Formal Methods and AI: Yet Another Entanglement

Kuldeep S. Meel

School of Computing, National University of Singapore

@Waterloo ML + Security + Verification Workshop

Aug 2019

• **Turing 1950:** ... one might have a complete system of logical inference "built in."
• **Turing 1950**: ... one might have a complete system of logical inference "built in."

• **Newell, Shaw, and Simon 1955**: Logic Theorist’

• **McCarthy, 1958**: Programming with Common Sense

• **Hayes-Roth, Waterman, and Lenat, 1982**: Building Expert Systems
• **Turing 1950:** ... one might have a complete system of logical inference "built in."

• **Newel, Shaw, and Simon 1955:** Logic Theorist’

• **McCarthy, 1958:** Programming with Common Sense

• **Hayes-Roth, Waterman, and Lenat, 1982:** Building Expert Systems

• **Hinton, 2019:** The final nail in the coffin of Symbolic AI....
• **Turing 1950**: ... one might have a complete system of logical inference "built in."

• **Newell, Shaw, and Simon 1955**: Logic Theorist’

• **McCarthy, 1958**: Programming with Common Sense

• **Hayes-Roth, Waterman, and Lenat, 1982**: Building Expert Systems

• **Hinton, 2019**: The final nail in the coffin of Symbolic AI....

• **Ephesians 5:14** Arise, you sleeper! Rise up from your coffin .....
• Core public agencies, such as those responsible for criminal justice, healthcare, welfare, and education (e.g., “high stakes” domains) should no longer use “black box” AI and algorithmic systems (AI Now Institute, 2018)


• Machine learning leads mathematicians to unsolvable problem (Nature, 2019)
• Part I Formal Methods for AI
  – Designing Interpretable Rules (Joint work with Bishwamittra Ghosh and Dmitri Malioutov; CP-18, AIES-19)
• **Part I Formal Methods for AI**
  – Designing Interpretable Rules (Joint work with Bishwamittra Ghosh and Dmitri Malioutov; CP-18, AIES-19)

• **Part II Functional Verification of Probabilistic Systems**
  – Quantitative Verification of Neural Networks (Joint work with Teodora Baluta, Shiqi Shen, Shweta Shine, and Prateek Saxena; CCS-19)
  – Quantitative Verification for Explanations (Joint work with Nina Narodytska, Aditya Shrotri, Alexey Ignatiev, and Joao Marques Silva; SAT-19)
  – Distribution Testing (Joint work with Sourav Chakraborty; AAAI-19)
• **Part I Formal Methods for AI**
  - Designing Interpretable Rules (Joint work with Bishwamittra Ghosh and Dmitri Malioutov; CP-18, AIES-19)

• **Part II Functional Verification of Probabilistic Systems**
  - Quantitative Verification of Neural Networks (Joint work with Teodora Baluta, Shiqi Shen, Shweta Shine, and Prateek Saxena; CCS-19)
  - Quantitative Verification for Explanations (Joint work with Nina Narodytska, Aditya Shrotri, Alexey Ignatiev, and Joao Marques Silva; SAT-19)
  - Distribution Testing (Joint work with Sourav Chakraborty; AAAI-19)

• **Part III AI for Formal Methods**
  - Data-Driven Design of SAT Solvers (Joint work with Mate Soos and Raghav Kulkarni; SAT-19)
The Need for Interpretable Models

• Core public agencies, such as those responsible for criminal justice, healthcare, welfare, and education (e.g., “high stakes” domains) should no longer use “black box” AI and algorithmic systems (AI Now Institute, 2018)
The Need for Interpretable Models

- Core public agencies, such as those responsible for criminal justice, healthcare, welfare, and education (e.g., “high stakes” domains) should no longer use “black box” AI and algorithmic systems (AI Now Institute, 2018)

- Medical and education domains see usage of techniques such as classification rules, decision rules, and decision lists.
The Need for Interpretable Models

- Core public agencies, such as those responsible for criminal justice, healthcare, welfare, and education (e.g., “high stakes” domains) should no longer use “black box” AI and algorithmic systems (AI Now Institute, 2018)

- Medical and education domains see usage of techniques such as classification rules, decision rules, and decision lists.

- Long history of interpretable classification models from data such as decision trees, decision lists, checklists etc with tools such as C4.5, CN2, RIPPER, SLIPPER

- Computational Intractability led prior work, mostly rooted in late 1980s and 1990s, to focus on greedy approaches
Objective  Learn rules that are accurate and interpretable.
Our Approach

Objective  Learn rules that are accurate and interpretable.

Approach  • The problem of rule learning is inherently an optimization problem
Our Approach

Objective  Learn rules that are accurate and interpretable.

Approach

• The problem of rule learning is inherently an optimization problem

• Can we take advantage of SAT revolution, in particular progress on MaxSAT solvers?
Binary Classification

- Features: \( \mathbf{x} = \{x^1, x^2, \cdots, x^m\} \)
- Input: Set of training samples \( \{\mathbf{X}_i, y_i\} \)
  - each vector \( \mathbf{X}_i \in \mathcal{X} \) contains valuation of the features for sample \( i \),
  - \( y_i \in \{0, 1\} \) is the binary label for sample \( i \)
- Output: Classifier \( \mathcal{R} \), i.e. \( y = \mathcal{R}(\mathbf{x}) \)
- Our focus: classifiers that can be represented as CNF Formulas \( \mathcal{R} := C_1 \land C_2 \land \cdots \land C_k \).
- Size of classifiers: \( |\mathcal{R}| = \sum_i |C_i| \)
Constraint Learning vs Machine Learning

Input  Set of training samples $\{X_i, y_i\}$

Output  Classifier $\mathcal{R}$

- Constraint Learning/Programming by Examples:

$$\min_{\mathcal{R}} |\mathcal{R}| \quad \text{such that} \quad \mathcal{R}(X_i) = y_i, \quad \forall i$$
Input  Set of training samples $\{X_i, y_i\}$

