Constrained Counting and Sampling: From Theory to Practice and Back

Kuldeep S. Meel

National University of Singapore

SPIN 2019

Collaborators (30+ across 10 countries)

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), 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)

Thanks to Joao Marques-Silva for slides on CDCL solving.

Model Checking of Software

- Ball and Rajamani; SPIN 2001: "safety properties of system software can be validated and invalidated using model checking....We model abstractions of C programs using boolean programs... [we use] Bebop, a tool for model checking boolean programs "
- Ball and Rajamani; SPIN 2000: "[In Bebop], we use Binary Decisions Diagrams (BDDs) to symbolically represent these summaries, which are binary relationships between sets of states."

Model Checking of Software

- Ball and Rajamani; SPIN 2001: "safety properties of system software can be validated and invalidated using model checking....We model abstractions of C programs using boolean programs... [we use] Bebop, a tool for model checking boolean programs "
- Ball and Rajamani; SPIN 2000: "[In Bebop], we use Binary Decisions Diagrams (BDDs) to symbolically represent these summaries, which are binary relationships between sets of states."
- Core underlying problem: **Boolean Satisfiability (SAT)**

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

History:

- William Stanley Jevons, 1835-1882: "I have given much attention, therefore, to lessening both the manual and mental labour of the process, and I shall describe several devices which may be adopted for saving trouble and risk of mistake."
- Ernst Schröder, 1841-1902: "Getting a handle on the consequences of any premises, or at least the fastest method for obtaining these consequences, seems to me to be one of the noblest, if not the ultimate goal of mathematics and logic."

Complexity of Boolean Reasoning

History:

- William Stanley Jevons, 1835-1882: "I have given much attention, therefore, to lessening both the manual and mental labour of the process, and I shall describe several devices which may be adopted for saving trouble and risk of mistake."
- Ernst Schröder, 1841-1902: "Getting a handle on the consequences of any premises, or at least the fastest method for obtaining these consequences, seems to me to be one of the noblest, if not the ultimate goal of mathematics and logic."
- Cook, 1971, Levin, 1973: Boolean Satisfiability is NP-complete.

History:

- William Stanley Jevons, 1835-1882: "I have given much attention, therefore, to lessening both the manual and mental labour of the process, and I shall describe several devices which may be adopted for saving trouble and risk of mistake."
- Ernst Schröder, 1841-1902: "Getting a handle on the consequences of any premises, or at least the fastest method for obtaining these consequences, seems to me to be one of the noblest, if not the ultimate goal of mathematics and logic."
- Cook, 1971, Levin, 1973: Boolean Satisfiability is NP-complete.
- Clay Institute, 2000: $1M Award!

Algorithmic Boolean Reasoning: Early History

- Davis and Putnam, 1958: "Computational Methods in The Propositional calculus", unpublished report to the NSA
- Davis and Putnam, JACM 1960: "A Computing procedure for quantification theory"
- Davis, Logemman, and Loveland, CACM 1962: "A machine program for theorem proving"
- Unit Propagation

Algorithmic Boolean Reasoning: Early History

- Davis and Putnam, 1958: "Computational Methods in The Propositional calculus", unpublished report to the NSA
- Davis and Putnam, JACM 1960: "A Computing procedure for quantification theory"
- Davis, Logemman, and Loveland, CACM 1962: "A machine program for theorem proving"
- Unit Propagation
  $F = (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b)$

  What can we deduce?

Algorithmic Boolean Reasoning: Early History

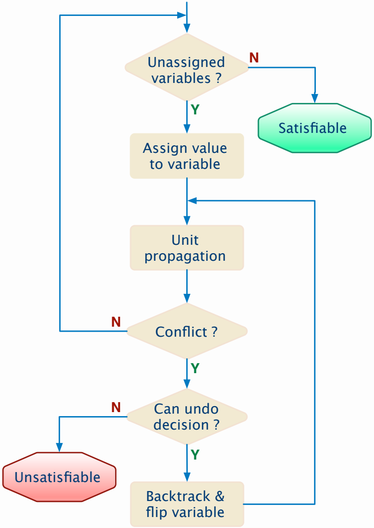
- Davis and Putnam, 1958: "Computational Methods in The Propositional calculus", unpublished report to the NSA
- Davis and Putnam, JACM 1960: "A Computing procedure for quantification theory"
- Davis, Logemman, and Loveland, CACM 1962: "A machine program for theorem proving"
- Unit Propagation
  $F = (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b)$

  What can we deduce?

  $s = 1$

The DPLL algorithm

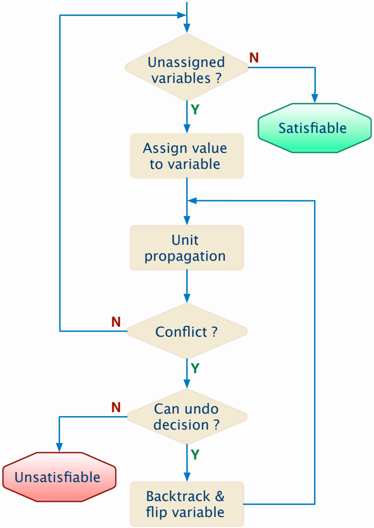
[DL60,DLL62]

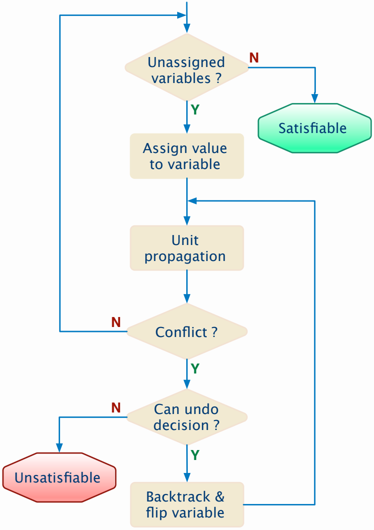


The DPLL algorithm

[DL60,DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

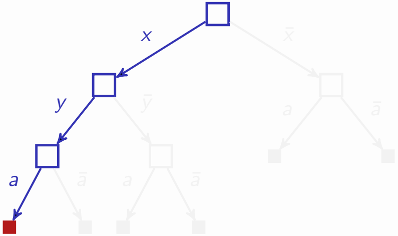




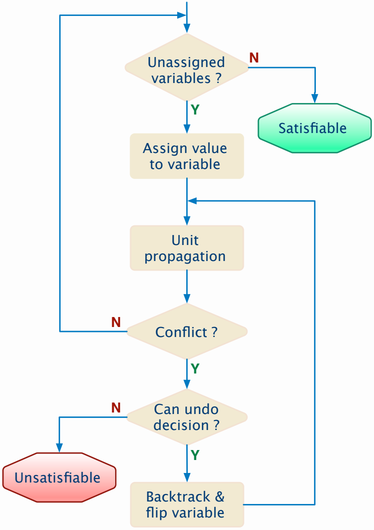
[DL60,DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	a	$\rightarrow b \rightarrow \bot$



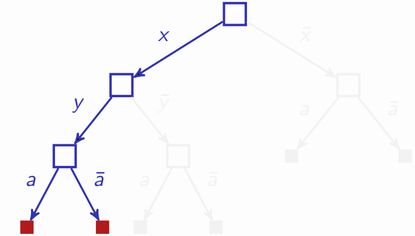
The DPLL algorithm

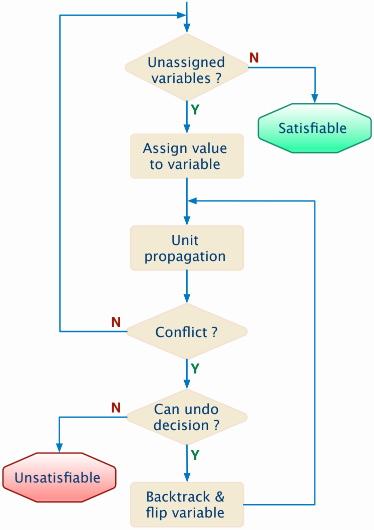


[DL60,DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	$\bar{a}$	$\rightarrow \bar{b} \rightarrow \bot$

