The Rise of Approximate Model Counting: A Child of SAT Revolution

Kuldeep S. Meel

National University of Singapore

Max Planck Institute for Security and Privacy

2^3 Years and Counting

S. Akshay (IITB, India), Teodora Baluta (NUS, SG), Fabrizio Biondi (Avast, CZ), Supratik Chakraborty (IITB, India), Alexis de Colnet (NUS, SG), Remi Delannoy (NUS, SG), Jeffrey Dudek (Rice,US), Leonardo Duenas-Osorio (Rice,US), Mike Enescu (Inria, France) Daniel Fremont (UCB, US), Dror Fried (Open U., Israel), Stephan Gocht (Lund U., Sweden), Rahul Gupta (IITK, India), Annelie Heuser (Inria, France), Alexander Ivrii (IBM, Israel), Alexey Ignatiev (IST, Portugal), Axel Legay (UCL, Belgium), Sharad Malik (Princeton, US), Joao Marques Silva (IST, Portugal), Rakesh Mistry (IITB, India), Nina Narodytska ((VMWare, US), Roger Paredes (Rice,US), Yash Pote (NUS, SG), Jean Quilbeuf(Inria, France), Subhajit Roy (IITK, India), Mate Soos (NUS, SG), Prateek Saxena (NUS, SG), Sanjit Seshia (UCB, US), Shubham Sharma (IITK, India), Aditya Shrotri(Rice,US), Moshe Vardi (Rice,US)

Special shout out to Mate Soos, maintainer of ApproxMC

Open source tool: `github.com/meelgroup/approxmc`

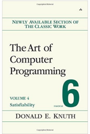
Boolean Satisfiability (SAT); Given a Boolean expression, using "and" ($\wedge$) "or", ($\vee$) and "not" ($\neg$), *is there a satisfying solution* (an assignment of 0's and 1's to the variables that makes the expression equal 1)?

Example:

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_1 \vee x_4)$$

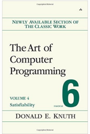
Solution: $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$

Modern SAT solvers are able to deal routinely with practical problems that involve millions of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)



The Tale of Triumph of SAT Solvers

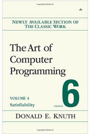
Modern SAT solvers are able to deal routinely with practical problems that involve millions of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)



Industrial usage of SAT Solvers: Model Checking, Planning, Genome Rearrangement, Telecom Feature Subscription, Resource Constrained Scheduling, Noise Analysis, Games, $\cdots$

The Tale of Triumph of SAT Solvers

Modern SAT solvers are able to deal routinely with practical problems that involve millions of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)



Industrial usage of SAT Solvers: Model Checking, Planning, Genome Rearrangement, Telecom Feature Subscription, Resource Constrained Scheduling, Noise Analysis, Games, $\cdots$

Now that SAT is "easy", it is time to look beyond satisfiability

Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$

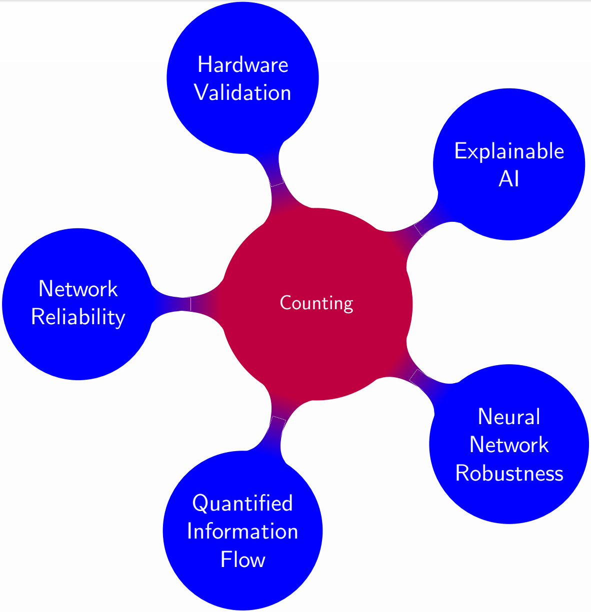
Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- Counting: Determine $|\mathsf{Sol}(F)|$
  - Approximation: $\Pr\left[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \leq c \leq |\mathsf{Sol}(F)|(1+\varepsilon)\right] \geq 1-\delta$

Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- Counting: Determine $|\mathsf{Sol}(F)|$
  - Approximation: $\Pr\left[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \leq c \leq |\mathsf{Sol}(F)|(1+\varepsilon)\right] \geq 1-\delta$
- Given $F := (X_1 \vee X_2)$
- $\mathsf{Sol}(F) = \{(0,1),(1,0),(1,1)\}$
- $|\mathsf{Sol}(F)| = 3$

Applications across Computer Science



Today's Menu

Testing of AI systems

Information Leakage

Network Reliability

Today's Menu

Testing of AI systems

Information Leakage Model Counting

Network Reliability

Today's Menu

Testing of AI systems

Information Leakage   Model Counting   Hashing Framework

Network Reliability

The Rise of Hashing-based Approach: Promise of Scalability and Guarantees
(S83,GSS06,GHSS07,CMV13b,EGSS13b,CMV14,CDR15,CMV16,ZCSE16,AD16 KM18,ATD18,SM19,ABM20,SGM20)

Testing of AI Systems

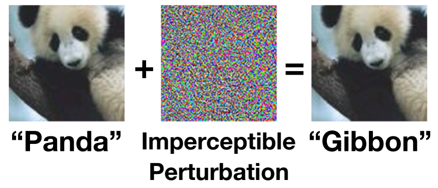
- Classical verification/testing setup for traditional systems
  - System captured as a model $M(\mathcal{I}, \mathcal{O})$ via logical constraints
  - Specification $\varphi(\mathcal{I}, \mathcal{O})$: relationship between input and output
  - Methodology: SAT (i.e., find one execution of M such that φ is not satisfied)

Testing of AI Systems

- Classical verification/testing setup for traditional systems
  - System captured as a model $M(\mathcal{I}, \mathcal{O})$ via logical constraints
  - Specification $\varphi(\mathcal{I}, \mathcal{O})$: relationship between input and output
  - Methodology: SAT (i.e., find one execution of M such that φ is not satisfied)
- Modern Machine Learning Systems
  - Model: A given neural network and an image
  - Specification: For all small perturbations, the model should not give different answers.

