An empirical study of QBF encodings: from treewidth to useful preprocessing

Luca Pulina and Armando Tacchella

DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
{Luca.Pulina | Armando.Tacchella}@unige.it

Abstract

Theoretical studies show that in some combinatorial problems, including satisfiability for quantified Boolean formulas (QBFs), there is a close relationship between classes of tractable instances and the treewidth of graphs describing their structure. In this paper [1] we investigate the practical relevance of such results for problems encoded as QBFs. We show that (an approximation of) treewidth is a predictor of empirical hardness, and that it is the only parameter among several other candidates which succeeds consistently in being so. We also provide evidence that QBF solvers benefit from a preprocessing phase geared towards reducing the treewidth of their input, and that this phase is a potential enabler for the solution of hard QBF encodings.

1 Introduction

Several theoretical studies deal with the relationship between the complexity of combinatorial problems and the treewidth (tw) of graphs representing their structure. The common trait of such studies is that the assumption of bounded values of tw yields tractable classes of problems which are intractable otherwise. This connection has been unveiled in the study of graph algorithms, and it emerged in other areas of application (see, e.g. [1]). In the context of the constraint satisfaction problem (CSP) the connection was first explored by Freuder [2], Dechter and Pearl [3], while more recent results (see, e.g., [4, 5, 6]) consider also the quantified constraint satisfaction problem (QCSP).

In this paper we are concerned with the practical relevance of the above results for problems that can be encoded as quantified Boolean formulas (QBFs). The satisfiability problem for QBFs (QSAT) is the subclass of the QCSP wherein all the variables range over a fixed Boolean domain. The importance of QSAT stems both from theoretical aspects – QSAT is the prototypical PSPACE-complete problem [7] – and from the fact that QBFs can provide compact Boolean encodings in several automated reasoning tasks, (see [8] for a comprehensive listing of domains and references). The interest in QSAT is also witnessed by a number of QBF encodings and solvers (see [8]), and by the presence of an annual competition of QBF solvers (QBFEVAL) [9].

From a practical standpoint, we see the results in [4, 5, 6] as gateways to the efficient solution of QBF encodings – many of which are also industrially relevant.

[1] Most of the results herein presented are also detailed in a paper accepted for the 15th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'08).

Proceedings of the 15th International RCRA workshop (RCRA 2008):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Udine, Italy, 12–13 December 2008

In particular, we build on [4] which relates the complexity of solving QCSPs to a generalization of tw that we call quantified treewidth (tw_p). Our main contributions, obtained considering data from the three most recent QBFEVAL events (2006-2008), are the following:

- Since computing tw is an NP-complete problem [10] it is not difficult to see that tw_p must be NP-hard at least; however, it turns out that tw_p can be *approximated* efficiently enough; to this purpose we introduce the proof-of-concept tool QuTE, a **Qu**antifed **T**reewidth **E**stimator, to compute upper bounds of tw_p.
- While bounding tw_p is only a *sufficient* condition for tractability and we have no clue whether increasing it will correspond to an increase in difficulty, we show that the approximation of tw_p computed by QuTE is a robust predictor – albeit in a statistical sense – of the performances exhibited by solvers when coping with QBF encodings; in this sense, the approximation of tw_p is a marker of empirical hardness, and it is the only parameter that succeeds consistently in being so among several other syntactic parameters which are plausible candidates.
- The result in [4] relates to a specific algorithm and it does not say much about QBF solvers using different approaches like search (see, e.g., [11, 12]), skolemization (see, e.g., [13]), or variable elimination (see, e.g., [14, 15]);[2] however, our experimental analysis shows that the significance of approximated tw_p is not related to some specific solver only.
- Finally, computing approximations of tw_p is useful; to show this we introduce QuBIS, a **Qu**antified **B**oolean formula **I**ncomplete **S**olver. QuBIS is incomplete in that, given an input QBF φ, it may either solve φ, or halt producing an equi-satisfiable QBF φ' whose treewidth is no larger than the treewidth of φ in most cases. Experiments with QuBIS show that preprocessing helps when it decreases the treewidth of QBFs, and the improvement can be so dramatic that formulas which cannot be solved by any solver (within the alloted time) before QuBIS preprocessing, can be solved afterwards.

The paper is structured as follows. In Section 2 we introduce basic definitions. In Section 3 we introduce the concept of empirical hardness and we show that the approximation of tw_p provided by QuTE is a marker of solver performances. In Section 4 we describe QuBIS in some detail, and show that it can effectively improve the performances of state-of-the-art QBF solvers. We conclude the paper in Section 5 with a summary about our current results and the related work.

[2] As shown in [16], the performances of Boolean satisfiability solvers based on variable elimination are very sensitive to different values of tw. As we confirm in Section 4, QBF solvers based on variable elimination are those that are most sensitive to tw_p and thus most closely resemble an implementation of the algorithm used in [4].

2 Preliminaries

In this section we consider the definition of QBFs and their satisfiability as given in the literature of QBF decision procedures (see, e.g., [11, 13, 14]), and we introduce notation from [4] to define graphs and associated parameters describing the structure of QBFs.

A *variable* is an element of a set P of propositional letters and a *literal* is a variable or the negation thereof. We denote with $|l|$ the variable occurring in the literal l, and with $\bar{l}$ the *complement* of l, i.e., $\neg l$ if l is a variable and $|l|$ otherwise. A literal is *positive* if $|l| = l$ and *negative* otherwise. A *clause* C is an n-ary $(n \geq 0)$ disjunction of literals such that, for any two distinct disjuncts l, l' in C, it is not the case that $|l| = |l'|$. A *propositional formula* is a k-ary $(k \geq 0)$ conjunction of clauses. A *quantified Boolean formula* is an expression of the form

$$Q_1 z_1 \dots Q_n z_n \Phi \tag{1}$$

where, for each $1 \leq i \leq n$, z_i is a variable, Q_i is either an existential quantifier $Q_i = \exists$ or a universal one $Q_i = \forall$, and Φ is a propositional formula in the variables $\{z_1, \dots, z_n\}$. The expression $Q_1 z_1 \dots Q_n z_n$ is the *prefix* and Φ is the *matrix* of (1). A literal l is *existential* if $|l| = z_i$ for some $1 \leq i \leq n$ and $\exists z_i$ belongs to the prefix of (1), and it is *universal* otherwise. For example, the following expression is a QBF:

$$\begin{aligned} \forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3 ((y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee \neg x_2 \vee \neg x_3) \wedge \\ (y_1 \vee \neg x_2 \vee x_3) \wedge (\neg y_1 \vee x_1 \vee x_3) \wedge \\ (\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge \\ (\neg y_1 \vee \neg x_1 \vee \neg y_2 \vee \neg x_3) \wedge \\ (\neg x_2 \vee \neg x_3)). \end{aligned} \tag{2}$$