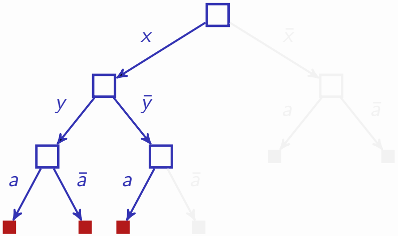




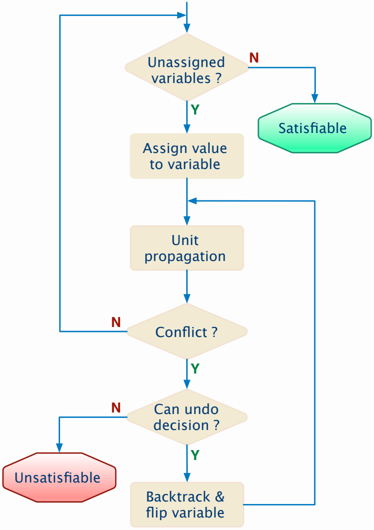
[DL60,DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	a	$\rightarrow b \rightarrow \bot$



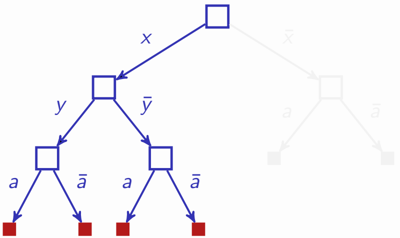
The DPLL algorithm



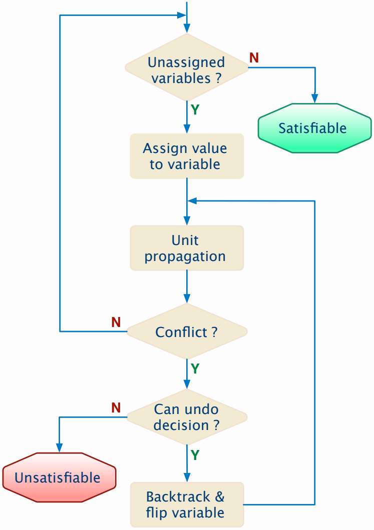
[DL60,DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	$\bar{a}$	$\rightarrow \bar{b} \rightarrow \bot$

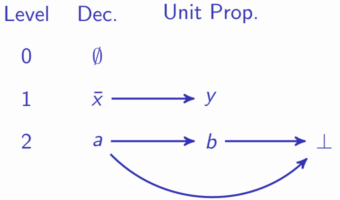


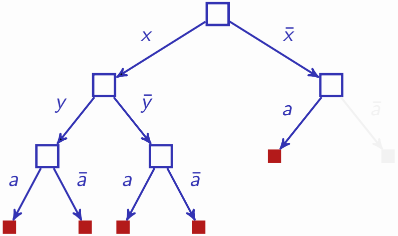
The DPLL algorithm



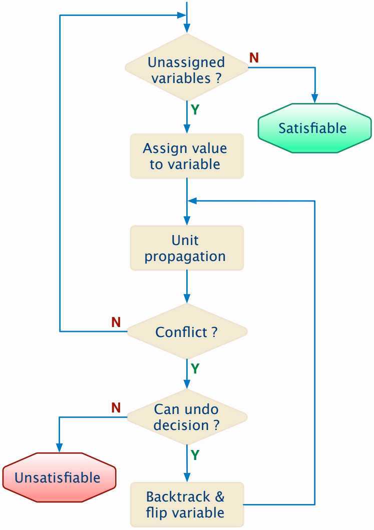
[DL60,DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$



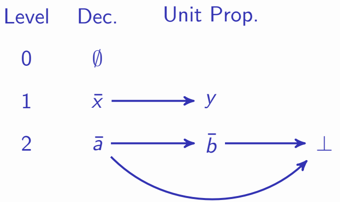


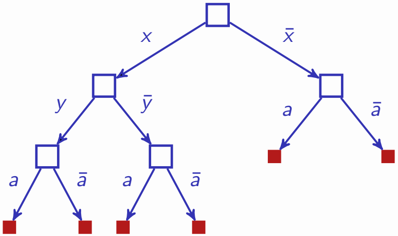
The DPLL algorithm



[DL60,DLL62]

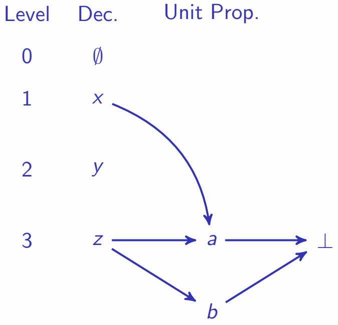
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$





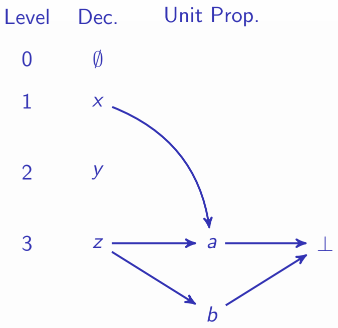
Clause learning

$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$



Clause learning

$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$

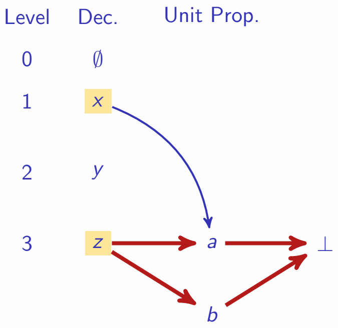


- Analyze conflict

[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]

Clause learning

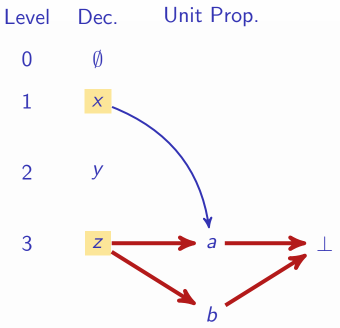
$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$



- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current

Clause learning

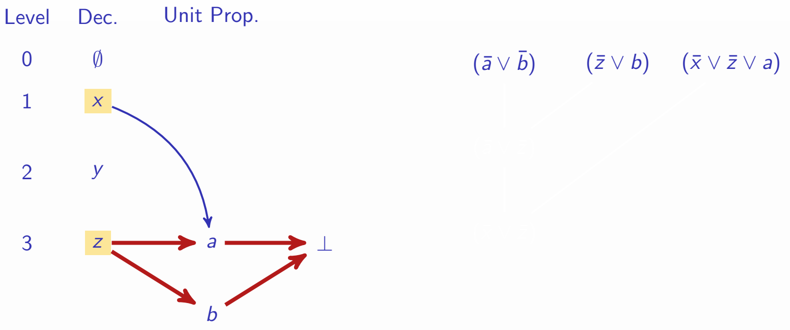
$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$



- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

Clause learning

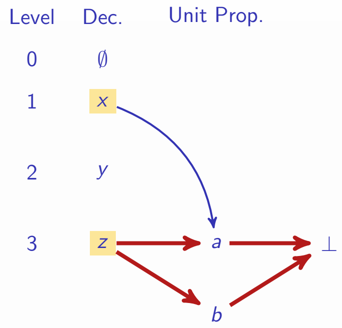
$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$

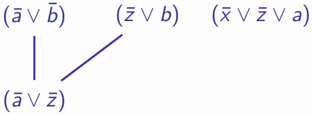


- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

Clause learning

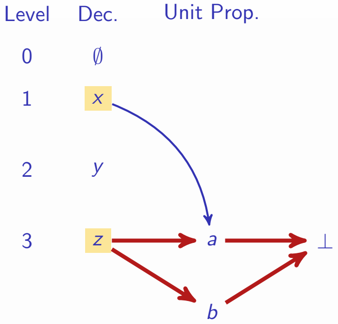
$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$

