Towards a Theory for Computing with SAT Solvers

What's the Power of a Satisfying Assignment?

Kuldeep S. Meel

University of Toronto

Joint work with Sourav Chakraborty, Diptarka Chakraborty, Remi Delannoy, and Gunjan Kumar

Relevant publication: LICS-22 and ICALP-23

Analysis of Algorithms

Knuth: "*People who analyze algorithms experience double happiness. Firstly, they encounter the beauty of mathematical patterns in elegant computational procedures. Secondly, they gain practical benefits as their theories enhance efficiency and economy in various applications.*"

Analysis of Algorithms

Knuth: "*People who analyze algorithms experience double happiness. Firstly, they encounter the beauty of mathematical patterns in elegant computational procedures. Secondly, they gain practical benefits as their theories enhance efficiency and economy in various applications.*"

Knuth's Approach to Algorithm Analysis (1962)

- A Scientific way to analyze algorithms
  - Identifying operation costs and input models
  - Examining operation frequencies
  - Calculating total runtime: $\sum_{\text{op}} \text{frequency}(\text{op}) \cdot \text{cost}(\text{op})$

- Objectives
  - Predicting algorithm performance
  - Facilitating algorithm comparison and improvement

Analysis of Algorithms

Knuth: "*People who analyze algorithms experience double happiness. Firstly, they encounter the beauty of mathematical patterns in elegant computational procedures. Secondly, they gain practical benefits as their theories enhance efficiency and economy in various applications.*"

Knuth's Approach to Algorithm Analysis (1962)

- A Scientific way to analyze algorithms
  - Identifying operation costs and input models
  - Examining operation frequencies
  - Calculating total runtime: $\sum_{\text{op}} \text{frequency}(\text{op}) \cdot \text{cost}(\text{op})$

- Objectives
  - Predicting algorithm performance
  - Facilitating algorithm comparison and improvement
  - Encounter beauty of mathematical patterns

A Theory for PTIME and Beyond

Edmonds, 1965: A Theory of Efficient Combinatorial Algorithms (PTIME)

A Theory for PTIME and Beyond

Edmonds, 1965: A Theory of Efficient Combinatorial Algorithms (PTIME)

Cook, 1971; Levin, 1973: Non-deterministic Polynomial Time (NP) Computations

Karp, 1972: The Surprising Power of NP-Completeness

A Theory for PTIME and Beyond

Edmonds, 1965: A Theory of Efficient Combinatorial Algorithms (PTIME)

Cook, 1971; Levin, 1973: Non-deterministic Polynomial Time (NP) Computations

Karp, 1972: The Surprising Power of NP-Completeness

Stockmeyer, 1972: A Theory of Polynomial Hierarchy Based on Oracles

A Rich Theory of Algorithms Relying on Oracles

- For four decades, the theoretical framework involving oracle-based computations has primarily remained within theoretical discussions
- Largely due to our inability to tackle NP-complete problems in practice until recently.

The SAT Revolution

Boolean Satisfiability (SAT): Given a Boolean formula, is there a solution, i.e., an assignment of 0's and 1's to the variables that makes the formula equal 1?

Example: $(X_1 \vee \neg X_2 \vee \neg X_3) \wedge (X_2 \vee \neg X_3)$ $\quad X_1 = 1, X_2 = 1, X_3 = 1$

The SAT Revolution

Boolean Satisfiability (SAT): Given a Boolean formula, is there a solution, i.e., an assignment of 0's and 1's to the variables that makes the formula equal 1?

Example: $(X_1 \vee \neg X_2 \vee \neg X_3) \wedge (X_2 \vee \neg X_3)$ $\quad X_1 = 1, X_2 = 1, X_3 = 1$

Knuth, 2016: These so-called "SAT solvers" can now routinely find solutions to practical problems that involve millions of variables and were thought until very recently to be hopelessly difficult.

[Late 2000's]: Now that SAT is "easy", it is time to look beyond satisfiability

- Development of algorithmic frameworks for optimization/MaxSAT, counting, sampling, and synthesis

Reducing the Chasm Between Oracles and Solvers

We will focus on the case of Boolean Formulas

NP Oracle: Return Yes ("Accept") if a formula φ is satisfiable else Return No

SAT Solver: Return a satisfying assignment if a formula φ is satisfiable else Returns No

- SAT solvers also return proof of unsatisfiablity but we will ignore that for now.

Need to analyze complexity with respect to oracles that capture behavior of SAT solvers

SAT Oracle: Return a satisfying assignment if φ is satisfiable else Return No

Not limited to Boolean formulas: Every decision procedure for an NP-complete problem that has been implemented always gives a witness whenever it can prove satisfying assignment

SAT Oracles and simulation by NP oracles

SAT Oracles can be simulated by n calls to NP oracles, so what's the big deal?

- Satisfying Assignment: Given a formula φ, find a satisfying assignment if it exists
  - Complexity: NP(n) ; SAT(1)

SAT Oracles and simulation by NP oracles

SAT Oracles can be simulated by n calls to NP oracles, so what's the big deal?

- Satisfying Assignment: Given a formula φ, find a satisfying assignment if it exists
  - Complexity: NP(n) ; SAT(1)

- There is **"ONLY"** a factor of n difference.
  - Factor n is the difference between today and 1995
  - On a formula with million variables, today's solvers are often 10^6 than a solver in 1995.
  - If all we had were the solvers from 1995, we wouldn't be bothered about computing with SAT solvers

SAT Oracles and simulation by NP oracles

SAT Oracles can be simulated by n calls to NP oracles, so what's the big deal?

- Satisfying Assignment: Given a formula φ, find a satisfying assignment if it exists
  - Complexity: NP(n) ; SAT(1)

- There is **"ONLY"** a factor of n difference.
  - Factor n is the difference between today and 1995
  - On a formula with million variables, today's solvers are often 10^6 than a solver in 1995.
  - If all we had were the solvers from 1995, we wouldn't be bothered about computing with SAT solvers

- Fine-Grained Analysis matters: It's not always exactly a factor n

SAT Oracles and simulation by NP oracles

SAT Oracles can be simulated by n calls to NP oracles, so what's the big deal?

- Satisfying Assignment: Given a formula φ, find a satisfying assignment if it exists
  - Complexity: NP(n) ; SAT(1)

- There is **"ONLY"** a factor of n difference.
  - Factor n is the difference between today and 1995
  - On a formula with million variables, today's solvers are often 10^6 than a solver in 1995.
  - If all we had were the solvers from 1995, we wouldn't be bothered about computing with SAT solvers

- Fine-Grained Analysis matters: It's not always exactly a factor n

- Satisfiability: Given a formula φ, is it satisfiable?
  - Complexity: NP(1) ; SAT(1)

SAT Oracles and simulation by NP oracles

SAT Oracles can be simulated by n calls to NP oracles, so what's the big deal?

- Satisfying Assignment: Given a formula φ, find a satisfying assignment if it exists
  - Complexity: NP(n) ; SAT(1)

- There is **"ONLY"** a factor of n difference.
  - Factor n is the difference between today and 1995
  - On a formula with million variables, today's solvers are often 10^6 than a solver in 1995.
  - If all we had were the solvers from 1995, we wouldn't be bothered about computing with SAT solvers

- Fine-Grained Analysis matters: It's not always exactly a factor n

- Satisfiability: Given a formula φ, is it satisfiable?
  - Complexity: NP(1) ; SAT(1)

- Objectives of Analysis of Algorithms: Predicting algorithm performance & facilitating algorithm improvement

- Algorithm analysis should be based on oracles that capture the physical reality

Sampling and Counting

Almost-Uniform Sampling: Given a Boolean formula φ and tolerance parameter ε output solutions σ such that

$$\frac{1}{(1+\varepsilon)|\mathsf{sol}(\varphi)|} \leq \Pr[\sigma \text{ is output}] \leq \frac{1+\varepsilon}{|\mathsf{sol}(\varphi)|}$$

Approximate Model Counting: Given a Boolean formula φ, tolerance parameter ε and confidence parameter δ, output c such that

$$\Pr\left[\frac{|\mathsf{sol}(\varphi)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathsf{sol}(\varphi)|\right] \geq 1-\delta$$

