Artificial Intelligence Research and Development
I. Sanz et al. (Eds.)

doi:10.3233/FAIA230705

General Boolean Formula Minimization with QBF Solvers

Eduardo CALÒ [a,1] and Jordi LEVY [b]
[a] *Utrecht University, The Netherlands*
[b] *IIIA, CSIC, Spain*
ORCiD ID: Eduardo Calò https://orcid.org/0000-0003-3881-8994, Jordi Levy https://orcid.org/0000-0001-5883-5746

Abstract. The minimization of propositional formulae is a classical problem in logic, whose first algorithms date back at least to the 1950s with the works of Quine and Karnaugh. Most previous work in the area has focused on obtaining minimal, or quasi-minimal, formulae in conjunctive normal form (CNF) or disjunctive normal form (DNF), with applications in hardware design. In this paper, we are interested in the problem of obtaining an equivalent formula in any format, also allowing connectives that are not present in the original formula. We are primarily motivated in applying minimization algorithms to generate natural language translations of the original formula, where using shorter equivalents as input may result in better translations. Buchfuhrer and Umans have proved that the (decisional version of the) problem is Σ_2^p-complete. We analyze three possible (practical) approaches to solving the problem. First, using brute force, generating all possible formulae in increasing size and checking if they are equivalent to the original formula by testing all possible variable assignments. Second, generating the Tseitin coding of all the formulae and checking equivalence with the original using a SAT solver. Third, encoding the problem as a Quantified Boolean Formula (QBF), and using a QBF solver. Our results show that the QBF approach largely outperforms the other two.

Keywords. SAT Solvers, QBF Solvers, Boolean Formula Minimization, Natural Language Processing

1. Introduction

The minimization of complex Boolean expressions is a longstanding problem in logic. The first algorithms developed in the 1950s, e.g., the works of Quine, McCluskey [1,2,3], and Karnaugh [4] paved the way for extensions and optimizations in the following years (e.g., the Petrick's method [5], and the Espresso heuristic logic minimizer [6], *i.a.*). These works have focused on obtaining minimal equivalent representations in specific canonical forms (e.g., conjunctive normal form (CNF) or disjunctive normal form (DNF)), and confined the studies to a limited set of connectives. Here, we are interested in the *general* Boolean formula minimization, where no assumptions are made in the form of the input

[1]Corresponding Author: Eduardo Calò, e.calo@uu.nl. This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 860621 and the MICINN project PROOFS (PID2019-109137GB-C21).

formula or the output. In fact, our minimization methods allow us to use distinct sets of connectives for the input and the output.

We frame Boolean minimization (i.e., finding the logically equivalent shortest formula(e) to a given one) as a Quantified Boolean Formulae (QBF) satisfiability problem and design an algorithm that consistently finds the shortest equivalents of a given formula. We compare this algorithm with a brute force baseline, and an approach based on SAT.

Motivation We have two main motivations behind our work. The first one is shared in [7], where the authors present `qbf2epr`, a tool that translates QBF to formulae in effective propositional logic (EPR). Their aim is to generate benchmarks for EPR and compare solvers for QBF and EPR. Similarly, our formula minimization problem, encoded as QBF, generates benchmarks for QBF solvers and allows us to compare SAT and QBF techniques. The automated deduction community is divided into sub-communities (e.g., SAT, QBF, SMT, MaxSAT, EPR), which try to solve distinct classes of problems, from SAT that is NP-complete, to EPR that is NEXPTIME-complete, passing by QBF that is PSPACE-complete, and each one has its own competition and set of benchmarks. However, many ideas that proved effective in one area (like learning in SAT) have been exported to others. In this sense, problems that could be solved with two distinct technologies, like ours, contribute to comparing the level of maturity reached in each area.

Our second motivation relates to a use case of minimization algorithms in natural language processing. Grasping the meaning of logical formalisms is a crucial task for many scholars, yet sometimes even experienced logicians might have trouble deciphering a complex formula. This problem is exacerbated with students of logic, particularly when they encounter unfamiliar formal systems [8]. Techniques from natural language generation [9,10], and in particular logic-to-text generation methodologies [11,12], can be used for simplifying and translating logical formulae into optimally intelligible text in natural languages (NLs) (such as English, Mandarin, or Korean), which can effectively explain formulae to systems' users. For example, given the following first-order formula:

$$\exists x(Problem(x) \wedge \forall y(Researcher(y) \rightarrow Interested(y,x)))$$

we want a system that can automatically generate a faithful and comprehensible explanation, via the following (or another semantically equivalent) text:

There is a problem that every researcher finds interesting.

