AQME’10 System Description

Luca Pulina and Armando Tacchella

University of Genoa – DIST - Viale Causa 13 – 16145 Genoa (Italy)

POS 2010 - Edinburgh, July 10, 2010
What is a quantified Boolean formula?

Consider a Boolean formula, e.g.,

\[(x_1 \lor x_2) \land (\neg x_1 \lor x_2)\]

Adding existential “∃” and universal “∀” quantifiers, e.g.,

\[\forall x_1 \exists x_2 (x_1 \lor x_2) \land (\neg x_1 \lor x_2)\]

yields a quantified Boolean formula (QBF).
What is the meaning of a QBF?

A QBF, e.g.,

$$\forall x_1 \exists x_2 (x_1 \lor x_2) \land (\neg x_1 \lor x_2)$$

is true if and only if

for every value of $x_1$ there exist a value of $x_2$ such that

$(x_1 \lor x_2) \land (\neg x_1 \lor x_2)$ is propositionally satisfiable

Given any QBF $\psi$:

- if $\psi = \forall x \varphi$ then $\psi$ is true iff $\varphi|_{x=0} \land \varphi|_{x=1}$ is true
- if $\psi = \exists x \varphi$ then $\psi$ is true iff $\varphi|_{x=0} \lor \varphi|_{x=1}$ is true
QBFs as a logic “assembly” language

This approach works fine as long as QBF solvers are robust!

Luca Pulina (UNIGE)

AQME’10 System Description

POS’10 - Edinburgh
QBFs as a logic “assembly” language

This approach works fine as long as QBF solvers are robust!
Are state-of-the-art QBF solvers robust?
Are state-of-the-art QBF solvers robust?
Are state-of-the-art QBF solvers robust?

Luca Pulina (UNIGE)

AQME’10 System Description

POS’10 - Edinburgh
Are state-of-the-art QBF solvers robust?
Are state-of-the-art QBF solvers robust?
Are state-of-the-art QBF solvers robust?
Are state-of-the-art QBF solvers robust?

Luca Pulina (UNIGE)

AQME’10 System Description
Are state-of-the-art QBF solvers robust?

Luca Pulina (UNIGE)
Are state-of-the-art QBF solvers robust?

Luca Pulina (UNIGE)

AQME’10 System Description
Are state-of-the-art QBF solvers robust?

NOT REALLY...
Goal: a robust QBF solver
Goal: a robust QBF solver
Goal: a robust QBF solver
Outline

1. Engineering a robust QBF solver
2. Designing a self-adaptive multi-engine
3. Experiments
4. Conclusions & future work
Outline

1. Engineering a robust QBF solver
2. Designing a self-adaptive multi-engine
3. Experiments
4. Conclusions & future work
Two approaches to yield a robust solver

**Brute force**

Given $m$ QSAT instances and $n$ solvers (engines)

1. Run each engine on a separate machine.
2. Stop all the engines as soon as one solves the instance, or all the engines exhaust resources.
3. Continue with the next instance (if any).
Two approaches to yield a robust solver

Brute force

Given $m$ QSAT instances and $n$ solvers (engines)

1. Run each engine on a separate machine.
2. Stop all the engines as soon as one solves the instance, or all the engines exhaust resources.
3. Continue with the next instance (if any).

Intelligence

Understand which engine is best for which QBFs

- Fairly old idea: asset allocation in economics.
- Looking for dynamically adaptive policies.

Algorithm portfolios: SAT, SMT, QBFs (see related work).
Intelligence = Learning (to choose engines)

\[
\phi 
\rightarrow \quad F(\phi) 
\rightarrow \quad \begin{array}{c}
\cdots \\
E_1 \\
\cdots \\
E_2 \\
\cdots \\
E_n
\end{array}
\]
Intelligence = Learning (to choose engines)
Intelligence = Learning (to choose engines)

\[ \phi \rightarrow F(\phi) \rightarrow E_1 \rightarrow E_2 \rightarrow \ldots \rightarrow E_n \rightarrow \text{result} \]
Intelligence = Learning (to choose engines)
Intelligence = Learning (to choose engines)

\[
F(\varphi) = \frac{\varphi}{E_1 E_2 E_3 \ldots E_n}
\]

Dataset

\[
\begin{array}{c}
\varphi_1 E_2 \\
\varphi_2 E_4 \\
\vdots \\
\varphi_m E_1
\end{array}
\]
Intelligence = Learning (to choose engines)

\[ F(\varphi) \]