Output Classifier $\mathcal{R}$

- **Constraint Learning/Programming by Examples:**

  $$\min_{\mathcal{R}} |\mathcal{R}| \quad \text{such that} \quad \mathcal{R}(X_i) = y_i, \quad \forall i$$

- **Machine Learning:**

  $$\min_{\mathcal{R}} |\mathcal{R}| + \lambda |\mathcal{E}_{\mathcal{R}}| \quad \text{such that} \quad \mathcal{R}(X_i) = y_i, \quad \forall i \notin \mathcal{E}_{\mathcal{R}}$$
Step 1  Discretization of Features
Step 2  Transformation to MaxSAT Query
Step 3  Invoke a MaxSAT Solver and extract $R$ from MaxSAT solution
Input  Features: $\mathbf{x} = \{x^1, x^2, \cdots x^m\}$ ; Training Data: $\{\mathbf{X}_i, y_i\}$ over $m$ features

Output  $\mathcal{R}$ of $k$ clauses

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{b_1^1, b_1^2, \cdots b_k^m \cdots b_k^m\}$, such that $\mathcal{R}_i = (b_i^1 x^1 \lor b_i^2 x^2 \cdots \lor b_i^m x^m)$

- For every sample $i$, we have noise variable $\eta_i$ to encode sample $i$ should be considered as noise or not.
Encoding to MaxSAT

Key Ideas

• $k \times m$ binary coefficients, denoted by $\{b_1^1, b_2^2, \ldots b_m^m \}$, such that $R_i = (b_1^i x^1 \lor b_2^i x^2 \ldots \lor b_m^i x^m)$

• $R(x \mapsto X_i) = \bigwedge_{i=1}^{k} R_i(x \mapsto X_i)$: Output of substituting valuation of feature vectors of $i$th sample
Encoding to MaxSAT

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{ b_1^1, b_2^1, \ldots b_m^1, \ldots b_1^m, \ldots b_m^m \}$, such that $R_i = (b_1^1 x^1 \lor b_2^1 x^2 \ldots \lor b_m^1 x^m)$

- $R(x \leftrightarrow X_i) = \bigwedge_{i=1}^k R_i(x \leftrightarrow X_i)$: Output of substituting valuation of feature vectors of $i$th sample

- For every sample $i$, we have noise variable $\eta_i$ to encode whether sample $i$ should be considered as noise or not.

- $D_i := (\neg \eta_i \rightarrow (y_i \leftrightarrow R(x \leftrightarrow X_i)))$; $W(D_i) = \top$
  
  If $\eta_i$ is False, $y_i$ is equivalent to prediction of the Rule
Encoding to MaxSAT

Key Ideas

• $k \times m$ binary coefficients, denoted by $\{b^1_1, b^2_1, \ldots b^m_1 \ldots b^m_k\}$, such that $\mathcal{R}_i = (b^1_i x^1 \lor b^2_i x^2 \ldots \lor b^m_i x^m)$

• $R(x \leftrightarrow X_i) = \bigwedge_{i=1}^k R_i(x \leftrightarrow X_i)$: Output of substituting valuation of feature vectors of $i$th sample

• For every sample $i$, we have noise variable $\eta_i$ to encode whether sample $i$ should be considered as noise or not.

• $D_i := (\neg \eta_i \rightarrow (y_i \leftrightarrow R(x \leftrightarrow X_i))); W(D_i) = \top$
  If $\eta_i$ is False, $y_i$ is equivalent to prediction of the Rule

• $V^j_i := (\neg b^j_i); \hspace{1cm} W(V^j_i) = 1$
  We want as few $b^j_i$ to be true as possible
Encoding to MaxSAT

Key Ideas

- \( k \times m \) binary coefficients, denoted by \( \{ b_{11}^1, b_{12}^2, \ldots b_{1m}^m \ldots b_{km}^m \} \), such that \( R_i = (b_{11}^1x^1 \lor b_{12}^2x^2 \ldots \lor b_{1m}^mx^m) \)

- \( R(x \mapsto X_i) = \bigwedge_{i=1}^{k} R_i(x \mapsto X_i) \): Output of substituting valuation of feature vectors of \( i \)th sample

- For every sample \( i \), we have noise variable \( \eta_i \) to encode whether sample \( i \) should be considered as noise or not.

- \( D_i := (\neg \eta_i \rightarrow (y_i \leftrightarrow R(x \mapsto X_i))) \); \( W(D_i) = \top \)
  If \( \eta_i \) is False, \( y_i \) is equivalent to prediction of the Rule

- \( V_i^j := (\neg b_i^j) \); \( W(V_i^j) = 1 \)
  We want as few \( b_i^j \) to be true as possible

- \( N_i := (\neg \eta_i) \); \( W(N_i) = \lambda \)
  We want as few \( \eta_i \) to be true as possible
Encoding to MaxSAT

1. \( R = \bigwedge_{i=1}^{k} R_i(x \mapsto X_i) \): Output of substituting valuation of feature vectors of \( i \)th sample

2. \( D_i := (\neg \eta_i \rightarrow (y_i \leftrightarrow R(x \mapsto X_i))) ; W(D_i) = \top \)

3. \( V_i^j := (\neg b_i^j) ; W(V_i^j) = 1 \)
   
   We want as few \( b_i^j \) to be true as possible

4. \( N_i := (\neg \eta_i) ; W(N_i) = \lambda \)
   
   We want as few \( \eta_i \) to be true as possible

Construction

Let \( Q^k = \bigwedge_i D_i \land \bigwedge_i N_i \land \bigwedge_{i,j} V_i^j \)

\( \sigma^* = \text{MaxSAT} (Q^k, W) \), then \( x^i \in \mathcal{R}_i \) iff \( \sigma^*(b_i^j) = 1 \).