What are the characteristics that a formula should have to become a suitable input for a logic-to-text translation system? One aspect that one might want to look at is length. Brevity has surrounded linguistic debate at least since [13]. Arguably, shorter utterances should be preferred over longer ones and unnecessary prolixity should be avoided. This principle might also apply to logical formulae. Intuitively, a short formula, rather than a longer logical equivalent, should be better suited to be translated into NL. In this paper, we tackle exclusively the logical aspect of the problem. For the application of the algorithm developed in this paper to the linguistic aspect of the problem, see [14]. We focus on propositional logic, a formalism in which equivalence is decidable, and limit

our examination to formulae's length,[2] aiming only for the shortest equivalents to a given formula.

1.1. Related Work

Formula Minimization Boolean formula minimization is a natural optimization problem in the second level of the Polynomial-Time Hierarchy Σ_2^p. Indeed, the problem is used by [15] to motivate the definition of the Polynomial Hierarchy. Its decisional version can be formulated as the following problem: Given a Boolean formula, prove the existence of a (smaller) formula (in the same set of variables) that gets the same evaluation of the given formula, for all possible assignments of the variables. The fact that both sets of quantified variables, as well as the time to evaluate the formulae, are bounded in the input proves its inclusion in Σ_2^p. As we will see in Section 2, this corresponds to our brute-force algorithm. It is assumed that both the given formula and its minimization are circuits or formulae of the same form. However, in our implementation, we leave open the possibility to use distinct sets of connectives. Apart from some completeness proofs for some particular forms of the input and output [16], the proof for the general form had eluded researchers until [17] proved Σ_2^p-completeness of the problem.

The optimization of complex Boolean expressions has been studied extensively in electronic circuits, where practical matters (i.e., complex circuits take up physical space and costs more resources in their implementation) make it crucial to find optimal circuit representations. Well-known minimization methods include the Quine-McCluskey algorithm [1,2,3] and the Karnaugh map [4]. In the Karnaugh map, Boolean results are transferred from a truth table onto a two-dimensional grid, where each cell position represents one combination of input conditions, while each cell value is the corresponding output value. Optimal groups of 0s and 1s are identified, which represent the terms of a canonical form that can be used to write a minimal expression. The Quine–McCluskey algorithm finds all the prime implicants of a function and uses them in a chart to find (i) the essential prime implicants of the function, and (ii) other prime implicants that are necessary to cover the function. The method is functionally identical to the Karnaugh map, but its tabular form makes it more efficient to employ in computer systems.

However, despite this long history of research and attempts to extend well-established methods (e.g., [18] tries to implement the `XOr` operator in the Quine-McCluskey algorithm), most work has focused on a limited set of connectives and canonical forms (e.g., CNF or DNF). For our scope, we need a more general approach where all connectives could be, in principle, taken into account.

Quantified Boolean Formulae Quantified Boolean Formulae (QBFs) are an extension of propositional logic, where universal and existential quantifications are allowed [19]. The use of quantifiers results in a greater expressive power than classic propositional logic. If all variables occurring in a QBF ϕ are bound, then ϕ is called *closed*. QBFs often assume a canonical prenex conjunctive normal form (PCNF) $\phi = \exists\vec{x}\forall\vec{y}\exists\vec{z}\cdots\psi$, where the portion containing only quantifiers and bound variables is called the *prefix*, followed by ψ that is a quantifier-free Boolean formula with conjunctions over clauses, called the *matrix*.

[2] We define length as the number of symbols (i.e., predicates and connectives, parentheses excluded) contained in a formula.

Algorithm 1: Brute-force algorithm

```
Input: φ
Output: a minimal equivalent formula ψ
1  Function equivalent(φ, ψ):
2      foreach Assignment I : Var(φ) → {0,1} do
3          if I(φ) ≠ I(ψ) then
4              return false
5      return true
6  Function main(φ):
7      foreach i = 1,...,|φ| do
8          foreach formula ψ s.t. |ψ| = i and Var(ψ) ⊆ Var(φ) do
9              if equivalent(φ, ψ) then
10                 return ψ
```

The QBF satisfiability problem [20] consists of determining, for a given QBF ϕ, the existence of an assignment for the free variables, such that ϕ evaluates to true under this assignment. Hence, ϕ is true iff, there exists a truth assignment to $\vec{x}$, such that, for all truth assignments to $\vec{y}$, there exists a truth assignment to $\vec{z}$,... such that ψ is true. Several QBF solvers have been developed over time,[3] and applications of QBFs technologies range from AI to planning [21,22,23]. QBF solvers only use to provide the instantiation of most externally existentially-quantified variables $\vec{x}$, since for the other ones, instantiation depends on previous universal variables $\vec{z} = f(\vec{y})$. In this work, we exploit QBFs to encode and solve the Boolean minimization problem.