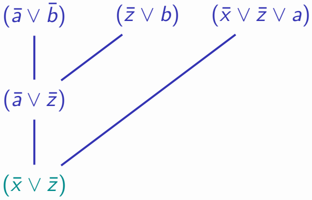




- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$

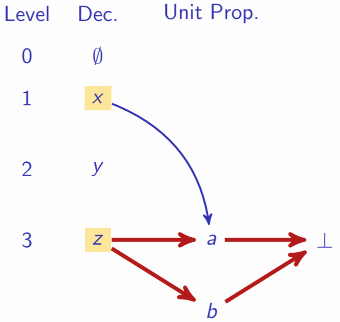


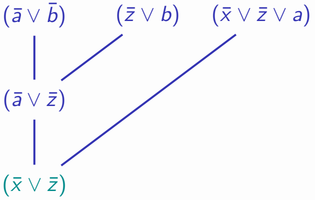


- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

Clause learning

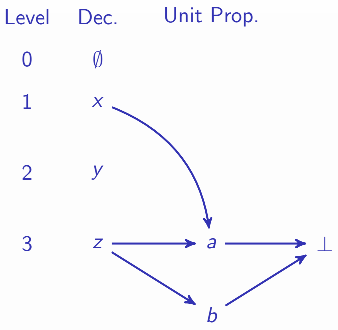
$(\bar{a} \vee \bar{b}) \wedge (\bar{z} \vee b) \wedge (\bar{x} \vee \bar{z} \vee a) \wedge (y \vee b)$



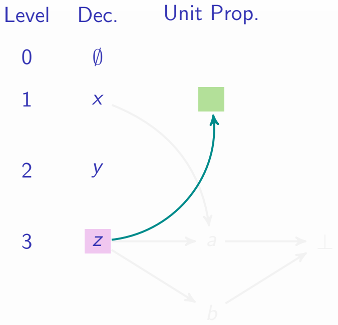


- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Clause learning – after backtracking

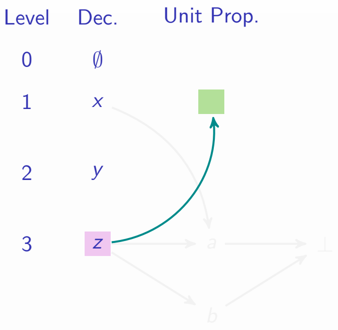


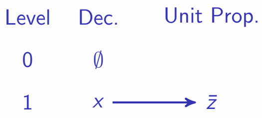
Clause learning – after backtracking



- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

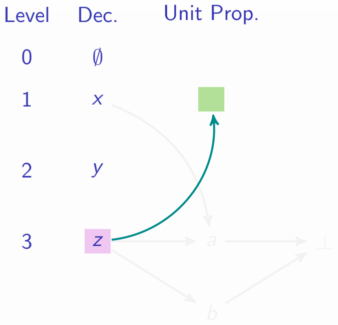
Clause learning – after backtracking

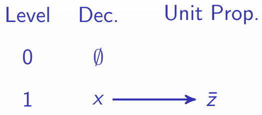




- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

Clause learning – after backtracking





- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are asserting [MSS96a,MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [ZMMM01]

The Modern CDCL SAT Solvers

- Clause learning & non-chronological backtracking [MSS96a,MSS99,BS97,Z97]

- Search restarts [GSK98,BMS00,H07,B08]

- Lazy data structures

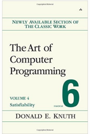
- Conflict-guided branching

- ...

The Modern CDCL SAT Solvers

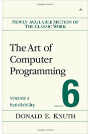
- Clause learning & non-chronological backtracking [MSS96a,MSS99,BS97,Z97]
  - Exploit UIPs [MSS96a,SSS12]
  - Minimize learned clauses [SB09,VG09]
  - Opportunistically delete clauses [MSS96a,MSS99,GN02]
- Search restarts [GSK98,BMS00,H07,B08]
- Lazy data structures
  - Watched literals [MMZZM01]
- Conflict-guided branching
  - Lightweight branching heuristics [MMZZM01]
  - Phase saving [S00,PD07]
- ...

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

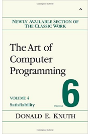
Modern SAT solvers are able to deal routinely with practical problems that involve millions of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)



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

Constrained Counting and Sampling

- 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 $\}$

Constrained Counting and Sampling

- 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 $\}$

- Constrained Counting: Determine $|\mathsf{Sol}(F)|$
- Constrained Sampling: Randomly sample from $\mathsf{Sol}(F)$ such that $\Pr[\text{y is sampled}] = \frac{1}{|\mathsf{Sol}(F)|}$

Constrained Counting and Sampling

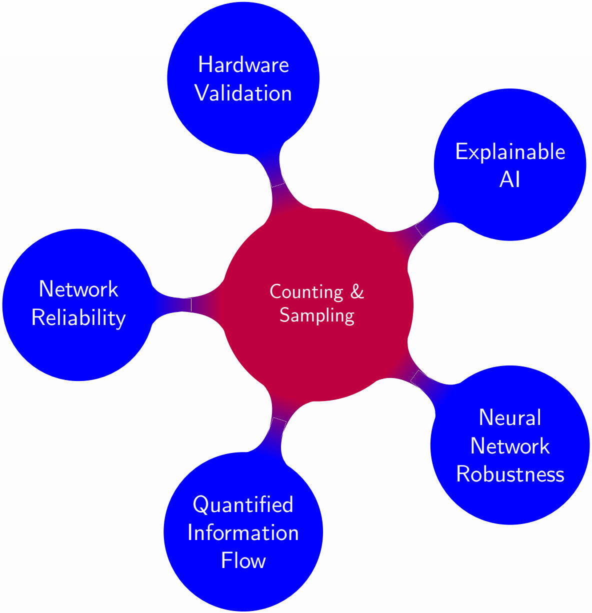
- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
  - Weight Function W: $\{0, 1\}^n \mapsto [0, 1]$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- $W(F) = \Sigma_{y \in \mathsf{Sol}(F)} W(y)$
- Constrained Counting: Determine $W(F)$
- Constrained Sampling: Randomly sample from $\mathsf{Sol}(F)$ such that $\Pr[\text{y is sampled}] = \frac{W(y)}{W(F)}$

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
  - Weight Function W: $\{0,1\}^n \mapsto [0,1]$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- $W(F) = \Sigma_{y \in \mathsf{Sol}(F)} W(y)$
- Constrained Counting: Determine $W(F)$
- Constrained Sampling: Randomly sample from $\mathsf{Sol}(F)$ such that $\Pr[\text{y is sampled}] = \frac{W(y)}{W(F)}$
- Given
  - $F := (X_1 \vee X_2)$
  - $W[(0,0)] = W[(1,1)] = \frac{1}{6}; W[(1,0)] = W[(0,1)] = \frac{1}{3}$
- $\mathsf{Sol}(F) = \{(0,1),(1,0),(1,1)\}$

Constrained Counting and Sampling

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
  - Weight Function W: $\{0,1\}^n \mapsto [0,1]$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- $W(F) = \Sigma_{y \in \mathsf{Sol}(F)} W(y)$
- Constrained Counting: Determine $W(F)$
- Constrained Sampling: Randomly sample from $\mathsf{Sol}(F)$ such that $\Pr[\text{y is sampled}] = \frac{W(y)}{W(F)}$
- Given
  - $F := (X_1 \vee X_2)$
  - $W[(0,0)] = W[(1,1)] = \frac{1}{6}; W[(1,0)] = W[(0,1)] = \frac{1}{3}$
- $\mathsf{Sol}(F) = \{(0,1),(1,0),(1,1)\}$
- $W(F) = \frac{1}{3} + \frac{1}{3} + \frac{1}{6} = \frac{5}{6}$