Dataset

\[ \varphi_1 E_2 \]
\[ \varphi_2 E_4 \]
\[ \ldots \]
\[ \varphi_m E_1 \]

Learning Algorithm
Intelligence = Learning (to choose engines)

\[ F(\varphi) \]

Dataset
\[
\varphi_1 E_2 \\
\varphi_2 E_4 \\
... \\
\varphi_m E_1
\]

Learning Algorithm

\[ E_1 \to \]
\[ E_2 \to \]
\[ E_n \to \]
Intelligence = Learning (to choose engines)

\[ \varphi \rightarrow F(\varphi) \rightarrow E_1, E_2, \ldots, E_n \]

Dataset

\[ \varphi_1 \ E_2 \\
\varphi_2 \ E_4 \\
\ldots \\
\varphi_m \ E_1 \]

Learning Algorithm

Choose a dataset
Intelligence = Learning (to choose engines)
Intelligence = Learning (to choose engines)
Choosing datasets

- **QBFLIB** ([www.qbflib.org](http://www.qbflib.org)), a repository of QBFs
  - More than **15K formulas** in a standard format.
  - Artificially generated, toy problems, realistic encodings, challenge problems, ...

- **QBF solvers competitions** ([www.qbfeval.org](http://www.qbfeval.org))
  - A **subset** of the formulas available in QBFLIB.
  - **Up-to-date** performance data about QBF solvers.

Luca Pulina (UNIGE)

AQME’10 System Description

POS’10 - Edinburgh
Choosing datasets

- QBFLIB ([www.qbflib.org](http://www.qbflib.org)), a repository of QBFs
  - More than **15K formulas** in a standard format.
  - Artificially generated, toy problems, realistic encodings, challenge problems, ...
- QBF solvers competitions ([www.qbfeval.org](http://www.qbfeval.org))
  - A **subset** of the formulas available in QBFLIB.
  - **Up-to-date** performance data about QBF solvers.

**Our choice in AQME’10**

The whole QBFEVAL’08 dataset (3326 fixed structured formulas).
Representing QBFs

**Basic features** regarding:
- **Clauses**: total number, number of Horn clauses, \ldots
- **Variables**: total number, existential and universal, \ldots
- **Quantifiers**: alternations, \ldots
- **Literals**: total number, average per clause, \ldots
- \ldots

**Combined features**: ratios/products between basic features.
Representing QBFs

**Basic features** regarding:
- **Clauses**: total number, number of Horn clauses, \ldots
- **Variables**: total number, existential and universal, \ldots
- **Quantifiers**: alternations, \ldots
- **Literals**: total number, average per clause, \ldots
- \ldots

**Combined features**: ratios/products between basic features.

**Our choice in AQME’10**

109 cheap syntactic features for each QBF.
Choice of inductive models

Our desiderata:

- Deal with numerical attributes (QBF features) and multiple class labels (engines).
- No assumptions of normality or (in)dependence among the features.
- No complex parameter tuning, thanks!

We also implemented multivariate logistic regression, decision trees, and decision rules.

We select 1-NN for its robustness w.r.t. the inductive models above (see [Pulina and Tacchella, CP-DP’08]).
Choice of inductive models

Our desiderata:

- Deal with numerical attributes (QBF features) and multiple class labels (engines).
- No assumptions of normality or (in)dependence among the features.
- No complex parameter tuning, thanks!

Our choice in AQME’10

Nearest-neighbour (1-NN)

- We also implemented multivariate logistic regression, decision trees, and decision rules.
- We select 1-NN for its robustness w.r.t. the inductive models above (see [Pulina and Tacchella, CP-DP’08]).
Choosing reasoning engines

- QBFEVALs reveal **major** differences between
  - Heuristic search based solvers.
  - Hybrid solvers mainly based on other techniques (e.g., resolution, skolemization), but possibly including search.

- Which solvers to choose as basic engines?
  - Only the best “search” and “hybrid”?
  - All state of the art solvers?
  - Something in between?
Choosing reasoning engines

- QBFEVALs reveal **major** differences between
  - Heuristic search based solvers.
  - Hybrid solvers mainly based on other techniques (e.g., resolution, skolemization), but possibly including search.

- **Which** solvers to choose as basic engines?
  - Only the best “search” and “hybrid”?
  - All state of the art solvers?
  - Something in between?