1.2. Structure of the Paper

The rest of the paper is structured as follows. Section 2 introduces the algorithms that we employ in our experiments and the QBF encoding we develop. Section 3 illustrates the experiments we carry out, comparing the three aforementioned approaches, and shows the results. We present some reflections on possible future directions in Section 4.

2. Algorithms

In our experimentation, we analyze three algorithms that we will call brute-force, SAT-based, and QBF-based.

Brute-force Algorithm The brute-force algorithm (see Alg. 1) is the algorithm that we mention in Section 1.1 as proof that formula minimization is in Σ_2^p. Two formulae ϕ and ψ are equivalent iff $\phi \leftrightarrow \psi$ is a tautology. Like TAUT, the formula equivalence problem is CoNP-complete. Considering that we test the equivalence for all formulae ψ smaller than ϕ, the average time required by `equivalent(`ϕ, ψ`)` is the same as considering ψ

[3] http://www.qbflib.org

a random formula smaller than ϕ. Assuming that the probability of $I(\phi) = 1$ is $1/2$, this average time would be $\mathcal{O}(|\phi|)$ (notice that in this situation, half of the calls finish after checking one assignment, $1/4$ after checking two assignments, etc. hence, on average we would check $\sum_{i=1}^{2^{|\phi|}} i\frac{1}{2^i} < 2$ assignments in every call). However, the assumption $P(I(\phi) = 1) = 1/2$ is, in general, false.

We can also estimate the number of calls to this function as follows. The number of distinct complete trees of size n that we can construct with k binary symbols is k^n. If the trees can have any form, then the computation is more complicated. Let $\mathscr{C}$ be the set of possible binary symbols (hence, we are not considering `Not`) and $\mathscr{V}$ be the set of possible leaves. The number of forms of trees with m binary nodes and $m+1$ leaves is given by the recurrence $f(m) = \sum_{i=0}^{m-1} f(i) f(m-i-1)$ that define the Catalan numbers C_m. The number of distinct trees will be $C_m |\mathscr{C}|^m |\mathscr{V}|^{m+1}$. Using Stirling approximation, this can be approximated as $\frac{4^m}{\sqrt{\pi} m^{3/2}} |\mathscr{C}|^m |\mathscr{V}|^{m+1}$. As a function of the tree size $n = 2m+1$, this is $\mathcal{O}((4|\mathscr{C}||\mathscr{V}|)^{n/2}/n^{3/2})$ calls to the `equivalent` function.

Algorithm 2: SAT-based algorithm

Input: ϕ
Output: a minimal equivalent formula ψ

```
1  Function tseitin(φ, x):
2      if φ = φ1 ∧ φ2 then
3          y1, y2 := freshvars()
4          return tseitin(φ1, y1) ∪ tseitin(φ2, y2) ∪ CNF({x ↔ y1 ∧ y2})
5      ··· /* Similarly for other connectives or variables */
6  Function equivalent(φ, ψ):
7      x1, x2 := freshvars()
8      Γ := tseitin(φ, x1) ∪ tseitin(ψ, x2) ∪ CNF({¬(x1 ↔ x2)})
9      return SAT(Γ) ≠ satisfiable
10 Function main(φ):
11     foreach i = 1, ..., |φ| do
12         foreach formula ψ s.t. |ψ| = i and Var(ψ) ⊆ Var(φ) do
13             if equivalent(φ, ψ) then
14                 return ψ
```

SAT-based Algorithm The second algorithm (see Alg. 2) is based on the use of a SAT solver and the Tseitin encoding of the two formulae that we want to prove equivalent. Given two formulae ϕ, ψ, we can find, in linear time $|\phi|+|\psi|$, a CNF formula Γ such that the two formulae are equivalent iff Γ *is not* satisfiable. Experiments show that, in practice, we still can get some gain with respect to the brute-force algorithm (see Section 3).

