The Second Coming of Logic in Artificial Intelligence

Kuldeep S. Meel

National University of Singapore

Acknowledgments to Moshe Y. Vardi for some of the slides.
I have serious allergy from electronic devices other than my own laptop.
So please turn off your devices.

Artificial Intelligence and Logic

Turing, 1950: "Opinions may vary as to the complexity which is suitable in the child machine. One might try to make it as simple as possible consistent with the general principles. Alternatively one might have a complete system of logical inference built in. In the latter case the store would be largely occupied with definitions and propositions. The propositions would have various kinds of status, e.g., well-established facts, conjectures, mathematically proved theorems, statements given by an authority, expressions having the logical form of proposition but not a belief-value"

Artificial Intelligence and Logic

- Newell, Shaw, and Simon, 1955: "Logic Theorist"

– Solved 38 of the first 52 theorems in Whitehead and Russell's Principia Mathematica

- McCarthy, 1958: "Programming with Common Sense "

- Shapiro, 1982: "Algorithmic Program Debugging"

- Hayes-Roth, Waterman, and DB Lenat , 1958: "Building Expert System"

Need tools to reason with logic

Aristotle's Syllogisms

- All men are mortal
- Socrates is a man

Socrates is a mortal

Boole's Symbolic Logic

Boole's insight: Aristotle's syllogisms are about *classes* of objects, which can be treated *algebraically*.

> "If an adjective, as 'good', is employed as a term of description, let us represent by a letter, as y, all things to which the description 'good' is applicable, i.e., 'all good things', or the class of 'good things'. Let it further be agreed that by the combination xy shall be represented that class of things to which the name or description represented by x and y are simultaneously applicable. Thus, if x alone stands for 'white' things and y for 'sheep', let xy stand for 'white sheep'.

Boolean Satisfiability

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

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.

P *vs.* NP: An Outstanding Open Problem

Does $P = NP$?

- *The* major open problem in theoretical computer science
- A major open problem in mathematics
  - A Clay Institute Millennium Problem
  - Million dollar prize!

What is this about? It is about computational complexity – how hard it is to solve computational problems.

Computational Problems

Example: *Graph* – $G = (V, E)$

- V – set of nodes
- E – set of edges

Two notions:

- Hamiltonian Cycle: a cycle that visits every *node* exactly once.
- Eulerian Cycle: a cycle that visits every *edge* exactly once.

Question: How hard it is to find a Hamiltonian cycle? Eulerian cycle?

Computational Complexity

Measuring complexity: How many (Turing machine) operations does it take to solve a problem of size n?

- *Size* of (V, E): number of nodes plus number of edges.

Complexity Class P: problems that can be solved in *polynomial time* – n^c for a *fixed* c

Examples:

- Is a number even?
- Is a number square?
- Does a graph have an Eulerian cycle?

What about the Hamiltonian Cycle Problem?

Hamiltonian Cycle

- **Naive Algorithm**: Exhaustive search – run time is $n!$ operations
- **"Smart" Algorithm**: Dynamic programming – run time is 2^n operations

Note: The universe is much younger than 2^{200} Planck time units!

Fundamental Question: Can we do better?

- Is Hamiltonian Cycle in P?

Checking Is Easy!

Observation: Checking if a *given* cycle is a Hamiltonian cycle of a graph $G = (V, E)$ is *easy*!

Complexity Class NP: problems where solutions can be *checked* in polynomial time.

Examples:

- HamiltonianCycle
- Factoring numbers

Significance: Tens of thousands of optimization problems are in NP!!!

- CAD, flight scheduling, chip layout, protein folding, ...

P vs. NP

- P: efficient *discovery* of solutions
- NP: efficient *checking* of solutions

The Big Question: Is $P = NP$ or $P \neq NP$?

- Is *checking* really easier than *discovering*?

Intuitive Answer: Of course, *checking* is easier than *discovering*, so $P \neq NP$!!!

- **Metaphor**: finding a needle in a haystack
- **Metaphor**: Sudoku
- **Metaphor**: mathematical proofs

Alas: We do not know how to *prove* that $P \neq NP$.

$$P \neq NP$$

Consequences:

- Cannot solve efficiently numerous important problems
- RSA encryption may be safe.

Question: Why is it so important to *prove* $P \neq NP$, if that is what is commonly believed?

Answer:

- If we cannot prove it, we do not really understand it.
- May be $P = NP$ and the "enemy" proved it and broke RSA!

$P = NP$

S. Aaronson, MIT: "If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in 'creative leaps,' no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss."

Consequences:

- Can solve efficiently numerous important problems.
- RSA encryption is not safe.

Question: Is it really possible that $P = NP$?

Answer: Yes! It'd require discovering a very clever algorithm, but it took 40 years to prove that LinearProgramming is in P.

Sharpening The Problem

NP-**Complete Problems**: hardest problems is NP

- HamilatonianCycle is NP-complete! [Karp, 1972]

Corollary: $P = NP$ if and only if HamiltonianCycle is in P

There are *thousands* of NP-complete problems. To resolve the $P = NP$ question, it'd suffice to prove that *one* of them is or is not in P.

History

- 1950-60s: Futile effort to show hardness of search problems.
- Stephen Cook, 1971: Boolean Satisfiability is NP-complete.
- Richard Karp, 1972: 20 additional NP-complete problems– 0-1 Integer Programming, Clique, Set Packing, Vertex Cover, Set Covering, Hamiltonian Cycle, Graph Coloring, Exact Cover, Hitting Set, Steiner Tree, Knapsack, Job Scheduling, ...
  – *All* NP-complete problems are polynomially equivalent!
- Leonid Levin, 1973 (independently): Six NP-complete problems
- M. Garey and D. Johnson, 1979: "Computers and Intractability: A Guide to NP-Completeness" - hundreds of NP-complete problems!
- Clay Institute, 2000: $1M Award!

Artificial Intelligence and Logic

- Newell, Shaw, and Simon, 1955: "Logic Theorist"

– Solved 38 of the first 52 theorems in Whitehead and Russell's Principia Mathematica

- McCarthy, 1958: "Programming with Common Sense "

- Shapiro, 1982: "Algorithmic Program Debugging"

- Hayes-Roth, Waterman, and DB Lenat , 1958: "Building Expert System"

Need tools to reason with logic

Reasoning with logic is intractable; death of Logical AI!

And
Logic strikes back!

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"

DPLL Method: Propositional Satisfiability Test

- Convert formula to conjunctive normal form (CNF)
- Backtracking search for satisfying truth assignment
- Unit-clause preference

Modern SAT Solving

CDCL = conflict-driven clause learning

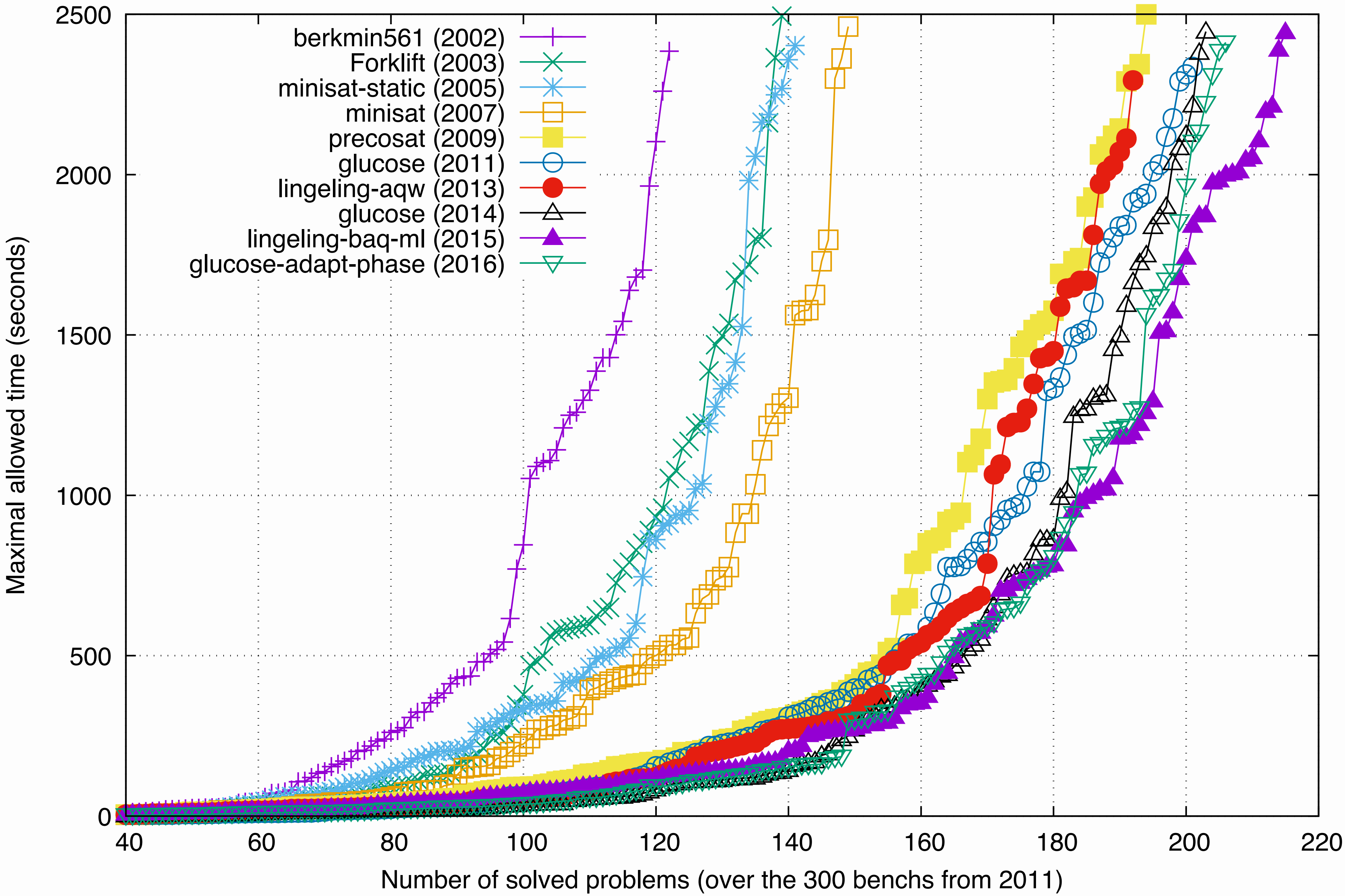
- Backjumping
- Smart unit-clause preference
- Conflict-driven clause learning
- Smart choice heuristic (brainiac vs speed demon)
- Restarts