We will assume ε to be a constant in this talk

The Power of Inter-reducibility of Sampling and Counting

Jerrum, Valiant, and Vazirani, 1986

Assumes access to a (approximate) model counter

- Can be implemented with $O(\log n \cdot \log \delta^{-1})$ NP-oracle calls

The Power of Inter-reducibility of Sampling and Counting

Jerrum, Valiant, and Vazirani, 1986

Assumes access to a (approximate) model counter

- Can be implemented with $O(\log n \cdot \log \delta^{-1})$ NP-oracle calls

For a given formula φ,

- Assign $x_1 = 1$ with probability $\frac{|\mathsf{sol}(\varphi \wedge x_1)|}{|\mathsf{sol}(\varphi)|}$
- Say we assigned $x_1 = 1$, now assign $x_2 = 1$ with probability $\frac{|\mathsf{sol}(\varphi \wedge x_1 \wedge x_2)|}{|\mathsf{sol}(\varphi \wedge x_1)|}$
- And so on until we have assigned all the variables

The Power of Inter-reducibility of Sampling and Counting

Jerrum, Valiant, and Vazirani, 1986

Assumes access to a (approximate) model counter

- Can be implemented with $O(\log n \cdot \log \delta^{-1})$ NP-oracle calls

For a given formula φ,

- Assign $x_1 = 1$ with probability $\frac{|\mathsf{sol}(\varphi \wedge x_1)|}{|\mathsf{sol}(\varphi)|}$
- Say we assigned $x_1 = 1$, now assign $x_2 = 1$ with probability $\frac{|\mathsf{sol}(\varphi \wedge x_1 \wedge x_2)|}{|\mathsf{sol}(\varphi \wedge x_1)|}$
- And so on until we have assigned all the variables

Complexity Analysis: n calls to counter where δ for counter needs to be set to $\mathcal{O}(2^{-n})$

- NP Oracle: $\mathcal{O}(n^2 \log n)$ queries
- SAT Oracle: $\mathcal{O}(n^2 \log n)$ queries
- JVV can not use the power of SAT oracle

Objectives of Analysis of Algorithms: Facilitating algorithm improvement

The Power of Inter-reducibility of Sampling and Counting

Jerrum, Valiant, and Vazirani, 1986

Assumes access to a (approximate) model counter

- Can be implemented with $O(\log n \cdot \log \delta^{-1})$ NP-oracle calls

For a given formula φ,

- Assign $x_1 = 1$ with probability $\frac{|\mathsf{sol}(\varphi \wedge x_1)|}{|\mathsf{sol}(\varphi)|}$
- Say we assigned $x_1 = 1$, now assign $x_2 = 1$ with probability $\frac{|\mathsf{sol}(\varphi \wedge x_1 \wedge x_2)|}{|\mathsf{sol}(\varphi \wedge x_1)|}$
- And so on until we have assigned all the variables

Complexity Analysis: n calls to counter where δ for counter needs to be set to $\mathcal{O}(2^{-n})$

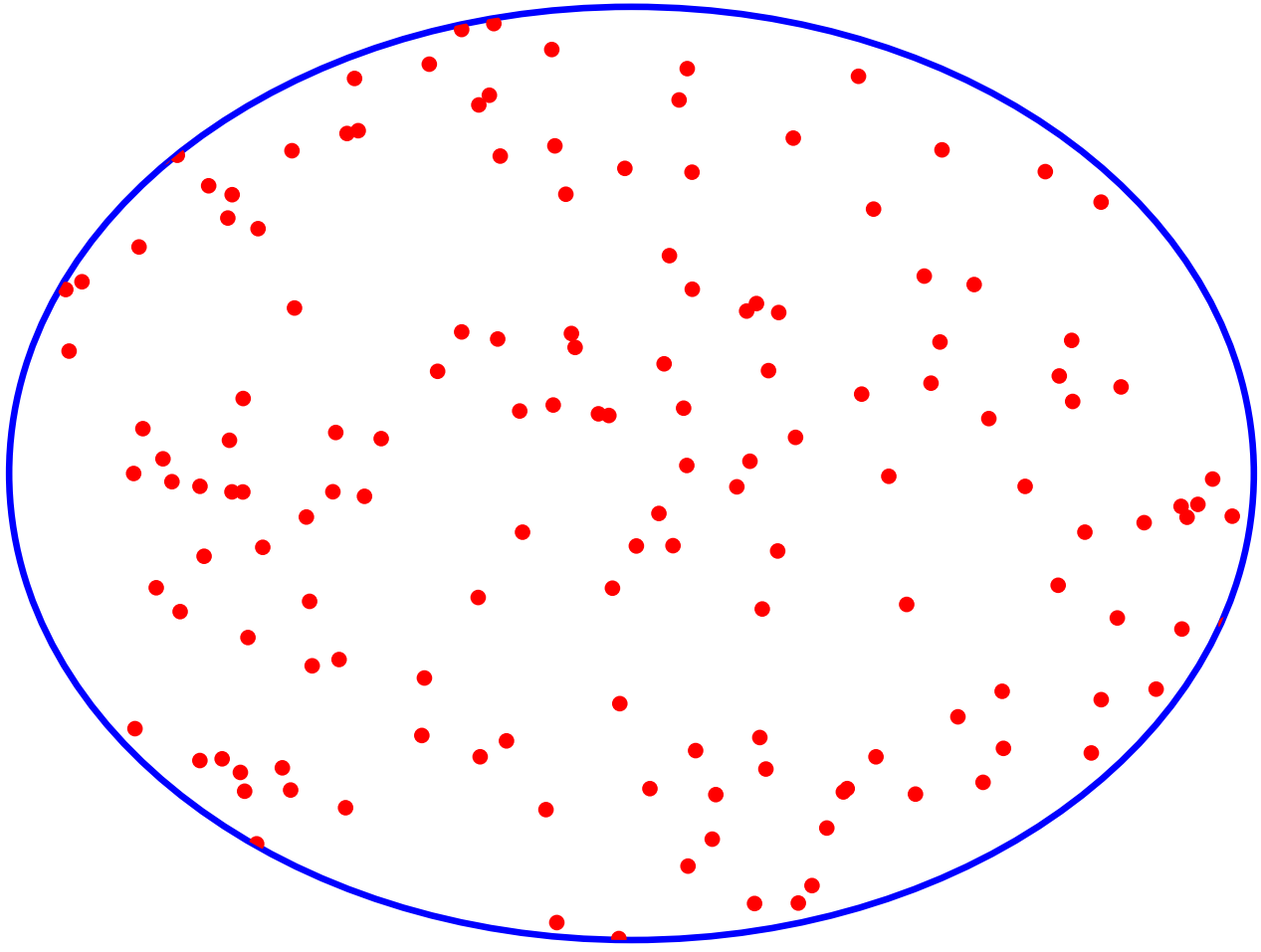
- NP Oracle: $\mathcal{O}(n^2 \log n)$ queries
- SAT Oracle: $\mathcal{O}(n^2 \log n)$ queries
- JVV can not use the power of SAT oracle