Remember, \( \mathcal{R}_i = (b_i^1 x^1 \lor b_i^2 x^2 \ldots \lor b_i^m x^m) \)
Theorem (Provable trade off of accuracy vs interpretability of rules)

Let $\mathcal{R}_1 \leftarrow \text{MLIC}(\mathbf{X}, \mathbf{y}, k, \lambda_1)$ and $\mathcal{R}_2 \leftarrow \text{MLIC}(\mathbf{X}, \mathbf{y}, k, \lambda_2)$, if $\lambda_2 > \lambda_1$ then $|\mathcal{R}_1| \leq |\mathcal{R}_2|$ and $|\mathcal{E}_{\mathcal{R}_1}| \geq |\mathcal{E}_{\mathcal{R}_2}|$. 
### Accuracy and training time of different classifiers

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Features</th>
<th>RF</th>
<th>SVC</th>
<th>RIPPER</th>
<th>MLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIMA</td>
<td>768</td>
<td>134</td>
<td>76.62</td>
<td>75.32</td>
<td>75.32</td>
<td>73.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(1.99)</td>
<td>(0.37)</td>
<td>(2.58 )</td>
<td>(0.74 )</td>
</tr>
<tr>
<td>Tom’s HW</td>
<td>28179</td>
<td>844</td>
<td>97.11</td>
<td>96.83</td>
<td>96.75</td>
<td>96.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(27.11)</td>
<td>(354.15)</td>
<td>(37.81)</td>
<td>(23.67 )</td>
</tr>
<tr>
<td>Adult</td>
<td>32561</td>
<td>262</td>
<td>84.31</td>
<td>84.39</td>
<td>83.72</td>
<td>80.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(36.64)</td>
<td>(918.26)</td>
<td>(37.66)</td>
<td>(25.07 )</td>
</tr>
<tr>
<td>Credit-default</td>
<td>30000</td>
<td>334</td>
<td>80.87</td>
<td>80.69</td>
<td>80.72</td>
<td>79.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(37.72)</td>
<td>(847.93)</td>
<td>(20.37)</td>
<td>(32.58 )</td>
</tr>
<tr>
<td>Twitter</td>
<td>49999</td>
<td>1050</td>
<td>95.16</td>
<td>Timeout</td>
<td>95.56</td>
<td>94.69</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(67.83)</td>
<td></td>
<td>(98.21)</td>
<td>(59.67 )</td>
</tr>
</tbody>
</table>

**Table:** For every cell in the last seven columns the top value represents the test accuracy (%) on unseen data and the bottom value surrounded by parenthesis represents the average training time (seconds).
## Size of interpretable rules of different classifiers

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RIPPER</th>
<th>MLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDBC</td>
<td>7.6</td>
<td>2</td>
</tr>
<tr>
<td>Adult</td>
<td>107.55</td>
<td>28</td>
</tr>
<tr>
<td>PIMA</td>
<td>8.25</td>
<td>4</td>
</tr>
<tr>
<td>Tom’s HW</td>
<td>30.33</td>
<td>4</td>
</tr>
<tr>
<td>Twitter</td>
<td>21.6</td>
<td>6</td>
</tr>
<tr>
<td>Credit</td>
<td>14.25</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table:** Size of the rule of interpretable classifiers.

Rule for WDBC Dataset:
- Tumor is diagnosed as malignant if the standard area of tumor $> 38.43$ OR
- largest perimeter of tumor $> 115.9$ OR
- largest number of concave points of tumor $> 0.1508$
Key Takaways

- A MaxSAT-based framework, MLIC, that **provably** trades off accuracy vs interpretability of rules
- The prototype implementation is capable of finding optimal (or high quality near-optimal) classification rules from large data sets with very small rules.

**Code:** https://github.com/meelgroup/mlic
  pip install rulelearning
Verification of AI
• Given a model $M$
  - $M$: A neural network to label images

• Specification $\phi$
  - $\phi$: Label stop sign as $\text{STOP}$
Imprecise Systems and The Classical Approach

- Given a model $M$
  - $M$: A neural network to label images
- Specification $\varphi$
  - $\varphi$: Label stop sign as STOP
- Check whether there exists an execution of $M$ that violates $\varphi$
  - Given a neural network, find if there exists a minor change to a image of stop sign such that $M$ incorrectly classifies?
• Given a model $M$
  - $M$: A neural network to label images

• Specification $\varphi$
  - $\varphi$: Label stop sign as STOP

• Check whether there exists an execution of $M$ that violates $\varphi$
  - Given a neural network, find if there exists a minor change to a image of stop sign such that $M$ incorrectly classifies?

• Yes but so what?
• The classical verification concerned with finding whether there exists one execution

• The Approach:
  – Represent $M$ and $\varphi$ as logical formulas and use constraint solver (SAT solvers)
  – Given a formula, a SAT solver checks if there exists a solution
  – $F = (x_1 \lor x_2)$, the SAT solver will return YES
• The classical verification concerned with finding whether there exists one execution

• The Approach:
  – Represent $M$ and $\varphi$ as logical formulas and use constraint solver (SAT solvers)
  – Given a formula, a SAT solver checks if there exists a solution
  – $F = (x_1 \lor x_2)$, the SAT solver will return YES

• We now care whether there exist too many?
  – Given a formula, we need to count (possibly subject to distributions)

• Challenges: Scalability, encodings, algorithms, quality of approximations
• The classical verification concerned with finding whether there exists one execution

• The Approach:
  – Represent $M$ and $\varphi$ as logical formulas and use constraint solver (SAT solvers)
  – Given a formula, a SAT solver checks if there exists a solution
    – $F = (x_1 \lor x_2)$, the SAT solver will return YES

• We now care whether there exist too many?
  – Given a formula, we need to count (possibly subject to distributions)

• Challenges: Scalability, encodings, algorithms, quality of approximations

• Underlying Core Problem: Distribution Testing
  Counting can be viewed as computing area/expectation.
• Samplers form the core of the state of the art probabilistic reasoning techniques. They generate distributions
• Samplers form the core of the state of the art probabilistic reasoning techniques. They generate distributions.