QBF-based Algorithm The third algorithm (see Alg. 3) is based on the use of a QBF solver. Here, instead of testing every possible minimal formula ψ, we test every possible depth δ. This supposes a significant improvement since there is a linear number of depths to try, instead of an exponential number of formula candidates. Second, instead of

Algorithm 3: QBF-based algorithm

Input: ϕ
Output: a minimal equivalent formula ψ

Function `scheme(`δ, z`):`
  $x_{false} := freshvar(\exists^{(1)})$
  $\Gamma := CNF(\{x_{false} \rightarrow \neg z\})$
  foreach $y \in \forall^{(1)}$ **do**
    $x_y := freshvar(\exists^{(1)})$
    $\Gamma := \Gamma \cup CNF(\{x_y \rightarrow (z \leftrightarrow y\})$
  if $\delta > 0$ **then**
    $z_1, z_2 := freshvar(\exists^{(2)})$
    $\Gamma := \Gamma \cup scheme(\delta - 1, z_1) \cup scheme(\delta - 1, z_2)$
    foreach $c \in \mathscr{C}$ **do**
      $x_c := freshvar(\exists^{(1)})$
      $\Gamma := \Gamma \cup CNF(\{x_c \rightarrow (z \leftrightarrow z_1\ c\ z_2)\})$
  return $\Gamma \cup CNF(\{x_{false} + \sum_{y \in \forall^{(1)}} x_y + \sum_{c \in \mathscr{C}} x_c = 1\})$

Function `equivalent(`ϕ, δ`):`
  $z_1, z_2 := freshvars(\exists^{(2)})$
  $\forall^{(1)} := Vars(\phi)$
  $\Gamma := tseitin(\phi, z_1) \cup scheme(\delta, z_2) \cup CNF(\{z_1 \leftrightarrow z_2\})$
  return $QBF(\Gamma) = true$

Function `main(`ϕ`):`
  foreach $\delta = 1, \ldots, depth(\phi)$ **do**
    if $equivalent(\phi, \delta)$ **then**
      return ψ *extracted from* Γ

a Tseitin encoding of the candidate, we compute a *scheme of the candidate*. This means that given a depth δ, we consider all terms of depth δ as possible candidates, without fixing the content of each node of the candidate. The equivalence between the original formula and this *scheme* can be encoded as a QBF formula with three quantifier alternations: $\exists\vec{x}.\forall\vec{y}.\exists\vec{z}.\Gamma$. In Alg. 3, these three sets of variables are represented as $\exists^{(1)}$, $\forall^{(1)}$, and $\exists^{(2)}$ and individual variables are named x, y, and z, respectively. If the QBF formula is true, the values we got for variables $x \in \exists^{(1)}$ will encode the minimal formula. Notice that QBF solvers only provide the instantiations of the most external existentially-quantified variables, since the values of the other existentially-quantified variables depend on more externally universally-quantified variables.[4]

These *schemes* are defined as follows. We assume that there is a maximal arity for all connectives; in our case 2 (although it could be generalized for any set of fixed-arity connectives). A scheme of depth δ is basically a complete tree of depth δ, therefore containing $2^{\delta+1} - 1$ nodes. For every node i of the scheme, and for every truth constant

[4] In the case of using a QBF solver unable to provide these instantiations, we cannot compute the minimal equivalent formula.

(we only consider the constant $false$), or variable y of the original formula, or connective c, we have a variable in x^i_{false}, x^i_y, $x^i_c \in \exists^{(1)}$. When $x^i_c \in \exists^{(1)}$ gets the value true, then at position i of the scheme, we have the connective c. Respectively, when variable $x^i_y \in \exists^{(1)}$ is true, we have variable y at position i, or constant $false$ when x^i_{false} is true. Additionally, we will have a variable x^i_{dummy} that gets the value true when at position i of the scheme there is not any content. The constraints:

$$x^i_{dummy} + x^i_{false} + \sum_{y \in \forall^{(1)}} x^i_y + \sum_{c \in \mathscr{C}} x^i_c = 1 \tag{1}$$

in the QBF formula will ensure that one, and only one, of them get the value true. Variables in $\forall^{(1)}$ are just the set of variables in the original formula. The original formula and the scheme are equivalent if *for all* assignments to these variables, both the original formula and scheme get the same evaluation. Variables in $z^i \in \exists^{(2)}$ encode the truth values for every possible subformula at position i of the scheme or of the Tseitin encoding of the original formula. The QBF formula will also contain restrictions of the form:

$$\begin{aligned} &x^i_{false} \rightarrow \neg z^i \\ &x^i_y \rightarrow (z^i \leftrightarrow y) \\ &x^i_{\texttt{Not}} \rightarrow (z^i \leftrightarrow \neg z^{i \cdot 1}) \\ &x^i_c \rightarrow (z^i \leftrightarrow z^{i \cdot 1}\, c\, z^{i \cdot 2}) \end{aligned} \tag{2}$$

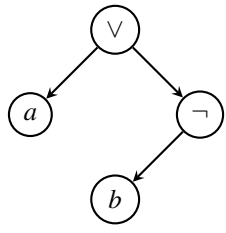
The intended meanings of these constraints are: If at position i of the scheme we have the constant $false$, the sub-scheme is evaluated to false, if there is an original variable $y \in \forall^{(1)}$, it is evaluated to y, and if there is a connective $c \in \mathscr{C}$, then the sub-scheme gets the same value as the connective c operated on the evaluations $z^{i \cdot 1}$ of the left-child of i and the evaluation $z^{i \cdot 2}$ of the right-child.

