Empowering ASPQ to Win in QBFEval 2018

Bernardo Cuteri[1], Carmine Dodaro[2], and Francesco Ricca[1]

[1] University of Calabria, Italy - lastname@mat.unical.it
[2] University of Genoa, Italy - dodaro@dibris.unige.it

Abstract. Answer Set Programming (ASP) is an established logic-based programming paradigm which has been successfully applied for solving complex problems since it features efficient implementations. In previous work we showed how to obtain ASPQ, a fairly effective 2QBF solver, by just resorting to state of the art ASP solvers. In this work we describe how we have optimized it to exploit some unexpressed potential of its solving strategy. The resulting empowered solver won the Hard Instances Track in the 2018 QBFEval.

Keywords: Answer Set Programming · Quantified Boolean Formulas · Hard Instances

1 Introduction

Answer Set Programming (ASP) [23] is a declarative programming paradigm that has been developed in the field of logic programming and nonmonotonic reasoning. The idea of ASP is to represent a given computational problem by means of a logic program whose stable models [36] (or answer sets) correspond to the desired solutions, and then to use an ASP solver to actually compute the stable models. Indeed, a robust solving technology has been developed [40, 2, 24, 31–35, 9, 10, 17–19, 43, 42, 38, 37], and useful extensions of the ASP semantics have been introduced [3, 14, 15, 6, 11]. As a matter of fact, ASP has been used in numerous scientific applications ranging from Artificial Intelligence [20, 29, 12, 1, 8]; Bioinformatics [25]; Databases [22, 41]; and Game Theory [16, 7]. Moreover, ASP is attracting increasing interest also beyond the scientific community, and counts already some successful application in industrial products [26, 27, 39]. ASP has become a popular choice for solving complex problems since it combines an expressive language with efficient implementations. Indeed, the results of the latest ASP Competition series [24] witness the continuous improvements achieved in the field of ASP solving.

The core language of ASP, which features disjunction in rule heads and nonmonotonic negation in rule bodies, can be used to solve all problems at the second level of the polynomial hierarchy. Given the large progress measured in the last few years in ASP solving, it is natural to ask whether this solving technology can applied profitably also for solving 2QBFs, the canonical problem for the second level of the polynomial hierarchy. Actually, we implemented a quite effective ASP-based 2QBF solver called ASPQ [13]. We obtained it reusing for a pragmatic goal a well-known theoretical result by Eiter and Gottlob. Indeed, to prove that answer set existence for a disjunctive logic programs is hard for the second level fo the polynomial hierarchy Eiter and Gottlob

provided a reduction from 2QBF in [28]. In ASPQ this transformation is implemented efficiently to actually solve an instance of 2QBF Φ by obtaining the corresponding ASP program $T^{eg}(\Phi)$ and then solving it with an ASP solver. Actually, ASPQ first calls CLASP, and if after a time threshould a solution cannot be found, it calls WASP. This is to exploit a somehow non overlapping behavior of the two solvers, that are based on different answer set checking techniques. ASPQ demonstrated to be a valid solver since the first participation in QBFEval [44]. Indeed, ASPQ entered as non-competitive participant the 2016 QBF Evaluation and obtained a fairly acceptable result in the 2QBF track, obtaining (virtually) the fifth place, thus performing better than various native QBF solvers. In 2017, we just updated the binaries of the ASP solvers to the last version, and fixed a bug of the script that did not allow in 2016 to exploit the entire time limits (the solver self-killed itself at 600 seconds), and ASPQ reached the fourth place at the 2QBF track. In this paper we describe how we have optimized it to better exploit the two state of the art ASP solvers used as internal engines. The resulting empowered version reached the fourth place in the 2QBF Track and won the Hard Instances Track in the 2018 QBFEval.

The paper is structured as follows: In section 2 we overview quantified boolean formulas; in Section 3 we recall the ASP language; in Section 4 we describe the Eiter and Gottlob transformation; in Section 5 we describe the main algorithm of our solver; in Section 6 we comment on the performance of ASPQ at the 2018 QBFEval; we draw the conclusion in Section 7.

2 Quantified Boolean Formulas

In this section, we introduce the logic of Quantified Boolean Formulas (QBFs), where variables can be existentially or universally quantified. Hence, QBF extends propositional logic, where all variables are existentially quantified, and the satisfiability problem comes from *NP*-complete to *PSPACE*-complete [45]. We start to introduce syntax and semantics of QBFs.