Key Tools: GRASP, 1996; Chaff, 2001

Current capacity: *millions* of variables

CDCL SAT solver improvement

[Source: Simon 2015]

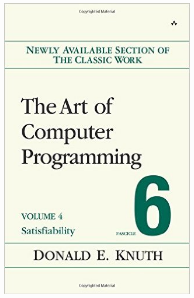


Applications of The CDCL SAT disruption

- Hundreds (thousands?) of practical applications

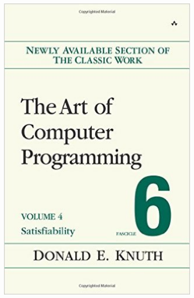
The Tale of Triumph of SAT Solvers

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



The Tale of Triumph of SAT Solvers

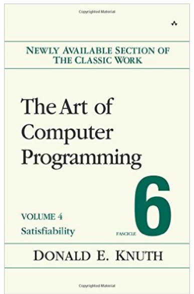
Modern SAT solvers are able to deal routinely with practical problems that involve many thousands of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)



Industrial usage of SAT Solvers: Hardware Verification, 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 many thousands of variables, although such problems were regarded as hopeless just a few years ago. (Donald Knuth, 2016)



Industrial usage of SAT Solvers: Hardware Verification, 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

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$

- $\mathrm{Sol}(F) = \{$ solutions of F $\}$

Constrained 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 $\}$
- Constrained Counting: Determine $|\mathsf{Sol}(F)|$

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: $\{0,1\}^n \mapsto [0,1]$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- $W(F) = \Sigma_{y \in \mathsf{Sol}(F)} W(y)$
- Constrained Counting: Determine $W(F)$

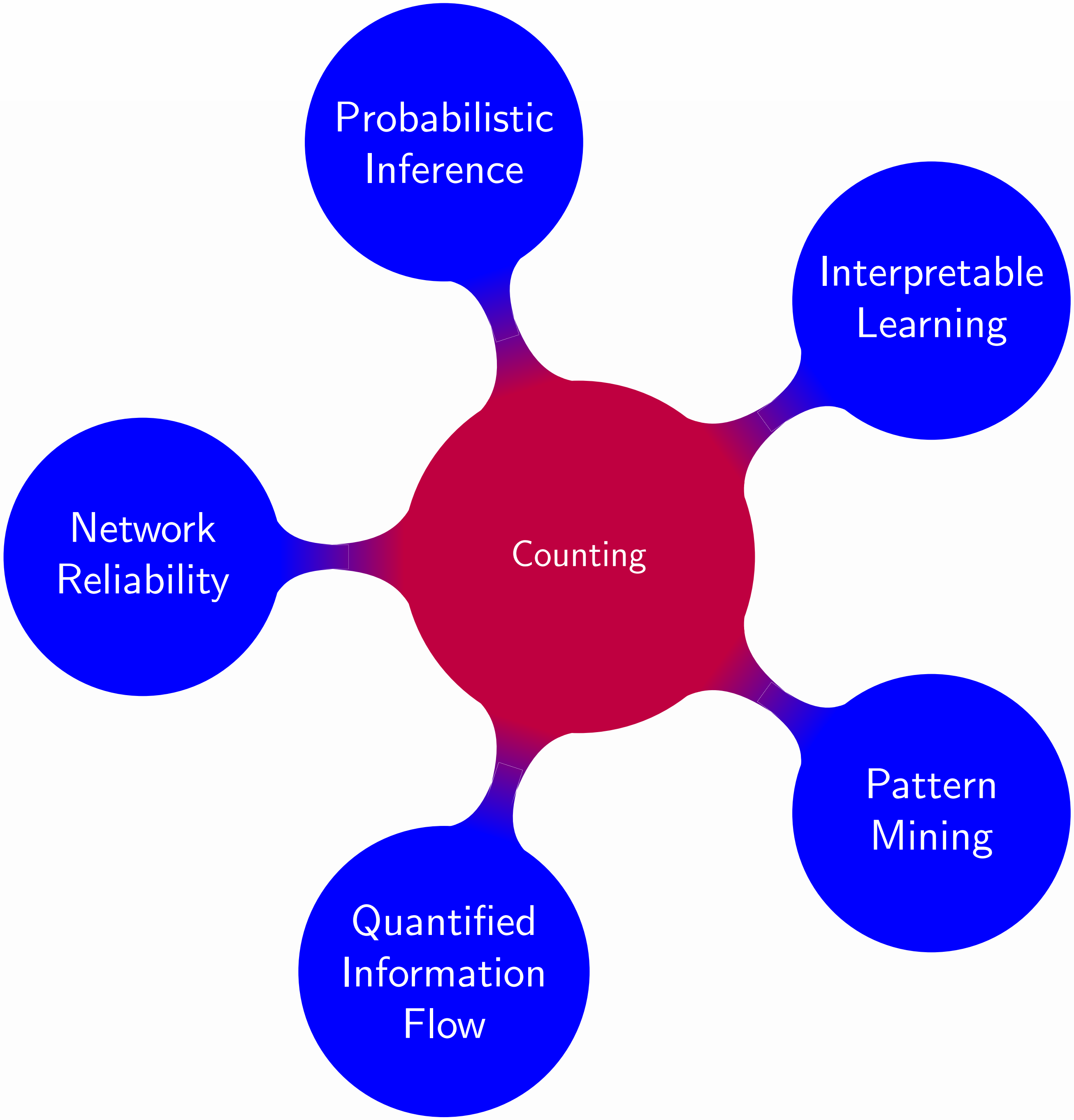
Constrained Counting

- Given
  - Boolean variables $X_1, X_2, \cdots X_n$
  - Formula F over $X_1, X_2, \cdots X_n$
  - Weight Function W: $\{0,1\}^n \mapsto [0,1]$
- $\mathsf{Sol}(F) = \{$ solutions of F $\}$
- $W(F) = \Sigma_{y \in \mathsf{Sol}(F)} W(y)$
- Constrained Counting: Determine $W(F)$
- 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

- 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 $W(F)$
- 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



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?