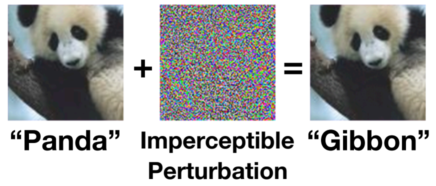
Testing of AI Systems

- Classical verification/testing setup for traditional systems
  - System captured as a model $M(\mathcal{I}, \mathcal{O})$ via logical constraints
  - Specification $\varphi(\mathcal{I}, \mathcal{O})$: relationship between input and output
  - Methodology: SAT (i.e., find one execution of M such that φ is not satisfied)
- Modern Machine Learning Systems
  - Model: A given neural network and an image
  - Specification: For all small perturbations, the model should not give different answers.



Testing of AI Systems

- Classical verification/testing setup for traditional systems
  - System captured as a model $M(\mathcal{I}, \mathcal{O})$ via logical constraints
  - Specification $\varphi(\mathcal{I}, \mathcal{O})$: relationship between input and output
  - Methodology: SAT (i.e., find one execution of M such that φ is not satisfied)
- Modern Machine Learning Systems
  - Model: A given neural network and an image
  - Specification: For all small perturbations, the model should not give different answers.



- Acceptable despite multiple executions with error
- Estimate the frequency of such behavior: Counting (BSSMS, 2019)

Quantitative Information Flow

Algorithm PC1 (SP, UI)

```
for i = 0; i <SP.length(); i++ do
    if SP[i] ≠ UI[i] then
        return "No"
    return "Yes"
end for
```

Algorithm PC2 (SP, UI)

```
match = "Yes"
for i = 0; i <SP.length(); i++ do
    if SP[i] ≠ UI[i] then
        match="No"
end for
return match
```

Quantification of Information Leakage between PC1 and PC2 via side-channels (such as time?)

- Annotate every line of program with time taken and perform symbolic analysis
- Shannon Entropy $= \sum_t \Pr[\text{finishtime} = t] \log \frac{1}{\Pr[\text{time}=t]}$

(Bang et al., 2016)

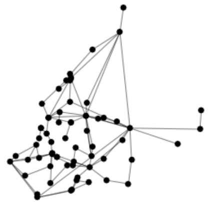


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \to [0, 1]$
- Compute Pr[s and t are disconnected]?

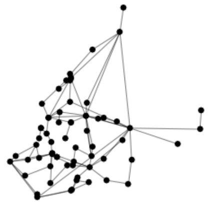


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \to [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a $0/1$ vector of size $|E|$
- $W(\pi) = \Pr(\pi)$

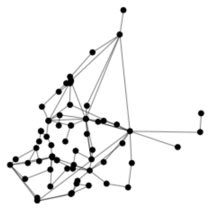


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \to [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a 0/1 vector of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where s and t are disconnected
  - Represented as a solution to set of constraints over edge variables

Reliability of Critical Infrastructure Networks

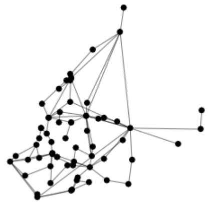


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \rightarrow [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a $0/1$ vector of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where s and t are disconnected
  - Represented as a solution to set of constraints over edge variables
- Pr[s and t are disconnected] $= \sum_{\pi_{s,t}} W(\pi_{s,t})$

Reliability of Critical Infrastructure Networks

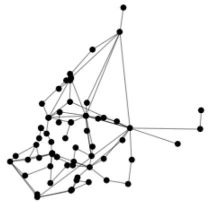


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \to [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a 0/1 vector of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where s and t are disconnected
  - Represented as a solution to set of constraints over edge variables
- Pr[s and t are disconnected] $= \sum_{\pi_{s,t}} W(\pi_{s,t})$

(DMPV, AAAI 17, ICASP-13, RESS 2019)

Constrained Counting

Prior Work

Strong guarantees but poor scalability

- Exact counters (Birnbaum and Lozinskii 1999, Jr. and Schrag 1997, Sang et al. 2004, Thurley 2006)
- Hashing-based approach (Stockmeyer 1983, Jerrum Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Bounding counters (Gomes et al. 2007,Kroc, Sabharwal, and Selman 2008, Gomes, Sabharwal, and Selman 2006, Kroc, Sabharwal, and Selman 2008)
- Sampling-based techniques (Wei and Selman 2005, Rubinstein 2012, Gogate and Dechter 2011)

Prior Work

Strong guarantees but poor scalability

- Exact counters (Birnbaum and Lozinskii 1999, Jr. and Schrag 1997, Sang et al. 2004, Thurley 2006)
- Hashing-based approach (Stockmeyer 1983, Jerrum Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Bounding counters (Gomes et al. 2007,Kroc, Sabharwal, and Selman 2008, Gomes, Sabharwal, and Selman 2006, Kroc, Sabharwal, and Selman 2008)
- Sampling-based techniques (Wei and Selman 2005, Rubinstein 2012, Gogate and Dechter 2011)

How to bridge this gap between theory and practice?

Obs 1 SAT Solver $\neq$ NP Oracle

Standing on the Shoulders of SAT Revolution

Obs 1 SAT Solver $\neq$ NP Oracle

- Return a satisfying assignment if satisfiable
- The performance of solver depends on the formulas

Standing on the Shoulders of SAT Revolution

Obs 1 SAT Solver $\neq$ NP Oracle

- Return a satisfying assignment if satisfiable
- The performance of solver depends on the formulas

Obs 2 Memoryfulness

- Incremental Solving: Often easier to solve F followed by G if we G can be written as $G = F \wedge H$
- If $F \rightarrow C$ then $(F \wedge H) \implies C$

Counting in Bochum

How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364K$)

Counting in Bochum

How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364K$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 364K/50

Counting in Bochum

How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364$K)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 364K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50

Counting in Bochum

How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364K$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 364K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT Query: Find a person who likes coffee

Counting in Bochum

How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364$K)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 364K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person y

Counting in Bochum

How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364K$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 364K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person y
- Attempt #2: Enumerate every person who likes coffee