Objectives of Analysis of Algorithms: Facilitating algorithm improvement

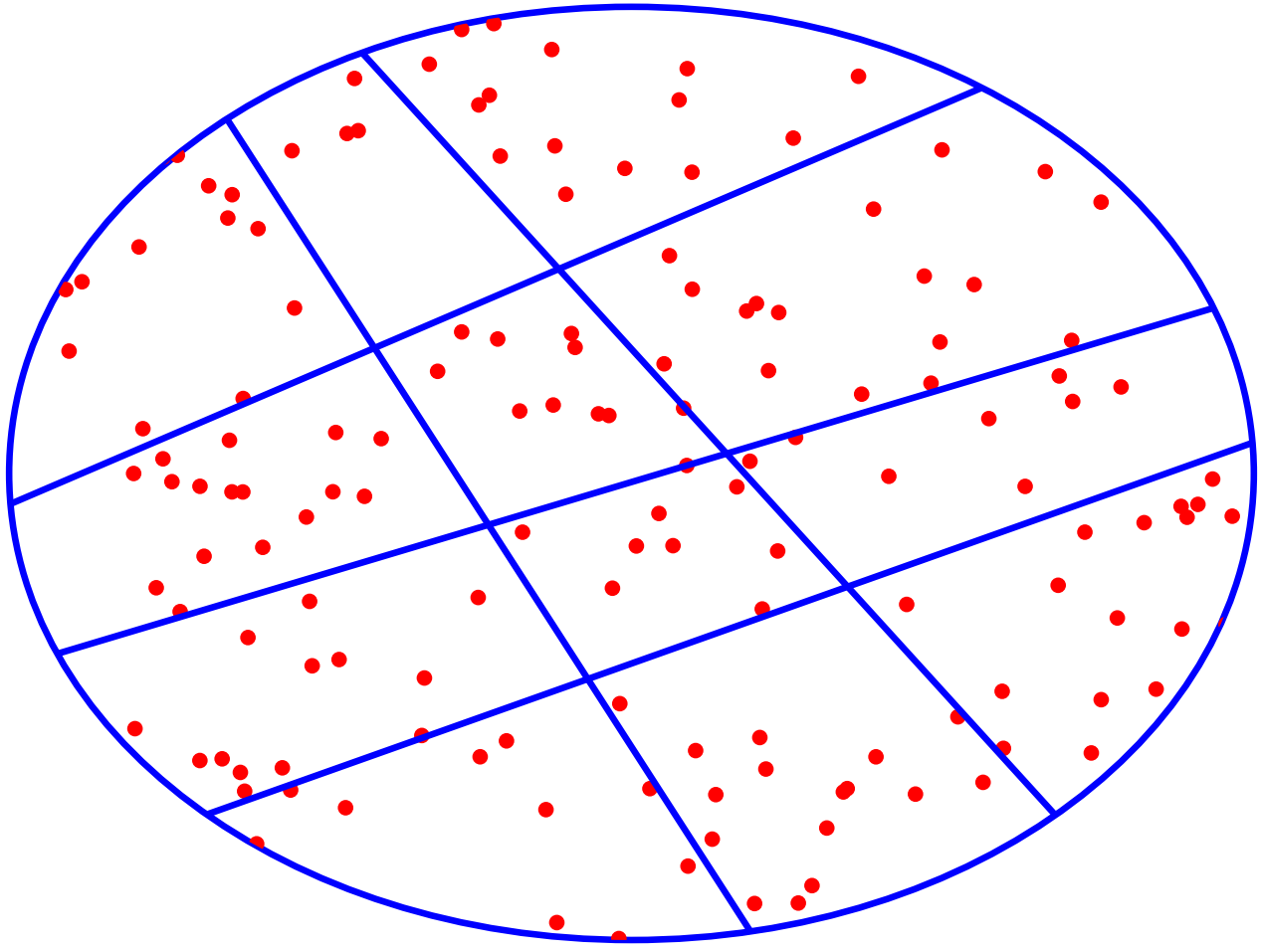
Can we get algorithms with better complexity ?

Delannoy and M; LICS-23 Yes, for query complexity with respect to SAT Oracles

UniSamp: Partitioning is all you Need

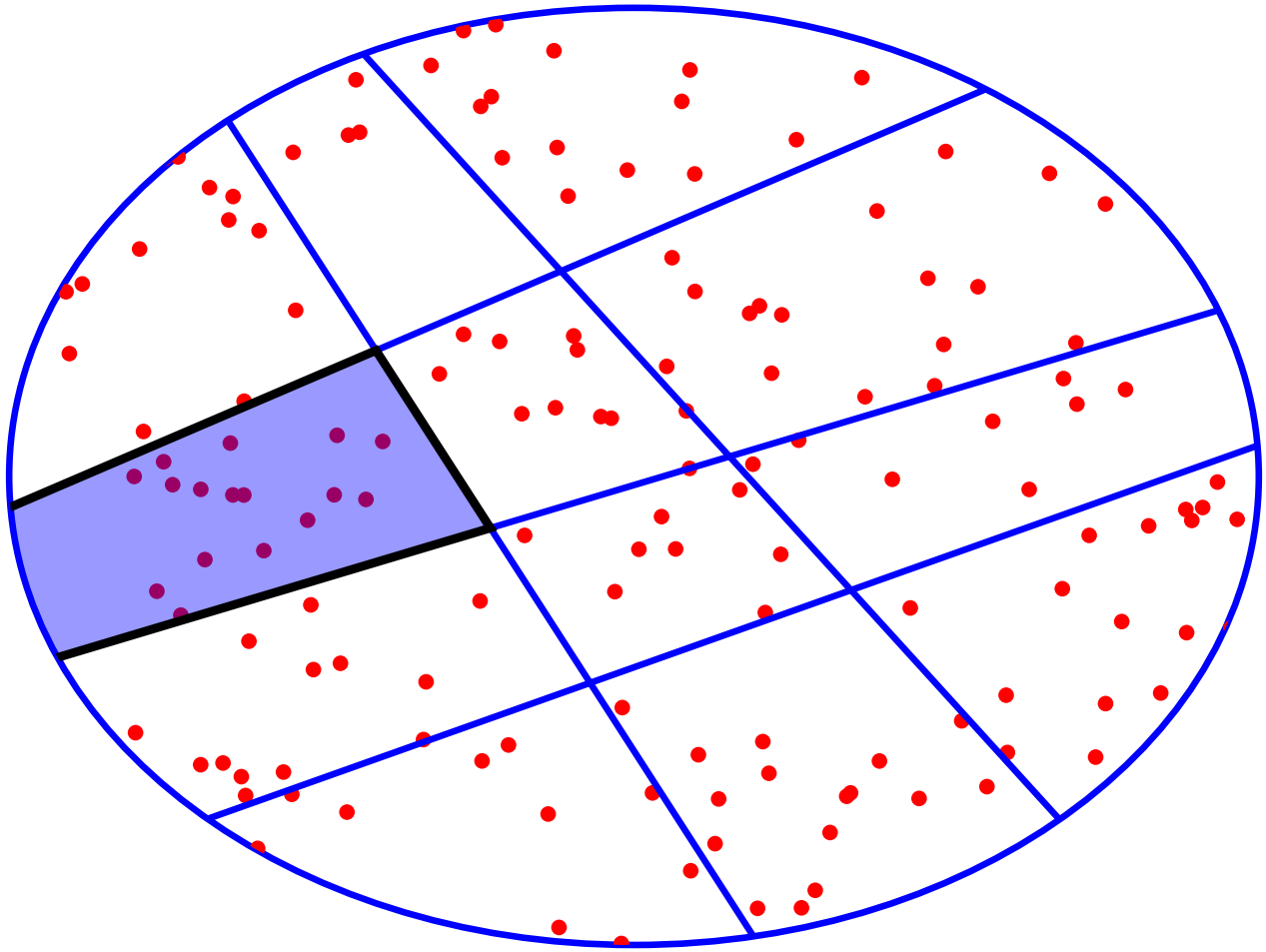


UniSamp: Partitioning is all you Need



UniSamp: Partitioning is all you Need

Pick a random cell



Enumerate Solutions in a randomly chosen Cell
Pick a solution uniformly at random from enumerated solutions

Challenges

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

- Rely on the power of 3-wise independent hash functions
- Can be implemented with XOR-based Hash functions

Challenge 2 How many cells?

- $\mathrm{thresh} = 200$
- Number of cells: $2^m \approx \frac{|\mathsf{sol}(\varphi)|}{\mathrm{thresh}}$
- Invoke an approximate counter to get an estimate of $|\mathsf{sol}(\varphi)|$

Challenges

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

- Rely on the power of 3-wise independent hash functions
- Can be implemented with XOR-based Hash functions

Challenge 2 How many cells?

