Formal Methods and AI: Yet Another Entanglement

Kuldeep S. Meel

School of Computing, National University of Singapore

@Waterloo ML + Security + Verification Workshop

Aug 2019

Join us in our mission: Positions for post-docs, long-term research assistants, and PhD students. Visit meelgroup.github.io for details.

The Formal Methods and AI Through Decades

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

The Formal Methods and AI Through Decades

- 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

- 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....

The Formal Methods and AI Through Decades

- 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....
- Ephesians 5:14 Arise, you sleeper! Rise up from your coffin .....

The Views from the Real World

- 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)
- How Do You Govern Machines That Can Learn? (New York Times, 2019)
- Machine learning leads mathematicians to unsolvable problem (Nature, 2019)

Formal Methods and AI

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

Formal Methods and AI

- 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)

Formal Methods and AI

- 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

Our Approach

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 $\mathcal{R} := C_1 \wedge C_2 \wedge \cdots \wedge C_k$.
- Size of classifiers: $|\mathcal{R}| = \Sigma_i |C_i|$

Constraint Learning vs Machine Learning

Input Set of training samples$\{\mathbf{X}_i, y_i\}$

Output Classifier $\mathcal{R}$

- Constraint Learning/Programming by Examples:

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

Constraint Learning vs Machine Learning

Input Set of training samples $\{\mathbf{X}_i, y_i\}$

Output Classifier $\mathcal{R}$

- Constraint Learning/Programming by Examples:

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

- Machine Learning:

$$\min_{\mathcal{R}} |\mathcal{R}| + \lambda|\mathcal{E}_{\mathcal{R}}| \quad \text{such that } \mathcal{R}(\mathbf{X}_i) = y_i, \quad \forall i \notin \mathcal{E}_{\mathcal{R}}$$

MLIC

Step 1 Discretization of Features

Step 2 Transformation to MaxSAT Query

Step 3 Invoke a MaxSAT Solver and extract $\mathcal{R}$ from MaxSAT solution

Encoding to MaxSAT

Input Features: $\mathbf{x} = \{x^1, x^2, \cdots x^m\}$; Training Data: $\{\mathbf{X}_i, y_i\}$ over m featues

Output $\mathcal{R}$ of k clauses

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{b_1^1, b_1^2, \cdots b_1^m \cdots b_k^m\}$, such that $\mathcal{R}_i = (b_i^1 x^1 \vee b_i^2 x^2 \ldots \vee b_i^m x^m)$
- For every sample i, we have noise variable η_i to encode sample i should be considered as noise or not.

Encoding to MaxSAT

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{b_1^1, b_1^2, \cdots b_1^m \cdots b_k^m\}$, such that $\mathcal{R}_i = (b_i^1 x^1 \vee b_i^2 x^2 \ldots \vee b_i^m x^m)$
- $R(\mathbf{x} \mapsto X_i) = \bigwedge_{l=1}^{k} R_l(\mathbf{x} \mapsto X_i)$: Output of substituting valuation of feature vectors of ith sample

Encoding to MaxSAT

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{b_1^1, b_1^2, \cdots b_1^m \cdots b_k^m\}$, such that $\mathcal{R}_i = (b_i^1 x^1 \vee b_i^2 x^2 \ldots \vee b_i^m x^m)$
- $R(\mathbf{x} \mapsto X_i) = \bigwedge_{l=1}^{k} R_l(\mathbf{x} \mapsto X_i)$: Output of substituting valuation of feature vectors of ith sample
- For every sample i, we have noise variable η_i to encode whether sample i should be considered as noise or not.
- $D_i := (\neg\eta_i \rightarrow (y_i \leftrightarrow R(\mathbf{x} \mapsto X_i))); W(D_i) = \top$
  If η_i is False, y_i is equivalent to prediction of the Rule