Reliability of Critical Infrastructure Networks

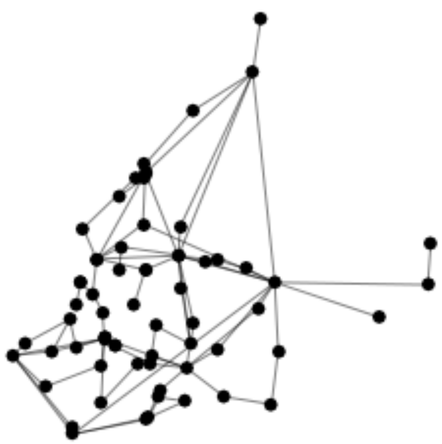


Figure: Plantersville, SC

- $G = (V, E)$; 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

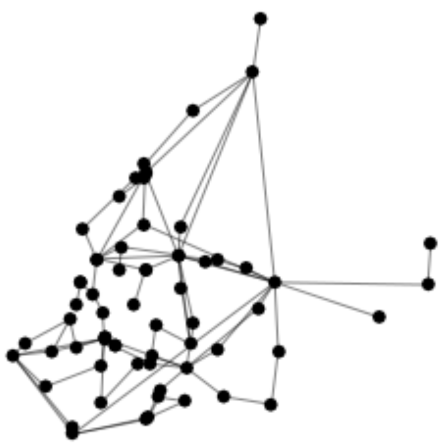


Figure: Plantersville, SC

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

Reliability of Critical Infrastructure Networks

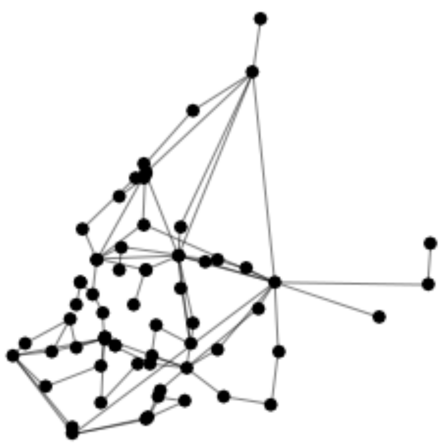


Figure: Plantersville, SC

- $G = (V, E)$; 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

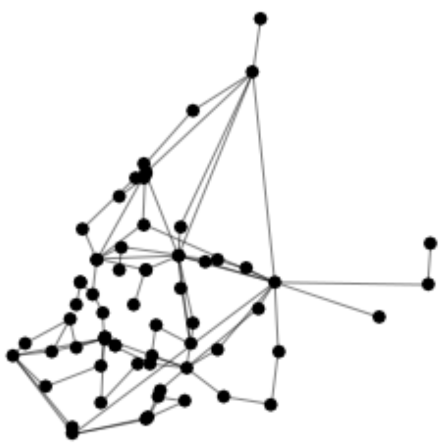


Figure: Plantersville, SC

- $G = (V, E)$; 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

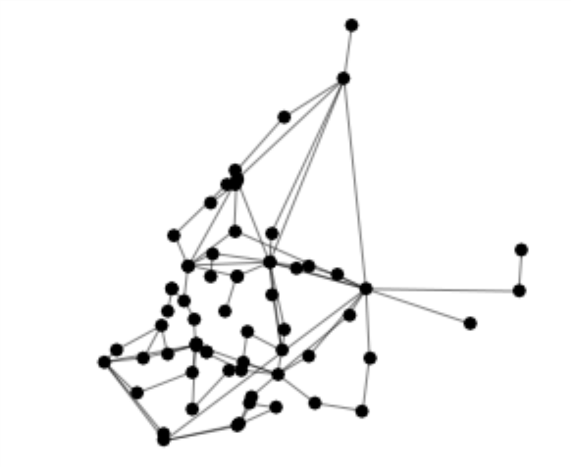


Figure: Plantersville, SC

- $G = (V, E)$; 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] $= \sum_{\pi_{s,t}} W(\pi_{s,t})$

(D**M**PV, AAAI 17)

Constrained Counting

Counting in Java

How many people in Java like coffee?

- Population of Java $= 141$M
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141$M)

Counting in Java

How many people in Java like coffee?

- Population of Java $= 141\text{M}$
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141\text{M}$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by $141\text{M}/50$

Counting in Java

How many people in Java like coffee?

- Population of Java $= 141\text{M}$
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141\text{M}$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by $141\text{M}/50$
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50

Counting in Java

How many people in Java like coffee?

- Population of Java $= 141$M
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 141M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP Query: Find a person who likes coffee

Counting in Java

How many people in Java like coffee?

- Population of Java $= 141$M
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 141M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP 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 Java

How many people in Java like coffee?

- Population of Java $= 141M$
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141M$)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by $141M/50$
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP 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

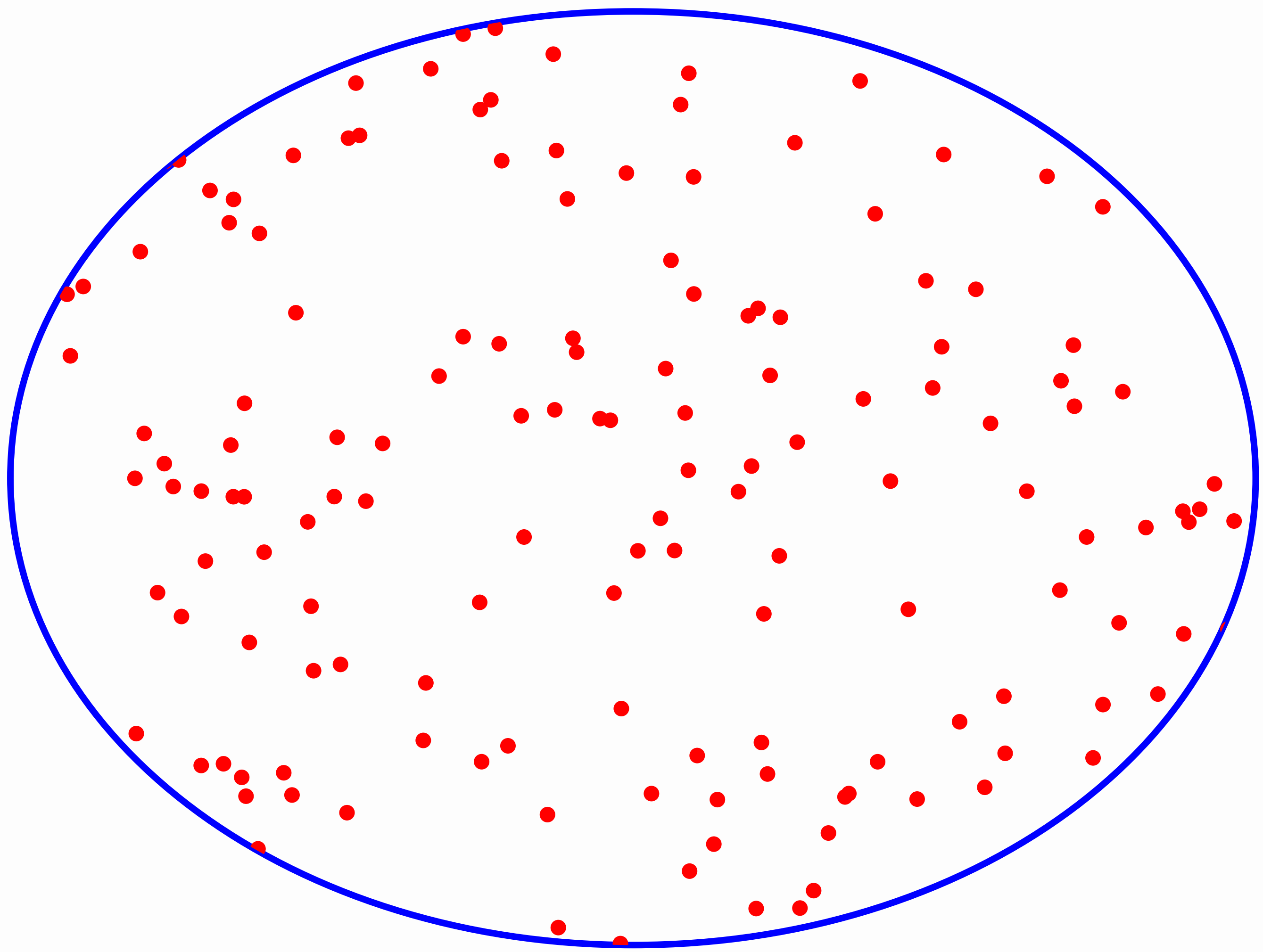
Counting in Java

How many people in Java like coffee?

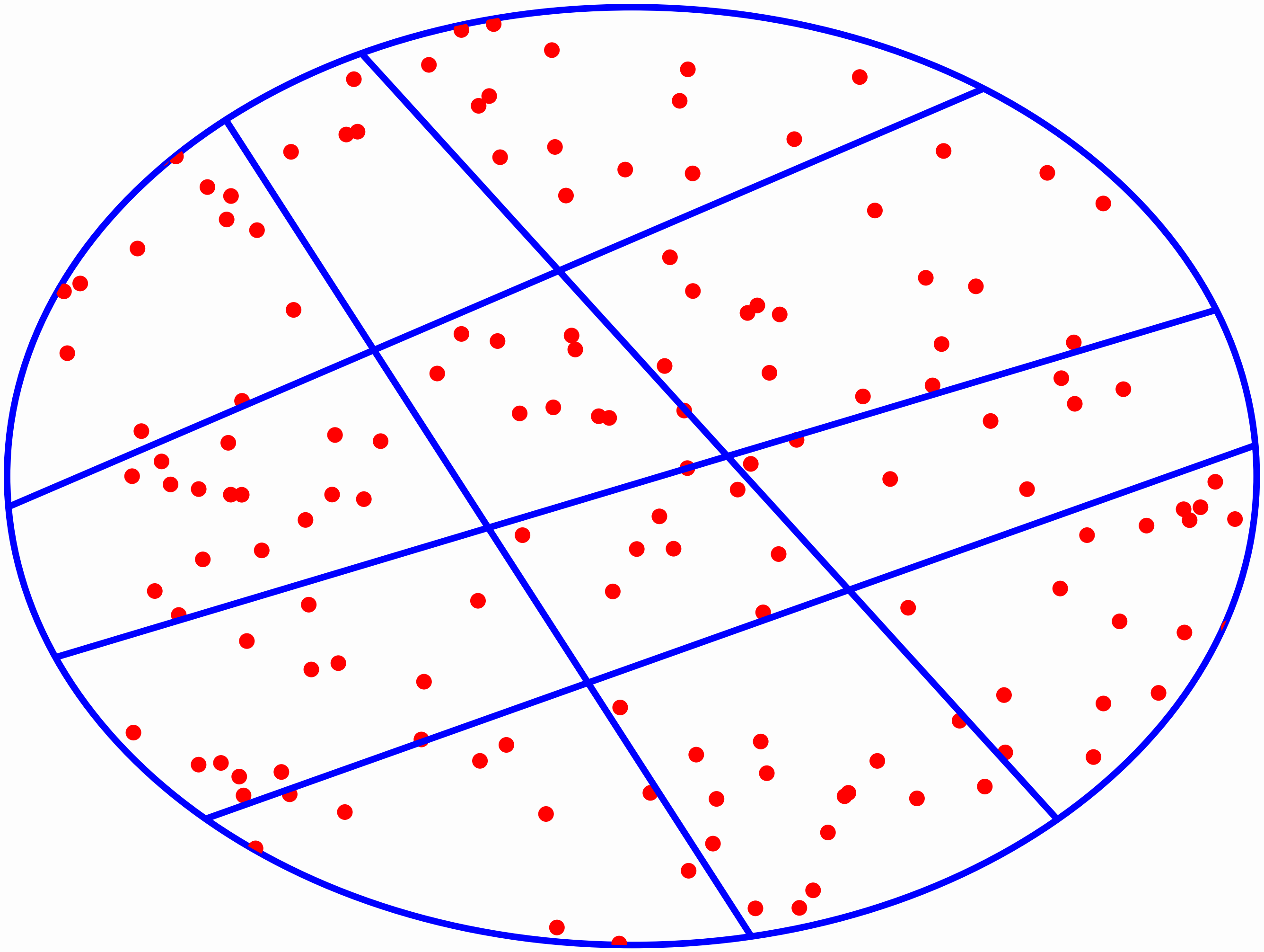
- Population of Java $= 141$M
- Assign every person a unique ($n =$) 26 bit identifier ($2^n = 141$M)
- Attempt #1: Pick 50 people and count how many of them like coffee and multiple by 141M/50
  - If only 5 people like coffee, it is unlikely that we will find anyone who likes coffee in our sample of 50
- NP 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