- $\mathrm{thresh} = 200$
- Number of cells: $2^m \approx \frac{|\mathsf{sol}(\varphi)|}{\mathrm{thresh}}$
- Invoke an approximate counter to get an estimate of $|\mathsf{sol}(\varphi)|$

Complexity Analysis: One call to counter, followed by enumeration of thresh solutions

- NP Oracle: $\mathcal{O}(\log n + n)$
- SAT Oracle: $\mathcal{O}(\log n)$ Enumeration requires only thresh calls to SAT oracle

Predicting Algorithm Performance and Comparison

	NP Oracle	SAT Oracle
JVV	$n^2 \log n$	$n^2 \log n$
UniSamp	$n + \log n$	$\log n$

Predicting Algorithm Performance and Comparison

	NP Oracle	SAT Oracle
JVV	$n^2 \log n$	$n^2 \log n$
UniSamp	$n + \log n$	$\log n$

Benchmark	Variables	Clauses	JVV Time (sec/sample)	UniSamp Time (sec/sample)
s27_new_3_2	17	31	0.38	0.02
4step	165	418	timeout	1.03
s420_15_7	366	994	timeout	1.56
GuidanceService2.sk	715	2181	timeout	1.00
GuidanceService.sk	988	3088	timeout	1.06
prod-2s	1113	4974	timeout	6.91
90-16-2-q	1216	1920	timeout	3254.91
UserServiceImpl.sk	1509	5009	timeout	0.72
blasted_squaring41	4185	13599	timeout	3348.58
ProcessBean.sk	4768	14458	timeout	5.87
doublyLinkedList.sk	6890	26918	timeout	6.28
01B-1	9159	39959	timeout	345.59
107.sk_3_90	8948	40147	timeout	7.51
LoginService.sk	8200	26689	timeout	7.13
sort.sk_8_52	12125	49611	timeout	11.46
enqueueSeqSK.sk	16466	58515	timeout	29.71
compress.sk	44901	166948	timeout	123.19
hash-4	188361	755882	timeout	315.34

SAT Oracle model predicts algorithm performance and facilitates algorithm comparison

Outline

Objectives

- Predicting algorithm performance
- Facilitating algorithm comparison and improvement
- Encounter beauty of mathematical patterns

Sampling

- SAT Oracle captures algorithm's performance and allowed comparison
- Enabled development of new algorithm

Outline

Objectives

- Predicting algorithm performance
- Facilitating algorithm comparison and improvement
- Encounter beauty of mathematical patterns

Sampling

- SAT Oracle captures algorithm's performance and allowed comparison
- Enabled development of new algorithm

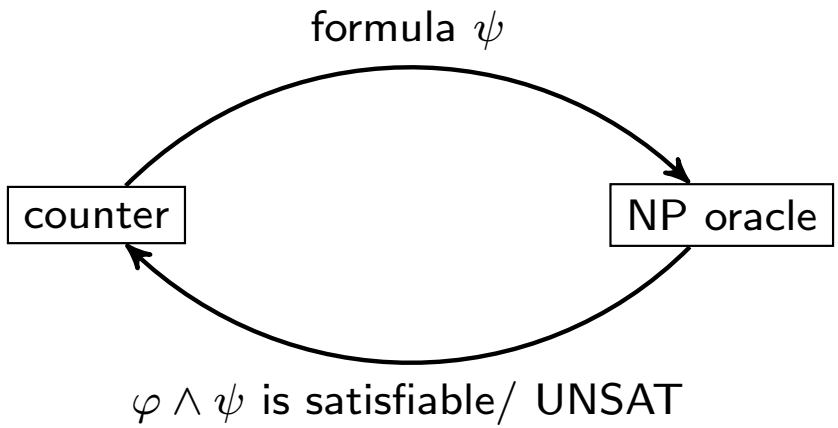
Counting: Given φ, ε and δ, output c such that

$$\Pr\left[\frac{|\mathsf{sol}(\varphi)|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathsf{sol}(\varphi)|\right] \geq 1-\delta$$

- NP-hard, so need a way to perform computations with NP oracles

The Computation Model for NP Queries

Stockmeyer's Model, 1983: **Input**: A formula φ, Decision oracle



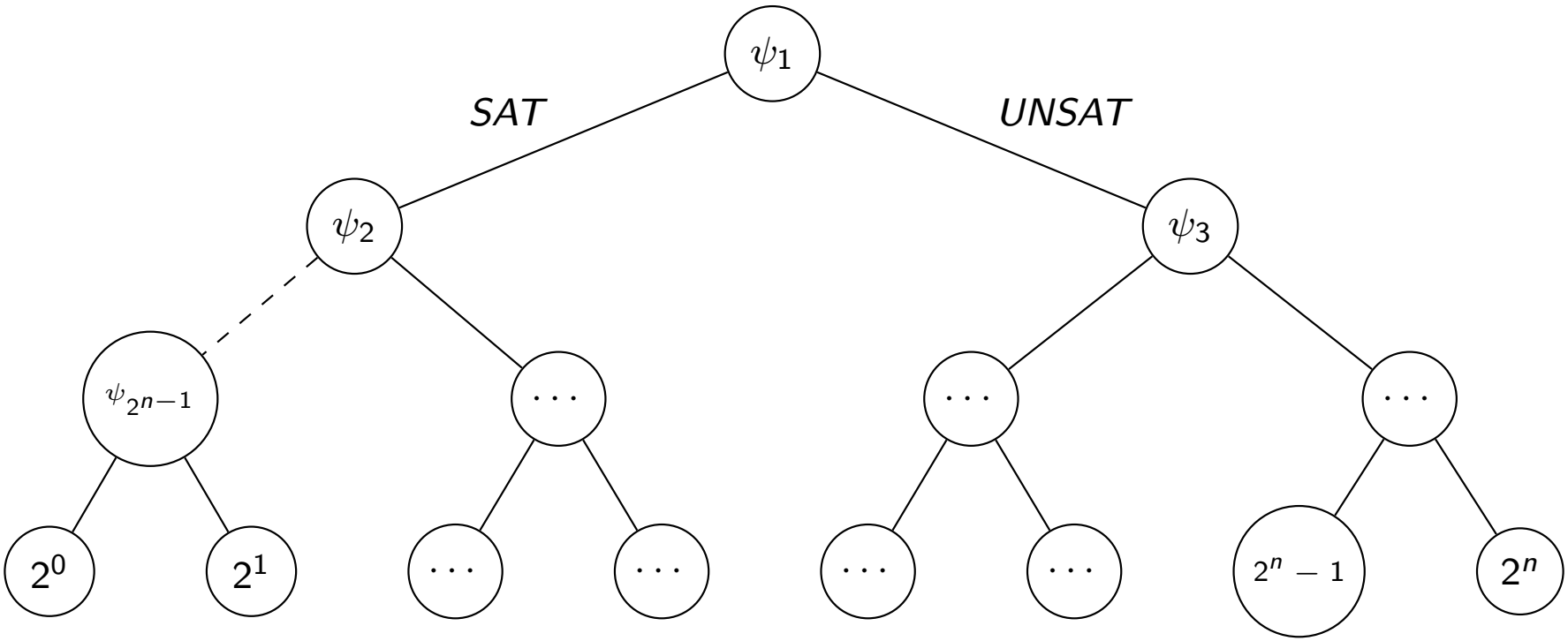
Adaptivity: Queries can be adaptive, i.e., ψ can be chosen aribtrarily depending on the answer of previous answers

