Assessing the Quality of Binomial Samplers: A Statistical Distance Framework

Uddalok Sarkar,
Indian Statistical Institute

Sourav Chakraborty,
Indian Statistical Institute

Kuldeep S. Meel
Georgia Institute of Technology

What is the program verification?

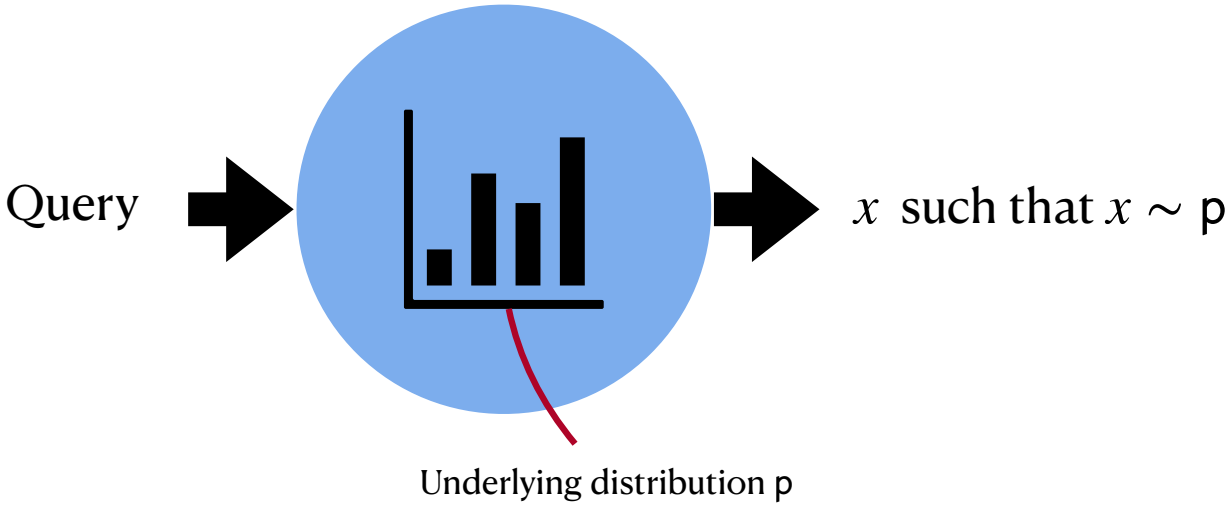
Program verification is the process of formally proving that a computer program meets its intended specifications or requirements. It aims to demonstrate, with mathematical rigor, that a program's behavior aligns with its desired functionality, ensuring correctness and reliability. Formal verification is a method that uses mathematical techniques to prove that a system or design behaves as expected.

What is a randomized program?

A randomized program is a computer program that uses randomness (or pseudo-randomness) in its execution.

What is a Sampler?

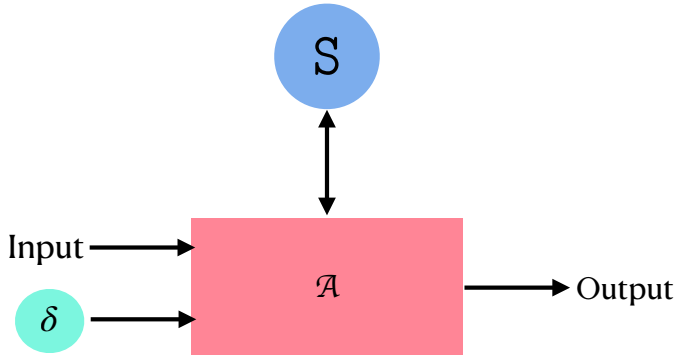
A Sampler is a probabilistic program that generates samples on demand from an underlying distribution.



Sampler : The heart of Randomized Algorithms

Randomized Algorithms needs:

1. Source of Randomness (S)
2. Confidence parameter (δ)

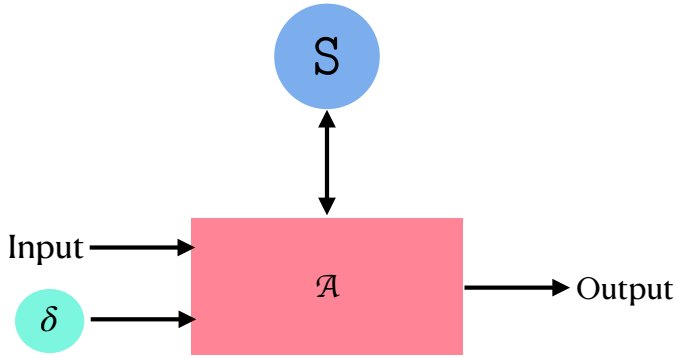


A **(Monte Carlo) Randomized Algorithm** uses randomness in its computation and provides a correct output with probability at least $1 - \delta$.

Sampler : The heart of Randomized Algorithms

Randomized Algorithms needs:

1. Source of Randomness (S)
2. Confidence parameter (δ)



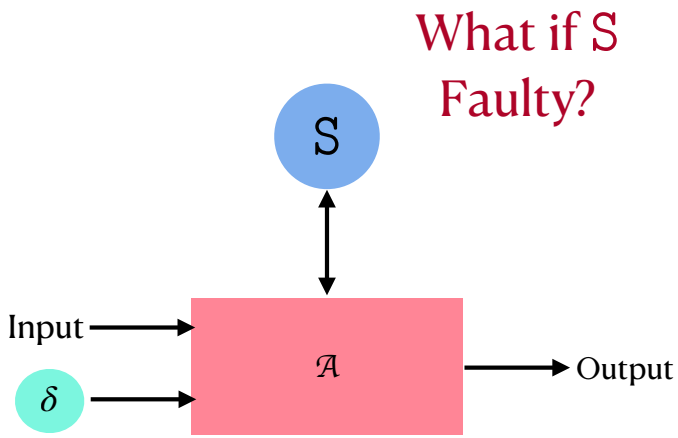
A **(Monte Carlo) Randomized Algorithm** uses randomness in its computation and provides a correct output with probability at least $1 - \delta$.

Assumes S is reliable.

Sampler : The heart of Randomized Algorithms

Randomized Algorithms needs:

1. Source of Randomness (S)
2. Confidence parameter (δ)



A **(Monte Carlo) Randomized Algorithm** uses randomness in its computation and provides a correct output with probability at least $1 - \delta$.

Assumes S is reliable.

Sampler : The heart of Randomized Algorithms

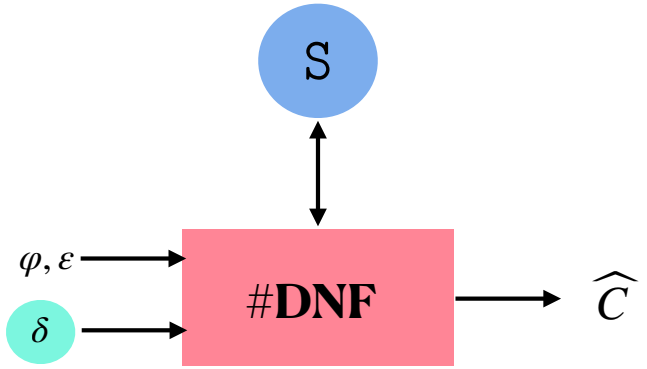
DNF Counting (#DNF)

DNF formula: Disjunction of Conjunctions

$$\varphi = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$$

$$sol(\varphi) = \{110,101,111,011\}$$

Compute $|sol(\varphi)|$



$$(1-\varepsilon)\,|sol(\varphi)| \leq \widehat{C} \leq (1+\varepsilon)\,|sol(\varphi)|$$

#**DNF** is a #P-Hard problem : Practicality requires randomization [MVC18, SACMO23]

A detour: A story of a DNF Counting Algorithm

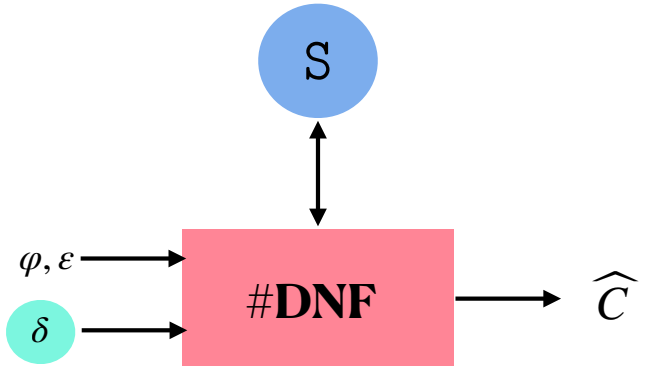
DNF Counting (#DNF)

DNF formula: Disjunction of Conjunctions

$$\varphi = F_1 \vee F_2 \vee F_3 \vee \ldots \vee F_m$$

Say, $\mathcal{S}_i =$ **satisfying assignments of** F_i

Then. $sol(\varphi) = \cup_{i=1}^{k} \mathcal{S}_i$



$$(1-\varepsilon)\,|sol(\varphi)| \leq \widehat{C} \leq (1+\varepsilon)\,|sol(\varphi)|$$

So DNF counting is a special case of estimating the union of sets.

In 2021 (Meel-Vinodchandran-C) stumbled upon a "simple" algorithm for estimating the size of union of sets, (called *Delphic Sets*).

Problem Setting - approximate $\#DNF$ counter

Input A Disjunctive Normal Form (DNF) φ over n variables $x_1, \ldots, x_n$.

- $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$.
- Each F_i is AND of literals.

Output Compute the number of assignments to the $x_1, \ldots, x_n$ that satisfy φ.

- Let $\mathcal{R}(\varphi) =$ set of satisfying assignments of φ
- Our focus: (ε, δ)-approximation, that is, output Est, such that

$$Pr\left[(1-\varepsilon)|\mathcal{R}(\varphi)| \leq \mathsf{Est} \leq (1+\varepsilon)|\mathcal{R}(\varphi)|\right] \geq 1-\delta$$

- FPRAS algorithm: Runs in time $\mathrm{poly}(n, m, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$

Problem Setting - approximate $\#DNF$ counter

Input A Disjunctive Normal Form (DNF) φ over n variables $x_1, \ldots, x_n$.