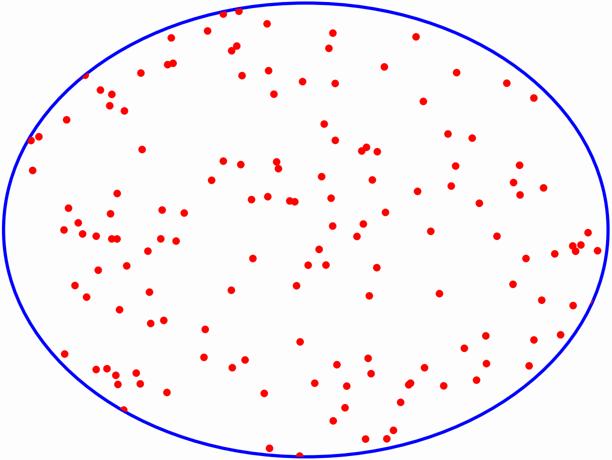
Counting in Bochum

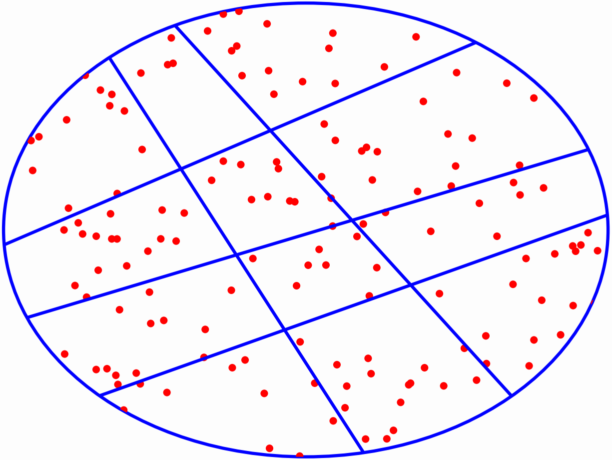
How many people in Bochum like coffee?

- Population of Bochum = 364K
- Assign every person a unique ($n =$) 19 bit identifier ($2^n = 364K$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 364K/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- SAT Query: Find a person who likes coffee
- A SAT solver can answer queries like:
  - Q1: Find a person who likes coffee
  - Q2: Find a person who likes coffee and is not person y
- Attempt #2: Enumerate every person who likes coffee
  - Potentially 2^n queries

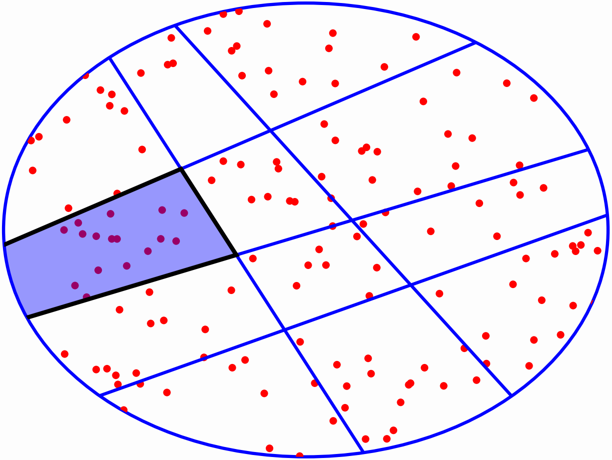
Can we do with lesser # of SAT queries – $\mathcal{O}(n)$ or $\mathcal{O}(\log n)$?

As Simple as Counting Dots





Pick a random cell



Estimate $=$ Number of solutions in a cell $\times$ Number of cells

Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

Challenge 2 How many cells?

Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Designing function h : assignments $\rightarrow$ cells (hashing)
- Solutions in a cell α: $\mathsf{Sol}(F) \cap \{y \mid h(y) = \alpha\}$

Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Designing function h : assignments $\rightarrow$ cells (hashing)
- Solutions in a cell α: $\mathsf{Sol}(F) \cap \{y \mid h(y) = \alpha\}$
- Deterministic h unlikely to work

Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Designing function h : assignments $\rightarrow$ cells (hashing)
- Solutions in a cell α: $\mathsf{Sol}(F) \cap \{y \mid h(y) = \alpha\}$
- Deterministic h unlikely to work
- Choose h randomly from a large family H of hash functions

  $\boxed{\text{Universal Hashing (Carter and Wegman 1977)}}$

2-wise independent Hashing

- Let H be family of 2-wise independent hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$

$$\forall y_1, y_2 \in \{0,1\}^n, \alpha_1, \alpha_2 \in \{0,1\}^m, h \xleftarrow{R} H$$
$$\Pr[h(y_1) = \alpha_1] = \Pr[h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)$$

$$\Pr[h(y_1) = \alpha_1 \wedge h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)^2$$

2-wise independent Hashing

- Let H be family of 2-wise independent hash functions mapping $\{0,1\}^n$ to $\{0,1\}^m$

$$\forall y_1, y_2 \in \{0,1\}^n, \alpha_1, \alpha_2 \in \{0,1\}^m, h \xleftarrow{R} H$$
$$\Pr[h(y_1) = \alpha_1] = \Pr[h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)$$

$$\Pr[h(y_1) = \alpha_1 \wedge h(y_2) = \alpha_2] = \left(\frac{1}{2^m}\right)^2$$

- The power of 2-wise independentity
  - Z be the number of solutions in a randomly chosen cell
  - $\mathsf{E}[Z] = \frac{|\mathsf{Sol}(F)|}{2^m}$
  - $\sigma^2[Z] \leq \mathsf{E}[Z]$

2-wise independent Hash Functions

- Variables: $X_1, X_2, \cdots X_n$
- To construct $h : \{0,1\}^n \rightarrow \{0,1\}^m$, choose m random XORs
- Pick every X_i with prob. $\frac{1}{2}$ and XOR them
  - $X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
  - Expected size of each XOR: $\frac{n}{2}$

2-wise independent Hash Functions

- Variables: $X_1, X_2, \cdots X_n$
- To construct $h : \{0,1\}^n \rightarrow \{0,1\}^m$, choose m random XORs
- Pick every X_i with prob. $\frac{1}{2}$ and XOR them
  - $X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
  - Expected size of each XOR: $\frac{n}{2}$
- To choose $\alpha \in \{0,1\}^m$, set every XOR equation to 0 or 1 randomly

$$X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} = 0 \qquad (Q_1)$$
$$X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} = 1 \qquad (Q_2)$$
$$\cdots \qquad (\cdots)$$
$$X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} = 1 \qquad (Q_m)$$

- Solutions in a cell: $F \wedge Q_1 \cdots \wedge Q_m$

2-wise independent Hash Functions

- Variables: $X_1, X_2, \cdots X_n$
- To construct $h : \{0,1\}^n \rightarrow \{0,1\}^m$, choose m random XORs
- Pick every X_i with prob. $\frac{1}{2}$ and XOR them
  - $X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2}$
  - Expected size of each XOR: $\frac{n}{2}$
- To choose $\alpha \in \{0,1\}^m$, set every XOR equation to 0 or 1 randomly

$$X_1 \oplus X_3 \oplus X_6 \cdots \oplus X_{n-2} = 0 \qquad (Q_1)$$
$$X_2 \oplus X_5 \oplus X_6 \cdots \oplus X_{n-1} = 1 \qquad (Q_2)$$
$$\cdots \qquad (\cdots)$$
$$X_1 \oplus X_2 \oplus X_5 \cdots \oplus X_{n-2} = 1 \qquad (Q_m)$$