Impact All the known approximate counting techniques fit in Stockmeyer's model

- Their queries are of the $\varphi \wedge \psi$ for input formula φ

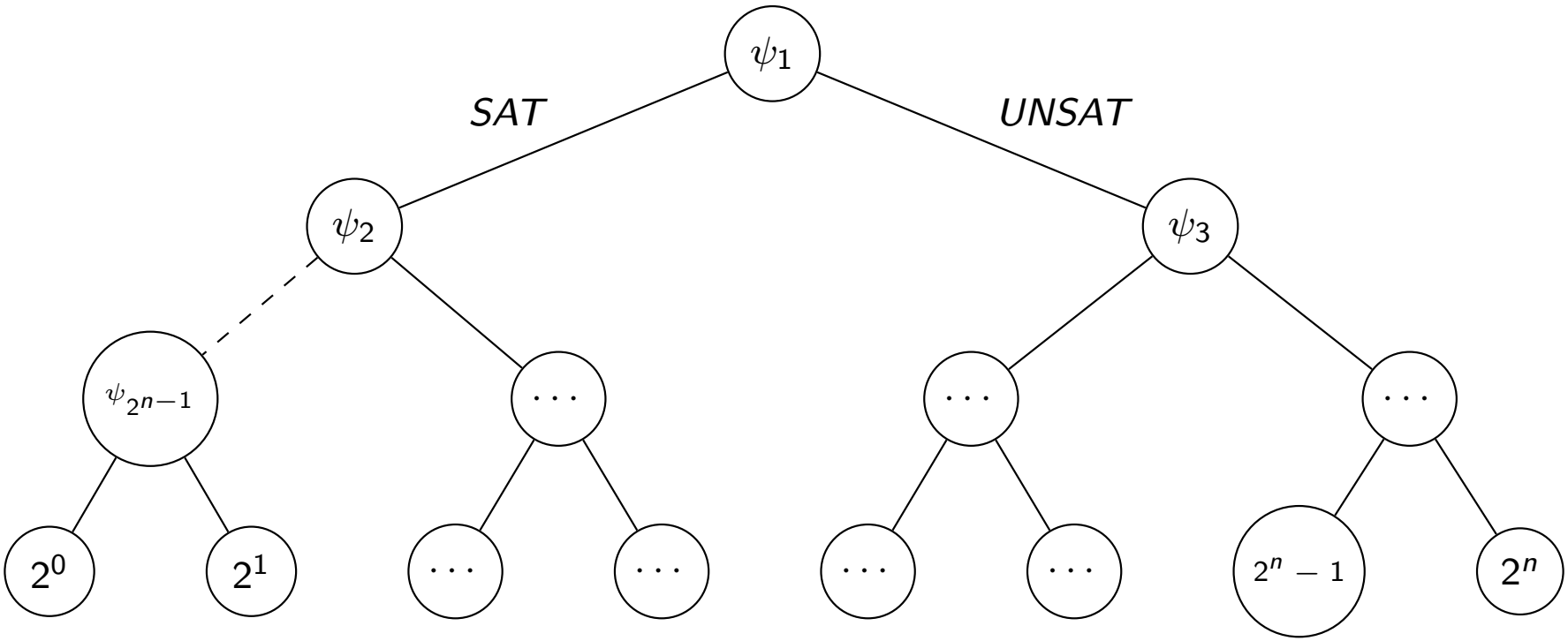
Lower and Upper Bounds

Stockmeyer, 1983: $\Theta(\log n)$ to NP Oracle are Necessary and Sufficient



Lower and Upper Bounds

Stockmeyer, 1983: $\Theta(\log n)$ to NP Oracle are Necessary and Sufficient



Since count $\in \{0, 1, \ldots, 2^n\}$, therefore a 2-approximation counter must be able to return $\log(2^n) = n$ possible values.

Thus, $2^q \geq n$, i.e., $q \geq \log n$.

From Theory to Practice

While Stockmeyer's method had the query complexity of $O(\log n)$, it did not lend itself to practical methods

ApproxMC (2013–Present): Design of a practical approximate model counter

- NP Oracle: $O(n \log n)$ queries
- SAT Oracle: $O(\log n)$ queries

A Key Observation: Complexity of Query $|\mathrm{sol}(\varphi)| \leq \mathrm{thresh}$ for some constant thresh

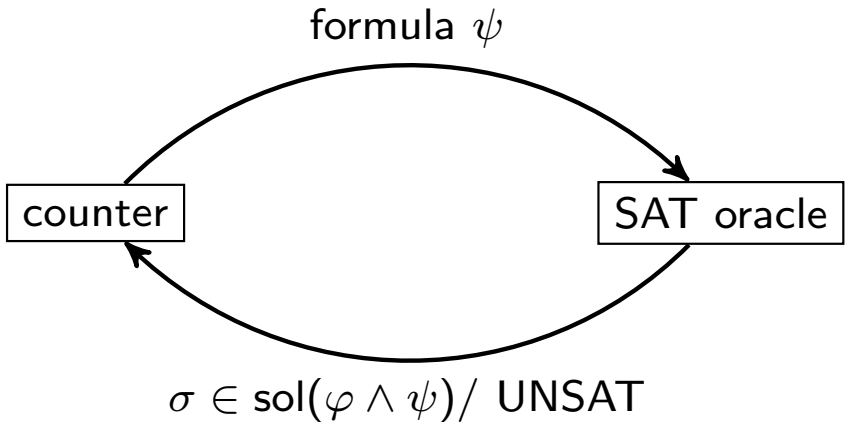
- Enumerate solutions one by one until we have found thresh solutions
  - NP Oracle: $O(n \cdot \mathrm{thresh})$
  - SAT Oracle: $O(\mathrm{thresh})$

SAT Oracle guides algorithm improvement

Stockmeyer's Model's Impact and SAT Oracle

- All the known approximate counting techniques fit in Stockmeyer's model
  - But, all these methods replace NP oracle calls with SAT Solver in practice.

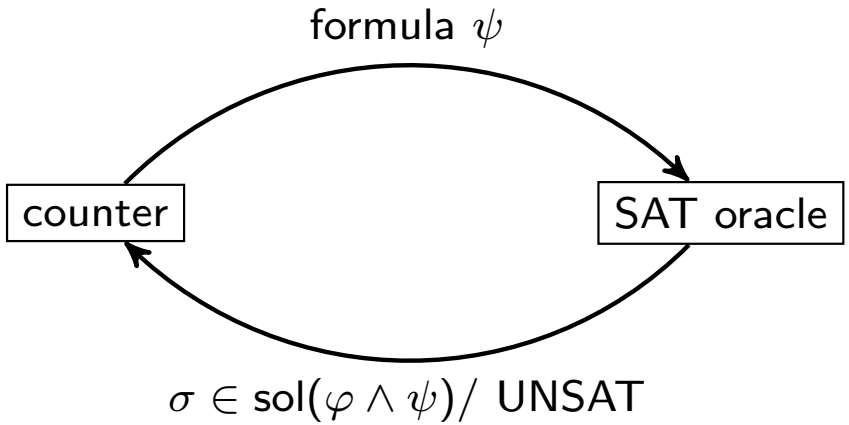
A Natural Extension to SAT Oracle



Stockmeyer's Model's Impact and SAT Oracle

- All the known approximate counting techniques fit in Stockmeyer's model
  - But, all these methods replace NP oracle calls with SAT Solver in practice.

A Natural Extension to SAT Oracle

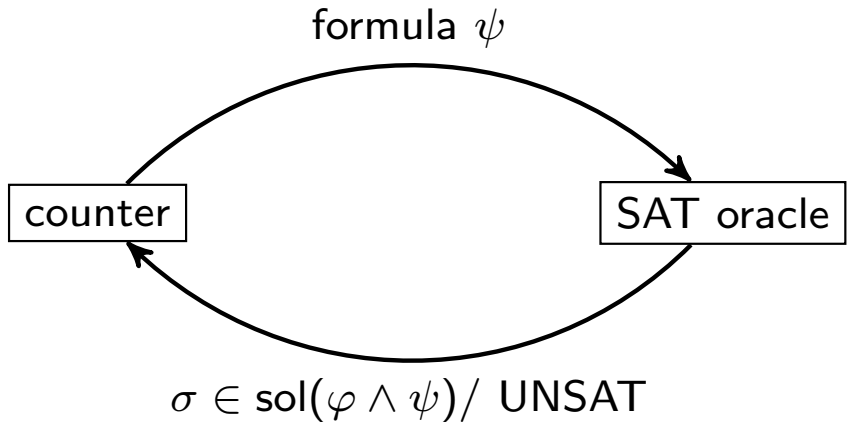


- How many SAT Oracle Queries are Necessary?
  - Upper bounds for NP oracles are also upper bounds for SAT Oracles
  - Lower bounds for NP oracles don't imply lower bounds for SAT Oracles
- Beyond curiosity: Is it possible to somehow use satisfying assignment to our advantage in designing future queries?