- $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$.
- Each F_i is AND of literals.

Output Compute the number of assignments to the $x_1, \ldots, x_n$ that satisfy φ.

- Let $\mathcal{R}(\varphi) =$ set of satisfying assignments of φ
- Our focus: (ε, δ)-approximation, that is, output Est, such that

$$Pr\left[(1-\varepsilon)|\mathcal{R}(\varphi)| \leq \mathsf{Est} \leq (1+\varepsilon)|\mathcal{R}(\varphi)|\right] \geq 1-\delta$$

- FPRAS algorithm: Runs in time $\mathrm{poly}(n, m, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$

Why this problem? Many practical applications.

Problem Setting - approximate $\#DNF$ counter

Input A Disjunctive Normal Form (DNF) φ over n variables $x_1, \ldots, x_n$.

- $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$.
- Each F_i is AND of literals.

Output Compute the number of assignments to the $x_1, \ldots, x_n$ that satisfy φ.

- Let $\mathcal{R}(\varphi) =$ set of satisfying assignments of φ
- Our focus: (ε, δ)-approximation, that is, output Est, such that

$$Pr\left[(1-\varepsilon)|\mathcal{R}(\varphi)| \leq \mathsf{Est} \leq (1+\varepsilon)|\mathcal{R}(\varphi)|\right] \geq 1-\delta$$

- FPRAS algorithm: Runs in time $\mathrm{poly}(n, m, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$

Why this problem? Many practical applications.

Why approximate counting? Exactly computing $|\mathcal{R}(\varphi)|$ is $\#P$-complete.

Problem Setting - approximate $\#DNF$ counter

Input A Disjunctive Normal Form (DNF) φ over n variables $x_1, \ldots, x_n$.

- $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$.
- Each F_i is AND of literals.

Output Compute the number of assignments to the $x_1, \ldots, x_n$ that satisfy φ.

- Let $\mathcal{R}(\varphi) =$ set of satisfying assignments of φ
- Our focus: (ε, δ)-approximation, that is, output Est, such that

$$Pr\left[(1-\varepsilon)|\mathcal{R}(\varphi)| \leq \mathsf{Est} \leq (1+\varepsilon)|\mathcal{R}(\varphi)|\right] \geq 1-\delta$$

- FPRAS algorithm: Runs in time $\mathrm{poly}(n, m, \frac{1}{\varepsilon}, \log(\frac{1}{\delta}))$

Why this problem? Many practical applications.

Why approximate counting? Exactly computing $|\mathcal{R}(\varphi)|$ is $\#P$-complete.

Objective Design theoretically correct algorithm that is also good in practice.

Rich History of work

- FPRAS for various counting algorithms
  - Stockmeyer (1983) and Sipser (1983)
- FPRAS using MCMC methods
  - Karp and Luby (1983)
  - Karp, Luby and Madras (1989)
- FPRAS using Hash function
  - Chakraborty, Meel and Vardi (2016)
  - Meel, Shrotri and Vardi (2017)

Rich History of work

- FPRAS for various counting algorithms
  - Stockmeyer (1983) and Sipser (1983)
- FPRAS using MCMC methods
  - Karp and Luby (1983)
  - Karp, Luby and Madras (1989)
- FPRAS using Hash function
  - Chakraborty, Meel and Vardi (2016)
  - Meel, Shrotri and Vardi (2017)

Current Status FPRAS algorithms with almost tight theoretical guarantee in time complexity is known.

Rich History of work

- FPRAS for various counting algorithms
  - Stockmeyer (1983) and Sipser (1983)
- FPRAS using MCMC methods
  - Karp and Luby (1983)
  - Karp, Luby and Madras (1989)
- FPRAS using Hash function
  - Chakraborty, Meel and Vardi (2016)
  - Meel, Shrotri and Vardi (2017)

Current Status FPRAS algorithms with almost tight theoretical guarantee in time complexity is known.

Major Challenge The algorithms are practically inefficient - because of the use of hash functions and MCMC methods.

Rich History of work

- FPRAS for various counting algorithms
  - Stockmeyer (1983) and Sipser (1983)
- FPRAS using MCMC methods
  - Karp and Luby (1983)
  - Karp, Luby and Madras (1989)
- FPRAS using Hash function
  - Chakraborty, Meel and Vardi (2016)
  - Meel, Shrotri and Vardi (2017)

Current Status FPRAS algorithms with almost tight theoretical guarantee in time complexity is known.

Major Challenge The algorithms are practically inefficient - because of the use of hash functions and MCMC methods.

Theorem (IJCAI 2023 (Soos-Aggarwal-C-Meel-Obremski)

We presented a new efficient approximate #DNF counter, called pepin, with (nearly) optimal time complexity that outperforms all the existing FPRAS algorithms when run on standard benchmark data

Key Idea

- *#DNF* is actually estimating volume of union of sets.
  - $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$, where each F_i is AND of literals.
  - $\mathcal{R}(\varphi) :=$ set of satisfying assignments of φ
  - Let $\mathcal{S}_i =$ set of all satisfying assignment of F_i

$$\mathcal{R}(\varphi) = \cup_{i=1}^{m} \mathcal{S}_i$$

Key Idea

- *#DNF* is actually estimating volume of union of sets.
  - $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$, where each F_i is AND of literals.
  - $\mathcal{R}(\varphi) :=$ set of satisfying assignments of φ
  - Let $\mathcal{S}_i =$ set of all satisfying assignment of F_i

$$\mathcal{R}(\varphi) = \cup_{i=1}^{m} \mathcal{S}_i$$

- Properties about the sets $\mathcal{S}_i$
  - Easy to compute the size $|\mathcal{S}_i|$
  - Easy to check if an assignment of the variables is in $\mathcal{S}_i$
  - Easy to draw a random element from $\mathcal{S}_i$

Key Idea

- *#DNF* is actually estimating volume of union of sets.
  - $\varphi := \mathsf{F}_1 \vee \mathsf{F}_2 \vee \cdots \vee \mathsf{F}_m$, where each F_i is AND of literals.
  - $\mathcal{R}(\varphi) :=$ set of satisfying assignments of φ
  - Let $\mathcal{S}_i =$ set of all satisfying assignment of F_i

$$\mathcal{R}(\varphi) = \cup_{i=1}^{m} \mathcal{S}_i$$

- Properties about the sets $\mathcal{S}_i$
  - Easy to compute the size $|\mathcal{S}_i|$
  - Easy to check if an assignment of the variables is in $\mathcal{S}_i$
  - Easy to draw a random element from $\mathcal{S}_i$

- pepin is based on an algorithm of Meel, Vinodchandran and Chakraborty (PODS 2021) for estimating volume of union of sets.
  - The Meel, Vinodchandran and Chakraborty (MVC) algorithm is an FPRAS with theoretical guarantees.
  - The algorithm is sampling based (no use of MCMC or hash function) - so expected to be efficient practically.

Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

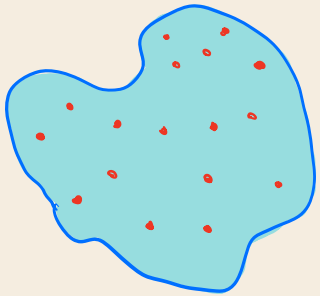
$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

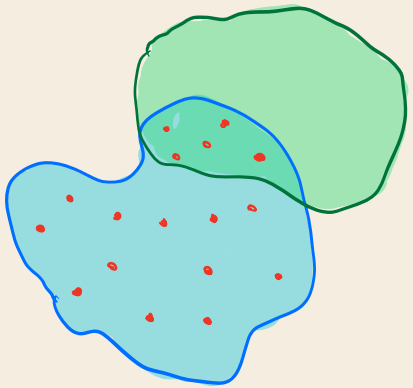


Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

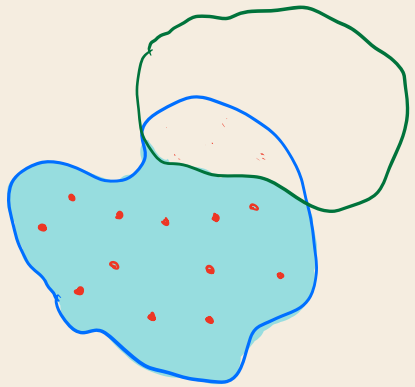


Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

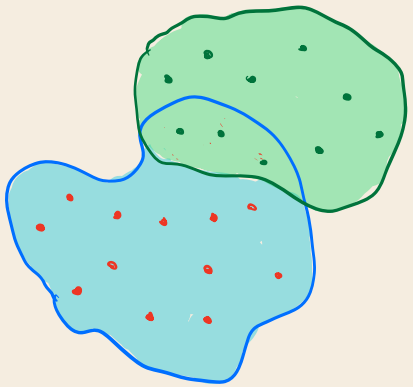


Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.



Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } \left|\cup_i \mathcal{S}_i\right|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

Trick 3 Picking every element in $\mathcal{S}_1$ with probability p

Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } |\cup_i \mathcal{S}_i|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

Trick 3 Picking every element in $\mathcal{S}_1$ with probability p

is same as

- Pick r according to the distribution Binomial($|S_1|, p$)
- Draw r distinct samples from S_1

Meel-Vinodchandran-C algorithm for union of sets

Trick 1 We want to pick every element in $\cup_i \mathcal{S}_i$ independently with probability p. If the final number of elements picked is R, then,

$$\frac{R}{p} \text{ is a "good" estimate of } |\cup_i \mathcal{S}_i|$$

Trick 2 $\mathcal{S}_1 \cup \mathcal{S}_2 = (\mathcal{S}_1 \setminus \mathcal{S}_2) \cup \mathcal{S}_2$.