• Usual technique for designing samplers is based on the Markov Chain Monte Carlo (MCMC) methods.
• Samplers form the core of the state of the art probabilistic reasoning techniques. They generate distributions

• Usual technique for designing samplers is based on the Markov Chain Monte Carlo (MCMC) methods.

• Since mixing times/runtime of the underlying Markov Chains are often exponential, several heuristics have been proposed for years.

• Statistical tests are employed to argue for quality of the output distributions. – Chi square test, KL divergence, Gini coefficient, ...

• But such statistical tests are often performed on a very small number of samples for which no theoretical guarantees exist.
• Samplers form the core of the state of the art probabilistic reasoning techniques. They generate distributions

• Usual technique for designing samplers is based on the Markov Chain Monte Carlo (MCMC) methods.

• Since mixing times/runtime of the underlying Markov Chains are often exponential, several heuristics have been proposed for years.

• Often statistical tests are employed to argue for quality of the output distributions.
  – Chi square test, KL divergence, Ginni coefficient, ...
• Samplers form the core of the state of the art probabilistic reasoning techniques. They generate distributions.

• Usual technique for designing samplers is based on the Markov Chain Monte Carlo (MCMC) methods.

• Since mixing times/runtime of the underlying Markov Chains are often exponential, several heuristics have been proposed for years.

• Often statistical tests are employed to argue for quality of the output distributions.
  – Chi square test, KL divergence, Ginni coefficient, ...

• But such statistical tests are often performed on a very small number of samples for which no theoretical guarantees exist.
A Uniform-Sampler, \( A \), is a randomized algorithm that outputs a random element of the set \( S \), such that, for any \( y \in S \)

\[
\Pr[y \text{ is output }] = \frac{1}{|S|},
\]
Definition

A Uniform-Sampler, \( \mathcal{A} \), is a randomized algorithm that outputs a random element of the set \( S \), such that, for any \( y \in S \)

\[
\Pr[y \text{ is output }] = \frac{1}{|S|},
\]

- Uniform sampling has wide range of applications in automated bug discovery, pattern mining, and so on.
Uniform Sampler for Discrete Sets

**Definition**

A **Uniform-Sampler, \( \mathcal{A} \), is a randomized algorithm that outputs a random element of the set \( S \), such that, for any \( y \in S \)**

\[
\Pr[y \text{ is output } ] = \frac{1}{|S|},
\]

- Uniform sampling has wide range of applications in automated bug discovery, pattern mining, and so on.
- Formal Methods/Software Engineer: Randomized Testing
  - Implicit representation of a set \( S \): Set of all solutions of \( \varphi \).
  - Given a CNF formula \( \varphi \), output a random solution of \( \varphi \).
- Several samplers available off the shelf: tradeoff between guarantees and runtime; (“random.randint(1,100)”)
What does Complexity Theory Tell Us

- "far" means total variation distance or the $\ell_1$ distance.

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]

\[ |S| \]
What does Complexity Theory Tell Us

- “far” means total variation distance or the $\ell_1$ distance.

Theorem (Batu-Fortnow-Rubinfeld-Smith-White (JACM 2013))

Testing whether a distribution is $\epsilon$-close to uniform has query complexity $\Theta(\sqrt{|S|}/\epsilon^2)$. [Paninski (Trans. Inf. Theory 2008)]

- If the output of a sampler is represented by 3 doubles, then $S > 2^{100}/49$. 

- If fewer than $\sqrt{S}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.
Definition (Conditional Sampling)

Given a distribution $A$ on $S$ one can

- Specify a set $T \subseteq S$,
- Draw samples according to the distribution $A|_T$, that is, $A$ under the condition that the samples belong to $T$. 
Beyond Black Box Testing

Definition (Conditional Sampling)

Given a distribution \( A \) on \( S \) one can

- Specify a set \( T \subseteq S \),
- Draw samples according to the distribution \( A|_T \), that is, \( A \) under the condition that the samples belong to \( T \).

Conditional sampling is at least as powerful as drawing normal samples. But how more powerful is it?
An algorithm for testing uniformity using conditional sampling:

1. Draw two elements \( x \) and \( y \) uniformly at random from the domain. Let \( T = \{ x, y \} \).

2. In the case of the "far" distribution, with probability \( \frac{1}{2} \), one of the two elements will have probability 0, and the other probability non-zero.

3. Note \( p_{\| T \|} = \sqrt{2} \) is a constant.

4. Now a constant number of conditional samples drawn from \( A|T \) is enough to identify that it is not uniform.
An algorithm for testing uniformity using conditional sampling:

1. Draw two elements $x$ and $y$ uniformly at random from the domain. Let $T = \{x, y\}$.

2. In the case of the “far” distribution, with probability $1/2$, one of the two elements will have probability 0, and the other probability non-zero.

3. Note $\sqrt{|T|} = \sqrt{2}$ is a constant.

4. Now a constant number of conditional samples drawn from $A|_T$ is enough to identify that it is not uniform.
What about other distributions?

Draw two elements $\sigma_1$ and $\sigma_2$ uniformly at random from the domain. Let $T = \{\sigma_1, \sigma_2\}$.

In the case of the “far” distribution, with probability almost 1, both the two elements will have the same probability $\epsilon$.

Probability that we will be able to distinguish the far distribution from the uniform distribution is very low.
What about other distributions?