When in a node of the scheme we put a `Not`, we only use one of the children, and in the case of putting a variable, we do not use any of the children. To avoid useless search in the QBF solver, we can force all these useless nodes to be fixed to the dummy value by adding the following constraints to the QBF formula:

$$\begin{aligned} &x^i_{\neg} \rightarrow x^{i \cdot 2}_{dummy} \\ &(x^i_y \vee x^i_{dummy} \vee x^i_{false}) \rightarrow (x^{i \cdot 1}_{dummy} \wedge x^{i \cdot 2}_{dummy}) \end{aligned} \tag{3}$$

In this encoding of schemes, from the assignment computed by the QBF solver for the variables in the most-externally existentially-quantified $\exists^{(1)}$ of the formula Γ, we can obtain the minimized formula ψ. For instance, if the variables set to true are the ones on the left, the formula ψ is the one on the right:

$$\begin{array}{ll} x^{\varepsilon}_{\texttt{Or}} & \\ x^1_a & x^2_{\texttt{Not}} \\ x^{1 \cdot 1}_{dummy} & x^{2 \cdot 1}_b \\ x^{1 \cdot 2}_{dummy} & x^{2 \cdot 2}_{dummy} \end{array}$$



Notice that the size of the QBF formula we get is $\mathcal{O}(2^{depth(\phi)} \cdot (|\mathscr{C}| + |Vars(\phi)|))$. Assuming that the original formula is balanced, in practice $2^{depth(\phi)} \approx |\phi|$. Therefore, we could consider it a polynomial encoding. Notice also that we do not make a profit from the commutativity and associativity of most connectives.

In this approach, we only bound the depth of the scheme. If we also want to limit its size, we can add the encoding of some cardinality constraint that bound the number of nodes in the schema that are distinct from the dummy:

$$\sum_i \neg x^i_{dummy} \leq size\_bound \tag{4}$$

Notice that this QBF-based algorithm may be used to minimize the depth of an equivalent formula, without using Eq. (4). If instead, we want to minimize the size of an equivalent formula, it is not so simple as adding Eq. (4). It is possible that, given an original formula ϕ, there exists an equivalent formula ψ, satisfying $size(\psi) < size(\phi)$, but where $depth(\psi) > depth(\phi)$, and it would not be found by the algorithm. Stricto sensu, we should try all schemes of depth $depth(\psi) \leq size(\phi)$ to avoid this situation and ensure that all smaller-sized equivalent candidates are considered. However, this would lead us to obtain QBF formulae of size $\mathcal{O}(2^{size(\phi)})$. In the experiments (see Section 3), we have observed that it is enough to consider all schemes of size $size(\psi) \leq size(\phi)$ and imposing a bound $depth(\psi) \leq \log_2 size(\psi) + 1$, to get the same results as with the other algorithms.

3. Experiments and Results

We conduct some experimentation with our algorithms. The three algorithms are implemented in Python 3 and are publicly available at `https://gitlab.nl4xai.eu/eduardo.calo/QBF-boolean-minimization`. In the case of the SAT-based algorithm, we use `Glucose 4` [24], a state-of-the-art SAT solver, via the Python module `python-sat`.[5] In the case of the QBF-based algorithm, we use the state-of-the-art `CAQE` [25,26] QBF solver, although any other QBF solver that accepts QDIMACS standard[6] input and output may be used.

For every size in $s = 1, \ldots, 20$, we generate 100 random formulae of size s and minimize them using the three algorithms. We make sure that all syntactically distinct formulae are generated with the same probability. We do it carefully to avoid any bias. However, we do not consider commutativity and associativity of connectives or other formula equivalences. We generate formulae of size s over a set of $\sqrt{s}$ variables[7] and connectives $\mathscr{C} = \{\texttt{Not}, \texttt{And}, \texttt{Or}\}$, and minimizations are searched among formulae with connectives $\mathscr{C}' = \{\texttt{Not}, \texttt{And}, \texttt{Or}, \texttt{Implies}\}$.[8] We also check that our conclusions are the same for another number of variables (e.g., half of the desired size, etc.) and that the implementations agree on the results in terms of formula size using any of the three methods.