Encoding to MaxSAT

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{b_1^1, b_1^2, \cdots b_1^m \cdots b_k^m\}$, such that $\mathcal{R}_i = (b_i^1 x^1 \vee b_i^2 x^2 \ldots \vee b_i^m x^m)$
- $R(\mathbf{x} \mapsto X_i) = \bigwedge_{l=1}^{k} R_l(\mathbf{x} \mapsto X_i)$: Output of substituting valuation of feature vectors of ith sample
- For every sample i, we have noise variable η_i to encode whether sample i should be considered as noise or not.
- $D_i := (\neg\eta_i \rightarrow (y_i \leftrightarrow R(\mathbf{x} \mapsto X_i))); W(D_i) = \top$
  If η_i is False, y_i is equivalent to prediction of the Rule
- $V_i^j := (\neg b_i^j); \qquad W\left(V_i^j\right) = 1$
  We want as few b_i^j to be true as possible

Encoding to MaxSAT

Key Ideas

- $k \times m$ binary coefficients, denoted by $\{b_1^1, b_1^2, \cdots b_1^m \cdots b_k^m\}$, such that $\mathcal{R}_i = (b_i^1 x^1 \vee b_i^2 x^2 \ldots \vee b_i^m x^m)$
- $R(\mathbf{x} \mapsto X_i) = \bigwedge_{l=1}^{k} R_l(\mathbf{x} \mapsto X_i)$: Output of substituting valuation of feature vectors of ith sample
- For every sample i, we have noise variable η_i to encode whether sample i should be considered as noise or not.
- $D_i := (\neg\eta_i \rightarrow (y_i \leftrightarrow R(\mathbf{x} \mapsto X_i))); W(D_i) = \top$
  If η_i is False, y_i is equivalent to prediction of the Rule
- $V_i^j := (\neg b_i^j); \qquad W\left(V_i^j\right) = 1$
  We want as few b_i^j to be true as possible
- $N_i := (\neg\eta_i); \qquad W(N_i) = \lambda$
  We want as few η_i to be true as possible

Encoding to MaxSAT

1. $R = \bigwedge_{l=1}^{k} R_l(\mathbf{x} \mapsto X_i)$: Output of substituting valuation of feature vectors of ith sample
2. $D_i := (\neg\eta_i \rightarrow (y_i \leftrightarrow R(\mathbf{x} \mapsto X_i))); W(D_i) = \top$
3. $V_i^j := (\neg b_i^j); \qquad W\left(V_i^j\right) = 1$

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

   We want as few η_i to be true as possible

Construction

Let $Q^k = \bigwedge_i D_i \wedge \bigwedge_i N_i \wedge \bigwedge_{i,j} V_i^j$

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

Remember, $\mathcal{R}_i = (b_i^1 x^1 \vee b_i^2 x^2 \ldots \vee b_i^m x^m)$

Provable Guarantees

Theorem (Provable trade off of accuracy vs interpretability of rules)

Let $\mathcal{R}_1 \leftarrow MLIC(\mathbf{X}, \mathbf{y}, k, \lambda_1)$ *and* $\mathcal{R}_2 \leftarrow MLIC(\mathbf{X}, \mathbf{y}, k, \lambda_2)$*, 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

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

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

Dataset	RIPPER	*MLIC*
WDBC	7.6	**2**
Adult	107.55	**28**
PIMA	8.25	**4**
Tom's HW	30.33	**4**
Twitter	21.6	**6**
Credit	14.25	**3**

Table: Size of the rule of interpretable classifiers.

Rule for WDBC Dataset:
Tumor is diagnosed as malignant if
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

(a) (b) (c) (d) (e)

- Given a model M
  - M: A neural network to label images
- Specification φ
  - φ: Label stop sign as **STOP**

(a) (b) (c) (d) (e)

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

Imprecise Systems and The Classical Approach

- Given a model M
  - M: A neural network to label images
- Specification φ
  - φ: Label stop sign as **STOP**
- Check whether there exists an execution of M that violates φ
  - 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?