2.1 Syntax

A *variable* x is an element of a set Γ of propositional letters and a *literal* l is a variable (e.g., x) or the negation of a variable (e.g., $\neg x$). We denote by $|l|$ the variable occurring in the literal l, and by $\neg l$ the complement of l , i.e., $\neg l = x$, if $l = \neg x$, and $\neg l = \neg x$, if $l = x$. Given a natural number $k > 0$, a *k-clause* is a disjunction of k literals, and a *propositional formula* of arity k is a conjunction of k-clauses. A *quantified Boolean formula* Φ is an expression of the form

$$Q_1 x_1 \dots Q_n x_n F \tag{1}$$

where, for each $i = 1, \dots, n$, x_i is a variable, Q_i is either an existential quantifier, that is $Q_i = \exists$ or a universal quantifier, that is $Q_i = \forall$, and F is a propositional formula in the variables x_1, ..., x_n, called the *matrix* of Φ. We say that l is an *existential literal*, if $|l| = x_i$ and $Q_i = \exists$, for some $i = 1, \dots, n$, otherwise we say that l is a *universal literal*. Finally, whenever there is $1 \leq k \leq n$ such that, $Q_1 = \dots = Q_k = \exists$, and $Q_{k+1} = \dots = Q_n = \forall$, we say that Φ is a 2QBF formula.

2.2 Semantics

First, given a literal l and a QBF formula of the form $\Phi = Qx\Psi$, where Ψ is a QBF formula, we denote by Φ_l the QBF formula obtained from Ψ by removing all the conjuncts in which l occurs, and removing $\neg l$ from the others. Moreover, we say that a clause is *contradictory*, whenever it does not contain existential literals.

Let Φ be a QBF formula of the form (1). We define the semantics of a QBF formula recursively as follows. If F contains a contradictory clause, then Φ is false; if F has no conjuncts, then Φ is true; if $\Phi = \exists x\Psi$, and Φ_x or $\Phi_{\neg x}$ are true, then Φ is true; if $\Phi = \forall x\Psi$, and Φ_x and $\Phi_{\neg x}$ are true, then Φ is true. The QBF satisfiability problem (QSAT) is to decide whether a given QBF formula is true or false.

Example 1. Consider the 2QBF formula $\Phi = \exists x \forall y F$, where $F = (x \vee y) \wedge (\neg x \vee \neg y)$. Therefore, $\Phi_x = \forall y(\neg y)$ and $\Phi_{\neg x} = \forall y(y)$. Then, from Φ_x, we obtain $(\Phi_x)_y = ()$ and $(\Phi_x)_{\neg y} = \emptyset$; and from $\Phi_{\neg x}$, we obtain $(\Phi_{\neg x})_y = \emptyset$ and $(\Phi_{\neg x})_{\neg y} = ()$. Hence, since $(\Phi_x)_{\neg y}$ is false and $(\Phi_x)_{\neg y}$ is true, thus Φ_x is false; and since $(\Phi_{\neg x})_y$ is true and $(\Phi_{\neg x})_{\neg y}$ is false, thus $\Phi_{\neg x}$ is also false. Therefore, we can conclude that Φ is false.

Example 2. Consider the 2QBF formula $\Phi = \exists x \forall y F$, where $F = (x \vee y) \wedge (x \vee \neg y)$. Therefore, $\Phi_x = \forall y \emptyset$ and $\Phi_{\neg x} = \forall y(y) \wedge (\neg y)$. Then, in particular, Φ_x is true. Therefore, we can already conclude that Φ is true.

3 Answer Set Programming

Answer Set Programming (ASP) [23] is a programming paradigm developed in the field of nonmonotonic reasoning and logic programming. In this section we overview the syntax and the semantics of ASP.

3.1 Syntax

Following the traditional grounding view [36], we concentrate on programs over a propositional signature Λ. A *disjunctive rule* r is of the form

$$a_1 \vee ... \vee a_l \leftarrow b_1, ..., b_m, \texttt{ not } b_{m+1}, ..., \texttt{ not } b_n, \tag{2}$$

where all a_i and b_j are atoms (from Λ) and $l \geq 0, n \geq m \geq 0$ and $l+n > 0$; `not` represents *negation-as-failure*, also known as *default negation*. The set $H(r) = \{a_1, ..., a_l\}$ is the *head* of r, while $B^+(r) = \{b_1, ..., b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$ are the *positive body* and the *negative body* of r, respectively; the *body* of r is $B(r) = B^+(r) \cup B^-(r)$. We denote by $At(r) = H(r) \cup B(r)$ the set of all atoms occurring in r. A rule r is a *fact*, if $B(r) = \emptyset$ (we then omit $\leftarrow$); a *constraint*, if $H(r) = \emptyset$; *normal*, if $|H(r)| \leq 1$; and *positive*, if $B^-(r) = \emptyset$. A *(disjunctive logic) program* P is a finite set of disjunctive rules. P is called *normal* [resp. *positive*] if each $r \in P$ is normal [resp. positive]. Finally, we denote by $At(P) = \bigcup_{r \in P} At(r)$ the set of all atoms occurring in the program P.