- Solutions in a cell: $F \wedge Q_1 \cdots \wedge Q_m$
- Performance of state of the art SAT solvers depends on the formulas (SAT Solvers != SAT oracles)

Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of F
  - $F := X_3 \iff (X_1 \vee X_2)$
  - X_1 and X_2 uniquely determines rest of the variables (i.e., X_3)
- Formally: if I is independent support, then $\forall \sigma_1, \sigma_2 \in \mathsf{Sol}(F)$, if σ_1 and σ_2 agree on I then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not

Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of F
  - $F := X_3 \iff (X_1 \vee X_2)$
  - X_1 and X_2 uniquely determines rest of the variables (i.e., X_3)
- Formally: if I is independent support, then $\forall \sigma_1, \sigma_2 \in \mathsf{Sol}(F)$, if σ_1 and σ_2 agree on I then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over I (CMV DAC14)

Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of F
  - $F := X_3 \iff (X_1 \vee X_2)$
  - X_1 and X_2 uniquely determines rest of the variables (i.e., X_3)
- Formally: if I is independent support, then $\forall \sigma_1, \sigma_2 \in \mathsf{Sol}(F)$, if σ_1 and σ_2 agree on I then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over I (CMV DAC14)
- Typically I is 1-2 orders of magnitude smaller than X
- Auxiliary variables introduced during encoding phase are *dependent* (Tseitin 1968)

Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of F
  - $F := X_3 \iff (X_1 \vee X_2)$
  - X_1 and X_2 uniquely determines rest of the variables (i.e., X_3)
- Formally: if I is independent support, then $\forall \sigma_1, \sigma_2 \in \mathsf{Sol}(F)$, if σ_1 and σ_2 agree on I then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over I (CMV DAC14)
- Typically I is 1-2 orders of magnitude smaller than X
- Auxiliary variables introduced during encoding phase are *dependent* (Tseitin 1968)

Algorithmic procedure to determine I?

Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of F
  - $F := X_3 \iff (X_1 \vee X_2)$
  - X_1 and X_2 uniquely determines rest of the variables (i.e., X_3)
- Formally: if I is independent support, then $\forall \sigma_1, \sigma_2 \in \mathsf{Sol}(F)$, if σ_1 and σ_2 agree on I then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over I (CMV DAC14)
- Typically I is 1-2 orders of magnitude smaller than X
- Auxiliary variables introduced during encoding phase are *dependent* (Tseitin 1968)

Algorithmic procedure to determine I?

- FP^{NP} procedure via reduction to Minimal Unsatisfiable Subset

Improved 2-wise Independent Hash Functions

- Not all variables are required to specify solution space of F
  - $F := X_3 \iff (X_1 \vee X_2)$
  - X_1 and X_2 uniquely determines rest of the variables (i.e., X_3)
- Formally: if I is independent support, then $\forall \sigma_1, \sigma_2 \in \mathsf{Sol}(F)$, if σ_1 and σ_2 agree on I then $\sigma_1 = \sigma_2$
  - $\{X_1, X_2\}$ is independent support but $\{X_1, X_3\}$ is not
- Random XORs need to be constructed only over I (CMV DAC14)
- Typically I is 1-2 orders of magnitude smaller than X
- Auxiliary variables introduced during encoding phase are *dependent* (Tseitin 1968)

Algorithmic procedure to determine I?

- FP^{NP} procedure via reduction to Minimal Unsatisfiable Subset
- Two orders of magnitude runtime improvement

(IMMV; CP15, Constraints16)

The Hope of Short XORs

- If we pick every variable X_i with probability p .
  - Expected Size of each XOR: np
  - $\mathsf{E}[Z_m] = \frac{|\mathsf{Sol}(F)|}{2^m}$
  - $\sigma^2[Z_m] \leq \mathsf{E}[Z_m] + \sum_{\sigma_1 \in \mathsf{Sol}(F)} \sum_{\substack{\sigma_2 \in \mathsf{Sol}(F) \\ w = d(\sigma_1, \sigma_2)}} r(w, m)$
    - where, $r(w, m) = \left(\left(\frac{1}{2} + \frac{(1-2p)^w}{2} \right)^m - \frac{1}{2^m} \right)$
  - For $p = \frac{1}{2}$, we have $\frac{\sigma^2[Z_m]}{\mathsf{E}[Z_m]} \leq 1$
- Earlier Attempts (GSS07,EGSS14,ZCSE16,AD17,ATD18)
  - $\sum_{\sigma_1 \in \mathsf{Sol}(F)} \sum_{\substack{\sigma_2 \in \mathsf{Sol}(F) \\ w = d(\sigma_1, \sigma_2)}} r(w, m) \leq \sum_{\sigma_1 \in \mathsf{Sol}(F)} \sum_{w=0}^{n} \binom{n}{w} r(w, m)$
  - $\binom{n}{w}$ grows very fast with n, so could not upper bound $\frac{\sigma^2[Z_m]}{\mathsf{E}[Z_m]}$
  - The weak bounds lead to significant slowdown: typically $100\times$ to $1000\times$ factor of slowdown! (ATD18,ABM20)

The Power of Isoperimetric Inequalities

- $\sum_{\sigma_1 \in \mathrm{Sol}(F)} \sum_{\substack{\sigma_2 \in \mathrm{Sol}(F) \\ w = d(\sigma_1, \sigma_2)}} r(w, m) = \sum_{w=0}^{n} C_F(w) r(w, m)$
- $C_F(w) = |\{\sigma_1, \sigma_2 \in \mathrm{Sol}(F) \mid d(\sigma_1, \sigma_2) = w\}|$

The Power of Isoperimetric Inequalities

- $\sum_{\sigma_1 \in \mathrm{Sol}(F)} \sum_{\substack{\sigma_2 \in \mathrm{Sol}(F) \\ w = d(\sigma_1, \sigma_2)}} r(w, m) = \sum_{w=0}^{n} C_F(w) r(w, m)$
- $C_F(w) = |\{\sigma_1, \sigma_2 \in \mathrm{Sol}(F) \mid d(\sigma_1, \sigma_2) = w\}|$
- Isoperimetric Inequalities! (Rashtchian and Raynaud 2019)