From Qualification to Quantification

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

From Qualification to Quantification

- The classical verification concerned with finding whether there exists one execution
- The Approach:
  - Represent M and φ as logical formulas and use constraint solver (SAT solvers)
  - Given a formula, a SAT solver checks if there exists a solution
  - $F = (x_1 \vee x_2)$, 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

From Qualification to Quantification

- The classical verification concerned with finding whether there exists one execution
- The Approach:
  - Represent M and φ as logical formulas and use constraint solver (SAT solvers)
  - Given a formula, a SAT solver checks if there exists a solution
  - $F = (x_1 \vee x_2)$, 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.

Distribution Testing: Samplers

- Samplers form the core of the state of the art probabilistic reasoning techniques. They generate distributions

Distribution Testing: Samplers

- 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.

Distribution Testing: Samplers

- 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.

Distribution Testing: Samplers

- 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, ...

Distribution Testing: Samplers

- 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.

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 $y \in S$

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

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 $y \in S$

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

- 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 φ.
  - Given a CNF formula φ, output a random solution of φ.
- Several samplers available off the shelf: tradeoff between guarantees and runtime; ("random.randint(1,100)")

- “far” means total variation distance or the ℓ_1 distance.

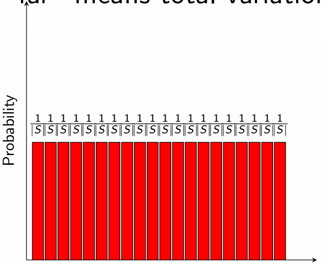


Figure: $\mathcal{U}$: Reference Uniform Sampler

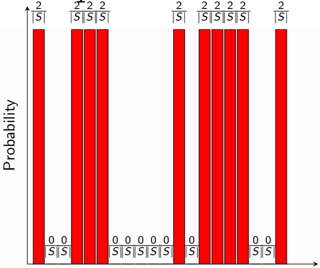


Figure: $\mathcal{A}$: 1/2-far from uniform Sampler

What does Complexity Theory Tell Us

- "far" means total variation distance or the ℓ_1 distance.

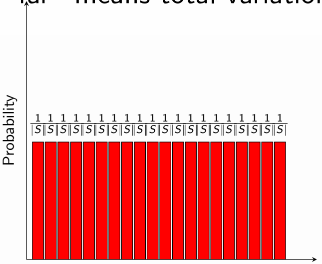


Figure: $\mathcal{U}$: Reference Uniform Sampler

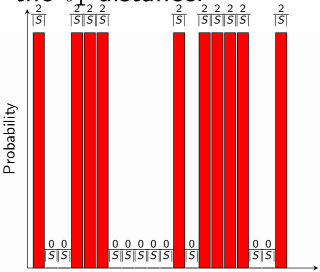


Figure: $\mathcal{A}$: 1/2-far from uniform Sampler

- If $< \sqrt{S}/100$ samples are drawn then with high probability you see only distinct samples from either distribution.

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

Testing whether a distribution is ϵ-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

Beyond Black Box Testing

Definition (Conditional Sampling)

Given a distribution $\mathcal{A}$ on S one can

- *Specify a set $T \subseteq S$,*
- *Draw samples according to the distribution $\mathcal{A}|_T$, that is, $\mathcal{A}$ under the condition that the samples belong to T.*

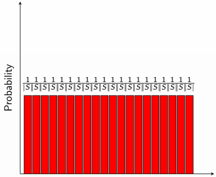
Definition (Conditional Sampling)

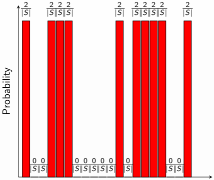
Given a distribution $\mathcal{A}$ on S one can

- *Specify a set $T \subseteq S$,*
- *Draw samples according to the distribution $\mathcal{A}|_T$, that is, $\mathcal{A}$ under the condition that the samples belong to T.*

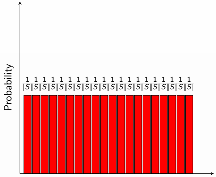
Conditional sampling is at least as powerful as drawing normal samples.
But how more powerful is it?