3.2 Semantics

Any set $I \subseteq \Lambda$ is called *interpretation*. An interpretation I is a *model* of a program P (denoted by $I \models P$) if, and only if, for each rule $r \in P$, $I \cap H(r) \neq \emptyset$ if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$ (denoted by $I \models r$). A model M of P is *minimal* if, and only if, no model $M' \subset M$ of P exists. We denote by $MM(P)$ the set of all minimal models of P. Given an interpretation I, we denote by P^I the so-called *Gelfond-Lifschitz reduct* [36] of P with respect to I, that is the set of rules $a_1 \vee ... \vee a_l \leftarrow b_1, ..., b_m$, obtained from rules $r \in P$ of form (2), such that $B^-(r) \cap I = \emptyset$. An interpretation I is called *answer set (or stable model)* of P, whenever $I \in MM(P^I)$. Finally, we denote by $AS(P)$ the set of all answer sets of P. A program P such that $AS(P) \neq \emptyset$ is called *coherent*, otherwise it is called *incoherent*.

Example 3. Consider the disjunctive logic program

$$P = \{a \leftarrow c, \texttt{not}\ b;\ b \vee c \leftarrow \texttt{not}\ d\}.$$

Therefore, the set of all minimal model of P is given by $MM(P) = \{\{d\}, \{b\}, \{a,c\}\}$. Instead, the set of all answer sets is $AS(P) = \{\{b\}, \{a,c\}\}$. Hence, P is a coherent logic program. Note that $I = \{d\}$ is not an answer set of P, since I is not a minimal model of $P^I = \{a \leftarrow c\}$.

Example 4. Consider the logic program

$$P_\Phi = \left\{ \begin{array}{ll} a \leftarrow d, \texttt{not}\ b; & d \leftarrow \texttt{not}\ e; \\ b \leftarrow \texttt{not}\ c; & c \leftarrow \texttt{not}\ a \end{array} \right\}.$$

Therefore, the set of all minimal model of P is given by $MM(P) = \{\{a,c,d\}, \{a,b,d\}, \{a,b,e\}, \{b,c,d\}, \{c,e\}\}$. However, $\{a,b,e\}$ and $\{c,e\}$ are not answer sets of P, as the rule $d \leftarrow \texttt{not}\ e$ is deleted in the reduct, and e can not belong to a minimal model of the reduct. Moreover, $P^{\{a,c,d\}} = \{a;\ d\}$; $P^{\{a,b,d\}} = \{b;\ d\}$; and $P^{\{b,c,d\}} = \{c;\ d\}$. Therefore, also $\{a,c,d\}$, $\{a,b,d\}$, and $\{b,c,d\}$ are not answer sets of P. Thus, P is an incoherent logic program.

4 Encoding 2QBF in ASP

In this section, we introduce the translation from 2QBFs to logic programs proposed by Eiter and Gottlob [28] to prove the Σ_2^P-hardness of checking whether a disjunctive logic program has some answer set.

To describe the translation, let $\Phi = \exists X \forall Y F$ be a quantified boolean formula, where we may assume that $X = \{x_1, \ldots, x_e\}$, $Y = \{y_1, \ldots, y_a\}$ and $F = D_1 \vee \ldots \vee D_m$, such that $D_i = L_{i,1} \wedge \ldots \wedge L_{i,a+e}$ and $L_{i,j}$ are literals over $X \cup Y$. For every atom $z \in X \cup Y$, we introduce a fresh atom z', and we set $\sigma(z) = z$ and $\sigma(\neg z) = z'$. Finally, we introduce one more fresh atom, say w, and define a disjunctive logic program P_Φ consisting of the following rules:

$$\begin{array}{ll} z \vee z' & \forall\, z \in X \cup Y \\ y \leftarrow w \ \text{ and } \ y' \leftarrow w & \forall\, y \in Y \\ w \leftarrow \sigma(L_{i,1}), \ldots, \sigma(L_{i,a+e}) & \forall\, i = 1, \ldots, m \\ w \leftarrow \texttt{not}\ w & \end{array}$$

Eiter and Gottlob in [28] proved the following result.