[5] `https://github.com/pysathq/pysat`

[6] `http://www.qbflib.org/qdimacs.html`

[7] Using s/c or s^c variables does not seem to affect substantially the results.

[8] We include `Implies` in $\mathscr{C}'$ to remark that the set of input and output connectives may be different (contrarily to other approaches).

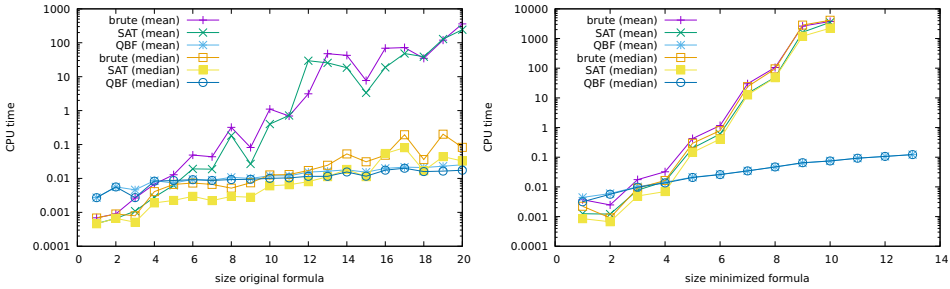


Figure 1. Average and median time required by the three algorithms with respect to the size of the original formula (left) and the resulting minimized formula (right).

We use a cluster with 11 calculating nodes with 2 Intel Xeon CPUs at $2.2GHz$ with 10 cores/CPU and $92GB$ of RAM. We set a time-out of $20,000s$. The brute-force and the SAT-based algorithms reach the time-out in some instances for $s = 15, 18, 19, 20$. These values are not considered in the computation of the mean and median times. Therefore, these mean and median values are abnormally low.

In Figure 1 (left), we show the average and median (logarithm of) CPU time required by each one of the algorithms as a function of the size of the input formula. We clearly observe that the QBF-based algorithm outperforms the other two algorithms, which seem to require exponential time on the size of the input. We also observe that the SAT-based is consistently better than the brute-force algorithm (a constant distance between the functions, in logarithmic axes, means an improvement of constant factor). This is quite surprising since, as we mention in Section 2, the computation of the formula equivalence can be done in linear average time.[9] It is also remarkable that, in the case of brute-force and SAT-based, there is a significant difference between the average and median time. The reason, as we comment in detail below, is the significant variability in the times required by each instance. The same effect produces a fluctuation in the values of the average time. We can conclude that, although in most of the instances (attending to the median), the three algorithms minimize the formula in less than one second, for sizes smaller than 20, just a few instances make brute-force and SAT-based require around $1h$ on average when the size is around 20.

In Figure 1 (right), we show the average and median (logarithm of) CPU time as a function of the size of the obtained minimal formula. Here the differences between the mean and median times are smaller. Hence, we can conclude that the size of the output determines the time required by the algorithms. Again, it is clear that the QBF-based algorithm outperforms the other two. We still observe that the median time is smaller than the average time, which indicates that significant variability still exists. Curiously, the times depend on the parity of the formula sizes: Even-size formulae are easier than odd-size formulae. The reason could be that, except in the case of negation, the rest of the connectives are binary.

In Figure 2 (left), we show how the average size of the minimized formula grows with respect to the size of the original formula. We observe that the growth is close to the square root of the original size. Recall that we generate random formulae of size s

[9] Equivalence checking of two propositional formulae is CoNP-complete in the worse case. However, for random formulae, and random assignments, the average time is decent.

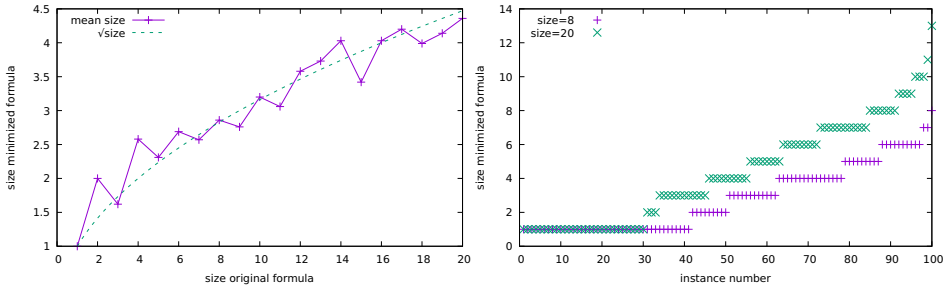


Figure 2. Average size of the minimized formula w.r.t. the size of the original formula (left) and distribution of minimized sizes for formulae of original size 8 and 20 (right).