**Our selection in AQME’10**

- **Search-based:** QuBE3.1, SSOLVE-UT, and 2CLSQ.
- **Hybrid:** QUANTOR2.11, and SKIZZO-0.9-STD.
Choosing reasoning engines

- QBFEVALs reveal **major** differences between
  - Heuristic search based solvers.
  - Hybrid solvers mainly based on other techniques (e.g., resolution, skolemization), but possibly including search.

- Which solvers to choose as basic engines?
  - Only the best “search” and “hybrid”?
  - All state of the art solvers?
  - Something in between?

---

**Our selection in AQME’10**

- Search-based: QuBE3.1, SSOLVE-UT, and 2CLSQ.
- Hybrid: QUANTOR2.11, and SKIZZO-0.9-STD.

---

“Vintage engines” offer us a baseline to compare the current progress in the development of QBF solvers.
Outline

1. Engineering a robust QBF solver
2. Designing a self-adaptive multi-engine
3. Experiments
4. Conclusions & future work
Designing a self-adaptive multi-engine

How could AQME’10 learn by its incorrect predictions?
Designing a self-adaptive multi-engine

How could AQME’10 learn by its incorrect predictions?

Retraining: adaptation schema applied to engine selection policies whenever they fail to give good predictions.
Retraining

\[ F(\varphi) \]

\[ \varphi \rightarrow F(\varphi) \rightarrow E_1 \rightarrow E_2 \rightarrow \cdots \rightarrow E_n \]

\[ \varphi_1 E_2 \]
\[ \varphi_2 E_4 \]
\[ \ldots \]
\[ \varphi_m E_1 \]

Dataset

Learning Algorithm
Retraining

\[ \varphi \xrightarrow{F(\varphi)} [\varphi_1 \ E_2 \ \varphi_2 \ E_4 \ \ldots \ \varphi_m \ E_1] \]

Dataset

Learning Algorithm

\[ E_1 \quad E_2 \quad E_3 \quad E_n \]
Retraining

\[
F(\varphi) \quad E_1 \\
\vdots \\
E_n
\]

Dataset

\[
\varphi_1 E_2 \\
\varphi_2 E_4 \\
\vdots \\
\varphi_m E_1 \\
\varphi_{m+1} E_1
\]

Learning Algorithm
Retraining

\[ F(\varphi) \]

Dataset

\[ \varphi_1 \quad E_2 \\
\varphi_2 \quad E_4 \\
\ldots \\
\varphi_m \quad E_1 \]

Learning Algorithm

\[ \varphi \]

\[ E_1 \quad E_2 \quad \ldots \quad E_n \]
Retraining

\[ F'(\varphi) \]

Dataset

\[ \varphi_1 E_2 \]
\[ \varphi_2 E_4 \]
\[ \ldots \]
\[ \varphi_m E_1 \]
\[ \varphi_{m+1} E_1 \]

Learning Algorithm

\[ E_1 \]
\[ E_2 \]
\[ \ldots \]
\[ E_n \]
Retraining policies

Critical points for AQME’10 performances:
- How much CPU time is granted to each engine.
- Which engine is called for retraining.
Retraining policies

Critical points for AQME’10 performances:
- How much CPU time is granted to each engine.
- Which engine is called for retraining.

Policies in AQME’10

- **Granted CPU time**: “Trust the Predicted Engine”
  - A fixed amount of CPU time is granted to the predicted solver.
  - If it fails, another engine is called (following the engine selection policy), with a granted amount of CPU time until the solver solves the input formula.
  - If the formula is not solved, the originally predicted engine is fired, with the time limit assigned to the remaining time.

- **Engine selection**: The engine to fire is selected according to the QBFEVAL’06 ranking.
AQME’10 architecture
AQME’10 architecture

System I/O:
* Input formula
* Classifier type and inductive model
* Policies
AQME’10 architecture