Theorem 1. *Let Φ be a 2QBF formula. Then, Φ is true if, and only if, P_Φ is coherent.*

We enlighten the translation described above through two examples.

Example 5. Consider the 2QBF formula

$$\Phi = \exists x \forall y \forall z((x \wedge y) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg y \wedge z)).$$

Therefore, the corresponding logic program is

$$P_\Phi = \left\{ \begin{array}{ccc} x \vee x'; & y \vee y'; & z \vee z'; \\ y \leftarrow w; & y' \leftarrow w; & \\ z \leftarrow w; & z' \leftarrow w; & \\ w \leftarrow x, y; & w \leftarrow x, y', z'; & w \leftarrow y', z; \\ & w \leftarrow \texttt{not}\ w & \end{array} \right\}$$

Hence, P_Φ has as unique answer set $\{x, y, y', z, z', w\}$, corresponding to set x to true in Φ. So that, according to Theorem 1, Φ is true.

Example 6. Consider the 2QBF formula

$$\Phi = \exists x \exists y \forall z((x \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z)).$$

Therefore, the corresponding logic program is

$$P_\Phi = \left\{ \begin{array}{ccc} x \vee x'; & y \vee y'; & z \vee z'; \\ z \leftarrow w; & z' \leftarrow w; & \\ w \leftarrow x, z'; & w \leftarrow x', y, z'; & \\ & w \leftarrow \texttt{not}\ w & \end{array} \right\}$$

Hence, P_Φ is incoherent. Indeed, there are only two choices to infer w, $\{x, z'\}$ and $\{x', y, z'\}$. Therefore, an interpretation candidate must contain one of the two sets. In both cases, it cannot be an answer set. Indeed, we have three candidate models: $I_1 = \{x, y, z, z', w\}$; $I_2 = \{x, y', z, z', w\}$; and $I_3 = \{x', y, z, z', w\}$. However, none can be an answer set. Indeed, $\{x, y, z\}$ is a minimal model of P^{I_1}; $\{x, y', z\}$ is a minimal model of P^{I_2}; and $\{x', y, z\}$ is a minimal model of P^{I_3}. In conclusion, according to Theorem 1, Φ is false.

5 The ASPQ 2QBF Solver

We first present the ASPQ main algorithm, and then introduce a result that allowed us to optimize ASPQ performance.

Algorithm 1: ASPQ-main

Input : A 2-*QBF* formula Φ
Output: SAT or UNSAT

```
1  begin
2      T_bloqqer := 120s; T_clasp := 60s          // QBFEval settings;
3      Φ := BLOQQER (T_bloqqer, Φ);
4      if Φ = ⊤ then return SAT;                  // solved by BLOQQER
5      if Φ = ⊥ then return UNSAT;                // solved by BLOQQER
6      Π := QDimacs2ASP(Φ);                       // encode the logic program
7      res := CLASP (T_clasp, Π);                 // run CLASP for T_clasp seconds
8      if res = UNKNOWN then res := WASP (Π);     // run WASP if unsolved
9      if res = COHERENT then return UNSAT;
10     if res = INCOHERENT then return SAT;
11     return UNKNOWN;
```

5.1 Main Algorithm

The main algorithm implemented in ASPQ is reported as Algorithm 1. The input formula Φ is first simplified by the preprocessor BLOQQER [21], which replaces Φ by a (usually) smaller equisatisfiable formula. The simplification process can take significant time in case of huge formulas. Hence, tool is allowed to run for at most $T_{bloqqer}$ seconds. Φ is not modified if BLOQQER exceeds the allotted time. Note that BLOQQER might be able to simplify the formula up to solving it. In that case, it (conventionally) returns a tautology for SAT formulas or a contradiction for UNSAT. This case is exploited to terminate immediately the computation and return the result. Otherwise, Φ is encoded as a propositional ASP program Π as detailed in Section 4. The program Π is subsequently provided as input of the ASP solver CLASP [30], which is executed for T_{clasp} seconds. If CLASP is not able to find an answer set within the allotted time, then WASP [5] is executed without time limits. The reason for using two solvers is to find in the observation that CLASP and WASP employ different strategies for solving disjunctive programs (see [4, 30] for details), which may solve different sets of instances. The system varies its performance for different selections of $T_{bloqqer}$, T_{clasp} , and T_{wasp}, in the following we describe a strategy to optimize this choice.