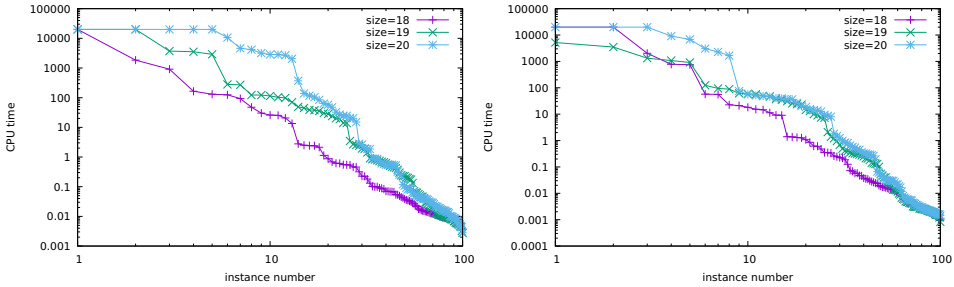


Figure 3. Distribution of times for the instances solved with the brute-force algorithm (left) and SAT-based algorithm (right).

and with $\sqrt{s}$ variables. Curiously, we observe that odd-size formulae are simplified more than even-size formulae, although the reason for this is not clear. In Figure 2 (right), we show the distribution of sizes of the minimized formulae (for original formulae of sizes 20 and 8).

As mentioned above, we observe significant variability in CPU times, for the brute-force and SAT-based algorithms. In Figure 3, we sort the instances in decreasing order of CPU time and represent, in double logarithmic axes, these times for the 100 instances. We observe that this representation is close to a line (truncated on the top due to time-outs) with an increasing (negative) slope when the size increases. This implies that the CPU time in these algorithms follows a power-law probability distribution, where the time required by a few instances is responsible for most of the average time. The standard solution in these situations is to use some kind of restart policy or some randomization of the algorithm. In our case, we could randomize the order of the candidates to minimal formulae. However, since we want to obtain the minimal equivalent formula, we cannot randomize the order of the sizes of formulae that we try.

4. Conclusion and Future Work

In this paper, we have analyzed the practical use of three algorithms for general Boolean formula minimization: a simple one that proves that the problem is in Σ_2^p, one based on the use of a SAT solver to check formula equivalences, and one that uses a Tseitin

encoding of a formula's scheme and a QBF solver. We show that the third one clearly outperforms the other two. The use of QBF solvers represents thus the state-of-the-art for the Boolean minimization problem.

We have done the experimentation using random formulae. We plan to expand the work using formulae that are not randomly generated, e.g., deriving from natural language, or realistic random formulae [27,28,29]. Moreover, our experiments have been limited to Boolean formulae. A natural extension of this work would be to see if this or similar methods could scale up to other (more expressive) formalisms, e.g., first-order logic (FOL). This would open up a range of interesting research questions, as in FOL, equivalence is undecidable. Adapting the QBF approach would probably not be feasible, yet, a semi-brute force approach, e.g., using a first-order theorem prover, could prove successful.

References

[1] Quine WV. The Problem of Simplifying Truth Functions. The American Mathematical Monthly. 1952;59(8):521-31.

[2] Quine WV. A Way to Simplify Truth Functions. The American Mathematical Monthly. 1955;62(9):627-31.

[3] McCluskey EJ. Minimization of Boolean functions. The Bell System Technical Journal. 1956;35(6):1417-44.

[4] Karnaugh M. The map method for synthesis of combinational logic circuits. Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics. 1953;72(5):593-9.

[5] Petrick SR. A direct determination of the irredundant forms of a Boolean function from the set of prime implicants. Air Force Cambridge Res Center Tech Report. 1956:56-110.

[6] Brayton RK, Hachtel GD, Hemachandra LA, Newton AR, Sangiovanni-Vincentelli ALM. A comparison of logic minimization strategies using ESPRESSO: An APL program package for partitioned logic minimization. In: Proceedings of the International Symposium on Circuits and Systems; 1982. p. 42-8.

[7] Seidl M, Lonsing F, Biere A. qbf2epr: A Tool for Generating EPR Formulas from QBF. In: Fontaine P, Schmidt RA, Schulz S, editors. PAAR-2012. Third Workshop on Practical Aspects of Automated Reasoning. vol. 21 of EPiC Series in Computing; 2013. p. 139-48.