The semantics of a QBF φ can be defined recursively as follows. A QBF clause is *contradictory* exactly when it does not contain existential literals. If the matrix of φ contains a contradictory clause then φ is false. If the matrix of φ has no conjuncts then φ is true. If $\varphi = Qz\psi$ is a QBF and l is a literal, we define φ_l as the QBF obtained from ψ by removing all the conjuncts in which l occurs and removing $\bar{l}$ from the others. Then we have two cases. If φ is $\exists z\psi$, then φ is true exactly when φ_z or $\varphi_{\neg z}$ are true. If φ is $\forall z\psi$, then φ is true exactly when φ_z and $\varphi_{\neg z}$ are true. The QBF satisfiability problem (QSAT) is to decide whether a given formula is true or false. It is easy to see that if φ is a QBF without universal quantifiers, solving QSAT is the same as solving propositional satisfiability (SAT).

A *relational signature* σ is a finite set of relation symbols, each of which has an associated arity. A (finite) *relational structure* $\mathbf{A}$ over σ consists of a universe A and a relation $R^{\mathbf{A}}$ over A for each relation symbol R of σ, such that the arity of $R^{\mathbf{A}}$ matches the arity associated to R. Accordingly, the QBF (2) can be rewritten

as:

$$\begin{aligned}\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3 (C_{000}(y_1, y_2, x_2) \wedge C_{0111}(y_1, y_2, x_2, x_3) \wedge \\ C_{010}(y_1, x_2, x_3) \wedge C_{100}(y_1, x_1, x_3) \wedge \\ C_{100}(y_1, y_2, x_2) \wedge C_{101}(y_1, y_2, x_2) \wedge \\ C_{1111}(y_1, x_1, y_2, x_3) \wedge C_{11}(x_2, x_3))\end{aligned} \tag{3}$$

over the signature $\sigma = \{C_{000}, C_{0111}, C_{010}, C_{100}, C_{101}, C_{1111}, C_{11}\}$ where each $C_w \in \sigma$ has arity $|w|$. Let ϕ be the expression (3). The QSAT problem for ϕ can be restated as the problem of checking the (first-order logic) entailment $\mathbf{B} \models \phi$, where $\mathbf{B}$ is a relational structure with signature σ and universe $B = \{0, 1\}$, such that, for each $C_w \in \sigma$, $C_w^{\mathbf{B}}$ is the relation containing all $|w|$-tuples over B except w, e.g., $C_{11}^{\mathbf{B}} = \{(0, 0), (0, 1), (1, 0)\}$.

Following [4] we further introduce the notion of *quantified relational structure* as a pair $(p, \mathbf{A})$ where $\mathbf{A}$ is a relational structure and p is a prefix, i.e., an expression of the form $Q_1 z_1 \ldots Q_n z_n$ where each Q_i is either $\exists$ or $\forall$, and $z_1, \ldots, z_n$ are exactly the elements of the universe of $\mathbf{A}$. The quantified relational structure $(p, \mathbf{A})$ associated to a QBF ϕ is obtained by letting p be the prefix of ϕ and letting $R^{\mathbf{A}}$ contain all tuples $(a_1, \ldots, a_k)$ such that $R(a_1, \ldots, a_k)$ appears as a conjunct in ϕ. Notice that a prefix $p = Q_1 z_1 \ldots Q_n z_n$ can be viewed as the concatenation of *quantifier blocks* where quantifiers in each block are the same, and consecutive blocks have different quantifiers. If $h \leq n$ is the number of blocks in p, then $h - 1$ is the *alternation depth* of p and, by extension, of the QBF having p as a prefix. If p consists of the blocks $Q_1 Z_1 \ldots Q_h Z_h$, then to each variable z we can associate a *level* $l(z)$ which is the index of the corresponding block, i.e., $l(z) = i$ for all the variables $z \in Z_i$. We also say that variable z *comes after* a variable v in p if $l(z) \geq l(v)$. For instance, the quantified relational structure associated to (3) is $(\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3, \mathbf{A})$, with universe $A = \{y_1, y_2, x_1, x_2, x_3\}$ and

$$\begin{array}{ll} C_{000}^{\mathbf{A}} = \{C_{000}(y_1, y_2, x_2)\} & C_{0111}^{\mathbf{A}} = \{C_{0111}(y_1, y_2, x_2, x_3)\} \\ C_{010}^{\mathbf{A}} = \{C_{010}(y_1, x_2, x_3)\} & C_{100}^{\mathbf{A}} = \{C_{100}(y_1, x_1, x_3), C_{100}(y_1, y_2, x_2)\} \\ C_{1111}^{\mathbf{A}} = \{C_{1111}(y_1, x_1, y_2, x_3)\} & C_{11}^{\mathbf{A}} = \{C_{11}(x_2, x_3)\}. \end{array} \tag{4}$$

A relational structure – and thus the structure of a QBF – can be described by a *Gaifman graph*. Given a relational structure $\mathbf{A}$, the Gaifman graph of $\mathbf{A}$ is the graph with vertex set equal to the universe A of $\mathbf{A}$ and with an edge (a, a') for every pair of different elements $a, a' \in A$ that occur together in some $\mathbf{A}$-tuple, i.e., in some element of $R^{\mathbf{A}}$ for some relation symbol R. A *scheme* for a quantified relational structure $(p, \mathbf{A})$ is a supergraph (A, E) of the Gaifman graph of $\mathbf{A}$ along with and ordering $a_1, \ldots, a_n$ of the elements of A such that (i) the ordering $a_1, \ldots, a_n$ preserves the order of p, i.e., if $i < j$ then a_j comes after a_i in p, and (ii) for any a_k, its lower numbered neighbors form a clique, that is, for all k, if $i < k$, $j < k$, $(a_i, a_k) \in E$ and $(a_j, a_k) \in E$, then $(a_i, a_j) \in E$.