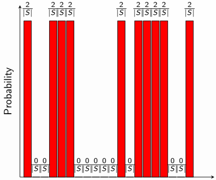
Testing Uniformity Using Conditional Sampling





Testing Uniformity Using Conditional Sampling

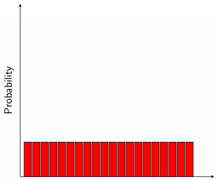


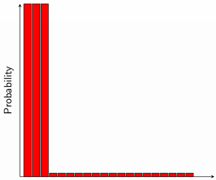


An algorithm for testing uniformity using conditional sampling:

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 $\mathcal{A}|_T$ is enough to identify that it is not uniform.

What about other distributions?



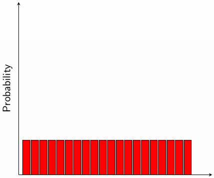


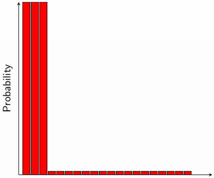
What about other distributions?

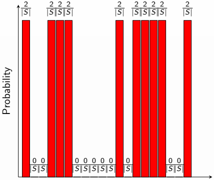
Previous algorithm fails in this case:

1. Draw two elements σ_1 and σ_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 ϵ.
3. Probability that we will be able to distinguish the far distribution from the uniform distribution is very low.

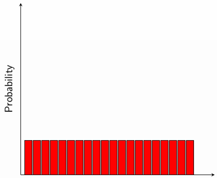
Testing Uniformity Using Conditional Sampling

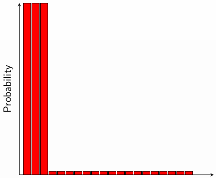


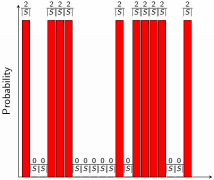




Testing Uniformity Using Conditional Sampling







1. Draw σ_1 uniformly at random from the domain and draw σ_2 according to the distribution $\mathcal{A}$. Let $T = \{\sigma_1, \sigma_2\}$.
2. In the case of the "far" distribution, with constant probability, σ_1 will have "low" probability and σ_2 will have "high" probibility.
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.

Barbarik

Input: A sampler under test $\mathcal{A}$, a reference uniform sampler $\mathcal{U}$, a tolerance parameter $\varepsilon > 0$, an intolerance parmaeter $\eta > \varepsilon$, a guarantee parameter δ

Output: ACCEPT or REJECT with the following guarantees:

- if the generator $\mathcal{A}$ is an ε-additive almost-uniform generator then Barbarik ACCEPTS with probability at least $(1 - \delta)$.
- if $\mathcal{A}$ is η-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 ε, η and δ, Barbarik need at most $K = \widetilde{O}(\frac{1}{(\eta-\varepsilon)^4})$ samples for any input formula φ, where the tilde hides a poly logarithmic factor of $1/\delta$ and $1/(\eta - \varepsilon)$.

- $\varepsilon = 0.6, \eta = 0.9, \delta = 0.1$
- 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

Instances	Size	UniGen2		SearchTreeSampler	
		Output	#Samples	Output	#Samples
71	1.14×2^{59}	A	1729750	R	250
blasted_case49	1.00×2^{61}	A	1729750	R	250
blasted_case50	1.00×2^{62}	A	1729750	R	250
scenarios_aig_insertion1	1.06×2^{65}	A	1729750	R	250
scenarios_aig_insertion2	1.06×2^{65}	A	1729750	R	250
36	1.00×2^{72}	A	1729750	R	250
30	1.73×2^{72}	A	1729750	R	250
110	1.09×2^{76}	A	1729750	R	250
scenarios_tree_insert_insert	1.32×2^{76}	A	1729750	R	250
107	1.52×2^{76}	A	1729750	R	250
blasted_case211	1.00×2^{80}	A	1729750	R	250
blasted_case210	1.00×2^{80}	A	1729750	R	250
blasted_case212	1.00×2^{88}	A	1729750	R	250
blasted_case209	1.00×2^{88}	A	1729750	R	250
54	1.15×2^{90}	A	1729750	R	250