Lemma

$$\sum_{w=0}^{n} C_F(w) r(w, m) \leq \sum_{w=0}^{n} \binom{8e\sqrt{n \cdot \ell}}{w} r(w, m) \ \textit{where}\ \ell = \log |\mathrm{Sol}(F)|$$

  - $\frac{\binom{n}{w}}{\binom{8e\sqrt{n \cdot \ell}}{w}} \approx \left(\frac{n}{\ell}\right)^{\frac{w}{2}}$

From Linear to Logarithmic Size XORs

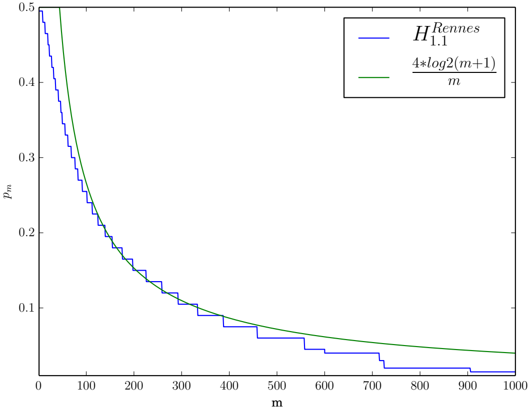
Theorem (MA, LICS-20)

For all q, k, $|\mathsf{Sol}(F)| \leq k \cdot 2^m$, $p = \mathcal{O}(\frac{\log m}{m})$ *we have*

$$\frac{\sigma^2[Z_m]}{\mathsf{E}[Z_m]} \leq q (a\ constant)$$

Recall, average size of XORs: $n \cdot p$
Improvement of p *from* $\frac{m/2}{m}$ *to* $\frac{\log m}{m}$

Sparse Hash Functions



$H_{1.1}^{Rennes}$: Sparse hash functions that guarantee $q = 1.1$

Handling CNF+XOR Formulas

- CNF + Sparse XORs are still CNF+XOR formulas.
- Translating XORs to CNF and performing CDCL is not sufficient

Handling CNF+XOR Formulas

- CNF + Sparse XORs are still CNF+XOR formulas.
- Translating XORs to CNF and performing CDCL is not sufficient
  - XORs can be solved by Gaussian elimination
- CryptoMiniSAT: Solver designed to perform CDCL and Gaussian Elimination in tandem (SNC09)
- BIRD2 (Blast, Inprocess, Recover, Detach, and Destroy): Tighter integration (SM19, SGM20)

Challenges

Challenge 1 How to partition into roughly equal small cells of solutions without knowing the distribution of solutions?

- Independent Support-based XORs
- Specialized CNF Solvers

Challenge 2 How many cells?

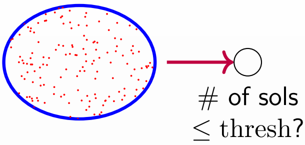
Challenge 2: How many cells?

- A cell is small if it has $\approx \mathrm{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions
- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$

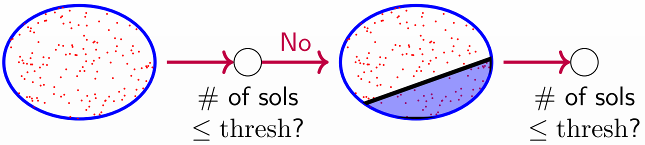
Challenge 2: How many cells?

- A cell is small if it has $\approx \mathrm{thresh} = 5(1+\frac{1}{\varepsilon})^2$ solutions
- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$
  - Check for every $m = 0, 1, \cdots n$ if the number of solutions $\leq \mathrm{thresh}$

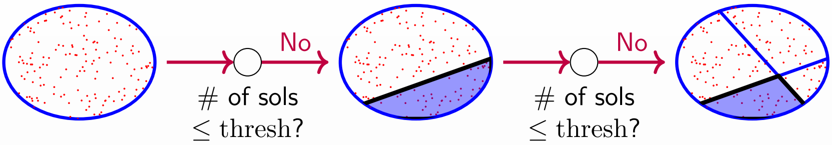
ApproxMC



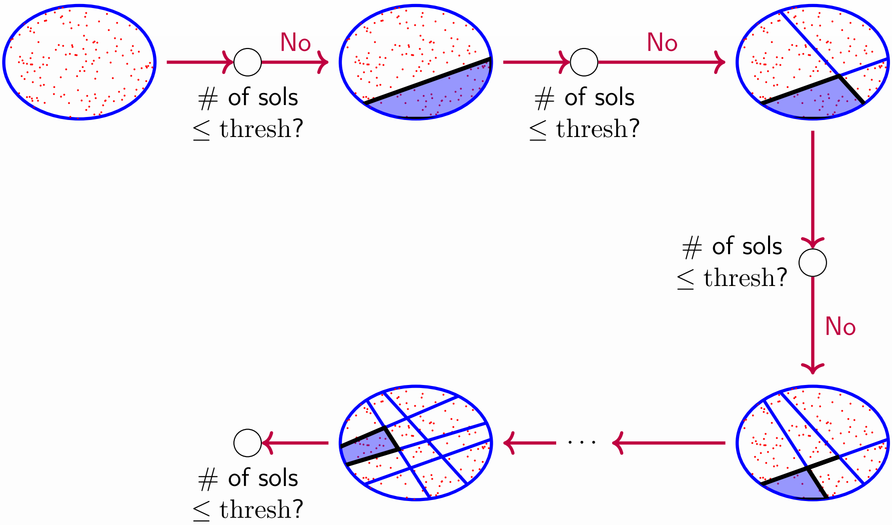
ApproxMC



ApproxMC

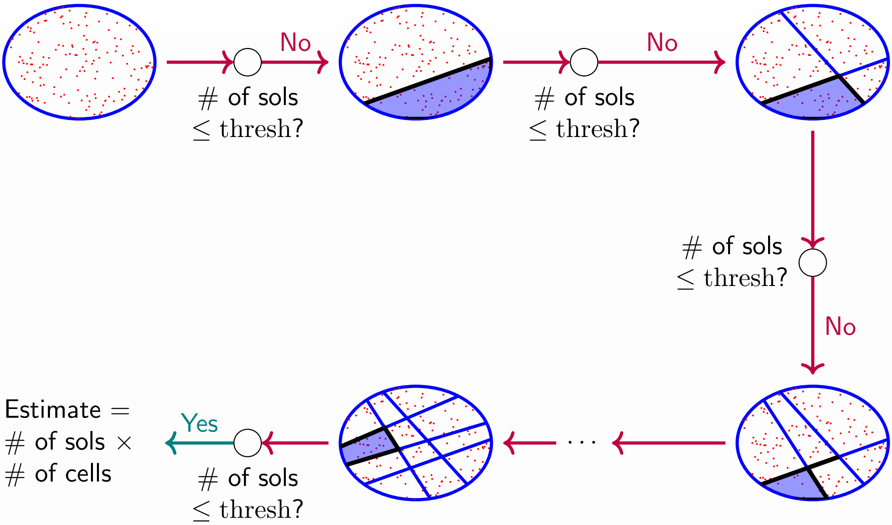


ApproxMC



No
# of sols
$\leq$ thresh?
No
# of sols
$\leq$ thresh?
# of sols
$\leq$ thresh?
No
...
# of sols
$\leq$ thresh?