Applications across Computer Science



Today's Menu

Testing of AI systems

Network Reliability

Hardware Validation

Today's Menu

Testing of AI systems

Network Reliability   Constrained Counting

Hardware Validation

Today's Menu

Testing of AI systems

Network Reliability   Constrained Counting   Hashing Framework

Hardware Validation

Today's Menu

Testing of AI systems		
Network Reliability	Constrained Counting	Hashing Framework
Hardware Validation	Constrained Sampling	

Testing of AI Systems

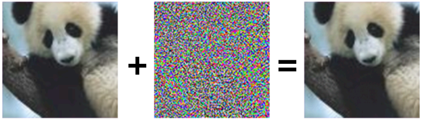
- Classical verification/testing setup for traditional systems
  - System captured as a model $M(\mathcal{I}, \mathcal{O})$ via logical constraints
  - Specification $\varphi(\mathcal{I}, \mathcal{O})$: relationship between input and output
  - Methodology: Find one execution of M such that φ is not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

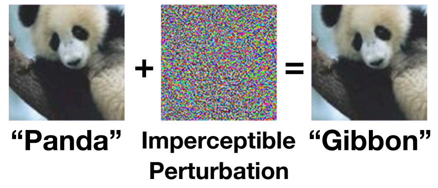
Testing of AI Systems

- 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: 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: 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: From satisfiability to counting

(BSSMS, 2019)

Can we reliably predict the effect of natural disasters on critical infrastructure such as power grids?

Can we reliably predict the effect of natural disasters on critical infrastructure such as power grids?

Can we predict likelihood of a region facing blackout?

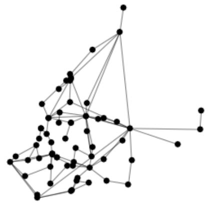


Figure: Plantersville, SC

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

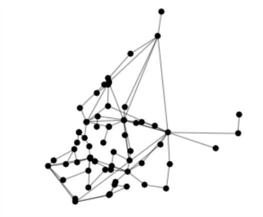


Figure: Plantersville, SC

- $G = (V, E)$; source node: s and terminal node t
- failure probability $g : E \rightarrow [0, 1]$
- 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

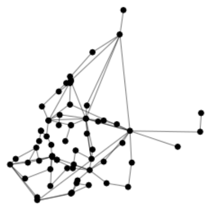


Figure: Plantersville, SC

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

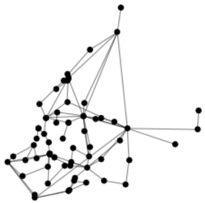


Figure: Plantersville, SC

- $G = (V, E)$; 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

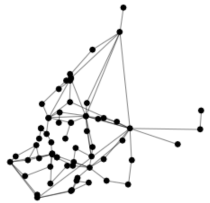


Figure: Plantersville, SC

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)

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?

Constrained Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
  - Weight Function W: $\{0,1\}^n \mapsto [0,1]$
- ExactCount(F, W): Compute $W(F)$?
  - #P-complete (Valiant 1979)

Constrained Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
  - Weight Function W: $\{0,1\}^n \mapsto [0,1]$
- ExactCount(F, W): Compute $W(F)$?
  - #P-complete (Valiant 1979)
- ApproxCount$(F, W, \varepsilon, \delta)$: Compute C such that

$$\Pr[\frac{W(F)}{1+\varepsilon} \leq C \leq W(F)(1+\varepsilon)] \geq 1-\delta$$

From Weighted to Unweighted Counting

Boolean Formula F and weight function $W : \{0,1\}^n \to \mathbb{Q}^{\geq 0}$

Boolean Formula F'

$$W(F) = c(W) \times |\mathsf{Sol}(F')|$$

- Key Idea: Encode weight function as a set of constraints

From Weighted to Unweighted Counting

Boolean Formula F and weight function $W : \{0,1\}^n \rightarrow \mathbb{Q}^{\geq 0}$

Boolean Formula F'

$$W(F) = c(W) \times |\mathsf{Sol}(F')|$$

- Key Idea: Encode weight function as a set of constraints
- Caveat: $|F'| = O(|F| + |W|)$

(CFMV, IJCAI15)

From Weighted to Unweighted Counting

Boolean Formula F and weight function $W : \{0,1\}^n \to \mathbb{Q}^{\geq 0}$

Boolean Formula F'

$$W(F) = c(W) \times |\mathsf{Sol}(F')|$$

- Key Idea: Encode weight function as a set of constraints
- Caveat: $|F'| = O(|F| + |W|)$

(CFMV, IJCAI15)

How do we estimate $|\mathsf{Sol}(F')|$?

Counting in Beijing

How many people in Beijing like coffee?

- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)

Counting in Beijing

How many people in Beijing like coffee?

- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 21.5M/50

Counting in Beijing

How many people in Beijing like coffee?

- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 21.5M/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 Beijing

How many people in Beijing like coffee?

- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 21.5M/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 Beijing

How many people in Beijing like coffee?

- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 21.5M/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 Beijing

How many people in Beijing like coffee?

- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 21.5M/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

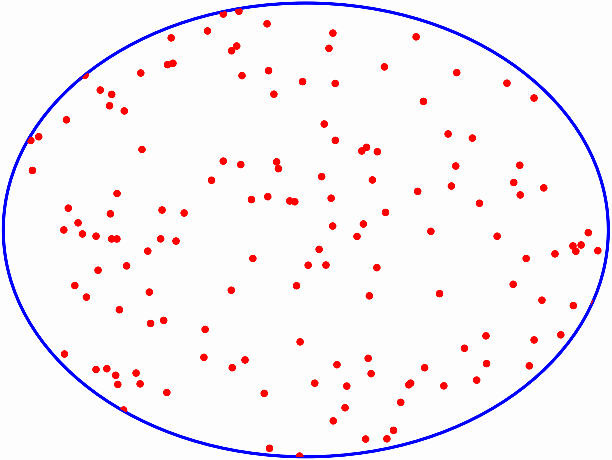
Counting in Beijing

How many people in Beijing like coffee?

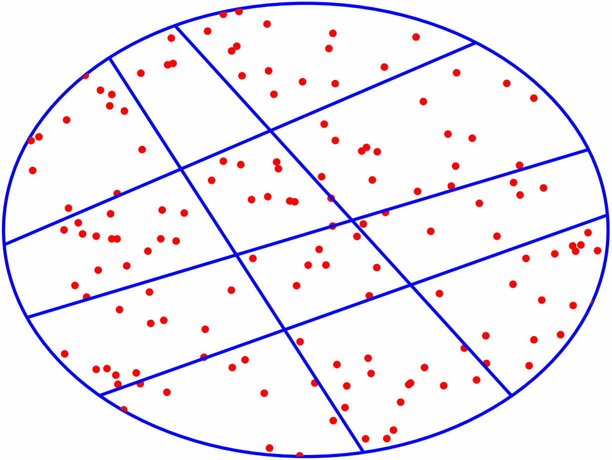
- Population of Beijing = 21.5M
- Assign every person a unique ($n =$) 25 bit identifier ($2^n = 21.5$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 21.5M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample 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 $\mathcal{O}(\log n)$?