The *width* w_p of a scheme is the maximum, over all vertices a_k, of the size of the set $\{i : i < k, (a_i, a_k) \in E\}$, i.e., the set containing all lower numbered neighbors of a_k. The *treewidth* tw_p of a quantified relational structure $(p, \mathbf{A})$ is the

minimum width over all schemes for $(p, \mathbf{A})$. Given the correspondence between relational structures and QBFs, we write $tw_p(\varphi)$ to denote the treewidth of the QBF φ.

3 Treewidth and empirical hardness

If we define the class QSAT[$tw_p < k$] as the restriction of the QSAT problem to all instances $((p, \mathbf{A}), \mathbf{B})$ where $(p, \mathbf{A})$ has quantified treewidth strictly less than k, then considering the definition of the polynomial-time algorithm k-consistency of [4] we can state the following:

Theorem 1 ([4]). *For all $k \geq 2$, establishing k-consistency is a decision procedure for QSAT[$tw_p < k$]*

From the above, it immediately follows that the class QSAT[$tw_p < k$] is a tractable subclass of the QSAT problem and, therefore, the QBFs corresponding to relational structures with a bounded tw_p are a tractable subclass of QSAT.

To understand the implications of Theorem 3 in the case of concrete QBF encodings and solvers, we define a notion of empirical hardness. Given a set of QBFs Γ, a set of QBF solvers Σ, and an implementation platform Π, we define *hardness* as a partial function $H_{\Gamma,\Sigma,\Pi} : \Gamma \to \{0, 1\}$ such that $H_{\Gamma,\Sigma,\Pi}(\varphi) = 1$ iff no solver in Σ can solve φ on Π; and $H_{\Gamma,\Sigma,\Pi}(\varphi) = 0$ iff all solvers in Σ can solve φ on Π. The parameters Γ and Σ take into account the dependency of H from the current state of the art in the available QBF encodings and solvers. The parameter Π denotes the dependency of H from the currently available hardware platforms, as well as the amount of time and space resources allotted to the solvers. In the remainder of this section Γ, Σ and Π are fixed, and we write $H(\varphi)$ to denote the hardness of φ. In particular, Π is a family of identical Linux workstations comprised of 8 Intel Core 2 Duo 2.13 GHz PCs with 4GB of RAM; the resources granted to the solvers are 600s of CPU time and 3GB of memory. As for Γ and Σ, we consider the sets of formulas and solvers that participated in the three most recent competitions of QBF solvers (QBFEVAL). Further details about the datasets and links to the events can be found in [17].

Understanding the relationship between $H(\varphi)$ and tw_p requires the computation of both quantities for all the formulas in Γ, but with our choice of Γ computing an exact value of tw_p is unfeasible, even for the smallest formulas. Therefore, we built the tool QuTE – whose implementation is sketched in Figure 1 – in order to provide us with approximate values of tw_p. As we can see in Figure 1, QuTE takes as input a QBF φ and returns an approximate value $\hat{tw}_p$ of the quantified treewidth by performing the following steps:

- The input QBF is converted to the corresponding Gaifman graph G (line 1) according to the construction presented in Section 2.
- An ordering σ is sought (line 2); the current implementation of FINDORDERING is the maximum cardinality search (MCS) algorithm of [18].

```
SORTBYPREFIX(φ, {v_1, v_2, ..., v_n})
 1 h ← number of quantifier blocks of φ
 2 Z_h ← h-th quantifier block of φ
 3 i ← 1; j ← 1; s ← 0
 4 while (h > 0) do
 5   if (v_i ∈ Z_h) then
 6     v'_j ← v_i; j ← j + 1; Z_h ← Z_h \ {v_i}
 7     if (Z_h = ∅) then
 8       h ← h - 1
 9       if (s ≠ 0) then i ← s; s ← 0
10   else
11     if (s = 0) then s ← i
12     while (v_i ∉ Z_h) i ← i + 1
13 return {v'_1, v'_2, ... v'_n}
```

```
FILLIN ((V, E), {v_1, v_2, ..., v_n})
 1 tw ← 0
 2 for v ∈ V do
 3   M(v) ← the set of vertices adjacent to v
 4 E' ← E
 5 for i ← 1 to n do
 6   if |M(v_i)| > tw then
 7     tw ← |M(v_i)|
 8   for u, w ∈ M(v_i) and (u, w) ∉ E' do
 9     E' ← E' ∪ (u, w)
10     M(u) ← M(u) ∪ {w}
11     M(w) ← M(w) ∪ {u}
12   for u ∈ M(v_i) do
13     M(u) ← M(u)\{v_i}
14 return tw
```

```
QUTE(φ)
 1 G ← Gaifman graph of φ
 2 σ ← FINDORDERING(G)
 3 σ' ← SORTBYPREFIX(φ, σ)
 4 tŵ_p ← FILLIN(G, σ')
 5 return tŵ_p
```

Figure 1: The algorithm of QUTE and the auxiliary functions SORTBYPREFIX and FILLIN.

- The function SORTBYPREFIX (line 3) transforms σ into another – possibly identical – ordering σ' which is compatible with the prefix of φ.
- Finally, the function FILLIN (line 4) computes the value of $\hat{tw}_p$ by computing a chordal completion of G in such a way that σ' becomes a perfect elimination scheme; since σ' is not guaranteed to yield the minimum value of tw over all possible chordal completions, it turns out that $\hat{tw}_p \geq tw_p$.