Previous algorithm fails in this case:

1. Draw two elements $\sigma_1$ and $\sigma_2$ uniformly at random from the domain. Let $T = \{\sigma_1, \sigma_2\}$.
2. In the case of the “far” distribution, with probability almost 1, both the two elements will have probability same, namely $\epsilon$.
3. Probability that we will be able to distinguish the far distribution from the uniform distribution is very low.
Testing Uniformity Using Conditional Sampling

Draw $\sigma_1$ uniformly at random from the domain and draw $\sigma_2$ according to the distribution $A$. Let $T = \{\sigma_1, \sigma_2\}$.

In the case of the "far" distribution, with constant probability, $\sigma_1$ will have "low" probability and $\sigma_2$ will have "high" probability.

We will be able to distinguish the far distribution from the uniform distribution using constant number of conditional samples from $A|_T$.

The constant depend on the farness parameter.
1. Draw $\sigma_1$ uniformly at random from the domain and draw $\sigma_2$ according to the distribution $\mathcal{A}$. Let $T = \{\sigma_1, \sigma_2\}$.

2. In the case of the “far” distribution, with constant probability, $\sigma_1$ will have “low” probability and $\sigma_2$ will have “high” probability.

3. We will be able to distinguish the far distribution from the uniform distribution using constant number of conditional samples from $\mathcal{A}|_T$.

4. The constant depend on the farness parameter.
Input: A sampler under test \( \mathcal{A} \), a reference uniform sampler \( \mathcal{U} \), a tolerance parameter \( \varepsilon > 0 \), an intolerance parameter \( \eta > \varepsilon \), a guarantee parameter \( \delta \)

Output: ACCEPT or REJECT with the following guarantees:

- if the generator \( \mathcal{A} \) is an \( \varepsilon \)-additive almost-uniform generator then Barbarik ACCEPTS with probability at least \( 1 - \delta \).
- if \( \mathcal{A} \) is \( \eta \)-far from a uniform generator and if non-adversarial sampler assumption holds then Barbarik REJECTS with probability at least \( 1 - \delta \).
Sample complexity

Theorem

Given $\epsilon$, $\eta$ and $\delta$, Barbarik need at most $K = \tilde{O}\left(\frac{1}{(\eta - \epsilon)^4}\right)$ samples for any input formula $\varphi$, where the tilde hides a poly logarithmic factor of $1/\delta$ and $1/(\eta - \epsilon)$.

- $\epsilon = 0.6$, $\eta = 0.9$, $\delta = 0.1$
- Maximum number of required samples $K = 1.72 \times 10^6$
- Independent of the number of variables
- To Accept, we need $K$ samples but rejection can be achieved with lesser number of samples.
Experimental Setup

- Three state of the art (almost-)uniform samplers
  - UniGen2: Theoretical Guarantees of almost-uniformity
  - SearchTreeSampler: Very weak guarantees
  - QuickSampler: No Guarantees

- Recent study that proposed Quicksampler perform unsound statistical tests and claimed that all the three samplers are indistinguishable

Code: https://github.com/meelgroup/barbarik
### Results-I

<table>
<thead>
<tr>
<th>Instances</th>
<th>Size</th>
<th>UniGen2</th>
<th>SearchTreeSampler</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Output</td>
<td>#Samples</td>
</tr>
<tr>
<td>71</td>
<td>$1.14 \times 2^{59}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case49</td>
<td>$1.00 \times 2^{61}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case50</td>
<td>$1.00 \times 2^{62}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>scenarios_aig_insertion1</td>
<td>$1.06 \times 2^{65}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>scenarios_aig_insertion2</td>
<td>$1.06 \times 2^{65}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>36</td>
<td>$1.00 \times 2^{72}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>30</td>
<td>$1.73 \times 2^{72}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>110</td>
<td>$1.09 \times 2^{76}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>scenarios_tree_insert</td>
<td>$1.32 \times 2^{76}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>107</td>
<td>$1.52 \times 2^{76}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case211</td>
<td>$1.00 \times 2^{80}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case210</td>
<td>$1.00 \times 2^{80}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case212</td>
<td>$1.00 \times 2^{88}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case209</td>
<td>$1.00 \times 2^{88}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>54</td>
<td>$1.15 \times 2^{90}$</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>Instances</td>
<td>Size</td>
<td>UniGen2</td>
<td>QuickSampler</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
<td>---------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td>Output</td>
<td>#Samples</td>
<td>Output</td>
</tr>
<tr>
<td>71</td>
<td>A</td>
<td>1729750</td>
<td>R</td>
</tr>
<tr>
<td>blasted_case49</td>
<td>1.14 x 2^{59}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case50</td>
<td>1.00 x 2^{61}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>scenarios_aig_insertion1</td>
<td>1.06 x 2^{65}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>scenarios_aig_insertion2</td>
<td>1.06 x 2^{65}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>36</td>
<td>1.00 x 2^{72}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>30</td>
<td>1.73 x 2^{72}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>110</td>
<td>1.09 x 2^{76}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>scenarios_tree_insert_insert</td>
<td>1.32 x 2^{76}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>107</td>
<td>1.52 x 2^{76}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case211</td>
<td>1.00 x 2^{80}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case210</td>
<td>1.00 x 2^{80}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case212</td>
<td>1.00 x 2^{88}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>blasted_case209</td>
<td>1.00 x 2^{88}</td>
<td>A</td>
<td>1729750</td>
</tr>
<tr>
<td>54</td>
<td>1.15 x 2^{90}</td>
<td>A</td>
<td>1729750</td>
</tr>
</tbody>
</table>
Key Takeaways

• We need new methodological approaches to verification of AI systems
• Sampling is a crucial component of the state of the art probabilistic reasoning systems
• Barbarik: Promise of strong theoretical guarantees with scalability to large instances
Key Takeaways

• We need new methodological approaches to verification of AI systems
• Sampling is a crucial component of the state of the art probabilistic reasoning systems
• Barbarik: Promise of strong theoretical guarantees with scalability to large instances
• Extend beyond uniform discrete distributions
AI for Formal Methods
• SAT is still NP-complete yet solvers tend to solve problems involving millions of variables
• The solvers of today are very complex and we understand very little on how to further improve the SAT solvers
• SAT is still NP-complete yet solvers tend to solve problems involving millions of variables
• The solvers of today are very complex and we understand very little on how to further improve the SAT solvers
• 50,000 hours of CPU time plus tens of human hours tuning parameters in CryptoMiniSAT for 2018 competition (won third place in SAT 2018 competition)
Data-Driven Design of SAT solver

- SAT solvers as composition of prediction engines (Liang et al)
  - Branching
  - Clause learning
  - Memory management
  - Restarts

Prior Work
- Machine learning to optimize behavior of prediction engines
- Focused on using runtime or proxy for runtime

CrystalBall
Whether it is possible to develop a framework to provide white-box access to execution of SAT solver, which can aid the developer to understand and synthesize algorithmic heuristics for modern SAT solvers?

