodaQBF to deal with. One of the advantages of our tool is that it uses the model counter as a black box. Thus we can explore various benchmarks with different solvers. In addition, any progress on such model counters can be utilized to improve our solver.

References

1. Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.
2. Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable and nearly uniform generator of sat witnesses. In *CAV*, 2013.
3. Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. A scalable approximate model counter. In *CP*, 2013.
4. Stanley Chan. *Introduction to Probability for Data Science*. Michigan Publishing, 2021.
5. Holger Dell, Dieter van Melkebeek, Valentine Kabanets, and Osamu Watanabe. Is valiant-vazirani's isolation probability improvable? In *CCC*, 2012.
6. Persi Diaconis et al. Trailing the dovetail shuffle to its lair. *Annals of Applied Probability*, 2(2):294–313, 1992.
7. Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.
8. Sanjay Gupta. Isolating an odd number of elements and applications in complexity theory. *Theory of Computing Systems*, 31:27–40, 1998.
9. Florian Lonsing and Armin Biere. Failed literal detection for QBF. In *SAT*, 2011.
10. Ashwin Naik, Kenneth W. Regan, and D. Sivakumar. On quasilinear-time complexity theory. In *STACS*, 1995.
11. Luca Pulina. QBFEVAL. 2024. `http://www.qbflib.org`.
12. Mate Soos and Kuldeep S. Meel. GANAK: Parity and modular counting with combinatorial decomposition. In *AAAI*, 2025.
13. Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
14. Leslie G. Valiant and Vijay V. Vazirani. Np is as easy as detecting unique solutions. In *STOC*, pages 458–463, 1986.