ApproxMC


No
# of sols
≤ thresh?
No
# of sols
≤ thresh?
# of sols
≤ thresh?
No
Estimate =
# of sols ×
# of cells
Yes
# of sols
≤ thresh?
· · ·

ApproxMC

- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$
  - Query 1: Is $\#(F \wedge Q_1) \leq \mathrm{thresh}$
  - Query 2: Is $\#(F \wedge Q_1 \wedge Q_2) \leq \mathrm{thresh}$
  - $\cdots$
  - Query n: Is $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_n) \leq \mathrm{thresh}$
- Stop at the first m where Query m returns YES and return estimate as $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_m) \times 2^m$
- Observation: $\#(F \wedge Q_1 \cdots \wedge Q_i \wedge Q_{i+1}) \leq \#(F \wedge Q_1 \cdots \wedge Q_i)$
  - If Query i returns YES, then Query $i+1$ must return YES

ApproxMC

- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$
  - Query 1: Is $\#(F \wedge Q_1) \leq \text{thresh}$
  - Query 2: Is $\#(F \wedge Q_1 \wedge Q_2) \leq \text{thresh}$
  - $\ldots$
  - Query n: Is $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_n) \leq \text{thresh}$
- Stop at the first m where Query m returns YES and return estimate as $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_m) \times 2^m$
- Observation: $\#(F \wedge Q_1 \cdots \wedge Q_i \wedge Q_{i+1}) \leq \#(F \wedge Q_1 \cdots \wedge Q_i)$
  - If Query i returns YES, then Query $i+1$ must return YES
  - Logarithmic search (# of SAT calls: $\mathcal{O}(\log n)$)
  - Incremental Search

ApproxMC

- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$
  - Query 1: Is $\#(F \wedge Q_1) \leq \mathrm{thresh}$
  - Query 2: Is $\#(F \wedge Q_1 \wedge Q_2) \leq \mathrm{thresh}$
  - $\cdots$
  - Query n: Is $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_n) \leq \mathrm{thresh}$
- Stop at the first m where Query m returns YES and return estimate as $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_m) \times 2^m$
- Observation: $\#(F \wedge Q_1 \cdots \wedge Q_i \wedge Q_{i+1}) \leq \#(F \wedge Q_1 \cdots \wedge Q_i)$
  - If Query i returns YES, then Query $i+1$ must return YES
  - Logarithmic search (# of SAT calls: $\mathcal{O}(\log n)$)
  - Incremental Search
- Will this work? Will the "m" where we stop be close to m^*?

ApproxMC

- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$
  - Query 1: Is $\#(F \wedge Q_1) \leq \mathrm{thresh}$
  - Query 2: Is $\#(F \wedge Q_1 \wedge Q_2) \leq \mathrm{thresh}$
  - $\cdots$
  - Query n: Is $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_n) \leq \mathrm{thresh}$
- Stop at the first m where Query m returns YES and return estimate as $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_m) \times 2^m$
- Observation: $\#(F \wedge Q_1 \cdots \wedge Q_i \wedge Q_{i+1}) \leq \#(F \wedge Q_1 \cdots \wedge Q_i)$
  - If Query i returns YES, then Query $i+1$ must return YES
  - Logarithmic search (# of SAT calls: $\mathcal{O}(\log n)$)
  - Incremental Search
- Will this work? Will the "m" where we stop be close to m^*?
  - Challenge Query i and Query j are not independent
  - Independence crucial to analysis (Stockmeyer 1983, $\cdots$)

ApproxMC

- We want to partition into 2^{m^*} cells such that $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$
  - Query 1: Is $\#(F \wedge Q_1) \leq \mathrm{thresh}$
  - Query 2: Is $\#(F \wedge Q_1 \wedge Q_2) \leq \mathrm{thresh}$
  - $\cdots$
  - Query n: Is $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_n) \leq \mathrm{thresh}$
- Stop at the first m where Query m returns YES and return estimate as $\#(F \wedge Q_1 \wedge Q_2 \cdots \wedge Q_m) \times 2^m$
- Observation: $\#(F \wedge Q_1 \cdots \wedge Q_i \wedge Q_{i+1}) \leq \#(F \wedge Q_1 \cdots \wedge Q_i)$
  - If Query i returns YES, then Query $i+1$ must return YES
  - Logarithmic search (# of SAT calls: $\mathcal{O}(\log n)$)
  - Incremental Search
- Will this work? Will the "m" where we stop be close to m^*?
  - Challenge Query i and Query j are not independent
  - Independence crucial to analysis (Stockmeyer 1983, $\cdots$)
  - Key Insight: The probability of making a bad choice of Q_i is very small for $i \ll m^*$

(CMV, IJCAI16)

Taming the Curse of Dependence

Let $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$ $(m^* = \log(\frac{|\mathsf{Sol}(F)|}{\text{thresh}}))$

Lemma (1)

ApproxMC terminates with $m \in \{m^* - 1, m^*\}$ with probability $\geq$ 0.8

Lemma (2)

For $m \in \{m^* - 1, m^*\}$, estimate obtained from a randomly picked cell lies within a tolerance of ε of $|\mathsf{Sol}(F)|$ with probability $\geq$ 0.8

Repeat $\mathcal{O}(\log(1/\delta))$ times and return the median

ApproxMC

Theorem (Correctness)

$$\Pr\left[\frac{|\mathrm{Sol}(F)|}{1+\varepsilon} \leq \mathit{ApproxMC}(F, \varepsilon, \delta) \leq |\mathrm{Sol}(F)|(1+\varepsilon)\right] \geq 1-\delta$$

Theorem (Complexity)

$\mathit{ApproxMC}(F, \varepsilon, \delta)$ makes $\mathcal{O}(\frac{\log n \log(\frac{1}{\delta})}{\varepsilon^2})$ calls to SAT oracle.