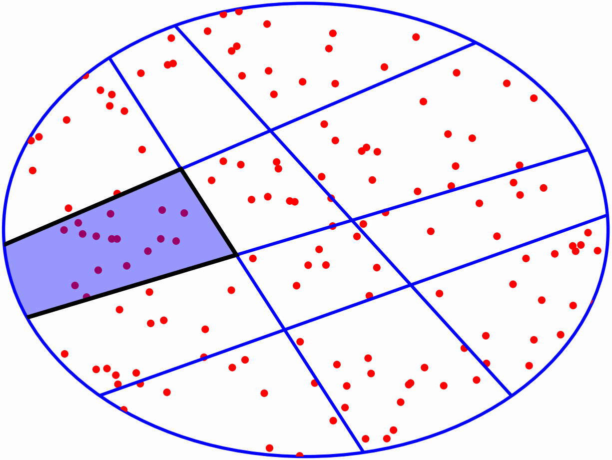
As Simple as Counting Dots



As Simple as Counting Dots



Pick a random cell



Estimate = Number of solutions in a cell × 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

Universal Hashing (Carter and Wegman 1977)

2-Universal Hashing

- Let H be family of 2-universal 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-Universal Hashing

- Let H be family of 2-universal 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-universality
  - 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-Universal 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-Universal 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

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

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

2-Universal 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 degrade with increase in the size of XORs (SAT Solvers != SAT oracles)

Improved Universal 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 Universal 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 Universal 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 $\forall \sigma_1, \sigma_2 \in \mathrm{Sol}(F)$, 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 Universal 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 Universal 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 Universal 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)

Challenges

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

- Independent Support-based 2-Universal Hash Functions

Challenge 2 How many cells?

Question 2: How many cells?

- A cell is small if it has about $\mathrm{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ solutions

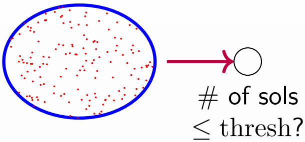
Question 2: How many cells?

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

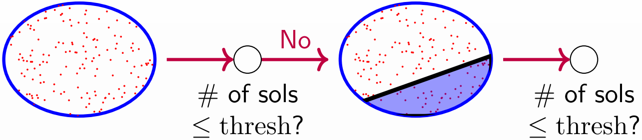
Question 2: How many cells?

- A cell is small if it has about $\mathrm{thresh} = 5(1 + \frac{1}{\varepsilon})^2$ 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 $\leq \mathrm{thresh}$

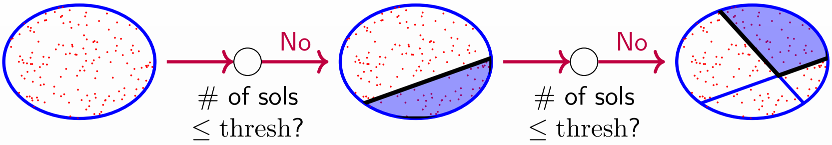
ApproxMC(F, ε, δ)



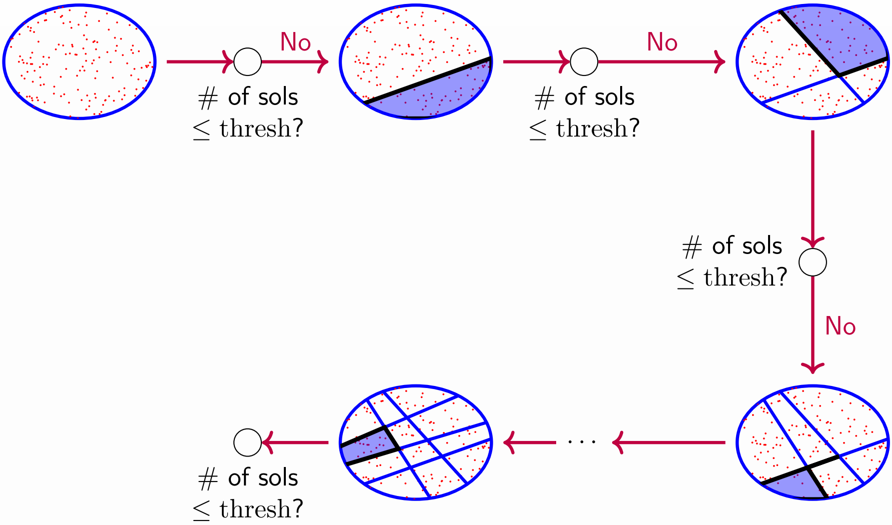
ApproxMC(F, ε, δ)



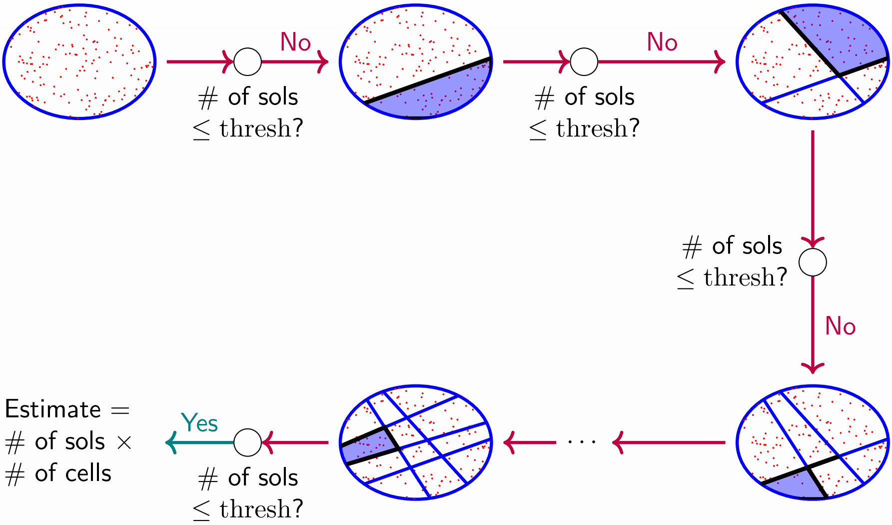
ApproxMC(F, ε, δ)



ApproxMC(F, ε, δ)



ApproxMC(F, ε, δ)



ApproxMC(F, ε, δ)

- 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(F, ε, δ)

- 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 $\#(F \wedge Q_1 \wedge Q_2) \leq \text{thresh}$
  - $\cdots$
  - 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: $\mathcal{O}(\log n)$)

ApproxMC(F, ε, δ)

- 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 $\#(F \wedge Q_1 \wedge Q_2) \leq \text{thresh}$
  - ...
  - 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: $\mathcal{O}(\log n)$)
- **Will this work? Will the "m" where we stop be close to m^*?**

ApproxMC(F, ε, δ)

- 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: $\mathcal{O}(\log n)$)
- **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(F, ε, δ)

- 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: $\mathcal{O}(\log n)$)
- **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 (F, ε, δ) *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*

ApproxMC(F, ε, δ)

Theorem (Correctness)

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

Theorem (Complexity)

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

- *Prior work required $\mathcal{O}(\frac{\boldsymbol{n} \log \boldsymbol{n} \log(\frac{1}{\delta})}{\varepsilon})$ calls to SAT oracle (Stockmeyer 1983)*

ApproxMC(F, ε, δ)

Theorem (Correctness)

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

Theorem (Complexity)

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

- *Prior work required $\mathcal{O}(\frac{\boldsymbol{n} \log \boldsymbol{n} \log(\frac{1}{\delta})}{\varepsilon})$ calls to SAT oracle (Stockmeyer 1983)*

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

If F is a DNF formula, then ApproxMC is FPRAS – fundamentally different from the only other known FPRAS for DNF (Karp, Luby 1983)

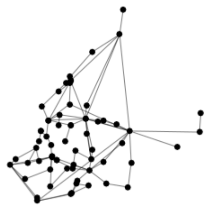
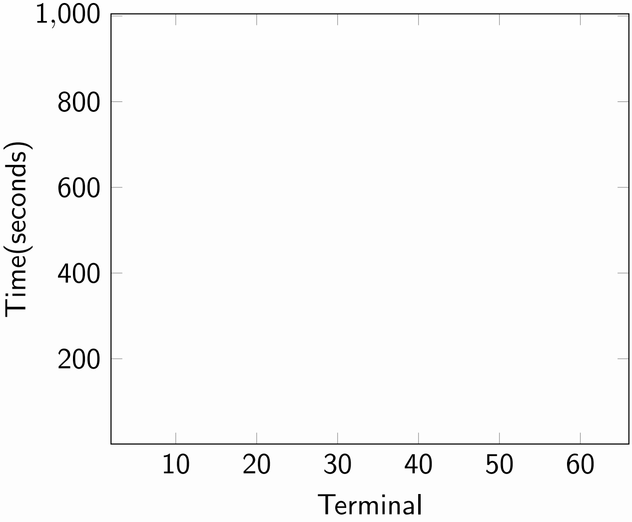