There are two important observations about QUTE. First, if FINDORDERING were able to guess σ in such a way that the chordalization performed by FILLIN yields the minimum maximal clique over all possible chordal completions, then $\hat{tw}_p = tw_p$. In practice, FINDORDERING is just an heuristic, but efficient heuristics – like MCS – do not guarantee a tight bound on the approximation, while more accurate algorithms (like, e.g., QuickBB [19]) are hopelessly slow in our case. We have experimented with several FINDORDERING – indeed, all those available in the TreeD library [20] on top of which QUTE is implemented – and we did not find substantial differences among different heuristics. Also, for graphs in which tw_p can be computed – random graphs of up to 30 nodes – we have seen that decreasing tw_p causes also $\hat{tw}_p$ to decrease and, if the graph is either very sparse or mostly connected, $tw_p = \hat{tw}_p$ on most samples. However, the question whether $\hat{tw}_p$ is a reasonably tight approximation of tw_p for the kind of graphs that we deal with is still an open point. Second, since by definition tw_p takes into account the prefix structure, σ' can be substantially different with respect to σ, and thus, in general, $tw_p \geq tw$. Moreover, all the QBFs we consider are in prenex CNF, meaning that all the bound variables are constrained to a total order. Knowing the "true" pre-

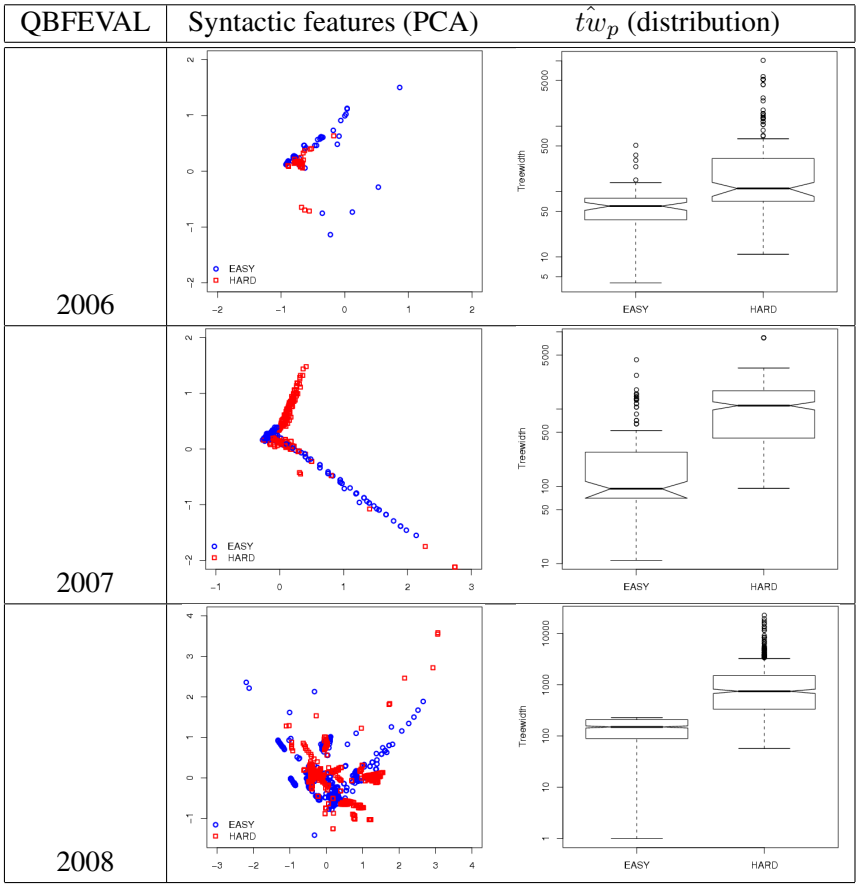

Figure 2: Hardness vs. syntactic features (left) and treewidth (right).

fix structure, i.e., the partial order among the bound variables, may allow, in some cases, to obtain a better approximation of tw_p than the one computed starting from the prenex QBF. We consider all the above issues related to improving QuTE[3] as topics for future research.

The main result of this section is presented in Figure 2 where we consider *hard* the formulas φ such that $H(\varphi) = 1$, and *easy* the formulas φ such that $H(\varphi) = 0$. Concerning the QBFEVAL 2006 dataset, we have 73 easy formulas, and 56 hard formulas. In the QBFEVAL 2007 dataset, easy and hard formulas are 215 and 263, respectively, while in the 2008 dataset there are 559 easy and 790 hard QBFs. The plots on the left-hand side of Figure 2 consider a set of 141 parameters that can be computed inexpensively like, e.g., the number of clauses, the number of variables and the alternation depth. The complete listing and a detailed description of such parameters, that we collectively call *syntactic features*, can be found in [21]. The plots on the right-hand side of Figure 2 are obtained by considering the distributions of $\hat{tw}_p$ computed using QuTE on easy and hard formulas.

[3]The latest C++ implementation of QuTE is available at [17].

For each QBFEVAL dataset, in Figure 2 (left) we show plots obtained by considering each formula as a point in the multidimensional space of syntactic features. Since it is impossible to visualize such a space, we consider its two-dimensional projection obtained by means of a principal components analysis (PCA) and considering only the first two principal components.[4] In Figure 2 (right) we show plots obtained by considering the distributions of $\hat{tw}_p$ computed for easy and hard formulas. For each distribution, we show a box-and-whiskers diagram representing the median (bold line), the first and third quartile (bottom and top edges of the box), the minimum and maximum (whiskers at the top and the bottom) of a distribution. Values laying farther away than the median ± 1.5 times the interquartile range are considered outliers and shown as dots on the plot.[5] An approximated 95% confidence interval for the difference in two medians is represented by the notches cut in the boxes: if the notches of two plots do not overlap, this is strong evidence that the two medians differ.

Considering the results shown in Figure 2 (right) we can conclude that $\hat{tw}_p$ is a robust marker of empirical hardness, since for all QBFEVAL datasets the distribution of $\hat{tw}_p$ varies significantly across hard and easy instances. To get a quantitative feeling of this, let us consider a simple Bayesian argument related to the problem of deciding whether a given QBF φ is hard or not. For the sake of our argument, the distributions in Figure 2 (right) represent $p(\hat{tw}_p|H)$, i.e., the probability density of $\hat{tw}_p$ given the hardness H. The proportion of hard instances in each dataset is the parameter r in $p(H) = r^H(1-r)^{(1-H)}$, i.e., the prior density of hardness H. Looking at prior information only, we decide that a QBF φ is hard exactly when $P(H=1) - P(H=0) > 0$, i.e., when $r > 0.5$. If we consider also the likelihood $p(\hat{tw}_p|H)$, then we can compute the posterior density $p(H|\hat{tw}_p)$ using Bayes' rule and then decide that a QBF φ is hard exactly when

$$P(\hat{tw}_p(\varphi)|H(\varphi)=1) \cdot P(H(\varphi)=1) > P(\hat{tw}_p(\varphi)|H(\varphi)=0) \cdot P(H(\varphi)=0)$$

We close our argument by stating that in all the QBFEVAL datasets the accuracy of the above criterion is strictly higher than looking at prior probabilities only. For instance, in the QBFEVAL'08 dataset we have $r = 0.54$ meaning that $P(H = 0) = 0.46$ and $P(H = 1) = 0.54$. A priori, given a formula φ in the QBFEVAL'08 dataset we would decide that it is hard ($H = 1$), with a 54% accuracy, i.e., only slightly more than tossing a fair coin. With the Bayesian approach, we obtain a 72% accuracy which is a definite increase in our predictive ability.