What CrystalBall is not about:
- Replacing experts

We envision a expert in loop framework
As a first step, we have focused on memory management: learnt clause deletion. All models are wrong. Some are useful.
Data-Driven Design of SAT solver

- SAT solvers as composition of prediction engines (Liang et al)
  - Branching
  - Clause learning
  - Memory management
  - Restarts

- Prior Work
  - Machine learning to optimize behavior of prediction engines
  - Focused on using runtime or proxy for runtime
Data-Driven Design of SAT solver

- SAT solvers as composition of prediction engines (Liang et al)
  - Branching
  - Clause learning
  - Memory management
  - Restarts

- Prior Work
  - Machine learning to optimize behavior of prediction engines
  - Focused on using runtime or proxy for runtime

Whether it is possible to develop a framework to provide white-box access to execution of SAT solver, which can aid the developer to understand and synthesize algorithmic heuristics for modern SAT solvers?
Data-Driven Design of SAT solver

• SAT solvers as composition of prediction engines (Liang et al)
  – Branching
  – Clause learning
  – Memory management
  – Restarts

• Prior Work
  – Machine learning to optimize behavior of prediction engines
  – Focused on using runtime or proxy for runtime

• CrystalBall  Whether it is possible to develop a framework to provide white-box access to execution of SAT solver, which can aid the developer to understand and synthesize algorithmic heuristics for modern SAT solvers?
Data-Driven Design of SAT solver

- SAT solvers as composition of prediction engines (Liang et al)
  - Branching
  - Clause learning
  - Memory management
  - Restarts

- Prior Work
  - Machine learning to optimize behavior of prediction engines
  - Focused on using runtime or proxy for runtime

- CrystalBall  Whether it is possible to develop a framework to provide white-box access to execution of SAT solver, which can aid the developer to understand and synthesize algorithmic heuristics for modern SAT solvers?

- What CrystalBall is not about?
  - Replacing experts
Data-Driven Design of SAT solver

- SAT solvers as composition of prediction engines (Liang et al)
  - Branching
  - Clause learning
  - Memory management
  - Restarts

- Prior Work
  - Machine learning to optimize behavior of prediction engines
  - Focused on using runtime or proxy for runtime

- CrystalBall
  Whether it is possible to develop a framework to provide white-box access to execution of SAT solver, which can aid the developer to understand and synthesize algorithmic heuristics for modern SAT solvers?

- What CrystalBall is not about?
  - Replacing experts

- We envision a expert in loop framework
Data-Driven Design of SAT solver

- SAT solvers as composition of prediction engines (Liang et al)
  - Branching
  - Clause learning
  - Memory management
  - Restarts

- Prior Work
  - Machine learning to optimize behavior of prediction engines
  - Focused on using runtime or proxy for runtime

- CrystalBall  Whether it is possible to develop a framework to provide white-box access to execution of SAT solver, which can aid the developer to understand and synthesize algorithmic heuristics for modern SAT solvers?

- What CrystalBall is not about?
  - Replacing experts

- We envision a expert in loop framework

- As a first step, we have focused on memory management: learnt clause deletion. All models are wrong. Some are useful.
The curse of learnt clauses

- Learnt clauses are very useful
- But they consume memory and can slowdown other components of SAT solving
- Not practical to keep all the learnt clauses
- Delete larger clauses
- Delete less used clauses
- Delete clauses based on Literal block distance

[E.g. MSS96a, MSS99]

[E.g. GN02, ES03]

[AS09]
• For inference, we want to do supervised learning
• For every clause, we need values of different features and a label
• The inference engine should learn the model to predict the label
For inference, we want to do supervised learning
For every clause, we need values of different features and a label
The inference engine should learn the model to predict the label

Components of CrystalBall

Feature Engineering
• For inference, we want to do supervised learning
• For every clause, we need values of different features and a label
• The inference engine should learn the model to predict the label

Components of CrystalBall

1. Feature Engineering
2. Labeling
For inference, we want to do supervised learning
For every clause, we need values of different features and a label
The inference engine should learn the model to predict the label

Components of CrystalBall
1. Feature Engineering
2. Labeling
3. Data collection
Architecture

- For inference, we want to do supervised learning
- For every clause, we need values of different features and a label
- The inference engine should learn the model to predict the label

Components of CrystalBall

1. Feature Engineering
2. Labeling
3. Data collection
4. Inference Engine
Part 1: Feature Engineering

- Global features: property of the CNF formula at the time of genesis
- Contextual features: computed at the time of generation of the clause and relate to the generated clause, e.g. LBD score
- Restart features: correspond to statistics (average and variance) on the size and LBD of clauses, branch depth, trail depth during the current and previous restart.
- Performance features: performance parameters of the learnt clause such as the number of times the solver played part of a 1stUIP conflict clause generation

Total # of features: 212
Part2: Labeling
Useful Clauses

- We focus on UNSAT formulas
  - SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it *discovers* satisfiable assignments.
Part2: Labeling
Useful Clauses

• We focus on UNSAT formulas
  – SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it discovers satisfiable assignments.