Trick 3 Picking every element in $\mathcal{S}_1$ with probability p

is same as

- Pick r according to the distribution Binomial$(|S_1|, p)$
- Draw r distinct samples from S_1

Trick 4 If the number r is "large",

- $r \leftarrow$ Binomial$(r, 1/2)$
- Throw each element already picked independently with probability $1/2$
- $p = p/2$

Meel-Vinodchandran-C algorithm for union of sets

Algorithm MVC

1: Thresh $\leftarrow \left(\frac{\log(12/\delta)+\log m}{\varepsilon^2}\right)$; **Initialize:** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
2: **for** $i = 1$ to m **do**
3: **for** all $s \in \mathcal{X}$ **do**
4: **if** $s \models \mathcal{S}_i$ **then** remove s from $\mathcal{X}$
5: $N_i \leftarrow \text{Binomial}(|\mathcal{S}_i|, p)$
6: **while** $N_i + |\mathcal{X}|$ is more than Thresh **do**
7: Remove each element of $\mathcal{X}$ with probability $1/2$
8: $N_i = \text{Binomial}(N_i, 1/2)$ and $p = p/2$
9: $k = 0$
10: **for** $j = 1$ to $N_i \log N_i$ **do**
11: $s \leftarrow$ a random sample from $\mathcal{S}_i$
12: **if** $s \notin \mathcal{X}$ **then**$\mathcal{X}$.Append(s) and $k = k + 1$
13: **if** $k == N_i$ **then break**
14: Output $\frac{|\mathcal{X}|}{p}$

Meel-Vinodchandran-C algorithm for union of sets

Algorithm MVC

1: Thresh $\leftarrow \left(\frac{\log(12/\delta)+\log m}{\varepsilon^2}\right)$; **Initialize:** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
2: **for** $i = 1$ to m **do**
3: **for** all $s \in \mathcal{X}$ **do**
4: **if** $s \models \mathcal{S}_i$ **then** remove s from $\mathcal{X}$
5: $N_i \leftarrow \text{Binomial}(|\mathcal{S}_i|, p)$
6: **while** $N_i + |\mathcal{X}|$ is more than Thresh **do**
7: Remove each element of $\mathcal{X}$ with probability $1/2$
8: $N_i = \text{Binomial}(N_i, 1/2)$ and $p = p/2$
9: $k = 0$
10: **for** $j = 1$ to $N_i \log N_i$ **do**
11: $s \leftarrow$ a random sample from $\mathcal{S}_i$
12: **if** $s \notin \mathcal{X}$ **then**$\mathcal{X}$.Append(s) and $k = k + 1$
13: **if** $k == N_i$ **then break**
14: Output $\frac{|\mathcal{X}|}{p}$

- But when we implemented it the results were erroneous.

Meel-Vinodchandran-C algorithm for union of sets

Algorithm MVC

1: Thresh $\leftarrow \left(\frac{\log(12/\delta)+\log m}{\varepsilon^2}\right)$; **Initialize:** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
2: **for** $i = 1$ to m **do**
3: **for** all $s \in \mathcal{X}$ **do**
4: **if** $s \models \mathcal{S}_i$ **then** remove s from $\mathcal{X}$
5: $N_i \leftarrow \text{Binomial}(|\mathcal{S}_i|, p)$
6: **while** $N_i + |\mathcal{X}|$ is more than Thresh **do**
7: Remove each element of $\mathcal{X}$ with probability $1/2$
8: $N_i = \text{Binomial}(N_i, 1/2)$ and $p = p/2$
9: $k = 0$
10: **for** $j = 1$ to $N_i \log N_i$ **do**
11: $s \leftarrow$ a random sample from $\mathcal{S}_i$
12: **if** $s \notin \mathcal{X}$ **then**$\mathcal{X}$.Append(s) and $k = k + 1$
13: **if** $k == N_i$ **then break**
14: Output $\frac{|\mathcal{X}|}{p}$

- But when we implemented it the results were erroneous.
- Why was it?

Challenges faced in Implementation

Algorithm MVC

1: Thresh $\leftarrow \left(\frac{\log(12/\delta)+\log m}{\varepsilon^2}\right)$; **Initialize:** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
2: **for** $i = 1$ to m **do**
3: **for** all $s \in \mathcal{X}$ **do**
4: **if** $s \models \mathcal{S}_i$ **then** remove s from $\mathcal{X}$
5: $N_i \leftarrow \text{Binomial}(|\mathcal{S}_i|, p)$
6: **while** $N_i + |\mathcal{X}|$ is more than Thresh **do**
7: Remove each element of $\mathcal{X}$ with probability $1/2$
8: $N_i = \text{Binomial}(N_i, 1/2)$ and $p = p/2$
9: $k = 0$
10: **for** $j = 1$ to $N_i \log N_i$ **do**
11: $s \leftarrow$ a random sample from $\mathcal{S}_i$
12: **if** $s \notin \mathcal{X}$ **then**$\mathcal{X}$.Append(s) and $k = k + 1$
13: **if** $k == N_i$ **then break**
14: Output $\frac{|\mathcal{X}|}{p}$

- Drawing from Binomial(N, p) has arbitrary precision issues when $Np << 1$.

Challenges faced in Implementation

Algorithm MVC

1: Thresh $\leftarrow \left(\frac{\log(12/\delta)+\log m}{\varepsilon^2}\right)$; **Initialize:** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
2: **for** $i = 1$ to m **do**
3: **for** all $s \in \mathcal{X}$ **do**
4: **if** $s \models \mathcal{S}_i$ **then** remove s from $\mathcal{X}$
5: $N_i \leftarrow \text{Binomial}(|\mathcal{S}_i|, p)$
6: **while** $N_i + |\mathcal{X}|$ is more than Thresh **do**
7: Remove each element of $\mathcal{X}$ with probability $1/2$
8: $N_i = \text{Binomial}(N_i, 1/2)$ and $p = p/2$
9: $k = 0$
10: **for** $j = 1$ to $N_i \log N_i$ **do**
11: $s \leftarrow$ a random sample from $\mathcal{S}_i$
12: **if** $s \notin \mathcal{X}$ **then**$\mathcal{X}$.Append(s) and $k = k + 1$
13: **if** $k == N_i$ **then break**
14: Output $\frac{|\mathcal{X}|}{p}$

- Drawing from Binomial(N, p) has arbitrary precision issues when $Np << 1$.
- Drawing and storing large number of samples from $\mathcal{S}_i$.

Challenges faced in Implementation

Algorithm MVC

1: Thresh $\leftarrow \left(\frac{\log(12/\delta)+\log m}{\varepsilon^2}\right)$; **Initialize:** $p \leftarrow 1$; $\mathcal{X} \leftarrow \emptyset$
2: **for** $i = 1$ to m **do**
3: **for** all $s \in \mathcal{X}$ **do**
4: **if** $s \models \mathcal{S}_i$ **then** remove s from $\mathcal{X}$
5: $N_i \leftarrow \text{Binomial}(|\mathcal{S}_i|, p)$
6: **while** $N_i + |\mathcal{X}|$ is more than Thresh **do**
7: Remove each element of $\mathcal{X}$ with probability $1/2$
8: $N_i = \text{Binomial}(N_i, 1/2)$ and $p = p/2$
9: $k = 0$
10: **for** $j = 1$ to $N_i \log N_i$ **do**
11: $s \leftarrow$ a random sample from $\mathcal{S}_i$
12: **if** $s \notin \mathcal{X}$ **then**$\mathcal{X}$.Append(s) and $k = k + 1$
13: **if** $k == N_i$ **then break**
14: Output $\frac{|\mathcal{X}|}{p}$

- Drawing from Binomial(N, p) has arbitrary precision issues when $Np << 1$.
- Drawing and storing large number of samples from $\mathcal{S}_i$.

Tackling of Challenges and Engineering Enhancements

Challenge 1: Drawing from Binomial(N, p) has arbitrary precision issues.

- Using a mixture of Poisson, Normal and Bernoulli distributions to "approximate" Binomial distribution and changing the algorithm to handle the extra error incurred.

Tackling of Challenges and Engineering Enhancements

Challenge 1: Drawing from Binomial(N, p) has arbitrary precision issues.

- Using a mixture of Poisson, Normal and Bernoulli distributions to "approximate" Binomial distribution and changing the algorithm to handle the extra error incurred.

$$\Pr[\mathcal{A} \text{ err when sampling from distribution } \mathrm{p}] \leq \Pr[\mathcal{A} \text{ err when sampling from distribution } \mathrm{q}] + D(\mathrm{p}, \mathrm{q})$$

Tackling of Challenges and Engineering Enhancements

Challenge 1: Drawing from Binomial(N, p) has arbitrary precision issues.

- Using a mixture of Poisson, Normal and Bernoulli distributions to "approximate" Binomial distribution and changing the algorithm to handle the extra error incurred.

$$\Pr[\mathcal{A} \text{ err when sampling from distribution } \mathrm{p}] \leq \Pr[\mathcal{A} \text{ err when sampling from distribution } \mathrm{q}] + D(\mathrm{p}, \mathrm{q})$$

Challenge 2: Drawing and storing large number of samples from $\mathcal{S}_i$.

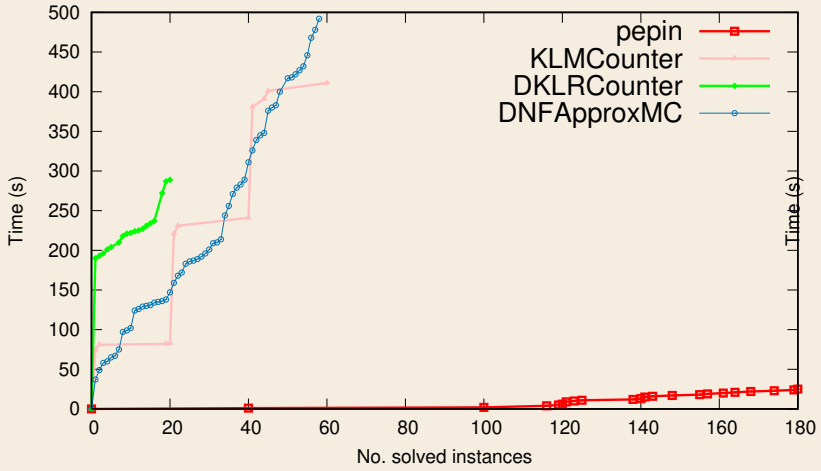
- Storing samples using a Dense Matrix-based Sample Storage system.
- The storage helps in doing a "lazy sampling" to reduce the number of effective samples drawn.