On the other hand, syntactic features are collectively unable to distinguish among easy and hard instances. As we can see in Figure 2 (left), with the partial exception of some hard instances in the QBFEVAL'07 dataset, there is a non-negligible chance that easy and hard instances share similar values of such features.

[4] Details about PCA and its use for visualizing multidimensional datasets are beyond the scope of this paper: see, e.g., Chap. 7 of [22] for an introduction to PCA and further references.

[5] In case outliers are detected, the whiskers extend up to the median $+1.5$ (resp. -1.5) times the interquartile range, while the maximum (resp. minimum) value becomes the highest (resp. lowest) outlier.

Group	Feature	Accuracy (%)
Treewidth ($\hat{tw}_p$)		72
Number of variables	Existential	49
	Universal	64
	Total	49
Number of sets	Total	54
Number of variables per set	Existential	45
	Universal	62
	Total	46
Clauses-to-Variables		43

Group	Feature	Accuracy (%)
Number of clauses	Unary	42
	Binary	53
	Horn	50
	Dual Horn	50
	Total	58
Number of occurrences per variable (mean)	Existential	59
	Universal	44
	Negative	42
	Positive	41
	Total	41

Table 1: Discriminative power of syntactic features considering posterior probabilities.

Technically, we say that in the space of syntactic features it is hard to find a discriminant – a curve in the PCA plots of Figure 2 – that allows us to tell easy instances from hard ones with sufficient accuracy. Repeating the Bayesian argument above for syntactic features on the QBFEVAL'08 dataset, we obtain the results of Table 1. Here we can see that posterior probability densities do not yield substantial improvements over prior probabilities except in the case of $\hat{tw}_p$ meaning that, overall, syntactic features are far from the predictive power of approximated treewidth. In some cases, including the "famous" clauses-to-variable ratio, conditioning hardness on a syntactic feature is even worse than tossing a fair coin.

4 Treewidth and useful preprocessing

In this section we introduce our tool QuBIS, an incomplete solver which, given an input QBF φ, may either solve it, or halt producing an equi-satisfiable QBF φ', where $\hat{tw}_p(\varphi')$ is no larger than $\hat{tw}_p(\varphi)$ in most cases. The purpose of this section is to show that QBF solvers benefit from getting the output of an incomplete run of QuBIS rather than being fed the original QBF as input. In other words, preprocessing helps when it decreases $\hat{tw}_p$ and, in practice, this is often true independently from the algorithm featured by the solver.

QuBIS is based on *Q-resolution* defined in [23] as an operation among clauses of a QBF. In particular, given two clauses $P \vee x$ and $R \vee \neg x$, where P and R are disjunctions of literals, the clause $P \vee R$ can be derived by Q-resolution subject to the constraints that (i) x is an existential variable, and (ii) P and R do not share any variable z such that $\neg z$ (resp. z) occurs in P and z (resp. $\neg z$) occurs in R. QuBIS uses Q-resolution to perform *variable elimination* on existential variables defined, e.g., in [14], as the operation whereby, given a QBF $Q_1 z_1 Q_2 z_2 \ldots \exists x \Phi$, the variable x can be resolved away by performing all resolutions on x, adding the resolvents to the matrix Φ and removing from Φ all the clauses containing x. Universal variables can be eliminated simply by deleting them once they have the highest prefix level. More precisely, given a matrix Φ, let $\Phi_{/z}$ be the same matrix whereby all the occurrences of z have been deleted. The QBF $Q_1 z_1 Q_2 z_2 \ldots \forall y \Phi$ is true exactly when the QBF $Q_1 z_1 Q_2 z_2 \ldots \Phi_{/y}$ is true, so y can be eliminated safely. From the above, it immediately follows that variable elimination, once it

Encoding	QBFs	Description
add	32	equivalence checking of partial implementations of circuits
circ	63	FPGA logic synthesis
count	24	model checking of counter circuits
cp	24	conformant planning domains
k	378	modal K formulas
katz	20	symbolic reachability of industrially relevant circuits
s	171	symbolic diameter evaluation of ISCAS89 circuits
tipdiam	203	symbolic diameter evaluation of circuits

Table 2: Encodings used to experiment with QuBIS.

Solver	add		circ		count		cp		k		katz		s		tipdiam	
	#	Time	#	Time	#	Time	#	Time	#	Time	#	Time	#	Time	#	Time
QMRes	20	1061.53	–	–	8	87.53	1	0.30	269	3072.22	7	51.90	6	36.00	58	2576.02
quantor	8	20.94	7	141.66	12	16.05	16	1710.01	259	922.56	–	–	17	1185.52	76	709.27
QuBE3.0	5	1.99	4	18.08	9	90.15	6	160.08	115	5552.56	–	–	1	0.07	71	1132.97
QuBE6.1	5	1.34	4	4.87	9	116.52	5	169.14	203	4376.43	6	26.21	62	3006.31	151	1493.88
sKizzo	14	814.80	6	79.11	12	5.88	7	287.40	348	7262.20	–	–	19	1089.13	133	8554.86
yQuaffle	4	0.86	4	0.58	9	3.99	8	267.22	142	5622.60	–	–	1	0.12	71	2162.84

Table 3: Performances of a selection of QBF solvers.

respects the prefix order, yields a decision procedure for QSAT (see, e.g., [15, 14]).