System I/O:
- Input formula
- Classifier type and inductive model
- Policies

Syntactic features extraction

INPUT

OUTPUT

INTERFACE

FEE

WEKA

POLICY MANAGER

MANAGER

ENGINE MANAGER

2clsQ

quantor

QuBE

sKizzo

sSolve
AQME’10 architecture

INPUT

INTERFACE

OUTPUT

System I/O:
* Input formula
* Classifier type and inductive model
* Policies

Syntactic features extraction

* Inductive models implementation
* Engine prediction

FEE

WEKA

POLICY MANAGER

MANAGER

ENGINE MANAGER

2clsQ

quantor

QuBE

sKizzo

sSolve
AQME’10 architecture

System I/O:
* Input formula
* Classifier type and inductive model
* Policies

Syntactic features extraction

* Inductive models implementation
* Engine prediction

Retraining policies
AQME’10 architecture
AQME’10 architecture

System I/O:
* Input formula
* Classifier type and inductive model
* Policies

Syntactic features extraction

* Inductive models implementation
* Engine prediction

Retraining policies

Interaction with the engines

Modules coordinator
Outline

1. Engineering a robust QBF solver
2. Designing a self-adaptive multi-engine
3. Experiments
4. Conclusions & future work
<table>
<thead>
<tr>
<th>Solver</th>
<th>MAIN</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>RND</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#</td>
<td>Time</td>
<td>#</td>
<td>Time</td>
<td>#</td>
<td>Time</td>
<td>#</td>
<td>Time</td>
<td>#</td>
<td>Time</td>
</tr>
<tr>
<td>AIGSOLVE</td>
<td>329</td>
<td>22786.60</td>
<td>NA</td>
<td>NA</td>
<td>37</td>
<td>1140.01</td>
<td>NA</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AQME’10</td>
<td>434</td>
<td>33346.60</td>
<td>128</td>
<td>2323.11</td>
<td>11</td>
<td>30132.40</td>
<td>407</td>
<td>20078.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEPQBF</td>
<td>370</td>
<td>21515.30</td>
<td>24</td>
<td>690.42</td>
<td>4</td>
<td>41448.00</td>
<td>342</td>
<td>12895.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEPQBF-PRE</td>
<td>356</td>
<td>18995.90</td>
<td>51</td>
<td>877.02</td>
<td>4</td>
<td>33371.90</td>
<td>343</td>
<td>9438.62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NENOFEX</td>
<td>225</td>
<td>13786.90</td>
<td>50</td>
<td>3545.65</td>
<td>3</td>
<td>30194.20</td>
<td>149</td>
<td>34502.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QMAIGA</td>
<td>361</td>
<td>43058.10</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUANTOR3.1</td>
<td>205</td>
<td>6711.37</td>
<td>48</td>
<td>3689.30</td>
<td>5</td>
<td>57960.90</td>
<td>134</td>
<td>2830.97</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRUQS’10</td>
<td>240</td>
<td>32839.70</td>
<td>132</td>
<td>1399.30</td>
<td>5</td>
<td>26257.30</td>
<td>117</td>
<td>15480.40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Best in the sense of numbers of problems solved within the CPU time limit
- Good performance in 2QBF and SH tracks.
Looking inside AQME’10

Luca Pulina (UNIGE)

<table>
<thead>
<tr>
<th>System</th>
<th>MAIN</th>
<th>2QBF</th>
<th>SH</th>
<th>RND</th>
</tr>
</thead>
<tbody>
<tr>
<td>2clsQ</td>
<td>28</td>
<td>–</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>QUANTOR2.11</td>
<td>106</td>
<td>24</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>QuBE3.1</td>
<td>145</td>
<td>11</td>
<td>2</td>
<td>146</td>
</tr>
<tr>
<td>sKIZZO</td>
<td>116</td>
<td>80</td>
<td>6</td>
<td>63</td>
</tr>
<tr>
<td>SSOLVE-UT</td>
<td>39</td>
<td>13</td>
<td>1</td>
<td>198</td>
</tr>
</tbody>
</table>

Retrainings 22 3 – 15
Looking inside AQME’10

Self-adaptation based on the characteristics of the test set.
Outline

1. Engineering a robust QBF solver
2. Designing a self-adaptive multi-engine
3. Experiments
4. Conclusions & future work
Conclusions

- A multiengine solver is a robust alternative to current state-of-the-art QBF solvers.
- Good performance achieved also using engines date back 2006.
- Retraining algorithm increases the performances in terms of number of solved formula.
- Performances “limited” by the State-of-the-art solver, i.e., the ideal solver that always fares the best time among all the considered solvers.
Future work

- Mechanism for the automatic integration of new engines.
- Implementation of new learning algorithms (see, e.g., D. Stern et al., AAAI 2010).
- Integration between different algorithms, not black-box engines (see, e.g., Pulina and Tacchella, FROCOS 2009).
Thank you!