Our final result

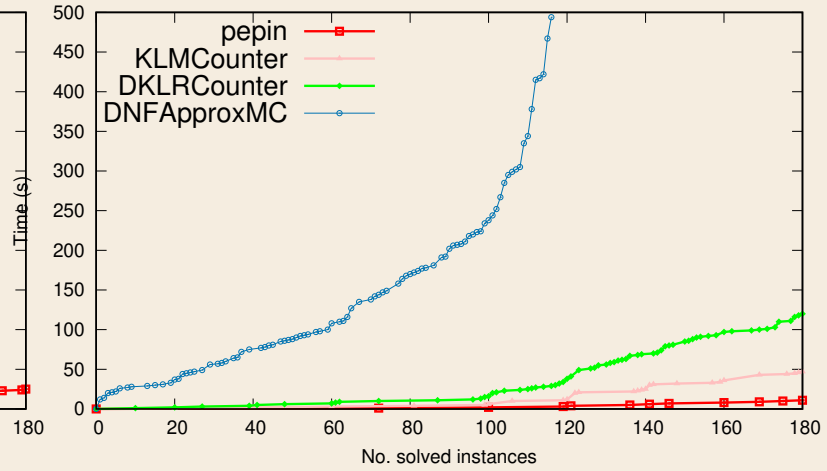
- We suitably modify the MVC algorithm and design our algorithm pepin
- We use various engineering enhancements for handling the samples.
- pepin is an FPRAS with theoretical guarantee.
- We implement pepin and study its performance with other state-of-the-art *#DNF* counters on standard benchmarks.

"*Engineering an Efficient Approximate DNF-Counter*"
Soos-Aggarwal-C-Meel-Obremsk [IJCAI 23]

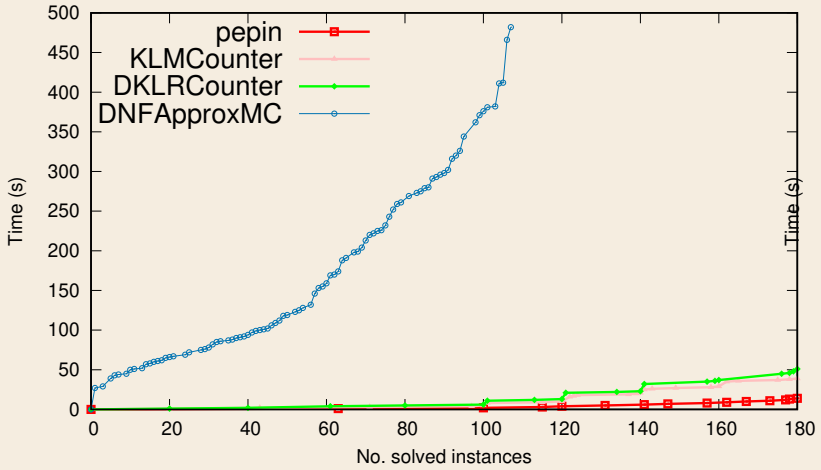
Performance of pepin against the other counters, with different cube widths.



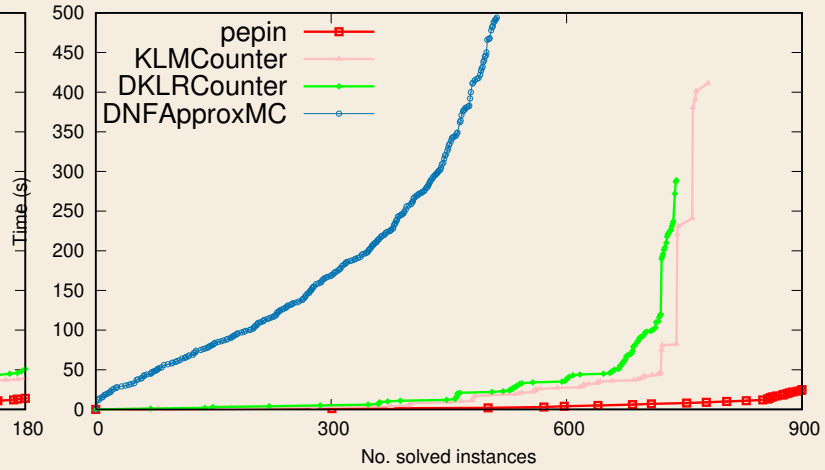
(a) Cube width 3



(b) Cube width 13



(c) Cube width 43



(d) All cube widths

Cube width is $\max_i\{\text{The number of literals in } F_i\}$.

After lots of HARD work ...

- We desgin and implement a FPRAS algorithm pepin for approximate $\#DNF$ counter that has theoretical guarantee and outperforms every other state-of-the-art $\#DNF$ counters in practice.

After lots of HARD work ...

- We desgin and implement a FPRAS algorithm pepin for approximate $\#DNF$ counter that has theoretical guarantee and outperforms every other state-of-the-art $\#DNF$ counters in practice.

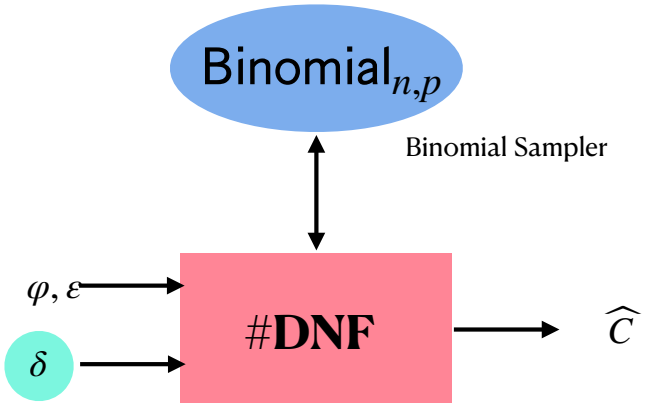
BUT,

WHAT DOES IT SAY ABOUT IMPLEMENTATION OF OTHER RANDOMIZED PROGRAMS?

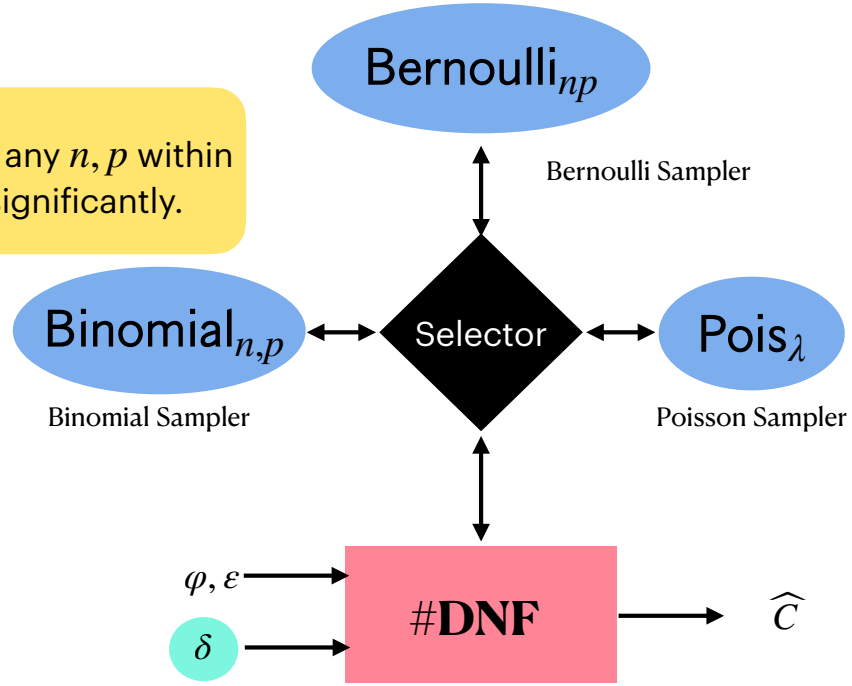
WHY CAN'T WE DESIGN PERFECT BINOMIAL SAMPLERS?

DNF Counter : Theory vs Practice

Because available library Binomial Samplers are not reliable for any n, p within the working domain, their practical implementation differs significantly.

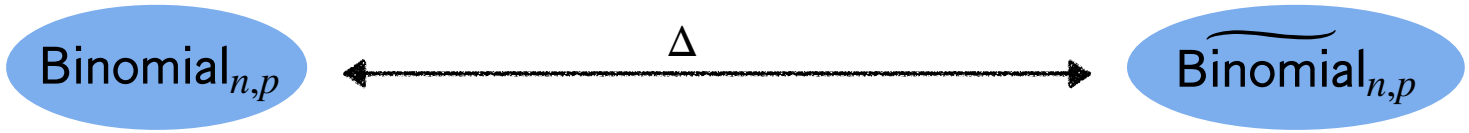


Theoretical Design of #DNF [MVC18]



Practical Design of #DNF [SACMO23]

Why Binomial Samplers are often not Reliable?

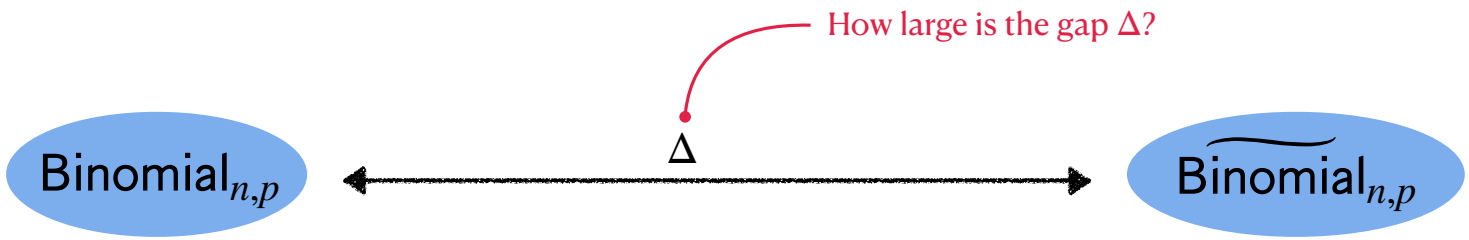


Sampling from Binomial Distribution
is **Hard** and **time consuming**

Library functions usually sample from a distribution
which is close to Binomial distribution

If we don't have a control over Δ we cannot implement randomized
algorithms with proper guarantees

Why Binomial Samplers are often not Reliable?