Stockmeyer's Model's Impact and SAT Oracle

- All the known approximate counting techniques fit in Stockmeyer's model
  - But, all these methods replace NP oracle calls with SAT Solver in practice.

A Natural Extension to SAT Oracle

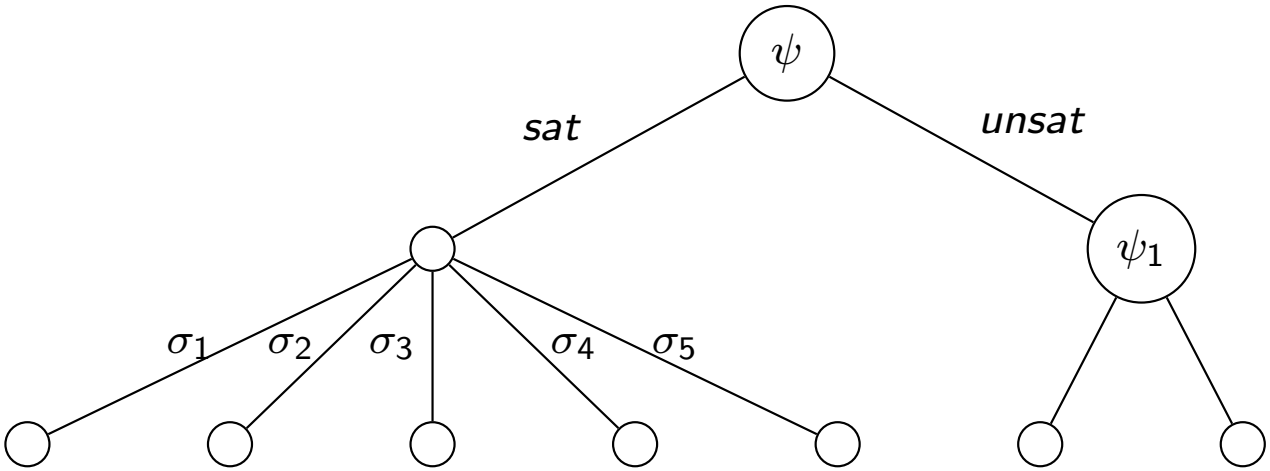


- How many SAT Oracle Queries are Necessary?
  - Upper bounds for NP oracles are also upper bounds for SAT Oracles
  - Lower bounds for NP oracles don't imply lower bounds for SAT Oracles
- Beyond curiosity: Is it possible to somehow use satisfying assignment to our advantage in designing future queries?

Theorem (Chakraborty, Chakraborty, Kumar, M.; ICALP-23)

Every approximate counter must makes $\tilde{\Omega}(\log n)$ *queries to SAT oracle*

SAT Oracles: The Limitation of Stockmeyer's Proof Technique



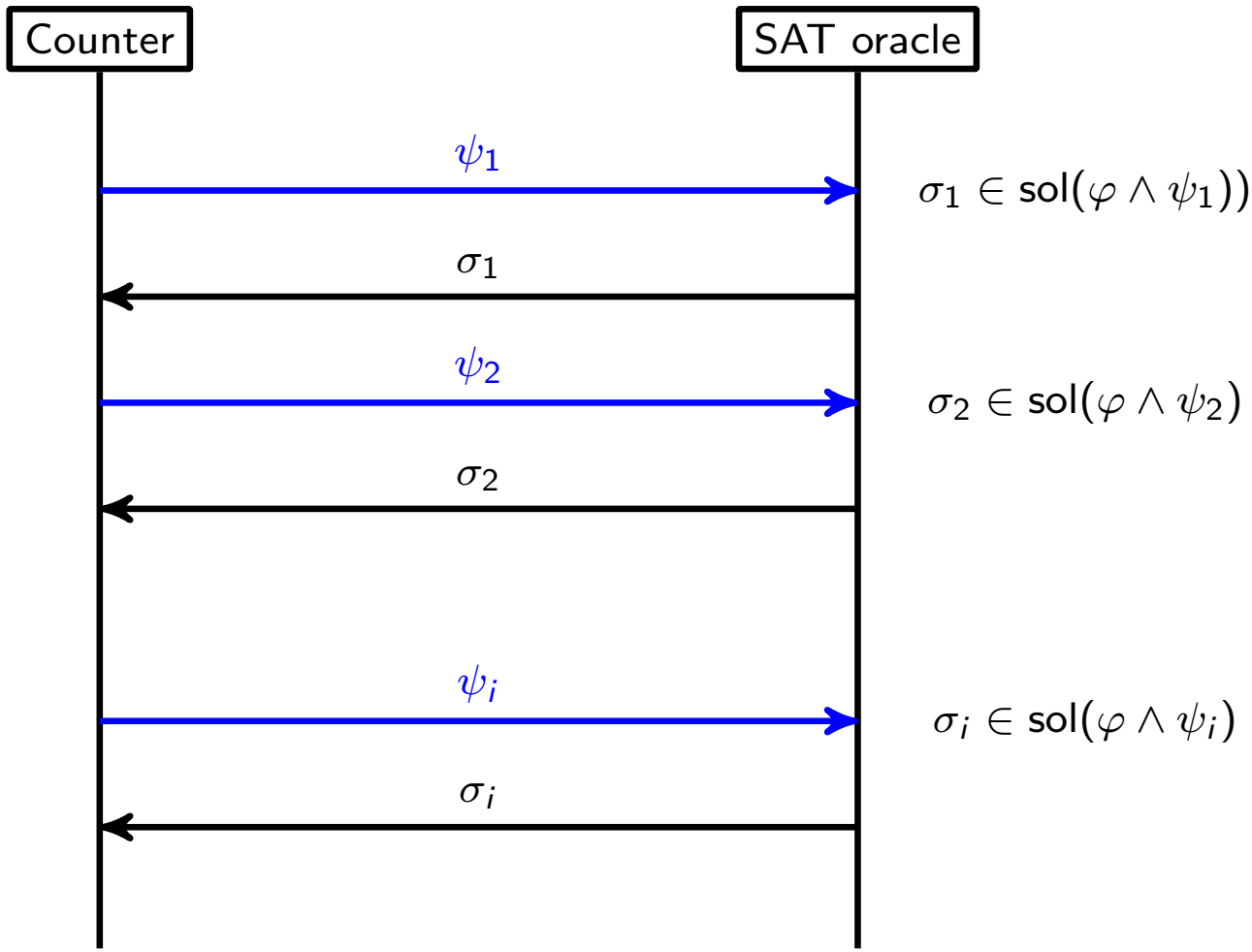
- degree of a node can be as large as the number of satisfying assignments in $\varphi \wedge \psi$, i.e., 2^n.
- Previous approach will give $(2^n)^q \geq n$, i.e., $q > \log n/n$.