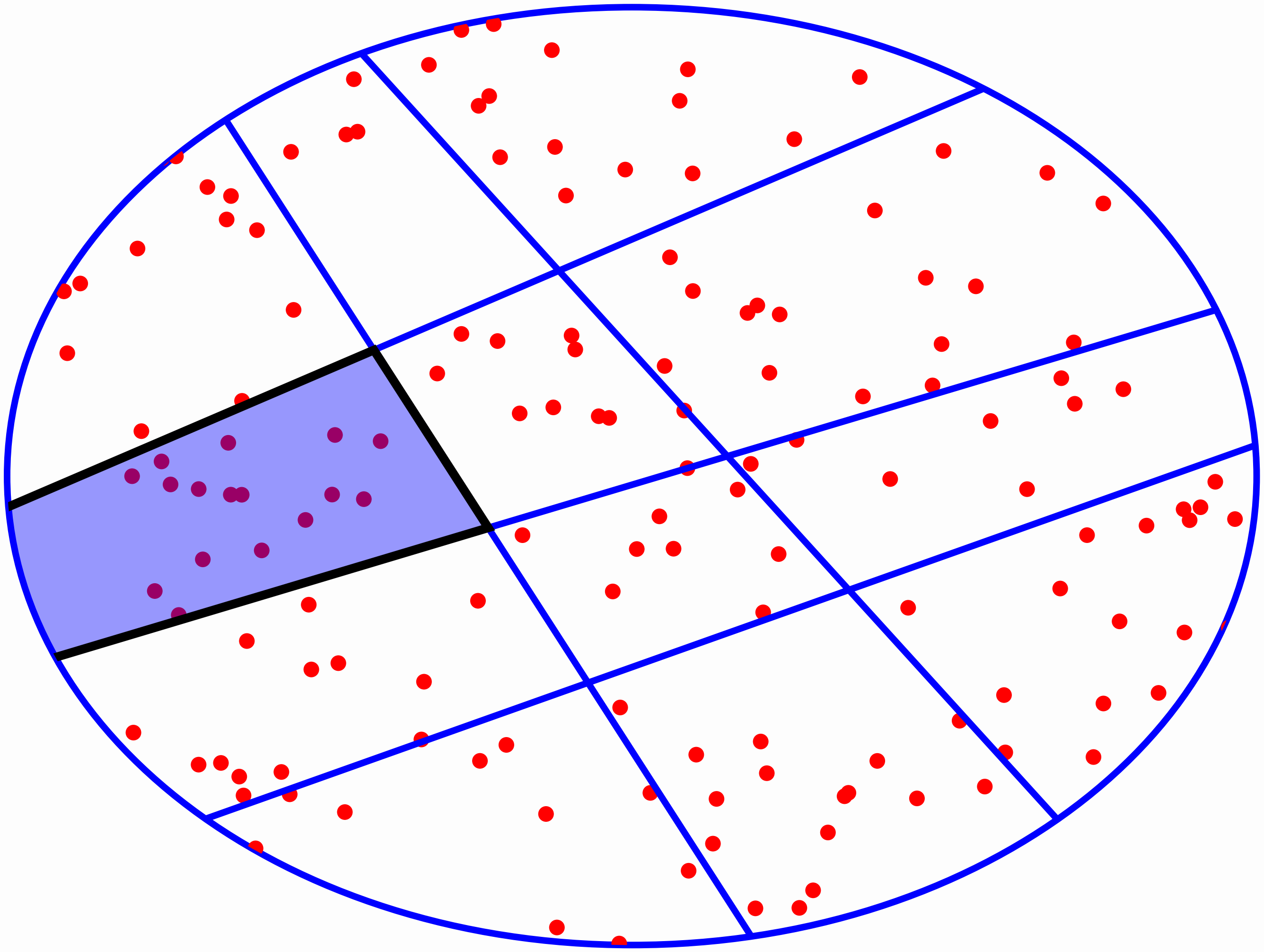


As Simple as Counting Dots



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

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

Challenges

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

- Universal Hash Functions

Challenge 2 How many cells?

Question 2: How many cells?

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

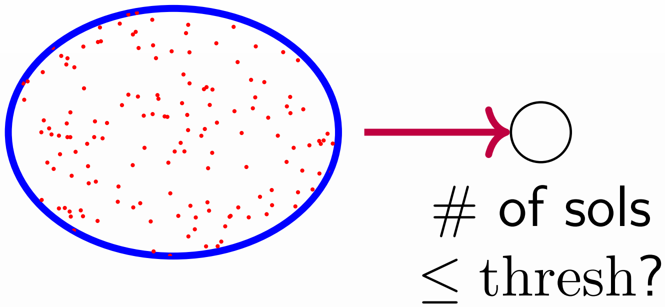
Question 2: How many cells?

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

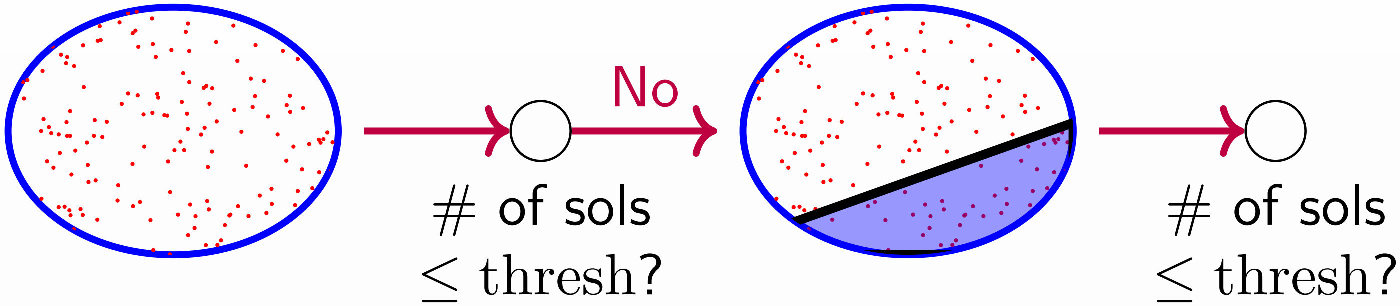
Question 2: How many cells?

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

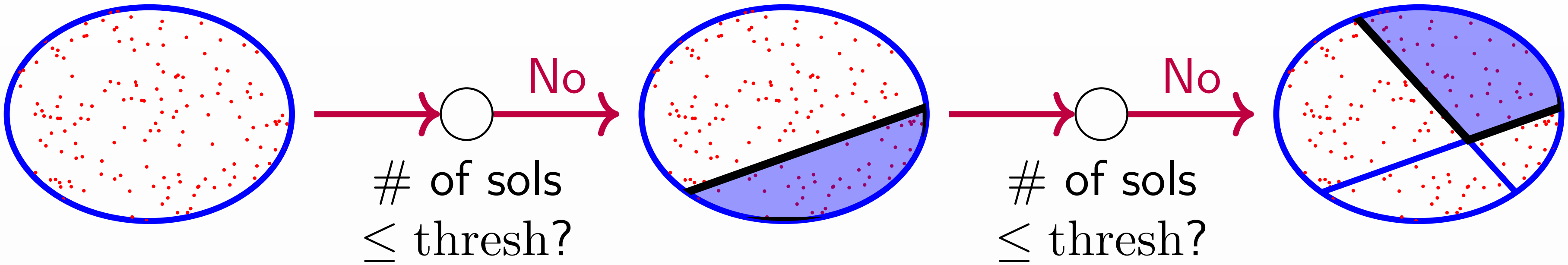
ApproxMC(F, ε, δ)