Sampling from Binomial Distribution
is **Hard** and **time consuming**

Library functions usually sample from a distribution
which is close to Binomial distribution

If we don't have a control over Δ we cannot implement randomized
algorithms with proper guarantees

Issue: We don't have a proper documented handle over the upper bound of Δ

Lets Investigate ...

1. **Sources of Deviation:** We need to identify the key factors contributing to deviations from the true Binomial Distribution.

2. **Quality Metric:** We need a metric to assess the quality of the sampling methods.

3. **Theoretical Bounds:** Can we derive bounds on the quality metric based on the **distribution parameters**: n, p and the **context parameters**: context precision and approximation schemes.

Exact Binomial Samplers

Having access to a sampler whose underlying distribution is exactly $\mathsf{Binomial}_{n,p}$

Bernoulli Sampling

[folklore]

- Toss n coins with bias p.
- count number of Heads.

Time: $O(n)$

Geometric Sampling

[Devroye, 1980]

- $Y_i \sim Geometric(p)$
- Minimum x with $\sum_{i=1}^{x+1} Y_i > n$
- $x \sim \mathsf{b}_{n,p}$

Time: $O(np)$

Sampling using $\mathsf{b}_{n,\frac{1}{2}}$

[Colton-Tsai, 2015]

- Designs efficient access to $\mathsf{Binomial}_{n,\frac{1}{2}}$
- For each bit of p, the algorithm draws samples from $\mathsf{b}_{n,\frac{1}{2}}$

Time: $O(\sqrt{n})$ preprocessing + $O(\log^2 n)$ sampling

Space: $O(\sqrt{n})$

Exact Binomial Samplers

Having access to a sampler whose underlying distribution is exactly $\mathsf{Binomial}_{n,p}$

Bernoulli Sampling

[folklore]

- Toss n coins with bias p.
- count number of Heads.

Time: $O(n)$

Geometric Sampling

[Devroye, 1980]

- $Y_i \sim Geometric(p)$
- Minimum x with $\sum_{i=1}^{x+1} Y_i > n$
- $x \sim \mathsf{b}_{n,p}$

Time: $O(np)$

Sampling using $\mathsf{b}_{n,\frac{1}{2}}$

[Colton-Tsai, 2015]

- Designs efficient access to $\mathsf{Binomial}_{n,\frac{1}{2}}$
- For each bit of p, the algorithm draws samples from $\mathsf{b}_{n,\frac{1}{2}}$

Time: $O(\sqrt{n})$ preprocessing + $O(\log^2 n)$ sampling

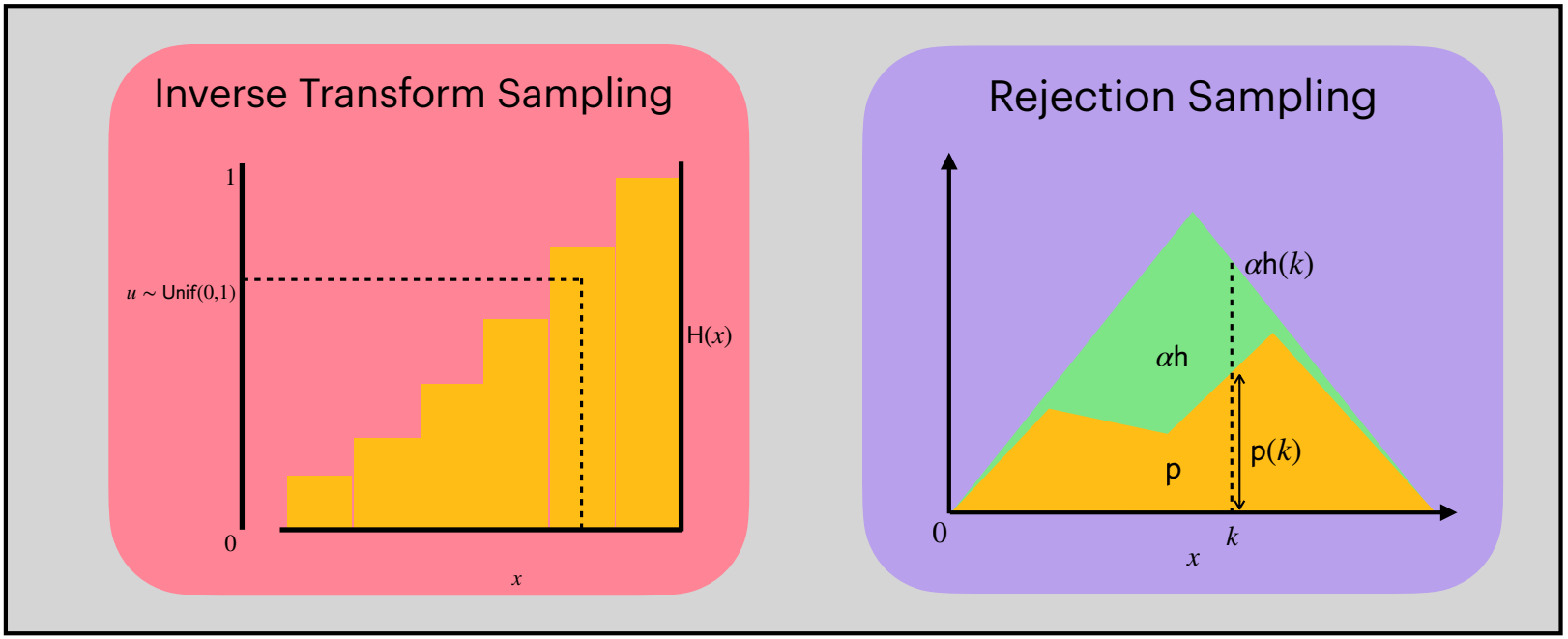
Space: $O(\sqrt{n})$

Other known exact samplers:

1. **Exponential distribution** [Karney, 2016]
2. **Normal distribution** [Karney, 2016]

Practical Binomial Samplers : Suffers from Exact Sampling

All library implemented functions are of this type



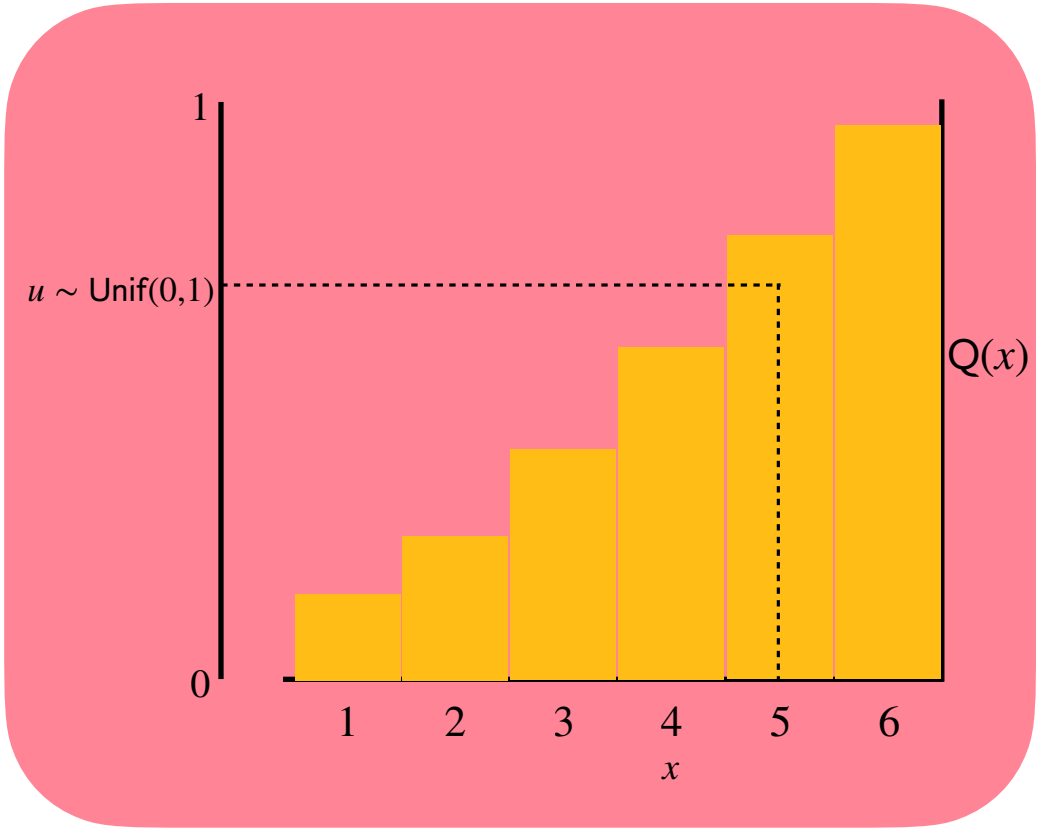
Transformed Rejection Sampling

Inverse Tranform Sampling

Target Distribution: q

1. Compute Cumulative distribution of $q : Q$
2. Draw uniform sample u from $[0,1]$
3. Compute $k \leftarrow \lfloor Q^{-1}(u) \rfloor$