QuBIS takes as input a QBF φ and two parameters: (i) an integer deg, the maximum degree allowed for a given variable considering the Gaifman graph of φ; (ii) an integer div, the maximum value of *diversity*, a parameter defined in [16] as the product of the number of positive and negative occurrences of a variable in φ. The role of deg is thus to bound the number of variables in a clause, while the role of div is to bound the (worst case) number of resolvents generated when eliminating an existential variable. Intuitively, QuBIS eliminates variables until the input QBF can be declared true, false or when eliminating variables is bound to increase the size of the resulting QBF beyond some critical threshold. More precisely, a variable qualifies for elimination only if it has the highest level in the prefix of φ, and it is a universal variable, or if it is an existential variable, its degree is no larger than deg and its diversity is no larger than div. Universal variables are eliminated simply by deleting all their occurrences from the matrix of φ, while existential variables are resolved away. In both cases, the resulting QBF is given as argument to a recursive call of QuBIS. QuBIS terminates when one of the following conditions is satisfied: (i) the matrix of φ is empty – in which case the input QBF is true; (ii) the matrix of φ contains a contradictory clause – in which case the input QBF is false; (iii) there are no variables that qualify for elimination in φ – in which case φ is returned as output. Therefore, QuBIS is a sound and complete decision procedure for the subclass of QBFs in which variables always qualify for elimination, while for all the other formulas QuBIS[6] behaves like a preprocessor.

The experiments detailed in this section are carried out on the same computing platforms described in Section 3, but here we focus on the 915 QBF encodings summarized in Table 2 and on the following QBF solvers (references available from [8]): QMRes, quantor, QuBE3.0, QuBE6.1, sKizzo, and yQuaffle.

[6] A proof-of-concept implementation in C++ of QuBIS can be downloaded from [17].

Our first experiment is to run the solvers on the QBF encodings, with the goal of confirming the validity of the selection above. In particular, we wish to show that the encodings considered are challenging enough given the current state of the art, and that the algorithms featured by the solvers are "orthogonal", i.e., solvers have complementary abilities across different families. Table 3 shows the results: the first column contains the solver names, and it is followed by eight groups of columns, one for each encoding. The columns "#" and "Time" contain, respectively, the number of formulas solved and the cumulative CPU seconds. A dash on both columns means that the solver did not solve any formula. Looking at Table 3 we see that our selection is indeed valid for our purposes. For instance, considering `circ` encodings we see that no single solver is able to solve more than about 10% of them. Still considering the percentage of QBFs solved by any single solver, we see that a similar result holds for `katz` encodings (about 30%) and `s` encodings (about 35%). Furthermore, there is no single solver dominating over all encodings: QMRES is best on `add` and second best on `k` encodings; QUANTOR is best on `count` (ex-aequo with SKIZZO) and third best on `k` encodings; QUBE6.1 is the strongest on `tipdiam` and `s` encodings – apparently due to internal preprocessing, given the performances of QUBE3.0; SKIZZO, on the other hand, is the strongest on `k` encodings. [7]

In the next experiment, for each solver we consider the formulas that it could not solve according to the results of Table 3. For each solver and each such formula φ, we compute $\hat{tw}_p(\varphi)$, preprocess φ with QUBIS – setting $deg = 20$ and $div = 2000$ – to yield a new QBF φ' and then compute $\hat{tw}_p(\varphi')$ with QUTE (both QUBIS and QUTE have a CPU time limit of 600 seconds). The goal of this experiment is to see whether QUBIS, used as a preprocessor, is able to decrease $\hat{tw}_p$ of (solver-wise) hard encodings. Table 4 shows the results. The table is split horizontally in two parts. In each part, the column "**Solver**" reports the name of a solver followed by four groups of columns, one for each encoding of Table 2. Each group contains five columns: H is the number of (solver-wise) hard formulas, H' is the number of such formulas preprocessed by QUBIS for which QUTE was able to estimate the treewidth, p is the number of formulas φ such that $\hat{tw}_p(\varphi') < \hat{tw}_p(\varphi)$. The columns μ and μ_q contain the mean value of $\hat{tw}_p$ across the formulas in H' before (μ) and after (μ_q) preprocessing .

Looking at Table 4, we can see that, on average, preprocessing with QUBIS decreases $\hat{tw}_p$. Indeed $\mu_q < \mu$ for all solvers and all encodings in Table 4. In several cases the set of encodings for which QUBIS is able to decrease $\hat{tw}_p$ almost coincides with the set that it is able to preprocess without exceeding its resource limits. This phenomenon is most evident for the families `add`, `circ` – where H' and p actually coincide for almost all solvers – `cp`, and `katz`. In some cases, e.g., two formulas in the `add` group and thirteen QUANTOR-hard encodings in the `k` group, $\hat{tw}_p$ can be decreased by one order of magnitude. Comparatively, there

[7] SKIZZO is also the strongest solver overall, with 539 encodings solved, 20% more than QUBE6.1 which comes second best.

Solver	add					circ					count					cp				
	H	H'	p	μ	μ_q	H	H'	p	μ	μ_q	H	H'	p	μ	μ_q	H	H'	p	μ	μ_q
QMRES	12	9	9	996	705	63	35	35	2055	1865	16	16	4	253	232	23	18	18	173	163
QUANTOR	24	21	21	541	398	56	28	28	2466	2234	12	12	3	311	309	8	4	4	447	438
QUBE3.0	27	24	23	507	364	59	31	31	2304	2090	15	15	4	254	229	18	13	13	211	203
QUBE6.1	27	24	23	507	364	59	31	31	2304	2090	15	15	4	254	229	19	14	14	203	193
SKIZZO	18	15	15	663	525	57	29	29	2425	2199	12	12	3	311	309	17	12	12	208	203
YQUAFFLE	28	25	24	490	353	59	31	31	2304	2090	15	15	4	254	229	16	11	11	221	215

	k					katz					s					tipdiam				
	H	H'	p	μ	μ_q	H	H'	p	μ	μ_q	H	H'	p	μ	μ_q	H	H'	p	μ	μ_q
QMRES	109	71	55	370	364	13	13	13	351	306	165	6	6	796	497	145	97	94	311	194
QUANTOR	119	101	53	378	372	20	20	18	276	240	154	–	–	–	–	127	80	79	352	219
QUBE3.0	263	194	76	182	178	20	20	18	275	240	170	9	9	557	353	132	85	83	345	214
QUBE6.1	175	151	59	93	89	14	14	14	331	289	109	–	–	–	–	52	20	20	546	347
SKIZZO	30	18	17	276	269	20	20	18	275	240	152	–	–	–	–	70	46	42	395	259
YQUAFFLE	236	185	74	204	200	20	20	18	275	240	170	9	9	557	353	132	85	84	342	213

Table 4: Results of treewidth analysis on QBF encodings and their preprocessed versions.

are only 38 cases in which the treewidth increased. Considering the total number of literals in a formula as a size indicator, we found out that the average size of such formulas is about one order of magnitude smaller than the size of the ones for which QUBIS is able to decrease $\hat{tw}_p$, while the number of quantifier blocks is, on average, a factor of two higher. Indeed, the net effect of QUBIS on these kind of formulas is to increase the size of clauses without eliminating any quantifier block, which means that $\hat{tw}_p$ may increase because minimal cliques (clauses) are bigger after preprocessing and the ordering on the vertices is as constrained as it was before preprocessing.