[8] Rector A, Drummond N, Horridge M, Rogers J, Knublauch H, Stevens R, et al. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In: Engineering Knowledge in the Age of the Semantic Web: 14th International Conference, EKAW 2004, Whittlebury Hall, UK, October 5-8, 2004. Proceedings 14. Springer; 2004. p. 63-81.

[9] Reiter E, Dale R. Building Natural Language Generation Systems. Studies in Natural Language Processing. Cambridge University Press; 2000.

[10] Gatt A, Krahmer E. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. Journal of Artificial Intelligence Research. 2018;61:65-170.

[11] Ranta A. Translating between language and logic: what is easy and what is difficult. In: Proceedings of the International Conference on Automated Deduction. Springer; 2011. p. 5-25.

[12] Calò E, van der Werf E, Gatt A, van Deemter K. Enhancing and Evaluating the Grammatical Framework Approach to Logic-to-Text Generation. In: Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM). Abu Dhabi, United Arab Emirates (Hybrid): Association for Computational Linguistics; 2022. p. 148-71.

[13] Grice HP. Logic and conversation. In: Speech acts. Brill; 1975. p. 41-58.

[14] Calò E, Levy J, Gatt A, Van Deemter K. Is Shortest Always Best? The Role of Brevity in Logic-to-Text Generation. In: Proceedings of the The 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023). Toronto, Canada: Association for Computational Linguistics; 2023. p. 180-92. Available from: https://aclanthology.org/2023.starsem-1.17.

[15] Garey MR, Johnson DS. Computers and intractability. vol. 174. freeman San Francisco; 1979.

[16] Umans C, Villa T, Sangiovanni-Vincentelli AL. Complexity of two-level logic minimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2006;25(7):1230-46.

[17] Buchfuhrer D, Umans C. The complexity of Boolean formula minimization. Journal of Computer and System Sciences. 2011;77(1):142-53.
[18] Turton BCH. Extending Quine-McCluskey for Exclusive-Or logic synthesis. IEEE Transactions on Education. 1996 Feb;39(1):81-5.
[19] Beyersdorff O, Janota M, Lonsing F, Seidl M. Quantified Boolean Formulas. In: Biere A, Heule M, van Maaren H, Walsh T, editors. Handbook of Satisfiability - Second Edition. vol. 336 of Frontiers in Artificial Intelligence and Applications. IOS Press; 2021. p. 1177-221.
[20] Giunchiglia E, Marin P, Narizzano M. Reasoning with quantified boolean formulas. In: Handbook of satisfiability. IOS Press; 2009. p. 761-80.
[21] Cashmore M, Fox M. Planning as QBF. International Conference on Automated Planning and Scheduling Doctoral Consortium (ICAPS 2010). 2010.
[22] Diptarama RY, Shinohara A. QBF encoding of generalized tic-tac-toe. In: 4th International Workshop on Quantified Boolean Formulas (QBF) Co-located with 19th International Conference on Theory and Applications of Satisfiability Testing (SAT), Bordeaux, France; 2016. p. 14-26.
[23] Shukla A, Biere A, Pulina L, Seidl M. A Survey on Applications of Quantified Boolean Formulas. In: 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI). Portland, OR, USA: IEEE; 2019. p. 78-84.
[24] Audemard G, Simon L. On the glucose SAT solver. International Journal on Artificial Intelligence Tools. 2018;27(01):1840001.
[25] Rabe MN, Tentrup L. CAQE: A Certifying QBF Solver. In: 2015 Formal Methods in Computer-Aided Design (FMCAD); 2015. p. 136-43.
[26] Tentrup L. CAQE and QuAbS: Abstraction Based QBF Solvers. Journal on Satisfiability, Boolean Modeling and Computation. 2019 Sep;11:155-210.
[27] Ansótegui C, Bonet ML, Levy J. Random SAT Instances à la Carte. In: Artificial Intelligence Research and Development, Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence, CCIA 2008. vol. 184 of Frontiers in Artificial Intelligence and Applications. IOS Press; 2008. p. 109-17. Available from: https://doi.org/10.3233/978-1-58603-925-7-109.
[28] Ansótegui C, Bonet ML, Levy J. Towards Industrial-Like Random SAT Instances. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009; 2009. p. 387-92. Available from: http://ijcai.org/Proceedings/09/Papers/072.pdf.
[29] Ansótegui C, Bonet ML, Levy J. Scale-Free Random SAT Instances. Algorithms. 2022;15(6):219. Available from: https://doi.org/10.3390/a15060219.