5.2 Optimizing the evaluation

We now abstract the ASPQ algorithm considering a setting in which more than two solvers have to be run in sequence, to draw a result that holds also in a more general setting.

Let $\{S_1,\ldots,S_n\}$ be a set of n solvers, $\{inst_1,\ldots,inst_m\}$ a set of m instances, T_j the maximal execution time given to the solver S_j, and $t^j_{inst_k}$ the execution time to solve the instance $inst_k$ by solver S_j.

Given a topological ordering $S = (S_1,\ldots,S_n)$ to run the solvers in series, we denote by m_j the number of solved instances by solver S_j with respect to the topological or-

dering S, i.e., the number of instances solved by S_j and not by S_i, for each $i < j$. Let $\{inst_1^j, \ldots, inst_{m_j}^j\}$ be the set of instances solved by solver S_j with respect to S.

Note that S can be seen as a new solver. Assume that $T = T_1 + \ldots + T_n$ is the maximal execution time given to the solver S. Hence, the total number of solved instances by S is given by $M = m_1 + \ldots + m_n$. We denote by T_S the total execution time to solve the M instances by S. Hence, the following result holds.

Theorem 2. *Let $(S_1, \ldots, S_n)$ be a topological ordering to run the solvers in series. Then,*

$$T_S = \sum_{k=1}^{n} \sum_{h=1}^{m_k} t^k_{inst^k_h} + \sum_{j=2}^{n} \left(T_{j-1} \sum_{i=j}^{n} m_i \right).$$

Proof. We give a constructive proof. Clearly, to solve instances in $\{inst_1^1, \ldots, inst_{m_1}^1\}$, the solver S spends the same time of the solver S_1. Hence, $t^1_{inst^1_1} + \ldots + t^1_{inst^1_{m_1}}$. Then, to solve instances in $\{inst_1^2, \ldots, inst_{m_2}^2\}$, the solver S spends the time of the solver S_2 to which is added the maximal execution time given to the solver S_1, that is T_1. Hence, $(T_1 + t^2_{inst^2_1}) + \ldots + (T_1 + t^2_{inst^2_{m_2}}) = T_1 m_2 + (t^2_{inst^2_1} + \ldots + t^2_{inst^2_{m_2}})$. At the end, to solve instances in $\{inst_1^n, \ldots, inst_{m_n}^n\}$, the solver S spends the time of the solver S_n to which is added the maximal execution time given to the solvers S_1, S_2, ..., S_{n-1}, that is $T_1 + \ldots + T_{n-1}$. Hence, $(T_1 + \ldots + T_{n-1} + t^n_{inst^n_1}) + \ldots + (T_1 + \ldots + T_{n-1} + t^n_{inst^n_{m_n}}) = (T_1 + \ldots + T_{n-1}) m_n + (t^n_{inst^n_1} + \ldots + t^n_{inst^n_{m_n}})$.

We have applied the result stated in Theorem 2 to optimize the performance of ASPQ. In particular, we have selected the time thresholds to minimize T_S on a preliminary experiment, and then used the result to configure the solver. The details are provided in the next section.

6 Implementation and Performance Assessment

In this section we describe the implementation of ASPQ and then we narrate the history of optimization of our system. Finally we comment on the official results of the 2QBFEvaluation where ASPQ performed well in the 2QBF Track and won the Hard Instances Track.

6.1 Implementation

ASPQ was implemented as a bash script that calls its four main modules implemented in separate commands, namely: Bloqqer, QDimacs2ASP, WASP and CLASP. The bash script follows faithfully Algorithm 1. Bloqqer, WASP and CLASP are called as external processes, and were obtained by downloading and compiling the sources from the respective websites. The QDimacs2ASP component is a genuine Java implementation of the Eiter-Gottlob transformation presented in Section 4 that takes as input an instance of 2QBF in the standard QDIMACS format used in QBFEvaluations, and outputs a grouns ASP program in *lparse* format. Time measurement are implemented resorting

Solver	Solved Instances
depqbfvariants-QxQBH	266
CADET	263
predyndep	261
ASPQ v.3	**258**
CAQE-hqspre	253
ghostq-cegar	253
CAQE-bloqqer-qdo	249
CAQE-bloqqer	246
ASPQ v.2	**240**
PortfolioDepQBFGhostQRaReQSQute	240
RAReQS	239
Qute-random	231
Qute-opt500	230
Qute-default	229
CUED2	227
CUED3	227
heretiq-simple	208
heretiq-cube	206
ghostq-plain	176
ijtihad	161
iProver-HQSpre-Bloqqer	148
depqbfvariants-qdo	73

Table 1. QBFEval 2018 Official Results of the 2QBF Track.

to the standard *time* command, that interrupts the execution of a subprocess if the computation exceeds the maximum allotted time. The intermediate results, produced by Bloqqer and QDimacs2ASP are stored in temporary files, that are destroyed once the system ends the computation. $T_{bloqqer}$, T_{clasp} , and T_{wasp} can be specified by modifying the three corresponding variables declared in the main bash script. The resulting implementation can be easily updated with newer versions of the internal components, and one can easily modify the maximum time thresholds by just editing the script with a text editor.