Another relevant fact from Table 4 is that there are cases in which (i) estimating tw_p is difficult and/or (ii) preprocessing is difficult. As it turns out, for some unprocessed formulas, e.g., in the family `s`, we are not even able to compute $\hat{tw}_p$, and for other formulas, e.g., in the family `circ`, we can compute $\hat{tw}_p$ for more formulas than the ones tamed by QUBIS. Needless to say, such families are quite challenging and, on average, their members feature relatively high values of $\hat{tw}_p$. One last observation about Table 4 is related to the fact that it may seem unlikely that a tool like QUBIS is able to consistently decrease $\hat{tw}_p$ – indeed, we have discussed cases where the converse is true. By definition of tw_p, if we let M be the size of the largest clause in a QBF φ, then we know that $M \leq tw_p(\varphi)$. Since variable elimination tends to generate large clauses, we would expect that in preprocessed formulas $\hat{tw}_p$ is higher than in the original ones. However, as discussed in [16], this is not necessarily the case, as long as the *connectivity* of φ does not increase. Clearly, for QUBIS to work properly as a preprocessor, the setting of the parameters deg and div is crucial – and not necessarily good in all the situations. Our setting in the above experiments reflects a good trade-off between efficiency in the usage of resources, and effectiveness in decreasing $\hat{tw}_p$.

In Table 5 we show the performances of QBF solvers on hard encodings after preprocessing. The Table is organized similarly to Table 4, and there are five columns in each group: H is the number of hard QBFs for each solver/encoding, S is the number of such formulas solved by QUBIS during preprocessing, "#", "Time$_p$", and "Time" contain, respectively, the total number of formulas solved

Solver	add					circ					count					cp				
	H	S	#	Time$_p$	Time	H	S	#	Time$_p$	Time	H	S	#	Time$_p$	Time	H	S	#	Time$_p$	Time
QMRES	12	–	2	0.06	41.47	63	–	–	–	–	16	–	–	–	–	23	–	–	–	–
QUANTOR	24	–	–	–	–	56	–	–	–	–	12	–	–	–	–	8	–	1	0.58	41.50
QUBE3.0	27	–	1	0.04	9.04	59	–	–	–	–	15	–	–	–	–	18	–	–	–	–
QUBE6.1	27	–	2	0.06	8.24	59	–	–	–	–	15	–	15	0.21	57.32	19	–	2	1.73	412.82
SKIZZO	18	–	–	–	–	57	–	9	2791.69	3274.30	12	–	–	–	–	17	–	–	–	–
YQUAFFLE	28	–	2	0.06	29.52	59	–	–	–	–	15	–	–	–	–	16	–	1	0.58	2.56

	k					katz					s					tipdiam				
	H	S	#	Time$_p$	Time	H	S	#	Time$_p$	Time	H	S	#	Time$_p$	Time	H	S	#	Time$_p$	Time
QMRES	109	19	30	9.68	1508.64	13	–	–	–	–	165	1	6	345.25	471.88	145	4	15	67.75	955.63
QUANTOR	119	–	35	1.72	2.76	20	–	–	–	–	154	–	–	–	–	127	3	4	0.01	67.19
QUBE3.0	263	55	88	11.10	3547.85	20	–	2	0.85	209.24	170	3	4	377.92	378.24	132	4	22	70.60	2061.39
QUBE6.1	175	14	50	0.87	4942.48	14	–	–	–	–	109	–	–	–	–	52	–	–	–	–
SKIZZO	30	–	1	0.17	67.54	20	–	2	0.85	172.17	152	–	–	–	–	70	–	10	6.25	2419.14
YQUAFFLE	236	37	63	6.22	4219.11	20	–	–	–	–	170	3	4	357.35	380.96	132	4	19	72.75	1048.42

Table 5: Performances of a selection of QBF solvers on preprocessed encodings.

(including the ones solved by QUBIS), the preprocessing time, and the cumulative CPU time in seconds (including preprocessing time). Looking at Table 5 (top), considering the group `add`, we can see that QMRES, QUBE6.1 and YQUAFFLE solve two previously unsolved formulas, while QUBE3.0 solves one. In particular, QMRES is able to solve two more encodings since they had an estimated quantified treewidth of 658 and 943 but QUBIS decreased it to 77 and 93, respectively. Considering the group `circ`, we can see that only SKIZZO is able to solve formulas (9 out of 29) that it found hard before preprocessing. Considering the group `count`, as in the case of `circ`, we see that only QUBE6.1 takes advantage of preprocessing by solving 15 previously unsolved formulas. Looking now at the `cp` group, we see that QUBE6.1 solves 2 hard formulas, while both QUANTOR and YQUAFFLE solve 1 hard formula.

Still with reference to Table 5, we consider now the encodings on the bottom. In the group `k`, preprocessing turns out to be very effective for most solvers, with the exception of SKIZZO because of a ceiling effect: indeed, SKIZZO alone is already quite effective on such formulas. On the other hand, search-based solvers benefit the most, and it is fair to say that QUBIS complements the shortcomings of these solvers on such encodings. However, it is interesting to notice that also variable-elimination based solvers like QUANTOR and QMRES improve their performances, which contributes to the thesis that decreasing $\hat{tw}_p$ is useful independently from the specific algorithm featured by the solver. As for the group `katz`, only QUBE3.0 and SKIZZO are able to solve 2 previously unsolved formulas. Similar results hold also for the `s` group, wherein QMRES is the solver which benefits the most from preprocessing. The group `tipdiam` is quite interesting in its own, since most solvers are able to benefit from preprocessing, again in an algorithm independent fashion.

We conclude the analysis of Table 5 by mentioning that, overall, the dataset therein considered consists of 272 hard – in the sense of Section 3 – formulas. QUBIS can preprocess 113 such formulas in a successful way, i.e., without exceeding its resource bounds. Noticeably, considering an ideal solver that always fares the best result for each pair solver/encoding *after* preprocessing, we have that

25 previously hard formulas can now be solved. Considering that QuBIS is still a proof-of-concept implementation, we view this as an indication that preprocessing geared towards reducing quantified treewidth is an enabler to deal with challenging QBF encodings.