Figure: Plantersville, SC



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

Timeout = 1000 seconds

(DMPV, AAAI17)

Reliability of Critical Infrastructure Networks

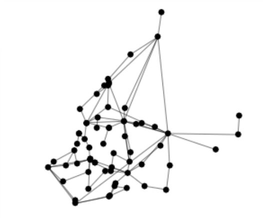
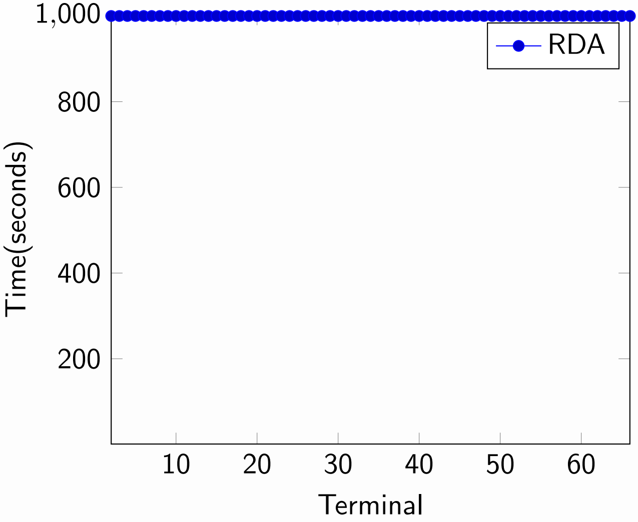


Figure: Plantersville, SC

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

Timeout = 1000 seconds



(DMPV, AAAI17)

Reliability of Critical Infrastructure Networks

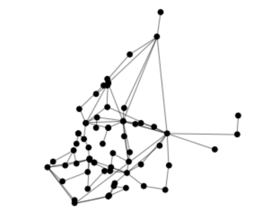
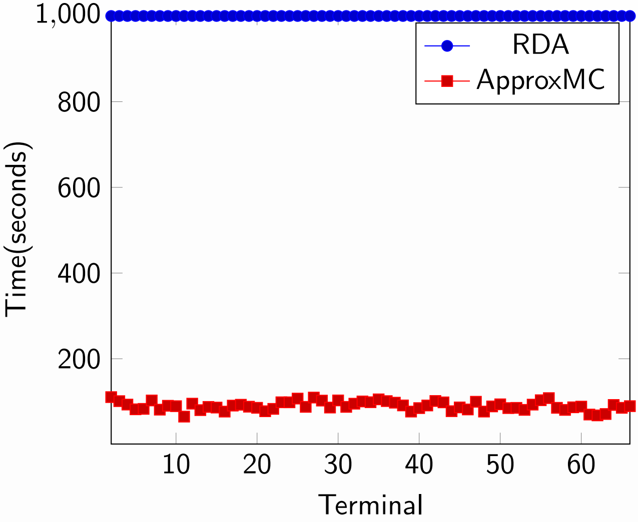


Figure: Plantersville, SC

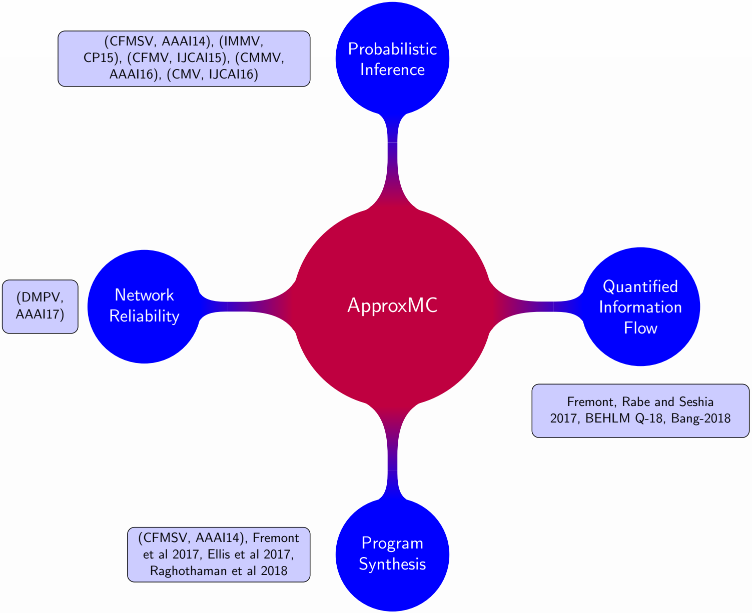
- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

Timeout = 1000 seconds



(DMPV, AAAI17)

Beyond Network Reliability



Network Reliability

Probabilistic Inference

Constrained Counting

Network Reliability

Probabilistic Inference

Constrained Counting

Hashing Framework

Network Reliability

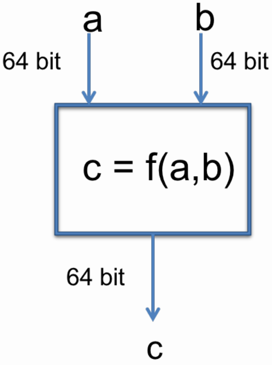
Probabilistic Inference

Hardware Validation

Constrained Counting

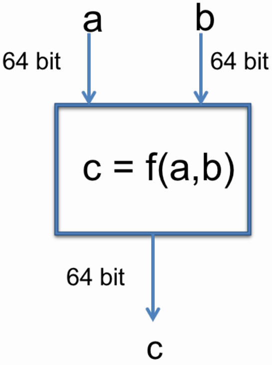
Hashing Framework

Hardware Validation



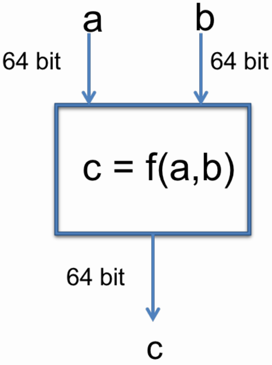
- Design is simulated with test vectors (values of *a* and *b*)
- Results from simulation compared to intended results

Hardware Validation

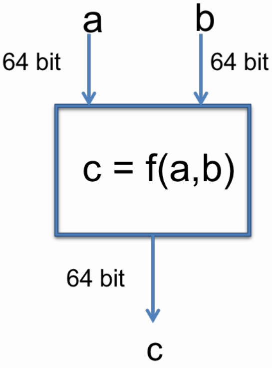


- Design is simulated with test vectors (values of *a* and *b*)
- Results from simulation compared to intended results
- Challenge: How do we generate test vectors?
  - 2^{128} combinations for a toy circuit

Hardware Validation



- Design is simulated with test vectors (values of a and b)
- Results from simulation compared to intended results
- Challenge: How do we generate test vectors?
  - 2^{128} combinations for a toy circuit
- Use constraints to represent *interesting* verification scenarios



Constraints

- Designers:
  - $a +_{64} 11 * 32b = 12$
  - $a <_{64} (b >> 4)$
- Past Experience:
  - $40 <_{64} 34 + a <_{64} 5050$
  - $120 <_{64} b <_{64} 230$
- Users:
  - $232 * 32a +_{64} b! = 1100$
  - $1020 <_{64} (b/_{64}2) +_{64} a <_{64} 2200$

Test vectors: random solutions of constraints

Constrained Sampling

- Given:
  - Set of Constraints F over variables $X_1, X_2, \cdots X_n$
- Uniform Sampler

$$\forall y \in \mathsf{Sol}(F), \mathsf{Pr}[\text{y is output}] = \frac{1}{|\mathsf{Sol}(F)|}$$