$$k \sim q$$



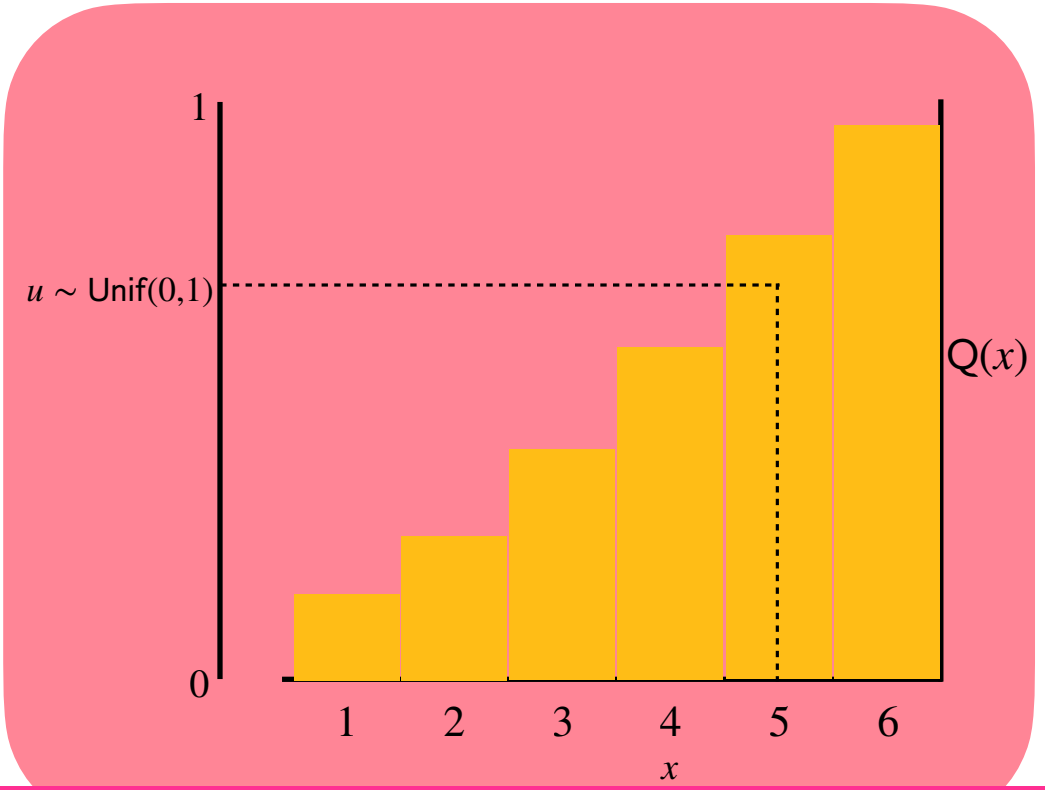
Sampler returns 5

Challenges

Target Distribution: q

1. Compute Cumulative distribution of q : Q
2. Draw uniform sample u from [0,1]
3. Compute $k \leftarrow \lfloor Q^{-1}(u) \rfloor$

$k \sim q$



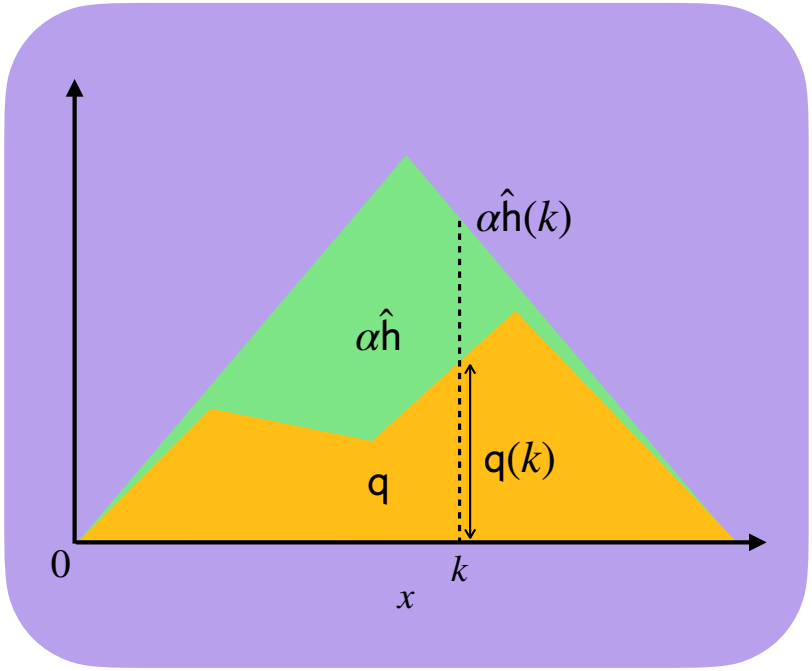
Computing the Q^{-1} function exactly can be very expensive.....

Rejection Sampling

Target Distribution: q

1. Sampling access to a hat distribution: $\hat{\mathsf{h}}$ such that $\forall k, \alpha\hat{\mathsf{h}}(k) \geq \mathsf{q}(k)$, for some $\alpha > 0$
2. Such that is $\hat{\mathsf{H}}^{-1}$ easy to compute
3. Draw $k \sim \hat{\mathsf{h}}$
4. Draw uniform sample u from $[0,1]$
5. Return k, if $u \leq \dfrac{\mathsf{q}(k)}{\alpha\hat{\mathsf{h}}(k)}$
6. Else goto 2

$$k \sim \mathsf{q}$$



Binomial Sampling: Transformed Rejection Approach

Target Distribution: $\text{Binomial}_{n,p}$

1. Consdier Hat distribution: $\hat{\text{h}}$
   such that $\forall k, \alpha\hat{\text{h}}(k) \geq \text{Binomial}_{n,p}(k)$ for some $\alpha > 0$
2. Compute Cumulative distribution of $\hat{\text{h}}$: $\hat{\text{H}}$
3. Draw uniform sample u
4. Compute $k \leftarrow \lfloor \hat{\text{H}}^{-1}(u) \rfloor$
5. Draw uniform sample v
6. Compute $r_k \leftarrow \dfrac{\text{Binomial}_{n,p}(k)}{\alpha\hat{\text{h}}(k)}$
7. Return k, if $v \leq r_k$ Else goto 3

Inverse Transform Sampling Component

Rejection Sampling Component

$k \sim \text{Binomial}_{n,p}$

Binomial Sampling: Transformed Rejection Approach

Target Distribution: $\mathsf{b}_{n,p}$

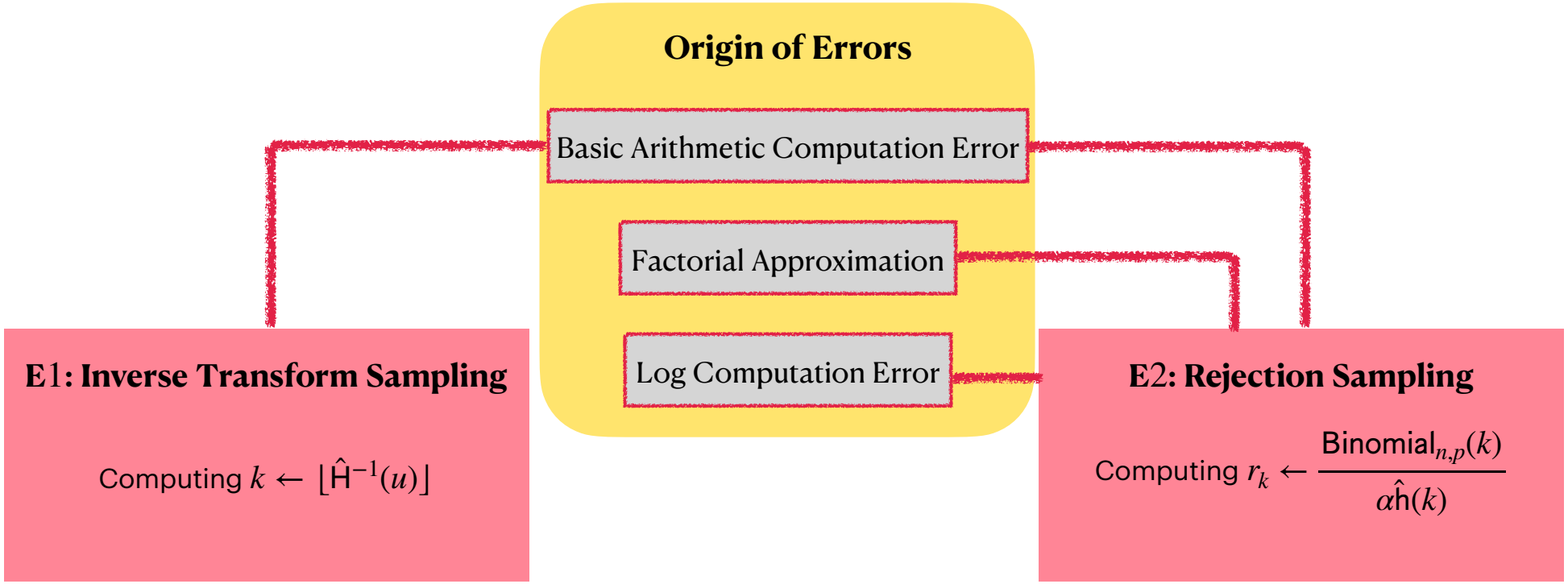
1. Consdier Hat distribution: $\hat{\mathsf{h}}$
   such that $\forall k, \alpha\hat{\mathsf{h}}(k) \geq \mathsf{Binomial}_{n,p}(k)$ for some $\alpha > 0$
2. Compute Cumulative distribution of $\hat{\mathsf{h}}$: $\hat{\mathsf{H}}$
3. Draw uniform sample u
4. Compute $k \leftarrow \lfloor \hat{\mathsf{H}}^{-1}(u) \rfloor$
5. Draw uniform sample v
6. Compute $r_k \leftarrow \dfrac{\mathsf{Binomial}_{n,p}(k)}{\alpha\hat{\mathsf{h}}(k)}$
7. Return k, if $v \leq r_k$ Else goto 3

$k \sim \widetilde{\mathsf{Binomial}}_{n,p}$ such that $\widetilde{\mathsf{Binomial}}_{n,p}$ and $\mathsf{Binomial}_{n,p}$ are **close**

Our Observation

Potential Sources of Errors in Sampling

Potential Errors and their Origin



Potential Errors and their Origin

Working precision: β

Set of all numbers in the context:

$$\mathbb{F} = \{m \cdot 2^e \ : \ 1/2 \leq |m| \leq 1 \text{ and } e \in \mathbb{Z}\}$$

m's precision is β:

$$m = 0.\underbrace{11110101....1}_{\beta}$$

Potential Errors and their Origin

Working precision: β

Set of all numbers in the context:

$$\mathbb{F} = \{m \cdot 2^e \ : \ 1/2 \leq |m| \leq 1 \text{ and } e \in \mathbb{Z}\}$$

m's precision is β:

$$m = 0.\underbrace{11110101....1}_{\beta}$$

Basic Arithmetic Computation Error

For $\circ \in \{+, -, \times, /, \sqrt{}\}$, we cannot ensure $\mathsf{rnd}(a \circ b) = a \circ b$ but,

$$\frac{|\mathsf{rnd}(a \circ b) - a \circ b|}{|a \circ b|} \leq \frac{1}{2^\beta}$$