ApproxMC

Theorem (Correctness)

$$\Pr\left[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \leq \mathit{ApproxMC}(F,\varepsilon,\delta) \leq |\mathsf{Sol}(F)|(1+\varepsilon)\right] \geq 1-\delta$$

Theorem (Complexity)

$\mathit{ApproxMC}(F,\varepsilon,\delta)$ makes $\mathcal{O}(\frac{\log n \log(\frac{1}{\delta})}{\varepsilon^2})$ calls to SAT oracle.

Theorem (FPRAS for DNF; (MSV, FSTTCS 17; CP 18, IJCAI-19))

If F is a DNF formula, then ApproxMC is FPRAS – different from the Monte-Carlo based FPRAS for DNF (Karp, Luby 1983)

Reliability of Critical Infrastructure Networks

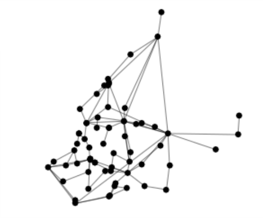
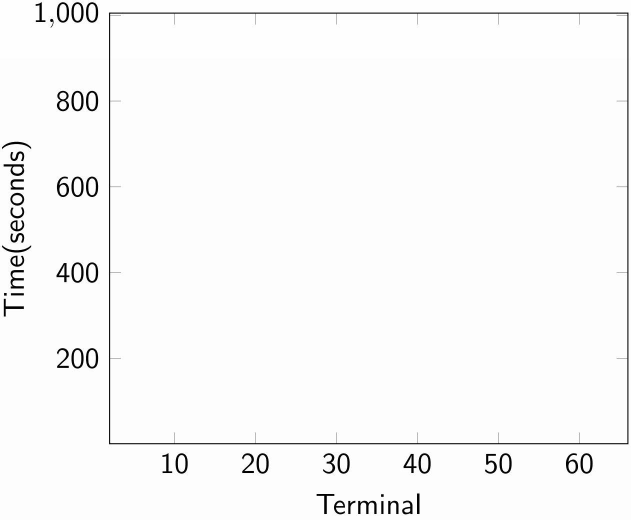


Figure: Plantersville, SC

- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds



(DMPV, AAAI17)

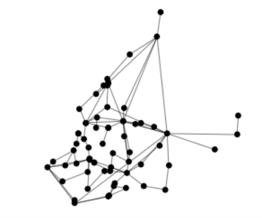
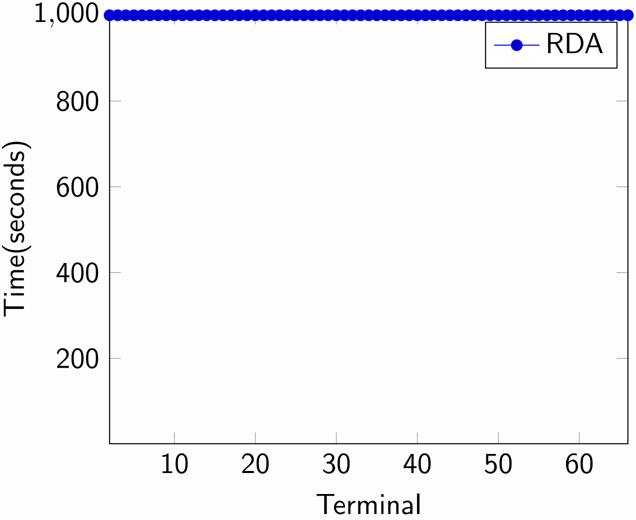


Figure: Plantersville, SC

- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds



(DMPV, AAAI17)

Reliability of Critical Infrastructure Networks

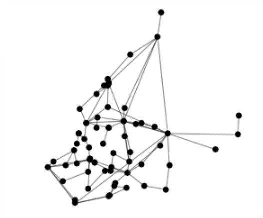
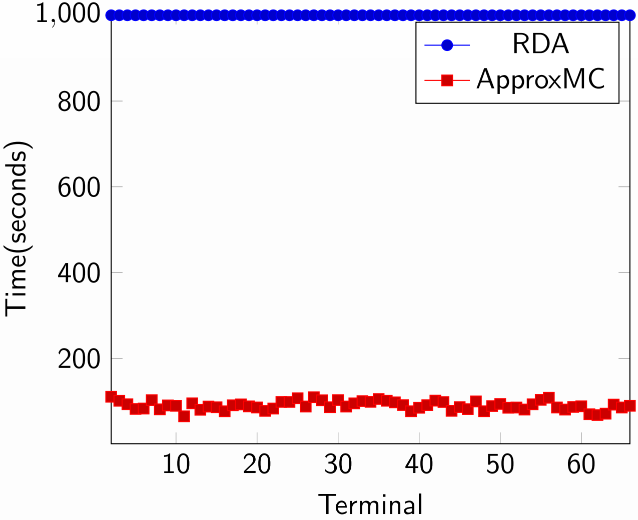


Figure: Plantersville, SC

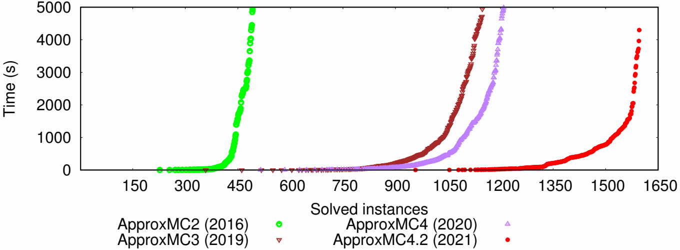
- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds



(DMPV, AAAI17)

Improvements Over the Years



Challenge Problems

Enabling “Counting Revolution”

Challenge Problems

Neural Network Robustness Handle 1000 neurons per layer

Civil Engineering Reliability for Los Angeles Transmission Grid

Security Leakage Measurement for C++ program with 1K lines

Enabling "Counting Revolution"

Challenge Problems

Neural Network Robustness Handle 1000 neurons per layer

Civil Engineering Reliability for Los Angeles Transmission Grid

Security Leakage Measurement for C++ program with 1K lines

Technical Directions

- Beyond SAT: Satisfiability Modulo Theory
- Formula-Dependent Sparser XOR constraints
- Tighter integration between solvers and algorithms

Enabling "Counting Revolution"

Challenge Problems

Neural Network Robustness Handle 1000 neurons per layer

Civil Engineering Reliability for Los Angeles Transmission Grid

Security Leakage Measurement for C++ program with 1K lines

Technical Directions

- Beyond SAT: Satisfiability Modulo Theory
- Formula-Dependent Sparser XOR constraints
- Tighter integration between solvers and algorithms

Questions?