Results-II

Instances	Size	UniGen2		QuickSampler	
		Output	#Samples	Output	#Samples
71	1.14×2^{59}	A	1729750	R	250
blasted_case49	1.00×2^{61}	A	1729750	R	250
blasted_case50	1.00×2^{62}	A	1729750	R	250
scenarios_aig_insertion1	1.06×2^{65}	A	1729750	R	250
scenarios_aig_insertion2	1.06×2^{65}	A	1729750	R	250
36	1.00×2^{72}	A	1729750	R	250
30	1.73×2^{72}	A	1729750	R	250
110	1.09×2^{76}	A	1729750	R	250
scenarios_tree_insert_insert	1.32×2^{76}	A	1729750	R	250
107	1.52×2^{76}	A	1729750	R	250
blasted_case211	1.00×2^{80}	A	1729750	R	250
blasted_case210	1.00×2^{80}	A	1729750	R	250
blasted_case212	1.00×2^{88}	A	1729750	R	250
blasted_case209	1.00×2^{88}	A	1729750	R	250
54	1.15×2^{90}	A	1729750	R	250

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

The Price of Success

- 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

The Price of Success

- 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

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 [E.g. MSS96a,MSS99]
- Delete less used clauses [E.g. GN02,ES03]
- Delete clauses based on Literal block distance [AS09]

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

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

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

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

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

- 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.

- 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.

- 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

- 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

- 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

- 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) $> t$.

- 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.

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

- 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.

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

Part 3: Data Collection

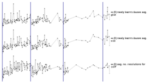
- Forward pass
  - The solver keeps track of features of each clause and dumps all the learnt clauses after we reach UNSAT.
  - genesis(C): The value of counter when C was learnt
  - expiry (C): The value of counter when C was last used in the UNSAT proof
- Backward pass
  - DRAT-trim is used to reconstruct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

Visualizing what happens during SAT solving has been a long-term goal of mine, and finally, I have managed to pull together something that I feel confident about. The system is fully explained in the liked image on the right, including how to read the graphs and why I made them. Here, I would like to talk about the challenges I had to overcome to create the system.



Visualizing the solving of mizh-md5-47-3.cnf

Gathering information

Gathering information during solving is challenging for two reasons. First, it's hard to know *what* to gather. Second, gathering the information should not affect overall speed of the solver (or only minimally), so the code to gather the information has to be well-written. To top it all, if much information is gathered, these have to be structured in a sane way, so it's easy to access later.

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 to decide about how to store/summarize 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 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 trivial to you what the graph meant, after all, you made it!

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-

https://www.msoos.org/2012/06/visualizing-sat-solving/ 1/2

Machine Learning and SAT

August 9, 2015 · Development, Research, SAT · glues, lingeling, 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 poor 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 without 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 glues 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' glues and then makes a decision. Looking at the code, the option actavgmax/stdmin/stdmax gives the cutoffs and the function lglneedacts calculates the values and decides. This has been in lingeling since 2011 (lingeling-587f).

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 claus-

https://www.msoos.org/2015/08/machine-learning-and-sat/ 1/2

- 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

- 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.

Accuracy

		Prediction	
		Throw	Keep
Ground	Throw	0.64	0.36
truth	Keep	0.11	0.89

Table: Confusion matrix

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.
- *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.
- *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*.

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

More Open Questions than Answers

- 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