6.2 Optimizing ASPQ

ASPQ entered the 2016 QBFEval in the 2QBF track as non competing system (it was submitted two days after the official solver submission deadline). We set $T_{bloqqer} = 60s$, so that preprocessing never occupies more than 10% of the allotted time (the timeout was set by the organizers to 600s); and we set $T_{clasp} = 120s$. This choice is motivated by the results of a preliminary experiment on the QBF instances used in ASP competitions, where CLASP solved the majority of instances within this time. After this experience, for QBFEval 2017 the timeout was increased to 900 seconds by the organizers, and we have updated the executables (with newer versions of Bloqqer, CLASP and WASP) and

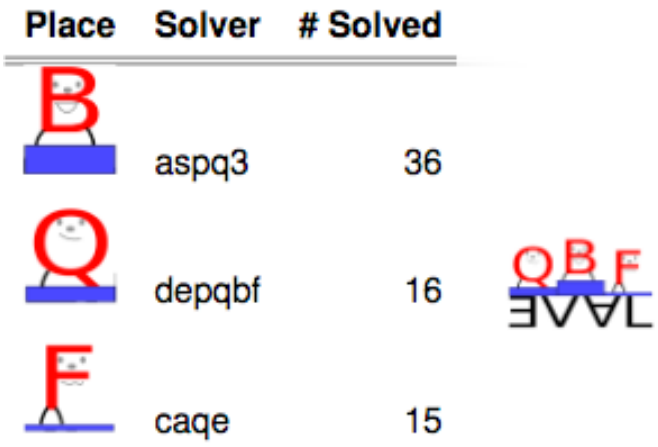

Fig. 1. A snaphot of the official results of the Hard Instances Track.

submitted the system with $T_{bloqqer} = 125s$ resulting in ASPQ v.2. The new setting was suggested by observing that Bloqqer processing is always beneficial for our system, and it terminates almost always in less than 125s in the QBFEval instances selected in 2016. An ugly bug limiting the execution to (the old timeout of) 600s influenced ASPQ v.2 performance in 2017. Thus, in 2018 we have fixed the bug and submitted both a fixed version of ASPQ (v.2) and a new one where we have optimized the thresholds by running once a python procedure that optimizes T_S of Theorem 2. We used as base the results obtained running each single component solver paired with Bloqqer, configured with different heuristics and choices of partial checks configurations (for the details on these parameters see [30, 5]), when run on the instances of QBFEval 2016 and 2017. The resulting optimized version uses $T_{bloqqer} = 125s$, $T_{clasp} = 850s$, where CLASP uses its default parameters, and WASP was configured with the `--forward-partialchecks` option.

6.3 Results in 2018 QBF Evaluation

ASPQ entered both the 2QBF track and the Hard Instances Track of QBFEval 2018.

2QBF Track. The results in the 2QBF track are summarized in Table 1, reporting the list of participants ordered by the number of solved instances (reported in a separate column). This elaboration has been done on the official results published in the QBFEval 2018 website. The optimized ASPQ v.3 reached the fourth place in the competition, solving only three, five and eight instances less than *predyndep*, *CADET*, and *depqbfvariants-QxQBH* occupying the first three places, respectively. The unoptimized (but fixed) ASPQ v.2 solved 18 instances less than ASPQ v.3 occupying the 9th place. All in all the optimization criterion presented in this paper allowed to improve significantly

our solver, and the resulting performance is basically aligned with the best options. It is worth noting that, all the other systems are based on native methods for solving QBF, whereas our implementation resorts to a translation, nonetheless it results to be very competitive.

Hard Instances Track. The results in the Hard Instances track are summarized in Figure 1, reporting a snapshot of the official slides presented at SAT 2018. Despite being able to solve only 2QBF instances (the track included also instances with more than two quantifiers) ASPQ v.3 could solve the majority of instances, more than twice as much as the solver in second position. Recall that, the Hard instances Track contained selected instances that no QBF solver could solve in previous edition from the CNF Track. This is an unexpected but impressive result that confirms the effectiveness of the optimized implementation of ASPQ and the efficiency of the ASP solvers employed by our system.