- Almost-Uniform Sampler

$$\forall y \in \mathsf{Sol}(F), \frac{1}{(1+\varepsilon)|\mathsf{Sol}(F)|} \leq \mathsf{Pr}[\text{y is output}] \leq \frac{(1+\varepsilon)}{|\mathsf{Sol}(F)|}$$

Prior Work

Strong guarantees but poor scalability

- Polynomial calls to NP oracle (Bellare, Goldreich and Petrank,2000)
- BDD-based techniques (Yuan et al 1999, Yuan et al 2004, Kukula and Shiple 2000)
- Reduction to approximate counting (Jerrum, Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Randomization in SAT solvers (Moskewicz 2001, Nadel 2011, Dutra Bachrach and Sen 2018)
- MCMC-based approaches (Sinclair 1993, Jerrum and Sinclair 1996, Kitchen and Kuehlmann 2007,...)
- Belief Networks (Dechter 2002, Gogate and Dechter 2006)

Prior Work

Strong guarantees but poor scalability

- Polynomial calls to NP oracle (Bellare, Goldreich and Petrank,2000)
- BDD-based techniques (Yuan et al 1999, Yuan et al 2004, Kukula and Shiple 2000)
- Reduction to approximate counting (Jerrum, Valiant and Vazirani 1986)

Weak guarantees but impressive scalability

- Randomization in SAT solvers (Moskewicz 2001, Nadel 2011, Dutra Bachrach and Sen 2018)
- MCMC-based approaches (Sinclair 1993, Jerrum and Sinclair 1996, Kitchen and Kuehlmann 2007,...)
- Belief Networks (Dechter 2002, Gogate and Dechter 2006)

How to bridge this gap between theory and practice?

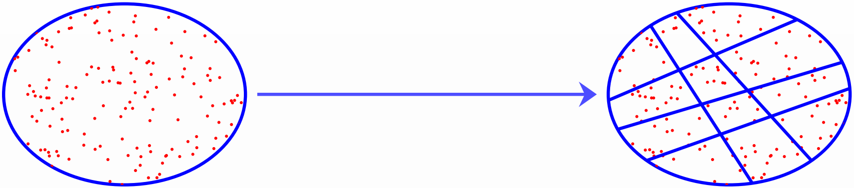
Close Cousins: Counting and Sampling

- Approximate counting and almost-uniform sampling are inter-reducible (Jerrum, Valiant and Vazirani, 1986)

Close Cousins: Counting and Sampling

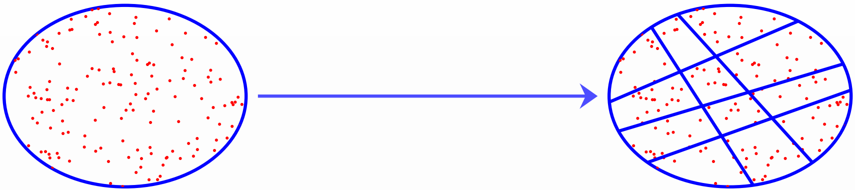
- Approximate counting and almost-uniform sampling are inter-reducible (Jerrum, Valiant and Vazirani, 1986)
- Is the reduction efficient?
  - Almost-uniform sampler (JVV) require linear number of approximate counting calls

Key Ideas



- Check if a randomly picked cell is *small*
  - If yes, pick a solution randomly from randomly picked cell

Key Ideas



- Check if a randomly picked cell is *small*
  - If yes, pick a solution randomly from randomly picked cell

Challenge: How many cells?

How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$ ($m^* = \log \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$)

How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$ ($m^* = \log \frac{|\mathsf{Sol}(F)|}{\mathrm{thresh}}$)
  - ApproxMC(F, ε, δ) returns C such that
  $$\Pr\left[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \le C \le |\mathsf{Sol}(F)|(1+\varepsilon)\right] \ge 1 - \delta$$
  - $\tilde{m} = \log \frac{C}{\mathrm{thresh}}$

How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$ ($m^* = \log \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$)
  - ApproxMC(F, ε, δ) returns C such that

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

  - $\tilde{m} = \log \frac{C}{\text{thresh}}$
  - Check for $m = \tilde{m} - 1, \tilde{m}, \tilde{m} + 1$ if a randomly chosen cell is *small*

How many cells?

- Desired Number of cells: $2^{m^*} = \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$ ($m^* = \log \frac{|\mathsf{Sol}(F)|}{\text{thresh}}$)
  - ApproxMC(F, ε, δ) returns C such that
  $$\Pr\left[\frac{|\mathsf{Sol}(F)|}{1+\varepsilon} \leq C \leq |\mathsf{Sol}(F)|(1+\varepsilon)\right] \geq 1 - \delta$$
  - $\tilde{m} = \log \frac{C}{\text{thresh}}$
  - Check for $m = \tilde{m} - 1, \tilde{m}, \tilde{m} + 1$ if a randomly chosen cell is *small*
  - Not just a practical hack required non-trivial proof

(CMV; DAC14),

(CFMSV; AAAI14, TACAS15),

(SGRM; LPAR18,TACAS19)

Theoretical Guarantees

Theorem (Almost-Uniformity)

$$\forall y \in \mathsf{Sol}(F),\ \frac{1}{(1+\varepsilon)|\mathsf{Sol}(F)|} \leq \Pr[y \textit{ is output}] \leq \frac{1+\varepsilon}{|\mathsf{Sol}(F)|}$$

Theoretical Guarantees

Theorem (Almost-Uniformity)

$$\forall y \in \mathsf{Sol}(F),\ \frac{1}{(1+\varepsilon)|\mathsf{Sol}(F)|} \leq \Pr[y \textit{ is output}] \leq \frac{1+\varepsilon}{|\mathsf{Sol}(F)|}$$

Theorem (Query)

For a formula F over n variables UniGen makes **one call** *to approximate counter*

Theoretical Guarantees

Theorem (Almost-Uniformity)

$$\forall y \in \mathsf{Sol}(F),\ \frac{1}{(1+\varepsilon)|\mathsf{Sol}(F)|} \leq \Pr[y \textit{ is output}] \leq \frac{1+\varepsilon}{|\mathsf{Sol}(F)|}$$

Theorem (Query)

For a formula F over n variables UniGen makes **one call** *to approximate counter*

- *Prior work required* **n** *calls to approximate counter (Jerrum, Valiant and Vazirani, 1986)*

	Relative Runtime
SAT Solver	1
Desired Uniform Generator	10

Experiments over 200+ benchmarks

	Relative Runtime
SAT Solver	1
Desired Uniform Generator	10
XORSample (2012 state of the art)	50000

Experiments over 200+ benchmarks

Three Orders of Improvement

	Relative Runtime
SAT Solver	1
Desired Uniform Generator	10
XORSample (2012 state of the art)	50000
UniGen	21

Experiments over 200+ benchmarks

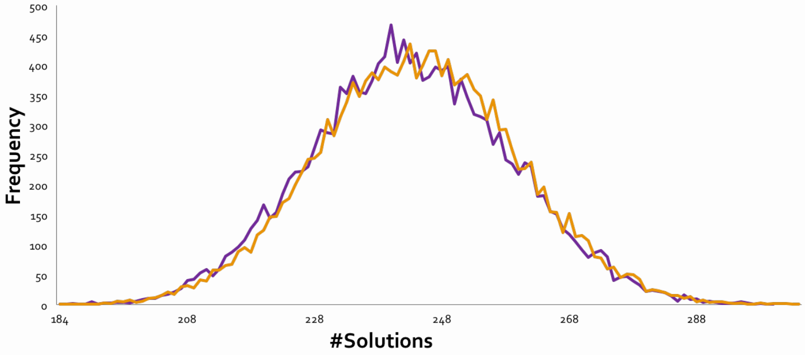
Three Orders of Improvement

	Relative Runtime
SAT Solver	1
Desired Uniform Generator	10
XORSample (2012 state of the art)	50000
UniGen	21