Proof Sketch of Our Lower Bound

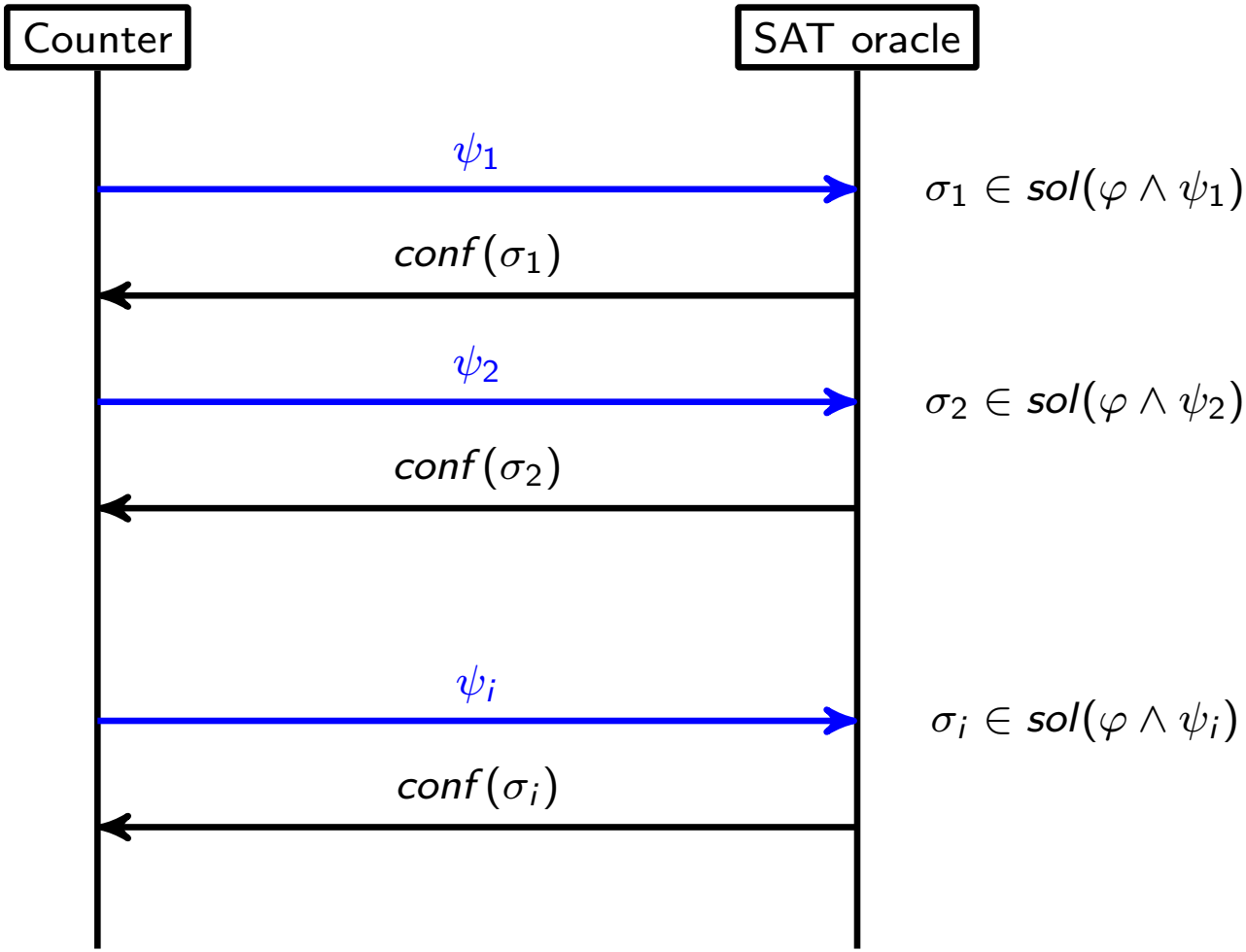
- Focus on a stronger SAT Oracle
  - The SAT Oracle will not return an arbitrary satisfying assignment but satisfying assignment chosen uniformly at random.

- A Restricted class of counting algorithms: Semi-oblivious counters
  - Lower bound for Semi-oblivious counters.
  - Show that they are equivalent to general counters

General Counter



Semi-Oblivious Counter

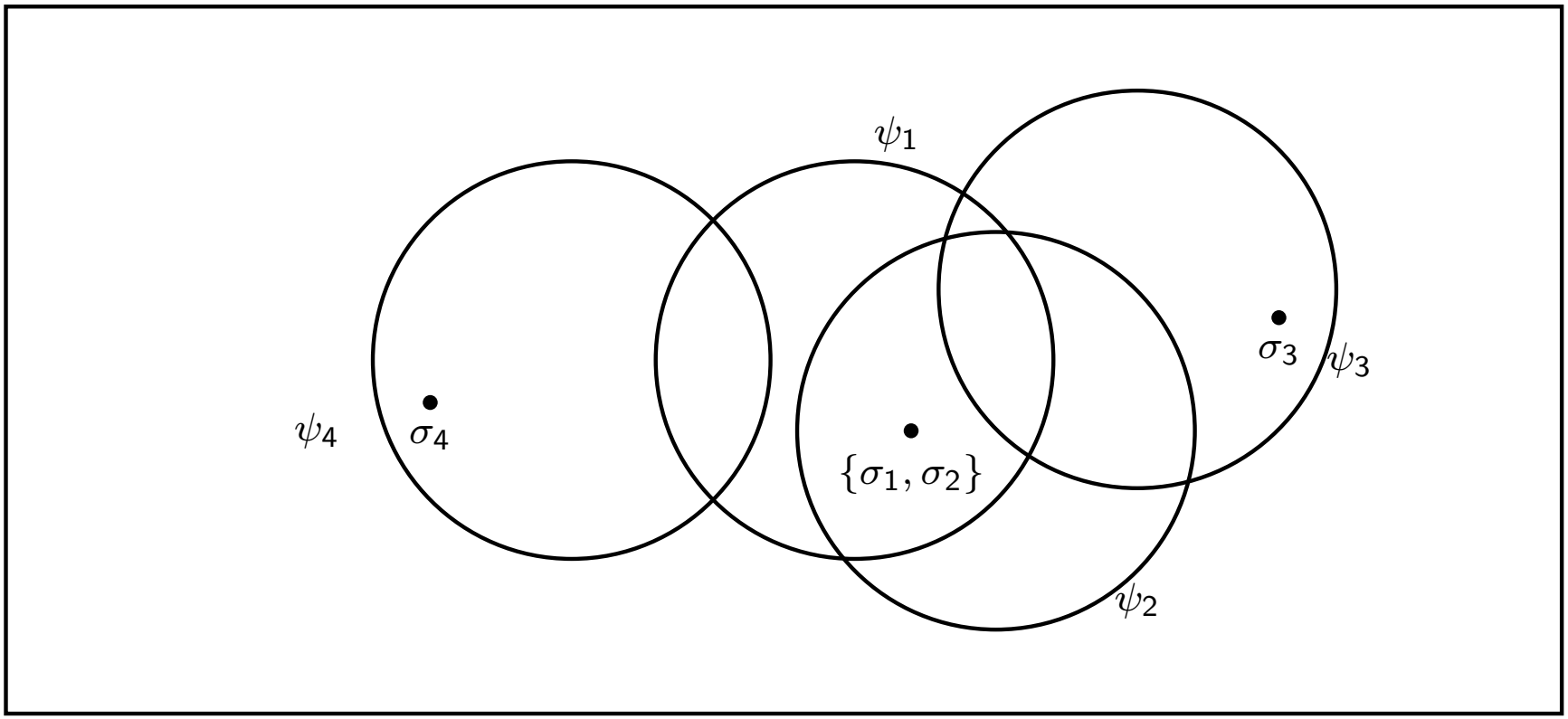


$conf(\sigma_i)$:

- Is $\sigma_i = \sigma_1, \sigma_i = \sigma_2, \ldots, \sigma_i = \sigma_{i-1}$?
- Does $\sigma_i \in sol(\psi_1), \sigma_i \in sol(\psi_2), \ldots, \sigma_i \in sol(\psi_{i-1})$

There are $2(i-1)$ possibilities for $conf(\sigma_i)$

Semi-Oblivious Counter



Information revealed to counter: (1) $\sigma_1 = \sigma_2 \neq \sigma_3 \neq \sigma_4$
(2) $\sigma_1 = \sigma_2 \in \psi_1, \psi_2, \notin \psi_3, \psi_4$; $\sigma_3 \in \psi_3, \notin \psi_1, \psi_2, \psi_4$; $\sigma_4 \in \psi_4, \notin \psi_1, \psi_2, \psi_3$

Achieving Lower Bound

Degree is at most $2^{2i} \leq 2^{2q}$.

If q is the number of queries then

$$(2^{2q})^q \geq \text{number of leaves} \geq n \implies q \geq \sqrt{\log n}$$

Achieving Lower Bound

Degree is at most $2^{2i} \leq 2^{2q}$.

If q is the number of queries then

$$(2^{2q})^q \geq \text{number of leaves} \geq n \implies q \geq \sqrt{\log n}$$

Using techniques from information theory, we improve this lower bound to $\log n$.

Lemma

Any semi-oblivious counter must make $\tilde{\Omega}(\log n)$ queries to a SAT oracle.