Reliability of Critical Infrastructure Networks

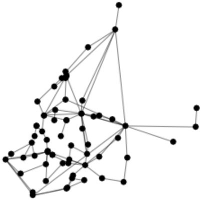


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \to [0, 1]$
- Compute Pr[s and t are disconnected]?

Reliability of Critical Infrastructure Networks

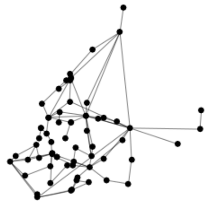


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \rightarrow [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a 0/1 vector of size $|E|$
- $W(\pi) = \Pr(\pi)$

Reliability of Critical Infrastructure Networks

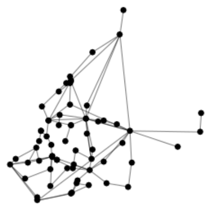


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \rightarrow [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a 0/1 vector of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where s and t are disconnected
  - Represented as a solution to set of constraints over edge variables

Reliability of Critical Infrastructure Networks

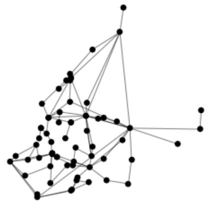


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \rightarrow [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a 0/1 vector of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where s and t are disconnected
  - Represented as a solution to set of constraints over edge variables
- Pr[s and t are disconnected] $= \sum_{\pi_{s,t}} W(\pi_{s,t})$

Reliability of Critical Infrastructure Networks

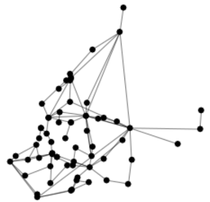


Figure: Plantersville, SC

Constrained Counting

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \rightarrow [0, 1]$
- Compute Pr[s and t are disconnected]?
- π : Configuration (of network) denoted by a 0/1 vector of size $|E|$
- $W(\pi) = \Pr(\pi)$
- $\pi_{s,t}$: configuration where s and t are disconnected
  - Represented as a solution to set of constraints over edge variables
- Pr[s and t are disconnected] $= \sum_{\pi_{s,t}} W(\pi_{s,t})$

(DMPV, AAAI 17, ICASP-13, RESS 2019)

Reliability of Critical Infrastructure Networks

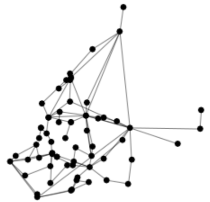
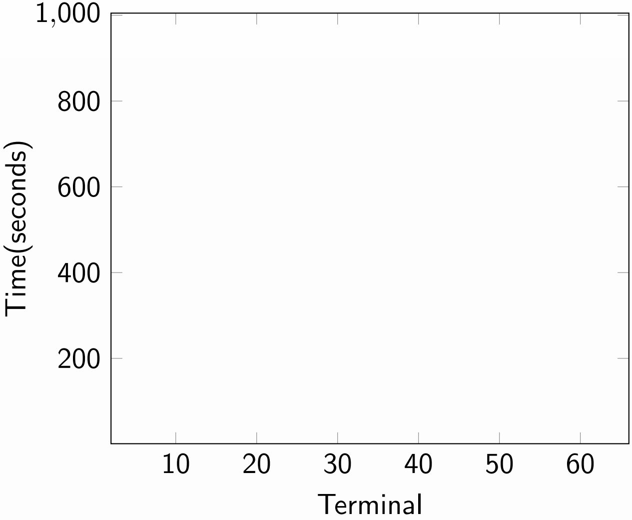


Figure: Plantersville, SC

- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds



(DMPV, AAAI17)

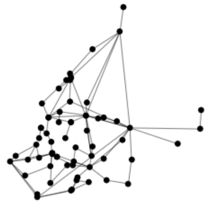
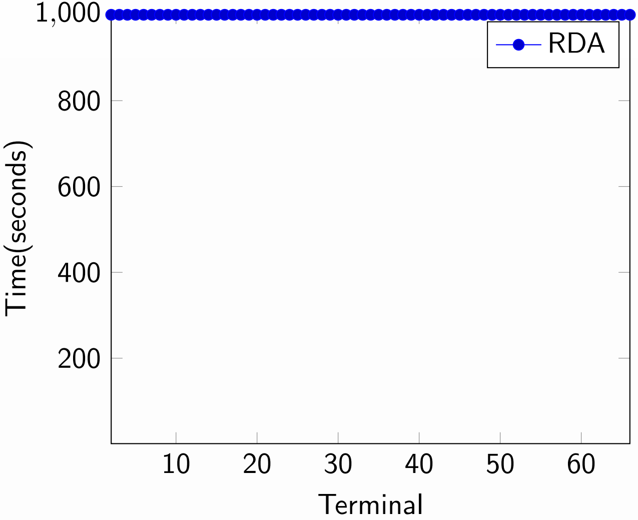


Figure: Plantersville, SC



- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds

(DMPV, AAAI17)

Reliability of Critical Infrastructure Networks

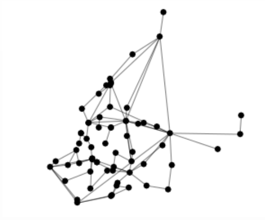
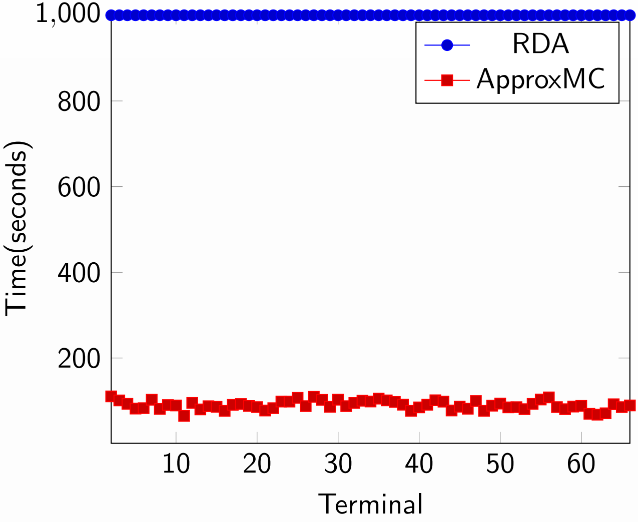


Figure: Plantersville, SC

- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds



(DMPV, AAAI17)