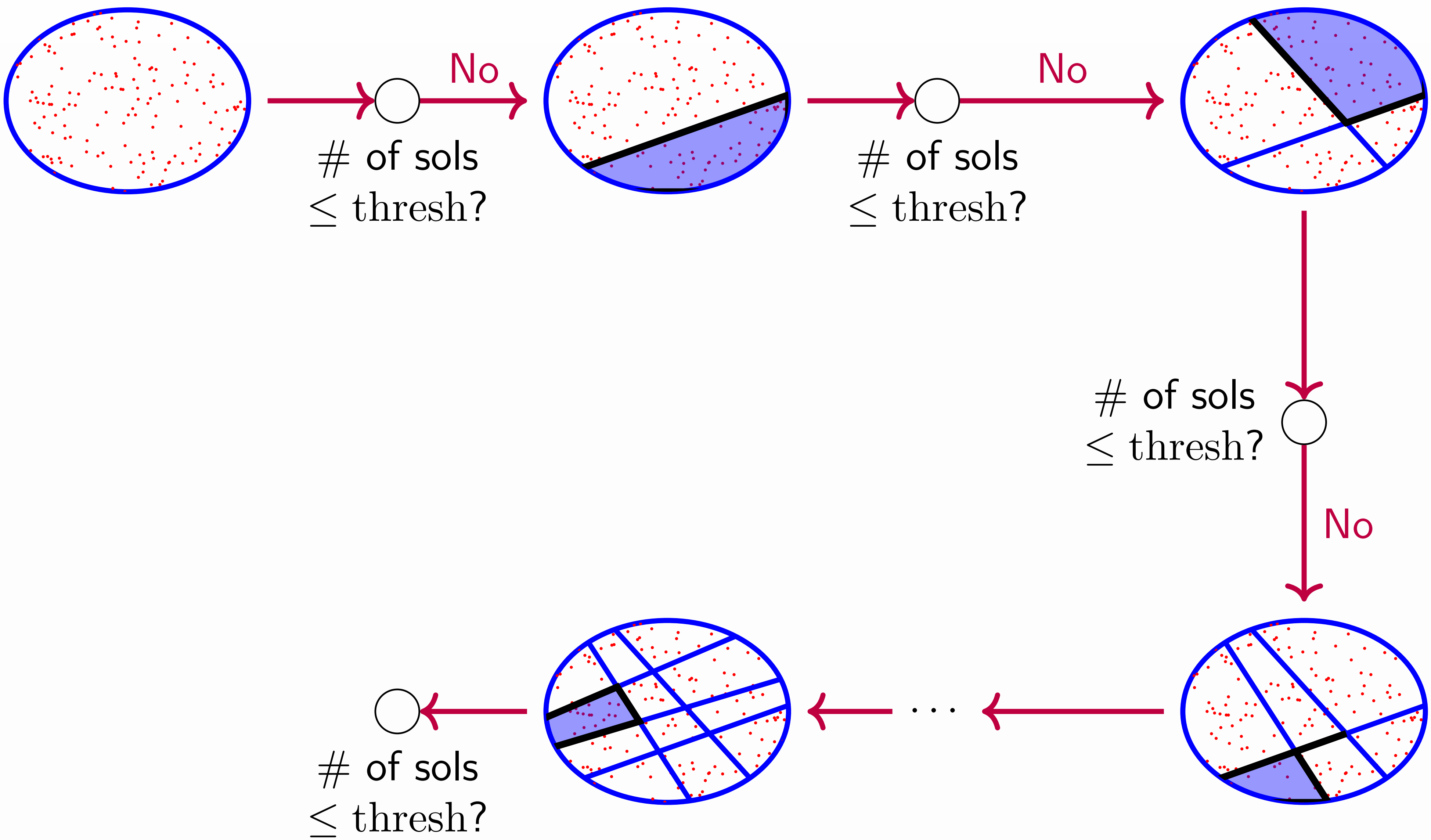
ApproxMC(F, ε, δ)



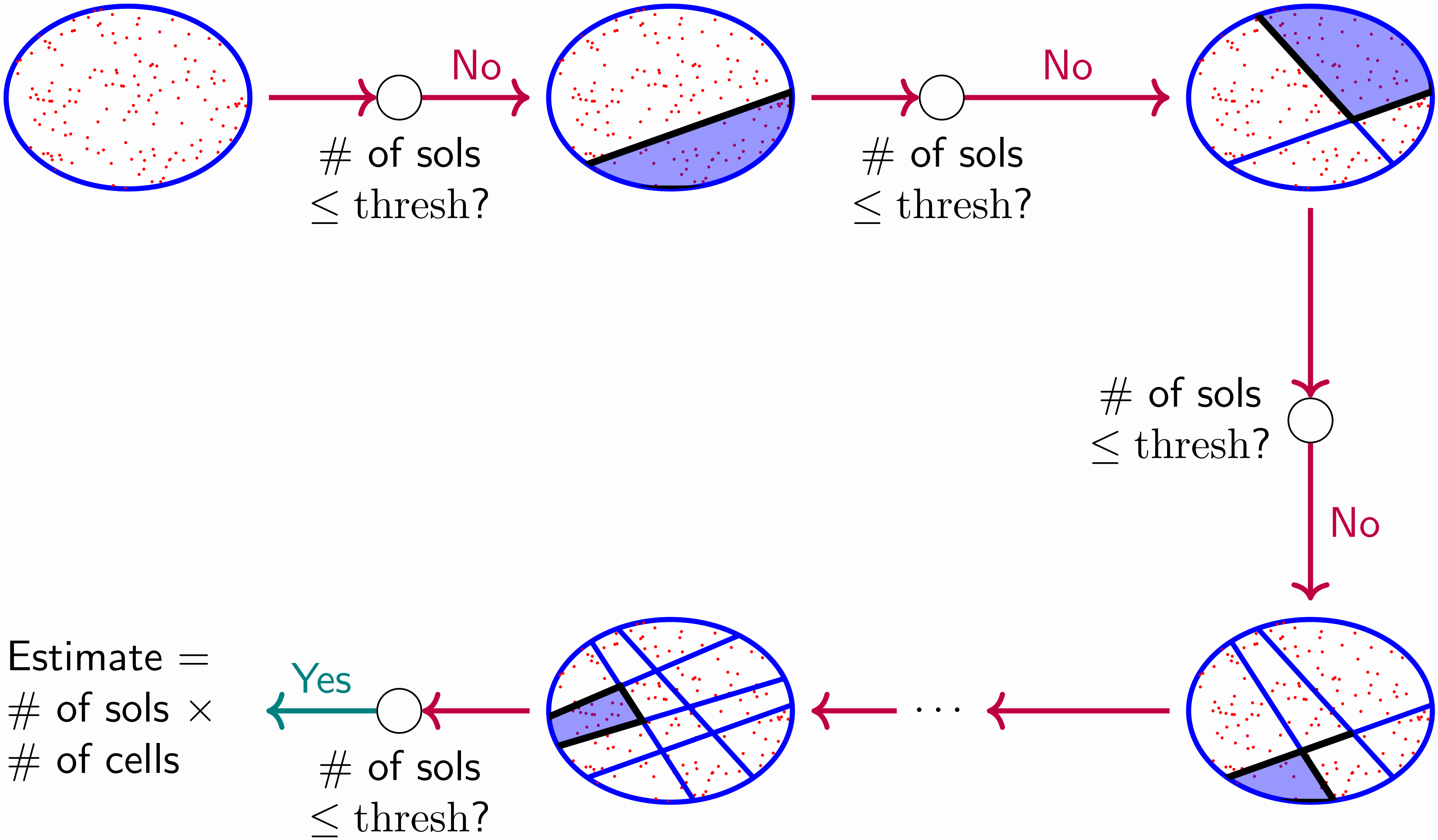
ApproxMC(F, ε, δ)



ApproxMC(F, ε, δ)



ApproxMC(F, ε, δ)



ApproxMC(F, ε, δ)