Achieving Lower Bound

Degree is at most $2^{2i} \leq 2^{2q}$.

If q is the number of queries then

$$(2^{2q})^q \geq \text{number of leaves} \geq n \implies q \geq \sqrt{\log n}$$

Using techniques from information theory, we improve this lower bound to $\log n$.

Lemma

Any semi-oblivious counter must make $\tilde{\Omega}(\log n)$ queries to a SAT oracle.

Lemma

A semi-oblivious counter can simulate a general counter..

Theorem

Any counter must make $\tilde{\Omega}(\log n)$ queries to a SAT oracle.

Main Lemma (Informal)

History (Transcript) till step i is a sequence:

$$\{(\psi_1, \sigma_1), (\psi_2, \sigma_2), \ldots, (\psi_i, \sigma_i)\}$$

Lemma (Informal)

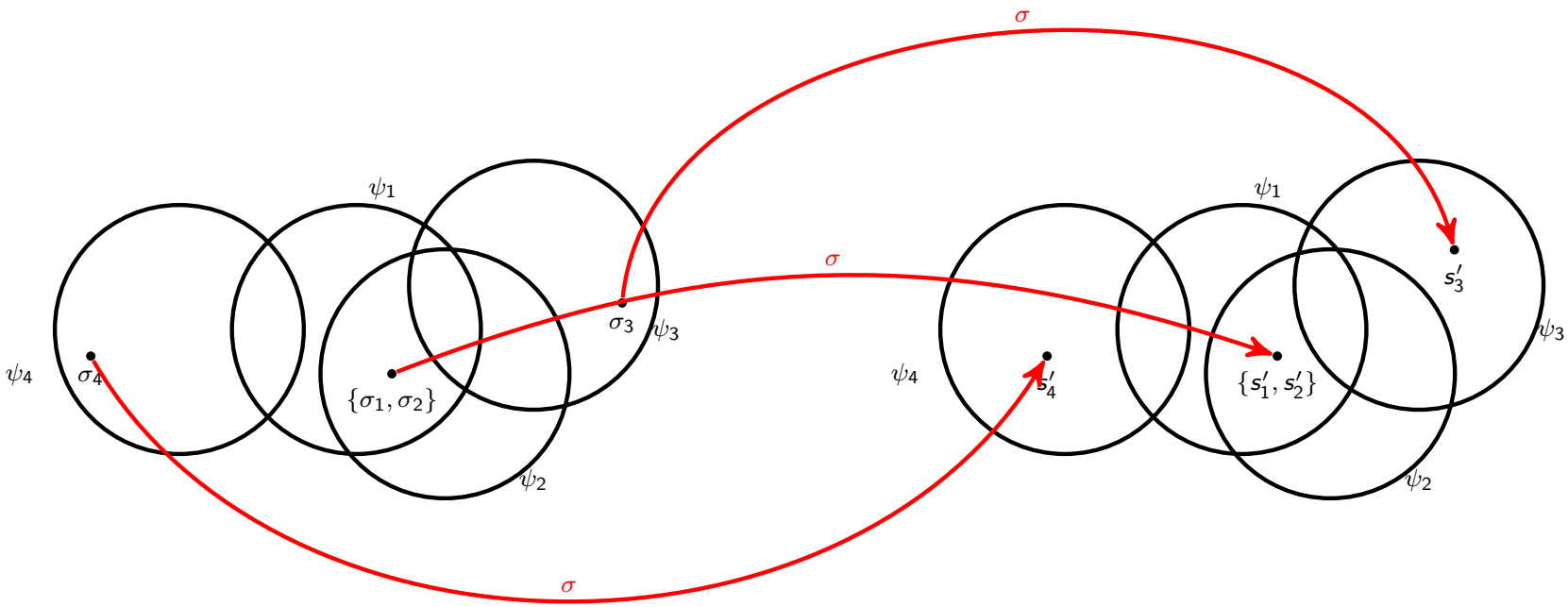
Consider two different histories such that the corresponding sample-configurations are the same. Then there is an isomorphic relabeling $\sigma : \{0,1\}^n \to \{0,1\}^n$ that maps one to other.

So any (general) counter (wlog) should behave in a similar way as long as configurations are the same.

In other words, suffices for the general counter to just see the configurations.

Isomorphic Relabelling

If configurations are the same then histories are equivalent up to relabeling, i.e., mutual information is very small.



$$\sigma(\{(\psi_1, \sigma_1), \ldots, (\psi_i, \sigma_i)\}) = \{(\psi_1, \sigma'_1), \ldots, (\psi_i, \sigma'_i)\}$$

Recap of Results

While Stockmeyer's method had the query complexity of $O(\log n)$, it did not lend itself to practical methods

ApproxMC (2013–Present): Design of a practical approximate model counter

- NP Oracle: $O(n \log n)$ queries
- SAT Oracle: $O(\log n)$ queries

Theorem

Every approximate counter must makes $\tilde{\Omega}(\log n)$ *queries to SAT oracle*

Reflections:

- Facilitating algorithm comparison and improvement
- Development of new techniques

Summary

Objectives of Analysis of Algorithms

- Predicting algorithm performance
- Facilitating algorithm comparison and improvement
- Encounter beauty of mathematical patterns

Sampling

- SAT Oracle captures algorithm's performance and allowed comparison
- Enabled development of new algorithm

Summary

Objectives of Analysis of Algorithms

- Predicting algorithm performance
- Facilitating algorithm comparison and improvement
- Encounter beauty of mathematical patterns

Sampling

- SAT Oracle captures algorithm's performance and allowed comparison
- Enabled development of new algorithm

Counting

- SAT Oracle captures algorithm's performance and allowed comparison
- Enabled development of new algorithm
- New Technical Challenges; more mathematical beauty to explore

Algorithm analysis should be based on oracles that capture the physical reality

Where do we go from here?

We have barely started: The Entire Polynomial Hierarchy is there to explore

MaxSAT: Given a unsatisfiable formula, what is the maximum number of clauses that can be satisfied?

- $\log n$ calls to NP oracle are necessary and sufficient
- What about SAT Oracle?

- Not just *mere curiosity*
  - Existing MaxSAT solvers rely on the underlying SAT solvers. so upper bounds can lead to practical runtime improvements

Where do we go from here?

We have barely started: The Entire Polynomial Hierarchy is there to explore

MaxSAT: Given a unsatisfiable formula, what is the maximum number of clauses that can be satisfied?

- $\log n$ calls to NP oracle are necessary and sufficient
- What about SAT Oracle?

- Not just *mere curiosity*
  - Existing MaxSAT solvers rely on the underlying SAT solvers. so upper bounds can lead to practical runtime improvements

Finding relationship between $P^{SAT[1]}$ versus $P^{SAT[\log n]}$ vs $P^{SAT[n]}$

Where do we go from here?

We have barely started: The Entire Polynomial Hierarchy is there to explore

MaxSAT: Given a unsatisfiable formula, what is the maximum number of clauses that can be satisfied?

- $\log n$ calls to NP oracle are necessary and sufficient
- What about SAT Oracle?

- Not just *mere curiosity*
  - Existing MaxSAT solvers rely on the underlying SAT solvers. so upper bounds can lead to practical runtime improvements

Finding relationship between $P^{SAT[1]}$ versus $P^{SAT[\log n]}$ vs $P^{SAT[n]}$

SAT Solvers also provide proofs. What's the power of SAT Oracle with Proofs?

Where do we go from here?

We have barely started: The Entire Polynomial Hierarchy is there to explore

MaxSAT: Given a unsatisfiable formula, what is the maximum number of clauses that can be satisfied?

- $\log n$ calls to NP oracle are necessary and sufficient
- What about SAT Oracle?

- Not just *mere curiosity*
  - Existing MaxSAT solvers rely on the underlying SAT solvers. so upper bounds can lead to practical runtime improvements

Finding relationship between $P^{SAT[1]}$ versus $P^{SAT[\log n]}$ vs $P^{SAT[n]}$

SAT Solvers also provide proofs. What's the power of SAT Oracle with Proofs?

Algorithm analysis should be based on oracles that capture the physical reality