Potential Errors and their Origin

Working precision: β

Set of all numbers in the context:

$$\mathbb{F} = \{m \cdot 2^e \ : \ 1/2 \leq |m| \leq 1 \text{ and } e \in \mathbb{Z}\}$$

m's precision is β:

$$m = 0.\underbrace{11110101....1}_{\beta}$$

Basic Arithmetic Computation Error

For $\circ \in \{+, -, \times, /, \sqrt{}\}$, we cannot ensure $\mathsf{rnd}(a \circ b) = a \circ b$ but,

$$\frac{|\mathsf{rnd}(a \circ b) - a \circ b|}{|a \circ b|} \leq \frac{1}{2^\beta}$$

Log Computation Error

Log computations require approximations like **Taylor Series Approximations**, or, **AGM method**.

$$|\log^{comp}(x) - \log(x)| \leq \frac{c\beta}{2^\beta}, \quad c > 0$$

Potential Errors and their Origin

Working precision: β

Set of all numbers in the context:

$$\mathbb{F} = \{m \cdot 2^e \ : \ 1/2 \leq |m| \leq 1 \text{ and } e \in \mathbb{Z}\}$$

m's precision is β:

$$m = 0.\underbrace{11110101....1}_{\beta}$$

Basic Arithmetic Computation Error

For $\circ \in \{+, -, \times, /, \sqrt{}\}$, we cannot ensure $\mathsf{rnd}(a \circ b) = a \circ b$ but,

$$\frac{|\mathsf{rnd}(a \circ b) - a \circ b|}{|a \circ b|} \leq \frac{1}{2^\beta}$$

Log Computation Error

Log computations require approximations like **Taylor Series Approximations**, or, **AGM method**.

$$|\log^{comp}(x) - \log(x)| \leq \frac{c\beta}{2^\beta}, \quad c > 0$$

Factorial Approximation

Computing $\mathsf{Binomial}_{n,p}(k)$ requires computing factorials like $n!, k!, (n-k)!$
Approximations like **Stirling's** approximation, **Lanczos'** approximations used.

$$\frac{|n! - comp(n!)|}{n!} \leq \zeta, \quad \text{where } \zeta \text{ depends on the approximation scheme}$$

How to quantify **Quality** of Samplers?

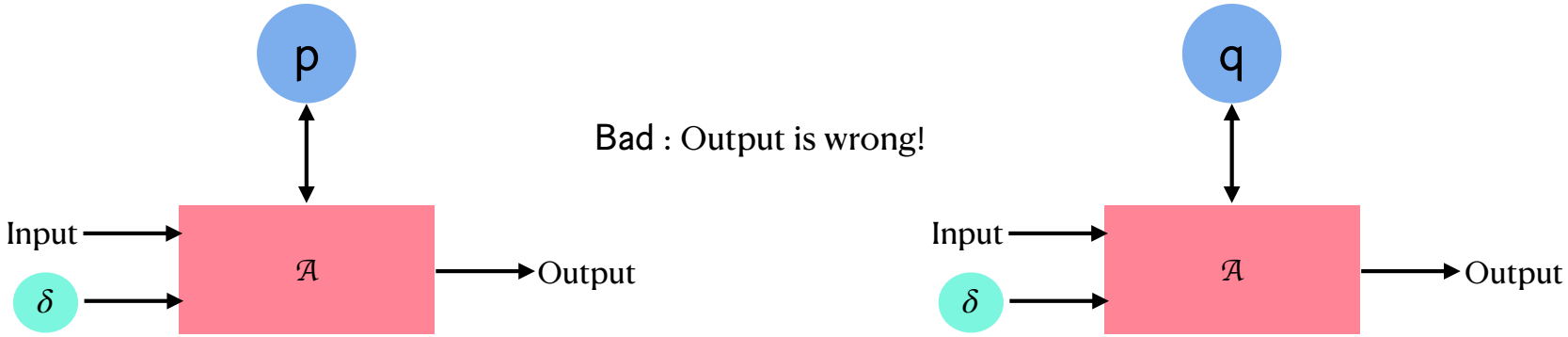
Natural quality metric: **Statistical Distance (D)**

$$\mathbf{D}(\mathsf{p}, \mathsf{q}) = \frac{1}{2} \sum_{x \in \Omega} \left| \mathsf{p}(x) - \mathsf{q}(x) \right|$$

How to quantify **Quality** of Samplers?

Natural quality metric: **Statistical Distance (D)**

$$\mathbf{D}(\mathrm{p},\mathrm{q}) = \frac{1}{2}\sum_{x\in\Omega}\left|\mathrm{p}(x) - \mathrm{q}(x)\right|$$



$$Pr_{\mathrm{q}}(\mathrm{Bad}) \leq Pr_{\mathrm{p}}(\mathrm{Bad}) + \mathbf{D}(\mathrm{p},\mathrm{q})$$

Analyzing **Quality** of Samplers = Bounding Statistical distance

E1: Inverse Transform Sampling

Computing $k \leftarrow \lfloor \hat{H}^{-1}(u) \rfloor$

Errors in this component cause **deviation in hat function** $\hat{h}$

We bound the pointwise deviation of $\hat{h}(k)$

E2: Rejection Sampling

Computing $r_k \leftarrow \dfrac{\text{Binomial}_{n,p}(k)}{\alpha \hat{h}(k)}$

Errors in this component cause **deviation in rejection ratio** r_k

We bound deviation of r_k

Finally Combining both bounds we bound the Statistical distance

Bounding Errors in Inverse Transform Sampling

Computing $k \leftarrow \lfloor \mathsf{H}^{-1}(u) \rfloor$ implies $k \sim \mathsf{h}$

Ideal Scenario

BUT!

H involves **error from arithmetic operations**. Therefore,

$$\left(1 - \frac{1}{2^\beta}\right) \lfloor \mathsf{H}^{-1}(u) \rfloor \leq k \leq \left(1 + \frac{1}{2^\beta}\right) \lfloor \mathsf{H}^{-1}(u) \rfloor$$

Actual Scenario

Bounding Errors in Inverse Transform Sampling

Computing $k \leftarrow \lfloor \mathsf{H}^{-1}(u) \rfloor$ implies $k \sim \mathsf{h}$

BUT!

H involves **error from arithmetic operations**. Therefore,

$$\left(1 - \frac{1}{2^\beta}\right) \lfloor \mathsf{H}^{-1}(u) \rfloor \leq k \leq \left(1 + \frac{1}{2^\beta}\right) \lfloor \mathsf{H}^{-1}(u) \rfloor$$

Ideal Scenario **Actual Scenario**

$$\lfloor \mathsf{H}^{-1}(u) \rfloor = k \xrightarrow{\text{Effects of Arithmetic Errors}} \text{Computed value of } \lfloor \mathsf{H}^{-1}(u) \rfloor \in \{k-1, k, k+1\}$$

$$\mathsf{h} \xrightarrow{\text{Effects of Arithmetic Errors}} \widetilde{\mathsf{h}} : \text{Deviated hat distribution}$$

Bounding Errors in Inverse Transform Sampling

Computing $k \leftarrow \lfloor \mathsf{H}^{-1}(u) \rfloor$ implies $k \sim \mathsf{h}$

BUT!

H involves **error from arithmetic operations**. Therefore,

$$\left(1 - \frac{1}{2^\beta}\right) \lfloor \mathsf{H}^{-1}(u) \rfloor \leq k \leq \left(1 + \frac{1}{2^\beta}\right) \lfloor \mathsf{H}^{-1}(u) \rfloor$$

Ideal Scenario **Actual Scenario**

$$\lfloor \mathsf{H}^{-1}(u) \rfloor = k \xrightarrow{\text{Effects of Arithmetic Errors}} \text{Computed value of } \lfloor \mathsf{H}^{-1}(u) \rfloor \in \{k-1, k, k+1\}$$

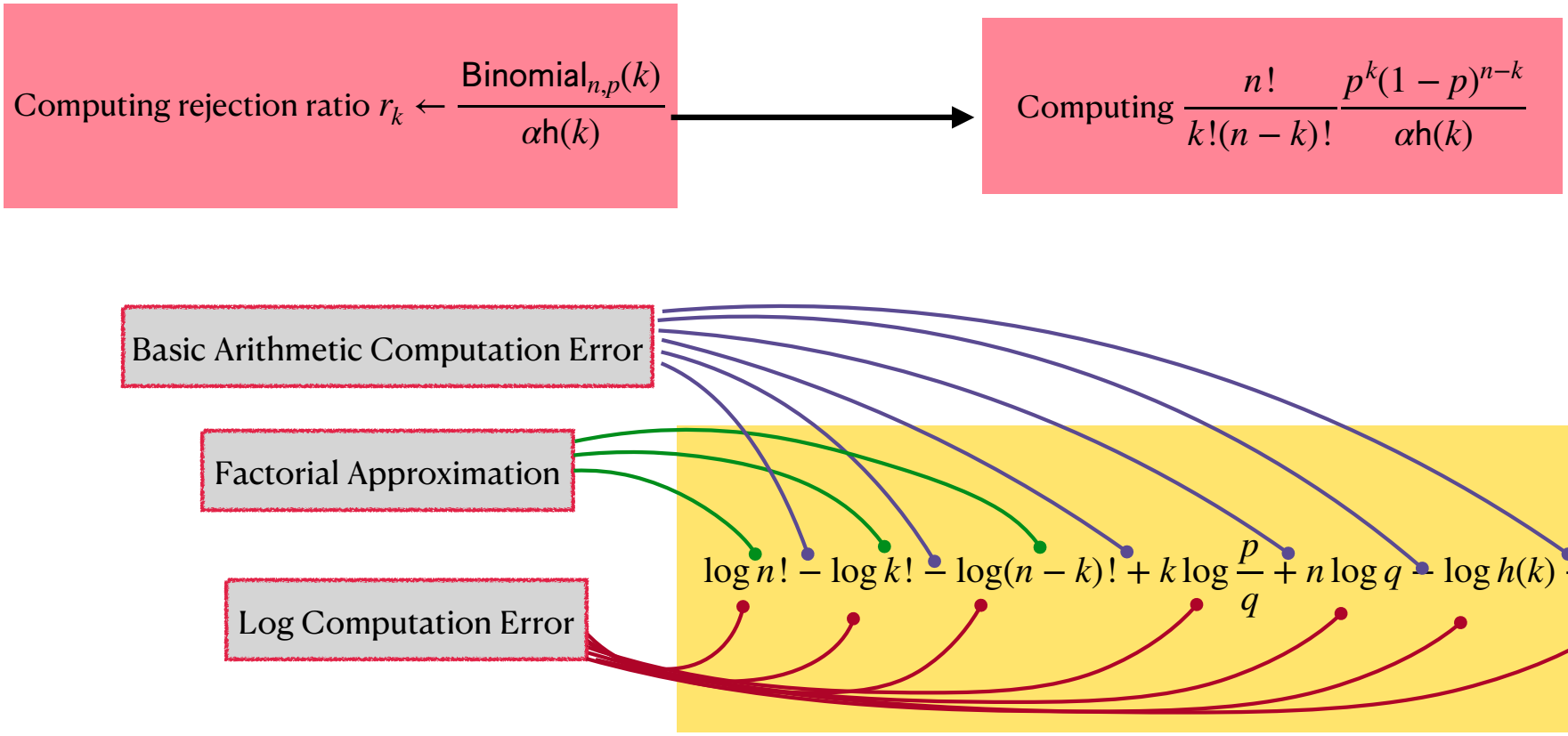
$$\mathsf{h} \xrightarrow{\text{Effects of Arithmetic Errors}} \widetilde{\mathsf{h}} : \text{Deviated hat distribution}$$