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)*

Reliability of Critical Infrastructure Networks

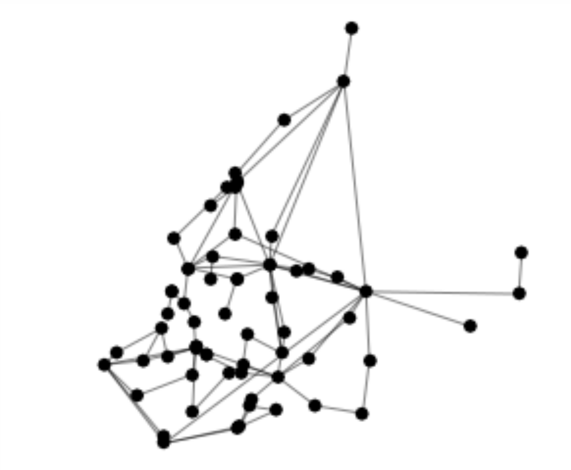
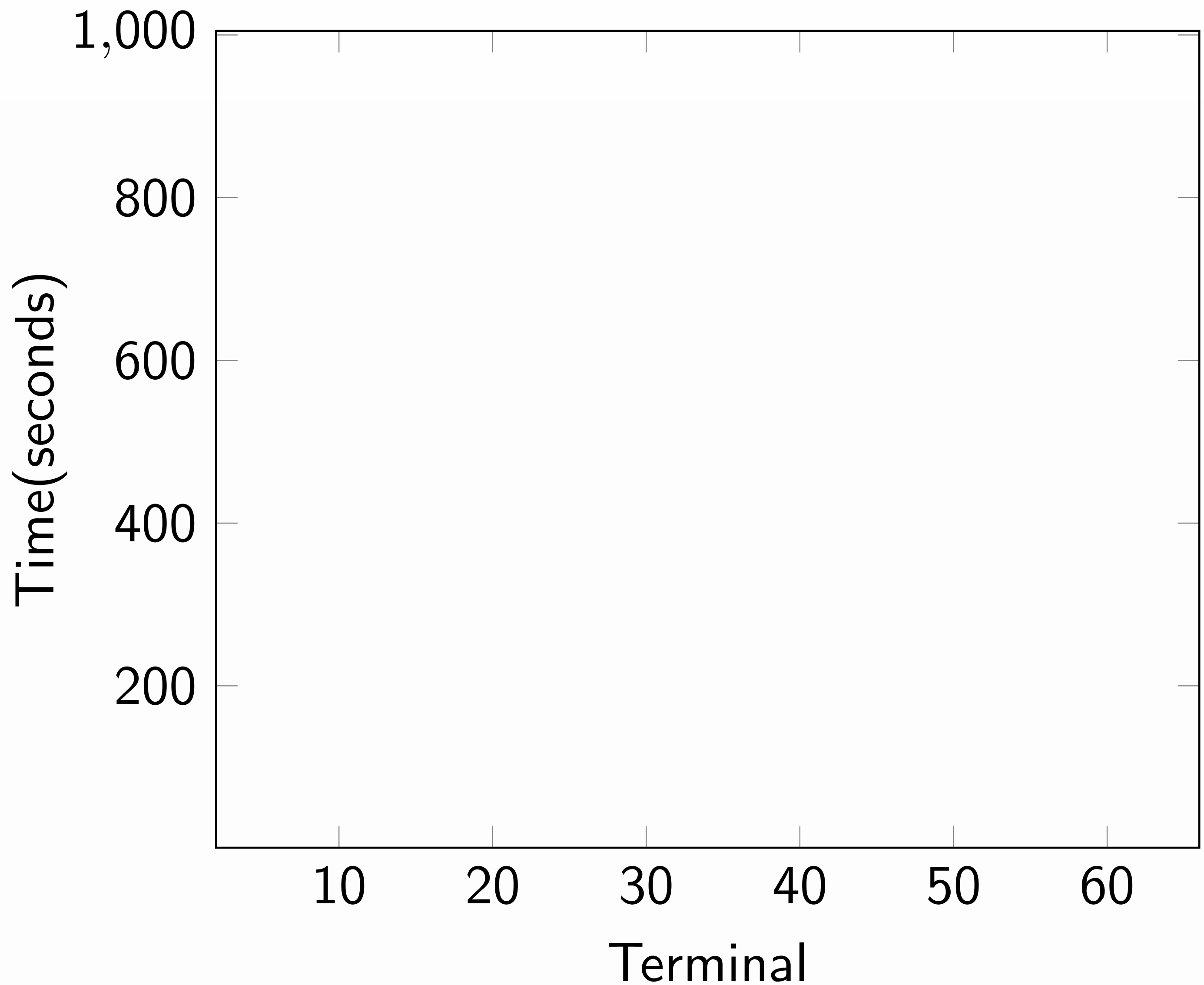


Figure: Plantersville, SC

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

Timeout = 1000 seconds



(D**M**PV, AAAI17)

Reliability of Critical Infrastructure Networks

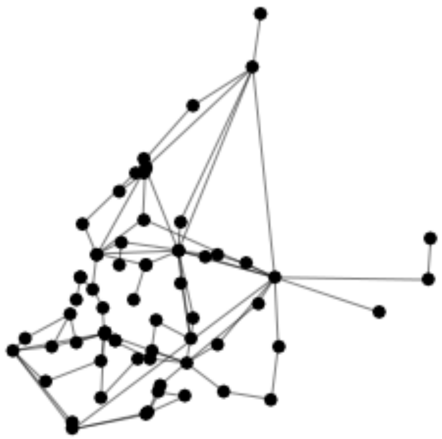
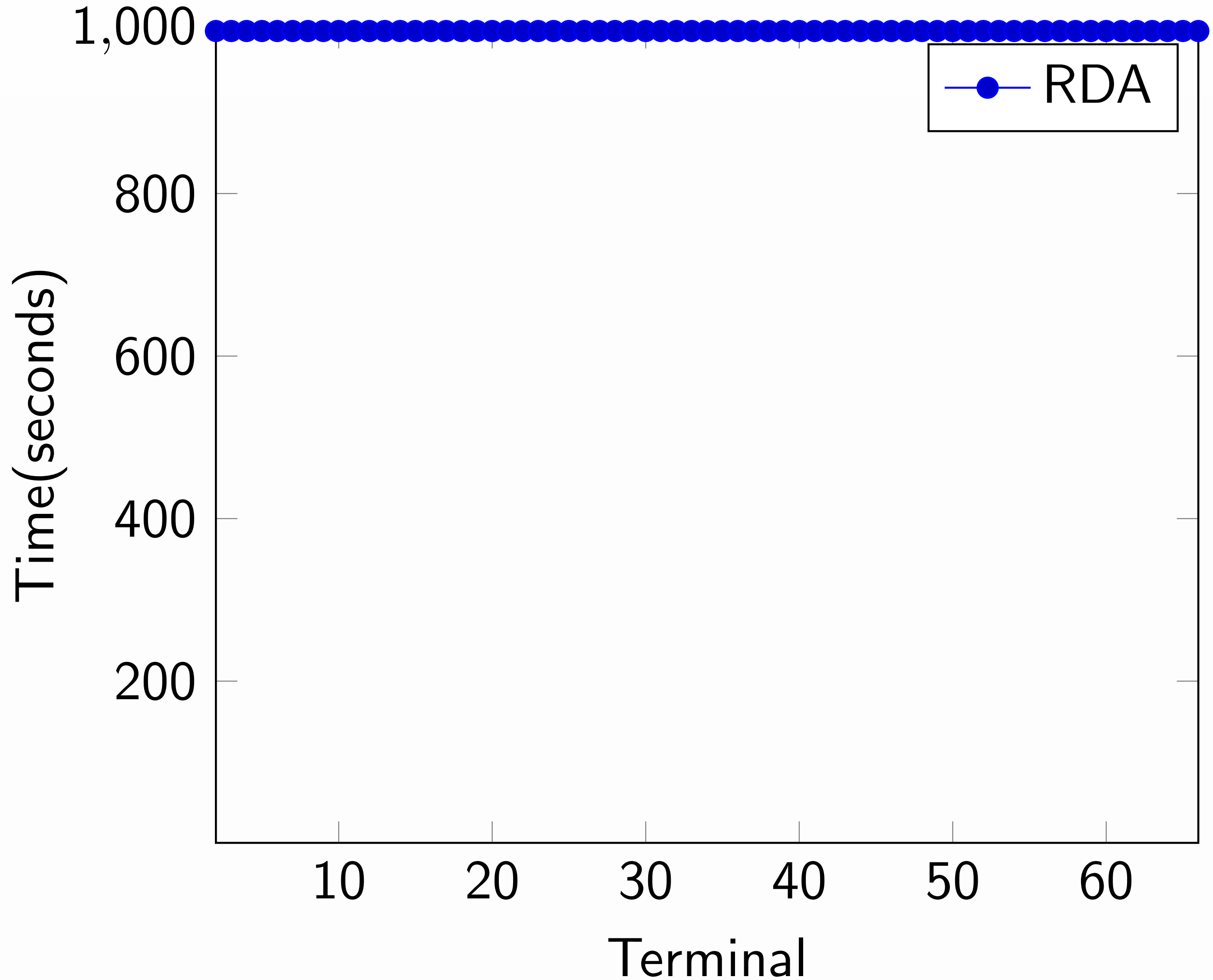


Figure: Plantersville, SC



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

Timeout = 1000 seconds

(D**M**PV, AAAI17)