5 Related work and conclusions

The empirical role of treewidth has been previously explored in the CSP [24] and SAT [16] literature. Before this paper, there was no such study in QBF, albeit the papers about QUANTOR and QMRes (see,e.g., [15, 14]) implicitly leverage the same concepts and are thus related to our contribution. From a theoretical standpoint, there are two other papers [5, 6] that together with [4] consider the relationship between structural restrictions of QBFs and the complexity of reasoning about them. Here, we follow [4] because its characterization of tw_p accounts nicely for the structure of the prefix and the structure of the formula in a single parameter. In [6], the prefix is taken into account by considering alternation depth, while treewidth accounts only for the structure of the matrix. In [5] a completely different kind of structural restriction, which is also incomparable with treewidth, is presented. From an experimental point of view, it may be interesting to check how the results of [6] and [5] apply to our setting, and whether they shed further insight on the behaviour of QBF solvers on hard encodings.

Concerning QuBIS, our direct source of inspiration has been the algorithm of Bounded Directional Resolution (BDR) presented in [16]. From an implementation point of view, even as a proof-of-concept implementation, QuBIS is more advanced than BDR. From an algorithmic point of view, the main difference between BDR and QuBIS is that our solver uses dynamic, rather than static, reordering of variables. Since QuBIS uses Q-resolution to eliminate variables, it is in this aspect similar to QMRes [15] and QUANTOR [14]. However, our approach differs from QMRes, because we do not use symbolic data structures, and also from QUANTOR since we never expand universal variables.

In this paper we have studied the practical relevance of $\hat{tw}_p$ as a marker of empirical hardness in QBFs. We have shown that such approximation is, in a statistical sense, a robust predictor of the difficulty encountered by solvers facing QBF encodings. We have shown that other purely syntactic features, alone or in combination among them, are not as good as $\hat{tw}_p$. Finally, we have shown that decreasing $\hat{tw}_p$ as done by QuBIS can enable QBF solvers to cope with hard encodings. Our future work will include looking for more accurate bounds when approximating tw_p, evaluating the impact of other preprocessors for QBFs on tw_p, and an evaluation of tw_p as a control parameter for multi-engine solvers.

References

[1] H.L. Bodlaender. Treewidth: Characterizations, applications, and computations. Technical report, Utrecht University, 2006.

[2] E. Freuder. Complexity of k-tree structured constraint satisfation problem. In *In proc. of AAAI'90*, 1990.

[3] R. Dechter and J. Pearl. Tree Clustering for constraint networks. *Artificial Intelligence*, pages 61–95, 1989.

[4] H. Chen and V. Dalmau. From Pebble Games to Tractability: An Ambidextrous Consistency Algorithm for Quantified Constraint Satisfaction. In *In proc. of 19th Int.l workshop on Computer Science Logic*, volume 3634 of *Lecture Notes in Computer Science*. Springer-Verlag, 2005.

[5] Georg Gottlob, Gianluigi Greco, and Francesco Scarcello. The Complexity of Quantified Constraint Satisfaction Problems under Structural Restrictions. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 150–155. Professional Book Center, 2005.

[6] G. Pan and M.Y. Vardi. Fixed-Parameter Hierarchies inside PSPACE. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 27–36. IEEE Computer Society, 2006.

[7] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *5th Annual ACM Symposium on the Theory of Computation*, pages 1–9, 1973.

[8] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `www.qbflib.org`.

[9] M. Narizzano, L. Pulina, and A. Taccchella. QBF solvers competitive evaluation (QBFEVAL), 2006. `http://www.qbflib.org/qbfeval`.

[10] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, pages 277–284, 1987.

[11] E. Giunchiglia, M. Narizzano, and A. Tacchella. Clause-Term Resolution and Learning in Quantified Boolean Logic Satisfiability. *Artificial Intelligence Research*, 26:371–416, 2006. Available on-line at `http://www.jair.org/vol/vol26.html`.

[12] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Proceedings of International Conference on Computer Aided Design (ICCAD'02)*, 2002.

[13] M. Benedetti. sKizzo: a Suite to Evaluate and Certify QBFs. In *20th Int.l. Conference on Automated Deduction*, volume 3632 of *LNCS*, pages 369–376. Springer Verlag, 2005.

[14] A. Biere. Resolve and Expand. In *Seventh Intl. Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, volume 3542 of *LNCS*, pages 59–70, 2005.

[15] G. Pan and M.Y. Vardi. Symbolic Decision Procedures for QBF. In *10th Conference on Principles and Practice of Constraint Programming (CP 2004)*, 2004.

[16] I. Rish and R. Dechter. Resolution versus search: Two strategies for sat. *Journal of Automated Reasoning*, 24(1/2):225–275, 2000.

[17] L. Pulina and A. Taccchella. MIND-Lab projects and related information, 2008. `http://www.mind-lab.it/projects`.

[18] R.E. Tarjan and M. Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 14(1):254–255, 1985.

[19] V. Gogate and R. Dechter. A Complete Anytime Algorithm for Treewidth. In *UAI '04, Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence*, pages 201–208. AUAI Press, 2004.

[20] Sathiamoorthy Subbarayan and Henrik Reif Andersen. Backtracking Procedures for Hypertree, HyperSpread and Connected Hypertree Decomposition of CSPs. In *IJCAI*, pages 180–185, 2007.

[21] L. Pulina and A. Tacchella. A multi-engine solver for quantified boolean formulas. In *13th Conference on Principles and Practice of Constraint Programming (CP 2007)*, volume 4741 of *LNCS*, pages 574–589. Springer Verlag, 2007.

[22] I.H. Witten and E. Frank. *Data Mining (2nd edition)*. Morgan Kaufmann, 2005.

[23] H. Kleine-Büning, M. Karpinski, and A. Flögel. Resolution for Quantified Boolean Formulas. *Information and Computation*, 117(1):12–18, 1995.

[24] J. Larrosa and R. Dechter. Boosting Search with Variable Elimination in Constraint Optimization and Constraint Satisfaction Problems. *Constraints*, 8(3):303–326, 2003.