Experiments over 200+ benchmarks

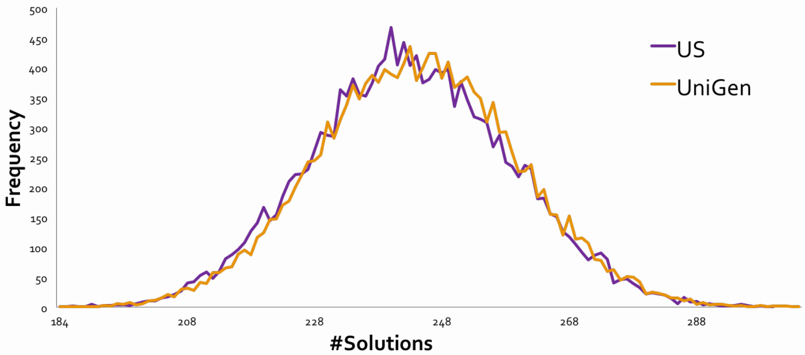
Closer to technical transfer

Quiz Time: Uniformity



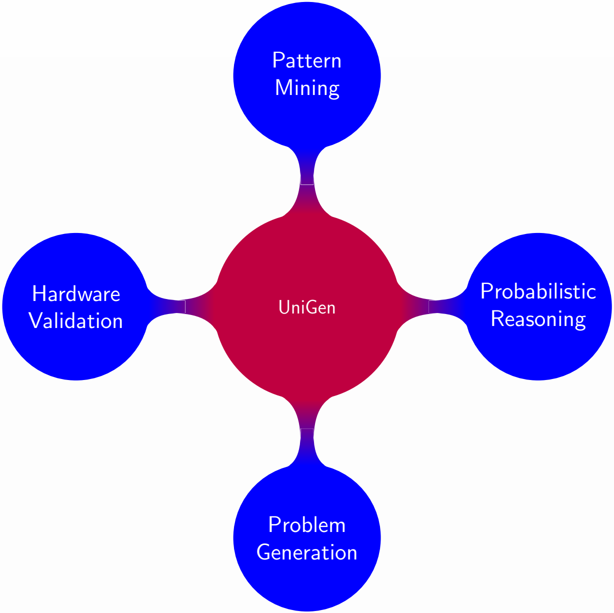
- Benchmark: case110.cnf; #var: 287; #clauses: 1263
- Total Runs: 4×10^6; Total Solutions : 16384

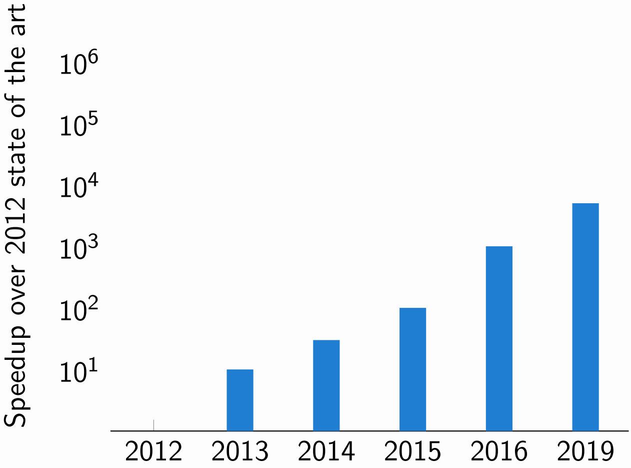
Statistically Indistinguishable



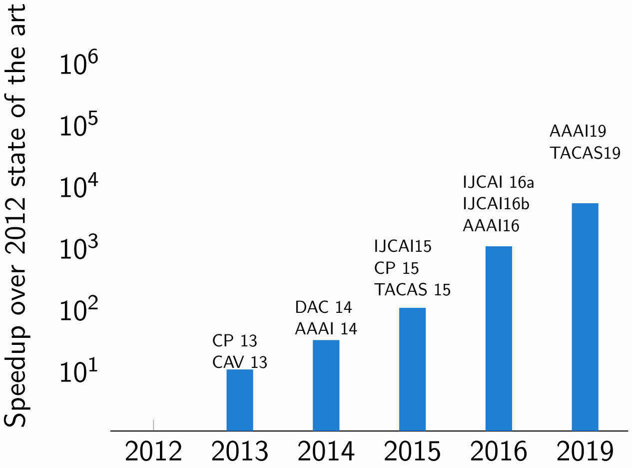
- Benchmark: case110.cnf; #var: 287; #clauses: 1263
- Total Runs: 4×10^6; Total Solutions : 16384

Usages of Open Source Tool: UniGen

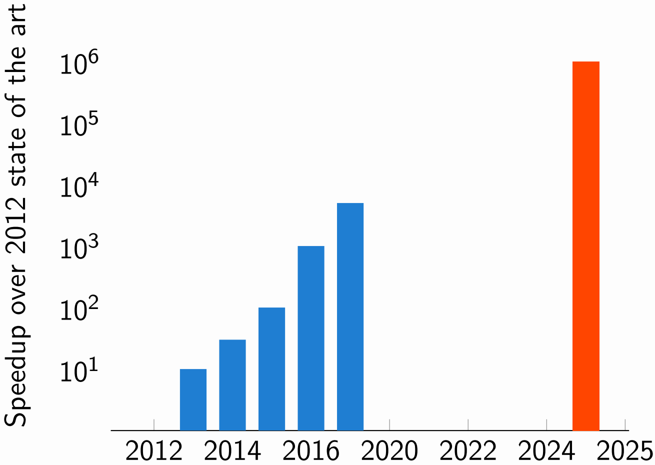



Speedup over 2012 state of the art
10^6
10^5
10^4
10^3
10^2
10^1
2012
2013
2014
2015
2016
2019

Mission 2025: Constrained Counting and Sampling Revolution



Mission 2025: Constrained Counting and Sampling Revolution



Requires combinations of ideas from theory, statistics and systems

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Neural Networks Handling 100K neurons

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Neural Networks Handling 100K neurons

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

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Neural Networks Handling 100K neurons

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

Hardware Verification Handling SMT formulas with 10K nodes

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Neural Networks Handling 100K neurons

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

Hardware Verification Handling SMT formulas with 10K nodes

The Potential of Hashing-based Framework

Machine Learning Probabilistic programming

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering: Reliability for Los Angeles Transmission Grid

Neural Networks: Handling 100K neurons

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

Hardware Verification: Handling SMT formulas with 10K nodes

The Potential of Hashing-based Framework

Machine Learning: Probabilistic programming

Theory: Classification of Approximate Counting Complexity

Mission 2025: Constrained Counting and Sampling Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Neural Networks Handling 100K neurons

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

Hardware Verification Handling SMT formulas with 10K nodes

The Potential of Hashing-based Framework

Machine Learning Probabilistic programming

Theory Classification of Approximate Counting Complexity

Databases Streaming algorithms

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)
- Handling weighted distributions: Connections to theory of integration

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)
- Handling weighted distributions: Connections to theory of integration
- Verification of sampling and counting (CM, AAAI19)

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)
- Handling weighted distributions: Connections to theory of integration
- Verification of sampling and counting (CM, AAAI19)
- Designing hardware accelerators – similar to advances in deep learning

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)
- Handling weighted distributions: Connections to theory of integration
- Verification of sampling and counting (CM, AAAI19)
- Designing hardware accelerators – similar to advances in deep learning

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)
- Handling weighted distributions: Connections to theory of integration
- Verification of sampling and counting (CM, AAAI19)
- Designing hardware accelerators – similar to advances in deep learning
- Understanding and applying sampling and counting to real world use-cases

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms (SM, AAAI19)
- Handling weighted distributions: Connections to theory of integration
- Verification of sampling and counting (CM, AAAI19)
- Designing hardware accelerators – similar to advances in deep learning
- Understanding and applying sampling and counting to real world use-cases

We can only see a short distance ahead but we can see plenty there that needs to be done (Turing, 1950)