Reliability of Critical Infrastructure Networks

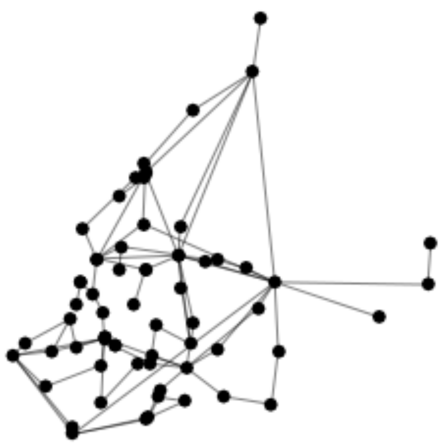
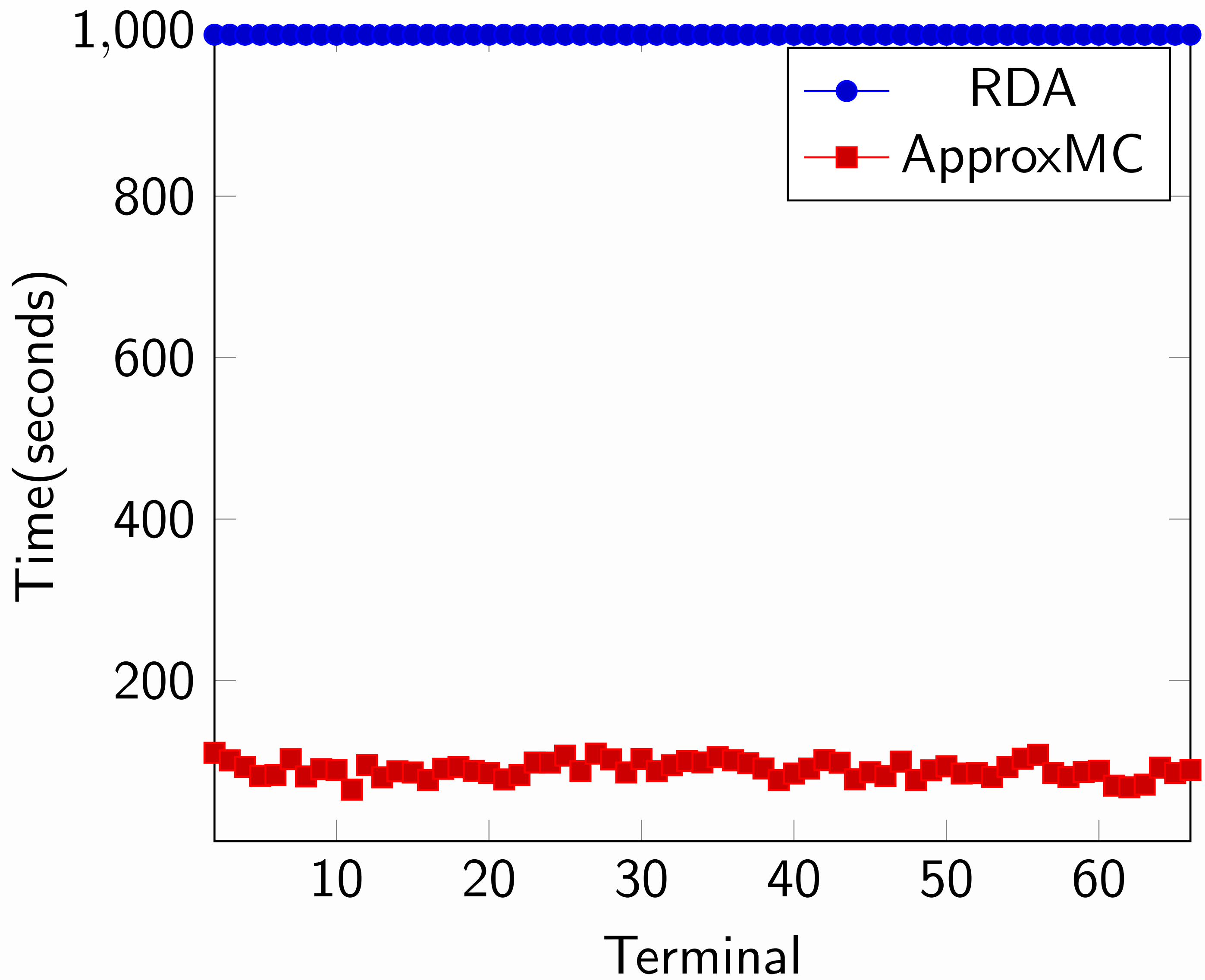


Figure: Plantersville, SC

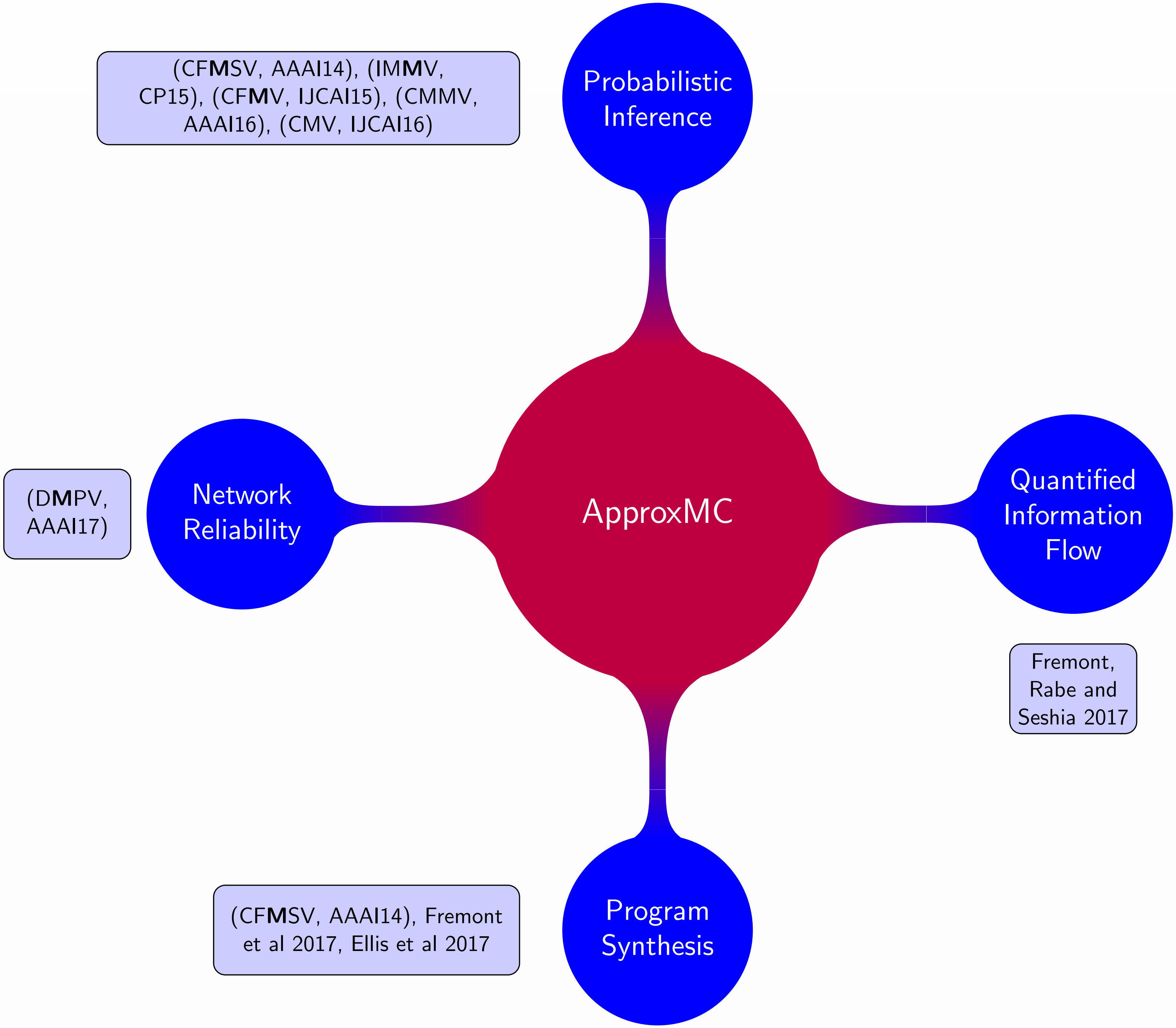
- $G = (V, E)$; source node: s
- Compute Pr[t is disconnected]?

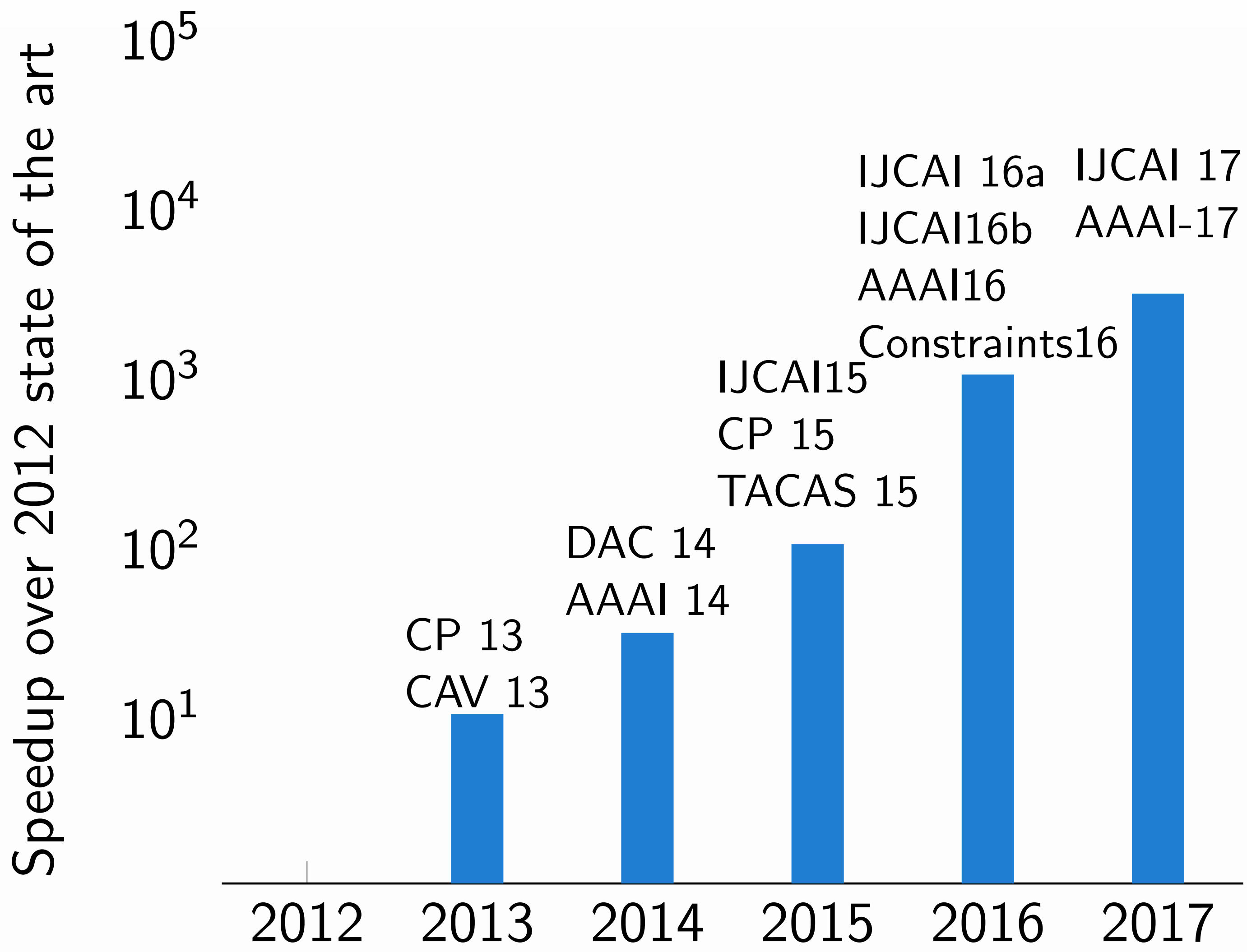
Timeout = 1000 seconds



(D**M**PV, AAAI17)