• A clause is useful if it is involved in the final UNSAT proof.
Part 2: Labeling

Useful Clauses

- We focus on UNSAT formulas
  - SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it discovers satisfiable assignments.

- A clause is useful if it is involved in the final UNSAT proof.

- For some cases, more than > 50% clauses are useful
Part 2: Labeling

Useful Clauses

- We focus on UNSAT formulas
  - SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it discovers satisfiable assignments.
- A clause is useful if it is involved in the final UNSAT proof.
- For some cases, more than > 50% clauses are useful.
- But we can only keep less than 5% of clauses in memory.
Part 2: Labeling

Useful Clauses

- We focus on UNSAT formulas
  - SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it discovers satisfiable assignments.

- A clause is useful if it is involved in the final UNSAT proof.

- For some cases, more than > 50% clauses are useful.

- But we can only keep less than 5% of clauses in memory
  
  Need to consider temporal aspect of usefulness

- We associate a counter with execution of SAT solver: incremented with every conflict

- expiry (C): The value of counter when C was last used in the UNSAT proof
Part2: Labeling
Useful Clauses

- We focus on UNSAT formulas
  - SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it discovers satisfiable assignments.

- A clause is useful if it is involved in the final UNSAT proof.

- For some cases, more than $>50\%$ clauses are useful.

- But we can only keep less than $5\%$ of clauses in memory.

Need to consider temporal aspect of usefulness.

- We associate a counter with execution of SAT solver: incremented with every conflict.

- expiry ($C$): The value of counter when $C$ was last used in the UNSAT proof.

- **Useful** A clause is useful in future at $t$ if expiry($C$) $> t$. 
Part 2: Labeling

Useful Clauses

- We focus on UNSAT formulas
  - SAT solver can be viewed as trying to find the proof of unsatisfiability. When the formula is satisfiable, it discovers satisfiable assignments.
- A clause is useful if it is involved in the final UNSAT proof.
- For some cases, more than > 50% clauses are useful
- But we can only keep less than 5% of clauses in memory

Need to consider temporal aspect of usefulness

- We associate a counter with execution of SAT solver: incremented with every conflict
- expiry (C): The value of counter when C was last used in the UNSAT proof
- **Useful** A clause is useful in future at t if expiry(C) > t.
- Can we predict every 10K conflicts for a clause C if C will be useful in future?
• Just record the trace of the solver?
• Works well for toy benchmarks.
• Just record the trace of the solver?
• Works well for toy benchmarks.
• We are interested in handling competition benchmarks – large benchmarks
• Need to reconstruct *approximate*/*inexact* trace
Part 3: Data Collection

- Just record the trace of the solver?
- Works well for toy benchmarks.
- We are interested in handling competition benchmarks – large benchmarks
- Need to reconstruct *approximate/inexact* trace `drat-trim`.
• Forward pass
  – The solver keeps track of features of each clause and dumps all the learnt clauses after we reach UNSAT.
  – genesis(C): The value of counter when C was learnt
  – expiry (C): The value of counter when C was last used in the UNSAT proof
Part 3: Data Collection

- **Forward pass**
  - The solver keeps track of features of each clause and dumps all the learnt clauses after we reach UNSAT.
  - genesis\( (C) \): The value of counter when \( C \) was learnt
  - expiry \( (C) \): The value of counter when \( C \) was last used in the UNSAT proof

- **Backward pass**
  - DRAT-trim is used to reconstruct the proof while satisfying the constraint while satisfying the constraint expiry\( (C) > \) genesis\( (C) \).
  - Key modifications
    - For every clause we attach a unique ID to every clause as the same clause can be learned twice, so it is important to track each clause
    - We supply genesis of a clause so that a clause is not used in the proof before its genesis
Looking back over the years

Visualizing SAT solving

June 16, 2012
Uncategorized
SAT, visualisation

Gathering information

It took me about 1-1.5 months to write the code to gather all information I wanted. It took a lot of time to correctly structure and decide about how to store/summarise the information gathered. There is much more gathered than shown on the webpage, but more about that below.

Selecting what to display, and how

This may sound trivial. Some would simply say: just display all information! But what we really want is not just plain information: what good is it to print 100'000 numbers on a screen? The data has to be displayed in a meaningful and visually understandable way.

Getting to the current layout took a lot of time and many-many discussions with all my friends and colleagues. I am eternally grateful for their input — it's hard to know how good a layout is until someone sees it for the first time, and completely misunderstands it. Then you know you have to change it: until then, it was trial and error.

What to display is a bit more complex. There is a lot of data gathered, but what is interesting? Naturally, I couldn't display everything, so I had to select. But selection may become a form of misrepresentation: if some important data isn't displayed, the system is effectively lying. So, I tried to add as much as possible that still made sense. This lead to a very large table of graphs, but I think it's still under-

Machine Learning and SAT

August 9, 2013
Development, Research, SAT, machine learning

I have lately been digging myself into a deep hole with machine learning. While doing that it occurred to me that the SAT community has essentially been trying to imitate some of ML in a somewhat way. Let me explain.

CryptoMiniSat and clause cleaning strategy selection

When CryptoMiniSat won the SAT Race of 2010, it was in large part because I realized that glucose at the time was essentially unable to solve cryptographic problems. I devised a system where I could detect which problems were cryptographic. It checked the activity stability of variables and if they were more stable than a threshold, it was decided that the problem was cryptographic. Cryptographic problems were then solved using a geometric restart strategy with clause activities for learnt database cleaning. Without this hack, it would have been impossible to win the competition.

It is clear that there could have been a number of ways to detect that a problem is cryptographic with out using such an elaborate scheme. However, that would have demanded a mixture of more features to decide. The scheme only used the average and the standard deviation.

