Artificial Intelligence 233 (2016) 73–83

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

On the query complexity of selecting minimal sets for monotone predicates

Mikoláš Janota[*], Joao Marques-Silva

IST/INESC-ID, Lisbon, Portugal

ARTICLE INFO

Article history:
Received 18 June 2014
Received in revised form 29 December 2015
Accepted 2 January 2016
Available online 7 January 2016

Keywords:
Query complexity
Monotone predicates
Minimal set
SAT
Backbone
Minimal unsatisfiable set
Minimal correction set
Independent variables

ABSTRACT

Propositional Satisfiability (SAT) solvers are routinely used for solving many function problems. A natural question that has seldom been addressed is: what is the number of calls to a SAT solver for solving some target function problem? This article improves upper bounds on the query complexity of solving several function problems defined on propositional formulas. These include computing the backbone of a formula and computing the set of independent variables of a formula. For the general case of monotone predicates, the article improves upper bounds on the query complexity of computing a minimal set when the number of minimal sets is constant. This applies for example to the computation of a minimal unsatisfiable subset (MUS) for CNF formulas, but also to the computation of prime implicants and implicates, with immediate application in a number of AI settings.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

The practical success of Boolean Satisfiability (SAT) solvers is demonstrated by an ever increasing number of applications. While some of these applications are naturally formulated as decision problems; others are not. In many settings SAT solvers are used for solving function problems. This is the case for example with computing a minimal unsatisfiable subset (MUS) of a CNF formula, a minimal correction subset (MCS) of a CNF formula, the backbone of a propositional formula, a prime implicant or implicate of a propositional formula, the largest autark assignment of a CNF formula, among many other function problems. Some of these function problems find important practical applications. For example MUSes are routinely used in abstraction refinement loops in software model checking (e.g. [1]) but also in ontology debugging [2,3], prime implicates and implicants are used in model checking [4,5], but also find a wide range of applications in AI [6], and backbones are used for example in configuration and in post-silicon fault localization [7] but also in optimization [8,9].

Despite the vast number of settings in which SAT solvers are used for solving function problems, the worst-case number of times a SAT solver is called for solving these problems is in general not known with sufficient accuracy. For example, it is not known what is the *best* worst-case number of calls to a SAT solver for computing the backbone of a propositional formula, or for computing the maximum autarky of a CNF formula, or for computing a minimal unsatisfiable subset, or for computing a maximally satisfiable subset, or for computing a prime implicant or implicate, among many other function problems. Moreover, the exact query complexity of computing an MUS has been an open research topic for around two

[*] Corresponding author.
E-mail addresses: mikolas@sat.inesc-id.pt (M. Janota), jpms@ieee.org (J. Marques-Silva).

http://dx.doi.org/10.1016/j.artint.2016.01.002
0004-3702/© 2016 Elsevier B.V. All rights reserved.

decades [10]. This article shows how worst-case number of SAT calls can be improved by using a SAT oracle with a witness. However, lower bounds for the number of calls are still left for future work.

We should note that lowering the number of SAT calls is not necessarily better from the practical perspective. Indeed, if a large number of SAT calls is replaced with a low number of very hard SAT calls it is not clear that the performance improves. However, the reduction of number of SAT calls effectively *delegates* some work to the SAT solver but also gives it more information about the problem, which is in general beneficial. In some cases, the best performance is obtained by hybrid approaches, where the number of SAT calls is parameterized, see for instance [11]. Hence, improving the upper bounds on the number of SAT calls opens possibilities for such algorithms.

This article extends recent work on unifying the approaches used for solving a large number of function problems related to propositional formulas [12,13], namely function problems that can be modeled as computing a minimal set subject to a monotone predicate. More concretely, this article investigates the number of times SAT solvers are called for solving function problems, and refines known upper bounds for a number of problems.

Practical algorithms for solving function problems are often modeled with polynomial time algorithms that call a SAT solver a polynomial number of times. A well-known measure of the hardness of these function problems is the number of times the SAT solver is called given the problem instance size, i.e. the query complexity of the function problem. (Other possible measures of hardness, including the size or hardness of problem instances, are not considered in this and most earlier work.) Following standard approaches in computational complexity, a SAT solver can be viewed as an *oracle*: given an instance, the SAT solver answers either positively (for satisfiable instances) or negatively (for unsatisfiable instances). However, the operation of SAT solvers differs *substantially* from the traditional NP oracle (e.g. [14,15]). In the case of positive answers, the SAT solver also returns a satisfying truth assignment (i.e. a *witness*). The ability of the SAT solver to return a witness on positive answers can have a profound impact on query complexity characterizations of function problems (related results have actually been investigated in the past, e.g. [16,17]). This article shows that for some function problems, the availability of witnesses enables reducing the worst case number of oracle queries from polynomial to logarithmic. As a result, this article proposes to use witness oracles [18,19] to model the operation of SAT solvers, as an alternative to the standard NP oracle, and to develop query complexity characterizations of function problems using witness oracles. These query complexity characterizations can then serve to compare available algorithms against the best known theoretical results.

The contributions of this article can be summarized as follows.[1] For function problems that can be represented as computing a minimal set subject to a monotone predicate in certain general forms [12,13], and have a constant number of minimal sets, the article shows that a minimal set can be computed with a logarithmic number of calls to a witness oracle. This result has a number of consequences, which the article also analyzes. For example, the backbone of a propositional formula or the independent variables of a propositional formula can be computed with a logarithmic number of calls to a witness oracle. In addition, a few additional results are included in the article, related to special cases of other function problems when the number of minimal sets is constant. Observe that, although the article addresses function problems solved with polynomial number of calls to a SAT solver, other *oracles* commonly used in practice could be considered. These include for example SMT, CSP, and QBF solvers.

The article is organized as follows. Section 2 introduces the notation used in the article. Section 3 introduces the studied problem and some of its particular instances. Section 4 develops query complexity results for certain forms of monotone predicates; it is shown that minimal sets for these predicates can be computed with a logarithmic number of calls to an NP oracle that provides a witness (the class $\mathsf{FP}^{\mathsf{NP}}[\mathsf{wit},\log]$). Section 5 extends the results of the previous section for function problems that have exactly a constant number of minimal sets. Section 6 relates to existing work and the article concludes in Section 7.

2. Preliminaries

This section introduces the notation and definitions used throughout the article.

2.1. Propositional logic

Standard propositional logic definitions are used throughout the article (e.g. [21,22]), including propositional formulas, truth assignments, etc. Some definitions are briefly reviewed in this section.

Sets are represented in calligraphic font, e.g. $\mathcal{R}, \mathcal{W}$. Propositional formulas are also represented in calligraphic font, e.g. $\mathcal{F}, \mathcal{H}, \mathcal{T}$. Propositional variables are represented with letters from the end of the alphabet, e.g. x, y, z, and indices can be used, e.g. x_1, y_1. A literal is a variable x_i or its negation $\neg x_i$.

The set of variables of a propositional formula $\mathcal{F}$ are represented by $\mathsf{var}(\mathcal{F})$. For simplicity, if not specified otherwise, the set of variables of a formula will be denoted by $X \triangleq \mathsf{var}(\mathcal{F})$. A clause c is a non-tautologous set of literals, interpreted as a disjunction of literals. A CNF formula $\mathcal{F}$ is set clauses, interpreted as a conjunction of clauses.

[1] A preliminary version of this article is available as an ECCC report [20].

Download English Version:

https://daneshyari.com/en/article/376787

Download Persian Version:

https://daneshyari.com/article/376787

Daneshyari.com