7 Conclusion

Answer Set Programming is an established logic-based programming paradigm that can be used to solve complex problems. ASP solving technology has steadily improved in the last few years, as demonstrated by ASP Competitions. Since the introduction of the first version of ASPQ, it was clear that ASP solving technology can be used fruitfully to solve 2QBF instances [13]. Indeed, ASPQ obtained fairly acceptable result in the 2QBF track in 2016 and 2017 editions of QBFEval [44]. The solver demonstrates that it is reasonable to exploit the capabilities of state of the art ASP solvers for solving 2QBF instances. Nonetheless, the first versions of ASPQ did not exploit completely the potential maximum performance of its architecture. In this paper, we describe how we have optimized ASPQ by tuning two main parameters of the algorithm. The resulting empowered solver significantly outperformed previous version and also won the Hard Instances Track in the 2018 QBFEval.

As far as future work is concerned, we are considering to tune our system by exploring the usage of the many different heuristics implemented by ASP solvers.

References

1. Adrian, W.T., Manna, M., Leone, N., Amendola, G., Adrian, M.: Entity set expansion from the web via ASP. In: ICLP-TC. OASICS, vol. 58, pp. 1:1–1:5 (2017)
2. Alviano, M., Amendola, G., Dodaro, C., Leone, N., Maratea, M., Ricca, F.: Evaluation of disjunctive programs in WASP. In: LPNMR. LNCS, vol. 11481, pp. 241–255. Springer (2019)
3. Alviano, M., Amendola, G., Peñaloza, R.: Minimal undefinedness for fuzzy answer sets. In: AAAI 2017. pp. 3694–3700 (2017)
4. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: LPNMR. LNCS, vol. 8148, pp. 54–66. Springer (2013)
5. Alviano, M., Dodaro, C., Leone, N., Ricca, F.: Advances in WASP. In: LPNMR. LNCS, vol. 9345, pp. 40–54. Springer (2015)
6. Amendola, G.: Dealing with incoherence in ASP: split semi-equilibrium semantics. In: DWAI@AI*IA. CEUR Workshop Proceedings, vol. 1334, pp. 23–32. CEUR-WS.org (2014)

7. Amendola, G.: Preliminary results on modeling interdependent scheduling games via answer set programming. In: RiCeRcA@AI*IA. CEUR Workshop Proceedings, vol. 2272 (2018)
8. Amendola, G.: Solving the stable roommates problem using incoherent answer set programs. In: RiCeRcA@AI*IA. CEUR Workshop Proceedings, vol. 2272 (2018)
9. Amendola, G., Dodaro, C., Faber, W., Leone, N., Ricca, F.: On the computation of paracoherent answer sets. In: AAAI'17. pp. 1034–1040 (2017)
10. Amendola, G., Dodaro, C., Faber, W., Pulina, L., Ricca, F.: Algorithm selection for paracoherent answer set computation. In: JELIA. LNCS, vol. 11468, pp. 479–489. Springer (2019)
11. Amendola, G., Dodaro, C., Faber, W., Ricca, F.: Externally supported models for efficient computation of paracoherent answer sets. In: AAAI. pp. 1720–1727. AAAI Press (2018)
12. Amendola, G., Dodaro, C., Leone, N., Ricca, F.: On the application of answer set programming to the conference paper assignment problem. In: AI*IA. LNCS, vol. 10037, pp. 164–178. Springer (2016)
13. Amendola, G., Dodaro, C., Ricca, F.: ASPQ: an asp-based 2qbf solver. In: QBF@SAT. CEUR Workshop Proceedings, vol. 1719, pp. 49–54 (2016)
14. Amendola, G., Eiter, T., Fink, M., Leone, N., Moura, J.: Semi-equilibrium models for paracoherent answer set programs. Artif. Intell. **234**, 219–271 (2016)
15. Amendola, G., Eiter, T., Leone, N.: Modular paracoherent answer sets. In: JELIA'14. pp. 457–471 (2014)
16. Amendola, G., Greco, G., Leone, N., Veltri, P.: Modeling and reasoning about NTU games via answer set programming. In: IJCAI. pp. 38–45 (2016)
17. Amendola, G., Ricca, F., Truszczynski, M.: Generating hard random boolean formulas and disjunctive logic programs. In: IJCAI. pp. 532–538. ijcai.org (2017)
18. Amendola, G., Ricca, F., Truszczynski, M.: A generator of hard 2qbf formulas and asp programs. In: KR. AAAI Press (2018)
19. Amendola, G., Ricca, F., Truszczynski, M.: Random models of very hard 2qbf and disjunctive programs: An overview. In: ICTCS. CEUR Workshop Proceedings (2018)
20. Balduccini, M., Gelfond, M., Watson, R., Nogueira, M.: The USA-Advisor: A Case Study in Answer Set Planning. In: LPNMR. LNCS, vol. 2173, pp. 439–442. Springer (2001)
21. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: CADE. LNCS, vol. 6803, pp. 101–115. Springer (2011)
22. Bravo, L., Bertossi, L.E.: Logic programs for consistently querying data integration systems. In: IJCAI. pp. 10–15 (2003)
23. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
24. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the Fifth Answer Set Programming Competition. Artif. Intell. **231**, 151–181 (2016)
25. Campeotto, F., Dovier, A., Pontelli, E.: A declarative concurrent system for protein structure prediction on GPU. J. Exp. Theor. Artif. Intell. **27**(5), 503–541 (2015)
26. Dodaro, C., Gasteiger, P., Leone, N., Musitsch, B., Ricca, F., Shchekotykhin, K.: Combining Answer Set Programming and domain heuristics for solving hard industrial problems (Application Paper). TPLP **16**(5-6), 653–669 (2016)
27. Dodaro, C., Leone, N., Nardi, B., Ricca, F.: Allotment problem in travel industry: A solution based on ASP. In: RR. LNCS, vol. 9209, pp. 77–92. Springer (2015)
28. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell. **15**(3-4), 289–323 (1995)
29. Gaggl, S.A., Manthey, N., Ronca, A., Wallner, J.P., Woltran, S.: Improved answer-set programming encodings for abstract argumentation. TPLP **15**(4-5), 434–448 (2015)
30. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp Series 3. In: LPNMR. LNCS, vol. 9345, pp. 368–383. Springer (2015)

31. Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F., Schaub, T.: Evaluation techniques and systems for answer set programming: a survey. In: IJCAI'18. pp. 5450–5456 (2018)
32. Gebser, M., Maratea, M., Ricca, F.: The Design of the Sixth Answer Set Programming Competition. In: LPNMR'15. pp. 531–544 (2015)
33. Gebser, M., Maratea, M., Ricca, F.: What's hot in the answer set programming competition. In: AAAI. pp. 4327–4329. AAAI Press (2016)
34. Gebser, M., Maratea, M., Ricca, F.: The design of the seventh answer set programming competition. In: Balduccini, M., Janhunen, T. (eds.) LPNMR. LNCS, vol. 10377, pp. 3–9. Springer (2017)
35. Gebser, M., Maratea, M., Ricca, F.: The sixth answer set programming competition. Journal of Artif. Intell. Res. **60**, 41–95 (2017)
36. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Comput. **9**(3/4), 365–386 (1991)
37. Giunchiglia, E., Leone, N., Maratea, M.: On the relation among answer set solvers. Ann. Math. Artif. Intell. **53**(1-4), 169–204 (2008)
38. Giunchiglia, E., Maratea, M.: On the Relation Between Answer Set and SAT Procedures (or, Between cmodels and smodels). In: ICLP. LNCS, vol. 3668, pp. 37–51 (2005)
39. Grasso, G., Leone, N., Manna, M., Ricca, F.: ASP at work: Spin-off and applications of the DLV system. In: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. LNCS, vol. 6565. Springer (2011)
40. Lierler, Y., Maratea, M., Ricca, F.: Systems, engineering environments, and competitions. AI Magazine **37**(3), 45–52 (2016)
41. Manna, M., Ricca, F., Terracina, G.: Taming primary key violations to query large inconsistent data via ASP. TPLP **15**(4-5), 696–710 (2015)
42. Maratea, M., Pulina, L., Ricca, F.: A multi-engine approach to answer-set programming. TPLP **14**(6), 841–868 (2014)
43. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in DLV: implementation, evaluation, and comparison to QBF solvers. J. Algorithms **63**(1-3), 70–89 (2008)
44. Marin, P., Narizzano, M., Pulina, L., Tacchella, A., Giunchiglia, E.: Twelve years of QBF evaluations: QSAT is pspace-hard and it shows. Fundam. Inform. **149**(1-2), 133–158 (2016)
45. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time: Preliminary report. In: STOC. pp. 1–9. ACM (1973)