Lingeling and clause cleaning strategy selection

The decision made by lingeling about whether to use glue or activities to clean learnt clauses is somewhat similar to my approach above. It calculates the average and the standard deviation of the learnt clauses' glue and then makes a decision. Looking at the code, the option actavgmax/stdmin/stdmax gives the cutoffs and the function lghneedacts calculates the values and decides. This has been in lingeling since 2011 [lingeling-SBT].

Probably a much better decision could be made if more data was taken into account (e.g. activities) but as a human, it’s simply hard to make a decision based on more than 2-3 pieces of data.

Enter machine learning

It is clear that the above schemes were basically trying to extract some feature from the SAT solver and then decide what features (glues/activities) to use to clear the learnt clause database. It is also clear that both have been extremely effective, it’s by no luck that they have been inside successful SAT solvers.

The question is, can we do better? I think yes. First of all, we don’t need to cut the problem into two steps. Instead, we can integrate the features extracted from the solver (variable activities, clause glue distribution, etc) and the features from the clause (glue, activities, etc.) and make a decision whether to keep the clause or not. This means we would make keep/throwaway decisions on individual clause-
Part 4: Inference Engine

What models to use

- Two constraints
  - Our 212 features are mixed or heterogeneous.
  - No straightforward manner to normalize all of our features.

- The SVM and other linear models require carefully normalized homogeneous features.

- We chose the random forest as the classifier for our inference engine
Experimental Setup

- All the UNSAT instances from SAT 2014-17.
- Each instance was ran with timeout of 20,000 seconds and CrystalBall finished execution for 260 instances.
- The number of learnt clauses for different problems varied from few hundreds to millions.
- We sampled 2000 data points from each benchmarks to ensure fair representation for each benchmark.
- We discarded 50 benchmarks that had less than 2000 data points.
- In total, we had 422K data points.
- Standard split into 70% training and 30% training.
<table>
<thead>
<tr>
<th></th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Throw</td>
</tr>
<tr>
<td>Ground truth</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>0.11</td>
</tr>
</tbody>
</table>

**Table:** Confusion matrix
The power of interpretable classifiers

Feature Ranking

1. **rdb0**: Number of times that the conflict took part in a 1UIP conflict generation since its creation.

2. **rdb0**: last touched diff: Number of conflicts ago that the clause was used during a 1UIP conflict clause generation.

3. **rdb0**: activity rel: Activity of the clause, relative to the activity of all other learned clauses at the point of time when the decision to keep or throw away the clause is made.

4. **rdb0**: sum uip1 used: Number of times that the clause took part in a 1UIP conflict generation since its creation.

5. **rdb1**: used for uip creation: Same as **rdb0** used for uip creation but instead of the current round, it is data from the previous round (i.e. 10k conflicts earlier).

LBD is not a top-5 feature.
The power of interpretable classifiers

Feature Ranking

1. `rdb0.used_for_uip_creation`: Number of times that the conflict took part in a 1UIP conflict generation since its creation.
2. `rdb0.last_touched_diff`: Number of conflicts ago that the clause was used during a 1UIP conflict clause generation.
3. `rdb0.activity_rel`: Activity of the clause, relative to the activity of all other learned clauses at the point of time when the decision to keep or throw away the clause is made.
4. `rdb0.sum_uip1_used`: Number of times that the clause took part in a 1UIP conflict generation since its creation.
5. `rdb1.used_for_uip_creation`: Same as `rdb0.used_for_uip_creation` but instead of the current round, it is data from the previous round (i.e. 10k conflicts earlier)

LBD is not a top-5 feature
Comparison with state of the art Solver

- 934 instances from SAT Competitions 2014-17 with a timeout of 5000 seconds.
Comparison with state of the art Solver

- 934 instances from SAT Competitions 2014-17 with a timeout of 5000 seconds.
- \textit{Maple\_LCM\_Dist} : 591 instances (2017 winning solver)
Comparison with state of the art Solver

- 934 instances from SAT Competitions 2014-17 with a timeout of 5000 seconds.
- \textit{Maple\_LCM\_Dist} : 591 instances (2017 winning solver)
- CryptoMiniSAT plus learned classifier: 612 instances
  - Solved SAT: 271
  - Solved UNSAT: 341
- The ratio of SAT to UNSAT instances is almost same to \textit{Maple\_LCM\_Dist}.
Comparison with state of the art Solver

- 934 instances from SAT Competitions 2014-17 with a timeout of 5000 seconds.
- *Maple\_LCM\_Dist*: 591 instances (2017 winning solver)
- CryptoMiniSAT plus learned classifier: 612 instances
  - Solved SAT: 271
  - Solved UNSAT: 341
- The ratio of SAT to UNSAT instances is almost same to *Maple\_LCM\_Dist*.
- Training was only on UNSAT instances – shows generalizability
• Design new features. For derivative features, you do not even need to rerun the solver
• Learn complex models
• Extend CrystalBall for branching, clause learning, and restarts
• An application area for interpretable machine learning
• Democratize the design of solvers; allows researchers without deep expertise in software engineering of SAT solvers to test out their ideas

Code: https://meelgroup.github.io/crystalball/
Conclusion

- Designing Interpretable Rules (Formal Methods for AI)
  - Joint work with Bishwamittra Ghosh and Dmitri Malioutov; CP-18, AIES-19

- Functional Verification of Probabilistic Systems (Beyond Traditional Verification Methodologies)
  - Quantitative Verification of Neural Networks (Joint work with Teodora Baluta, Shiqi Shen, Shweta Shinde, and Prateek Saxena; CCS-19)
  - Quantitative Verification for Explanations (Joint work with Nina Narodytska, Aditya Shrotri, Alexey Ignatiev, and Joao Marques Silva; SAT-19)
  - Distribution Testing (Joint work with Sourav Chakraborty; AAAI-19)

- Data-Driven Design of SAT Solvers (AI for Formal Methods)
  - Joint work with Mate Soos and Raghav Kulkarni; SAT-19

Thank You