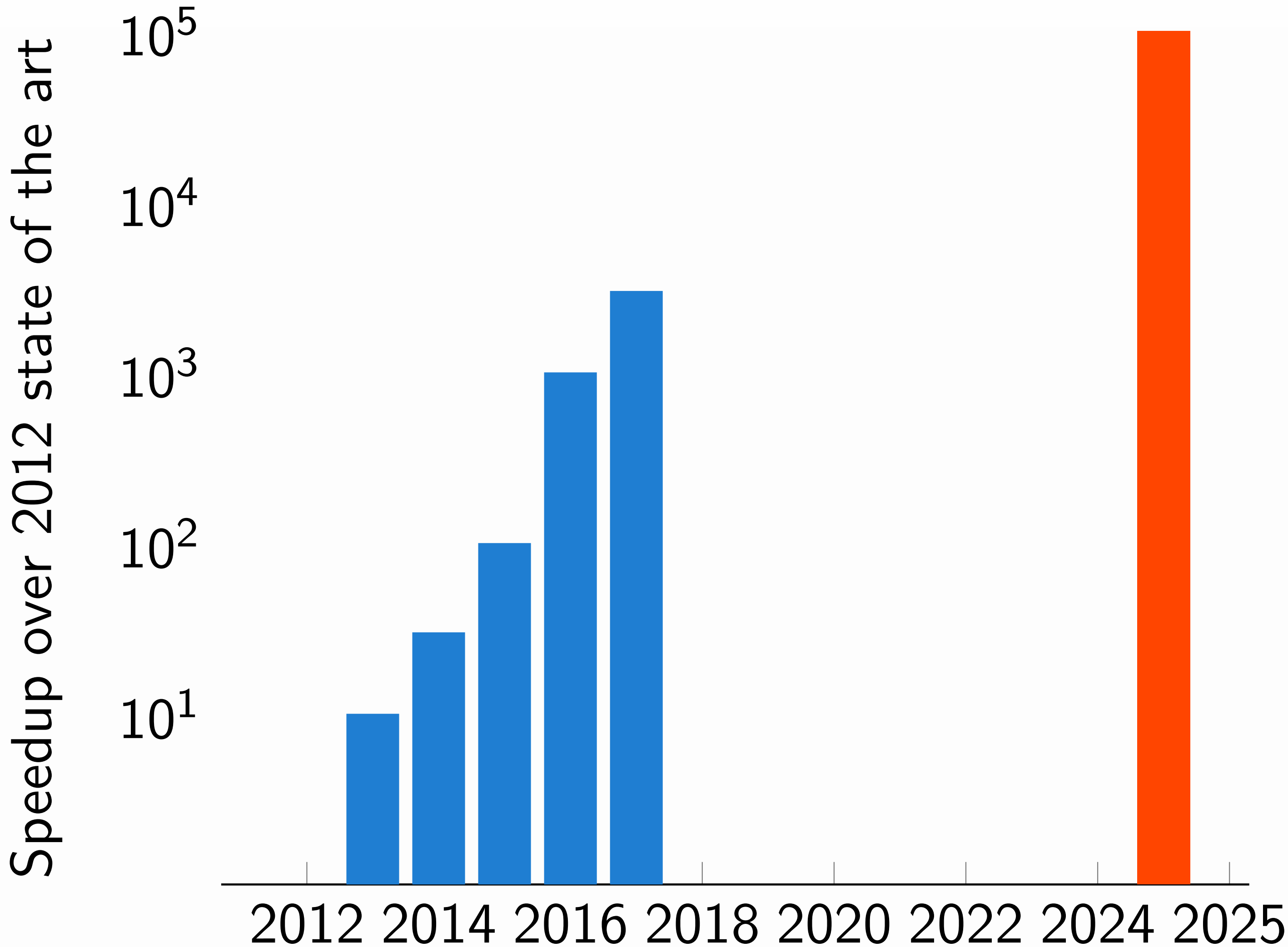
Beyond Network Reliability





Speedup over 2012 state of the art
10^5
10^4
10^3
10^2
10^1
CP 13
CAV 13
DAC 14
AAAI 14
IJCAI15
CP 15
TACAS 15
IJCAI 16a
IJCAI16b
AAAI16
Constraints16
IJCAI 17
AAAI-17
2012
2013
2014
2015
2016
2017

Mission 2025: Constrained Counting Revolution



Requires combinations of ideas from theory, statistics and systems

Mission 2025: Constrained Counting and Sampling Revolution

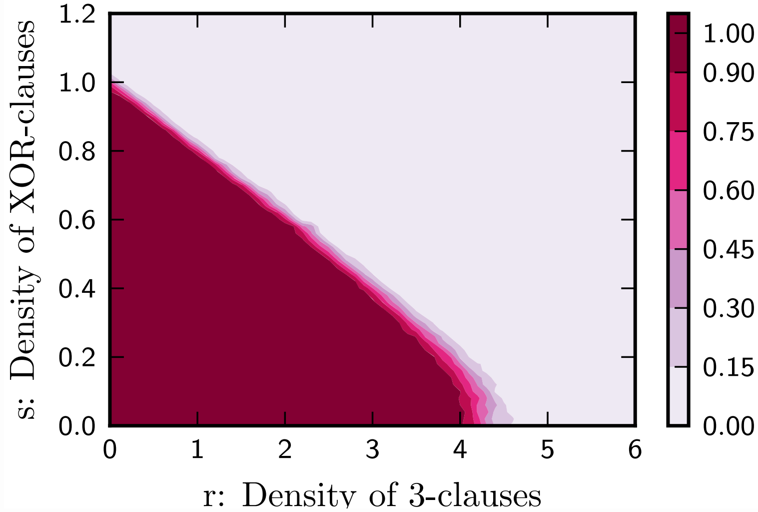
- Tighter integration between solvers and algorithms

Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms
- Exploring solution space structure of CNF+XOR formulas

(DMV, IJCAI16)

(DMV, IJCAI17)

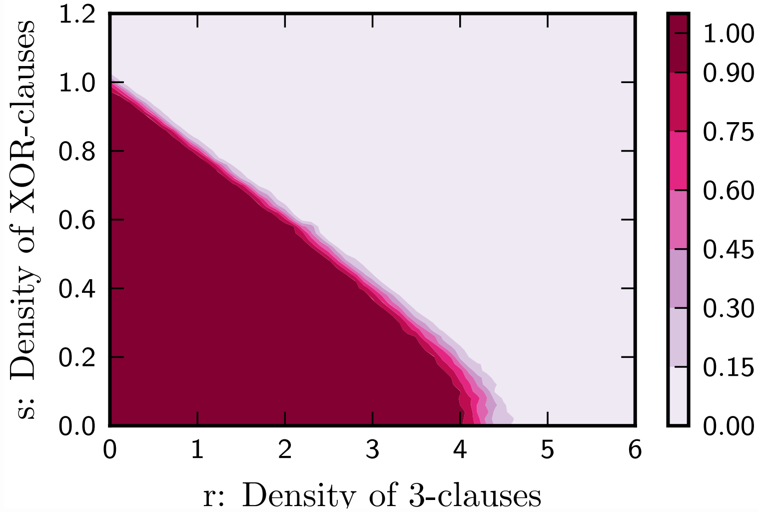


Mission 2025: Constrained Counting and Sampling Revolution

- Tighter integration between solvers and algorithms
- Exploring solution space structure of CNF+XOR formulas

(DMV, IJCAI16)

(DMV, IJCAI17)



- Beyond Boolean variables – without *bit blasting*

Mission 2025: Constrained Counting Revolution

Challenge Problems

Mission 2025: Constrained Counting Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

Mission 2025: Constrained Counting Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

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

Mission 2025: Constrained Counting Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

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

Machine Learning Inference for Bayesian network with 10K nodes

Mission 2025: Constrained Counting Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

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

Machine Learning Inference for Bayesian network with 10K nodes

The Potential of Hashing-based Framework

Programming Probabilistic programming

Mission 2025: Constrained Counting Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

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

Machine Learning Inference for Bayesian network with 10K nodes

The Potential of Hashing-based Framework

Programming Probabilistic programming

Theory Classification of Approximate Counting Complexity

Mission 2025: Constrained Counting Revolution

Challenge Problems

Civil Engineering Reliability for Los Angeles Transmission Grid

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

Machine Learning Inference for Bayesian network with 10K nodes

The Potential of Hashing-based Framework

Programming Probabilistic programming

Theory Classification of Approximate Counting Complexity

Databases Streaming algorithms

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