Bound 1

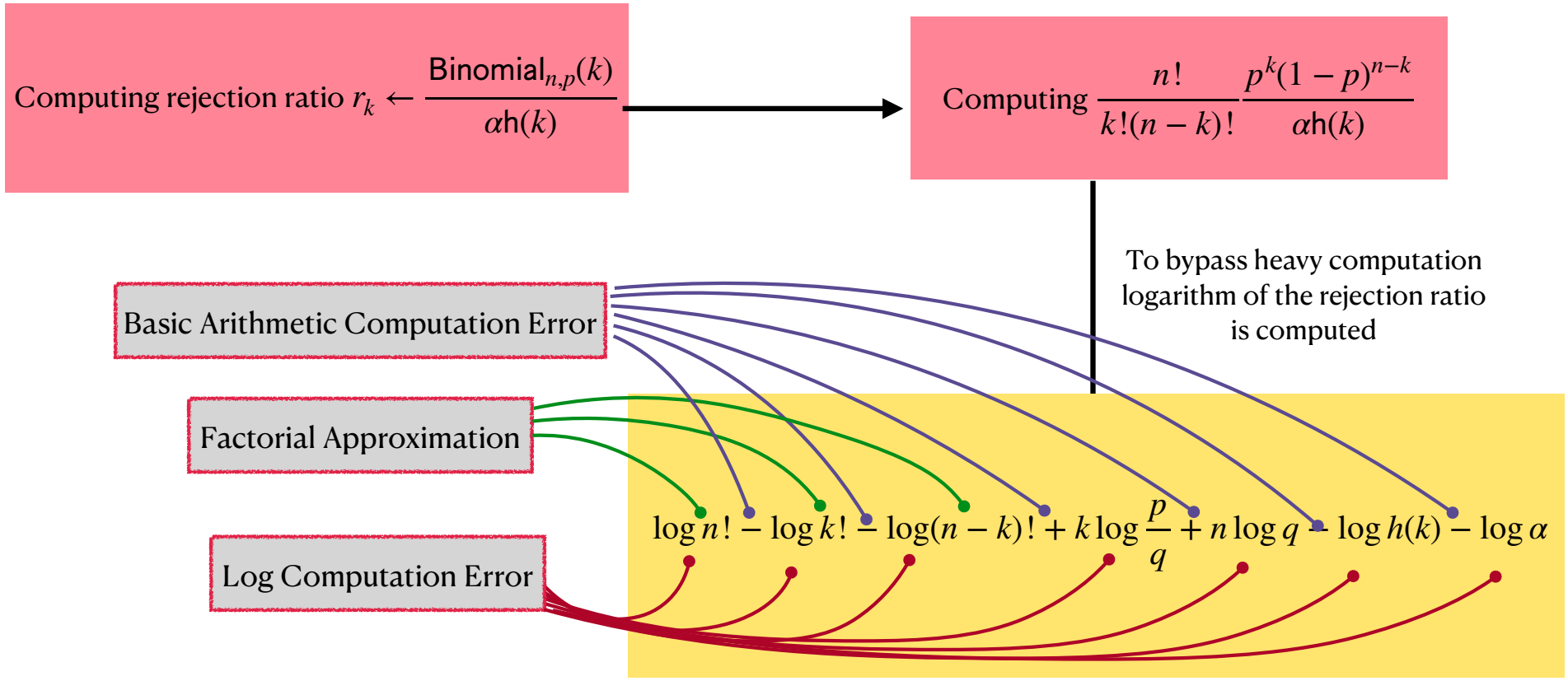
For all $k \in [0,\ n]$ we have $\left(1 - \frac{k}{2^\beta}\right) \leq \frac{\widetilde{\mathsf{h}}(k)}{\mathsf{h}(k)} \leq \left(1 + \frac{k w_k}{2^\beta}\right)$ where

$$w_k = \max\left(\frac{\mathsf{h}(k-1)}{\mathsf{h}(k)}, \frac{\mathsf{h}(k+1)}{\mathsf{h}(k)}\right)$$

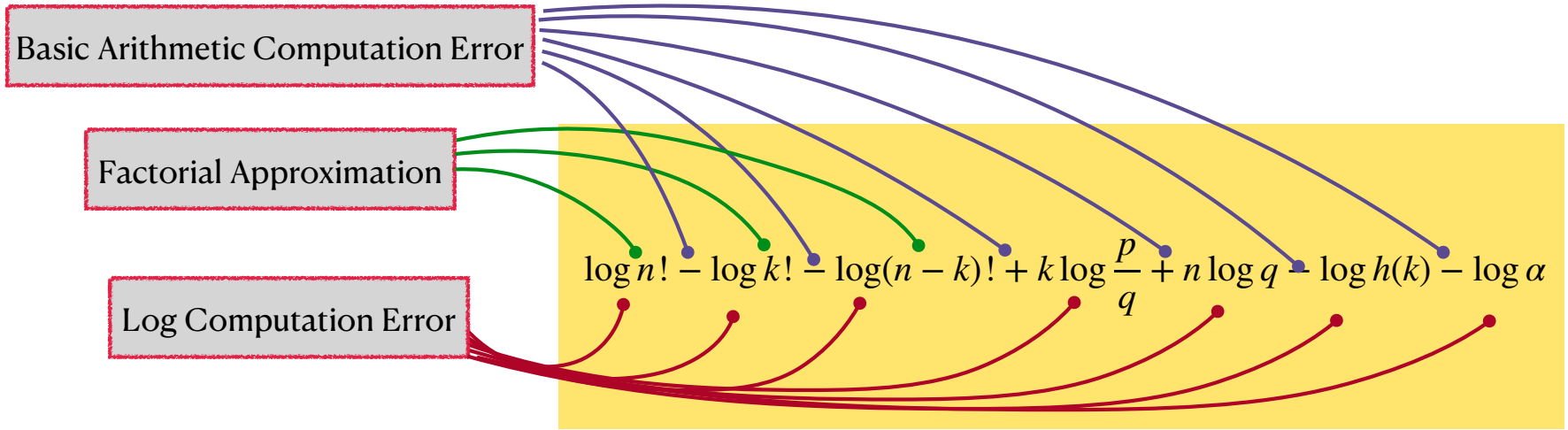
Bounding Errors in Rejection Sampling



Bounding Errors in Rejection Sampling



Bounding Errors in Rejection Sampling



Bound 2

$$\left(1 - O\left(\frac{n}{2^\beta}\right) - 15\zeta\right) \leq \frac{\widetilde{r}_k}{r_k} \leq \left(1 + O\left(\frac{n}{2^\beta}\right) + 15\zeta\right)$$

where ζ denotes the error bound due to the factorial approximation.

Our Main Technical Results

Our Contribution

Theorem 1

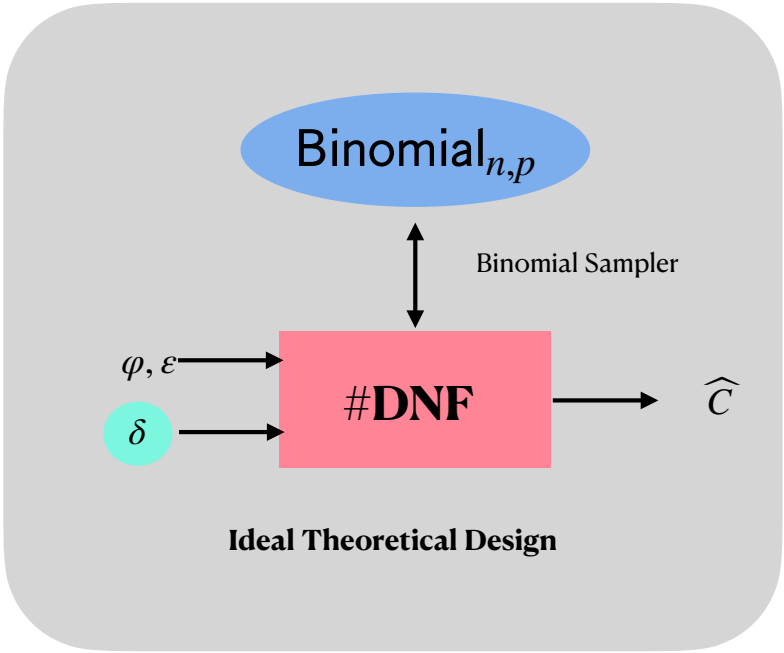
1. Let the precision of the context be $\beta \geq \max(2\lceil \log_2 n \rceil, \lceil -\log_2 p \rceil)$,
2. Let $\widetilde{\text{Binomial}}_{n,p}$ denote the distribution of the sampler.

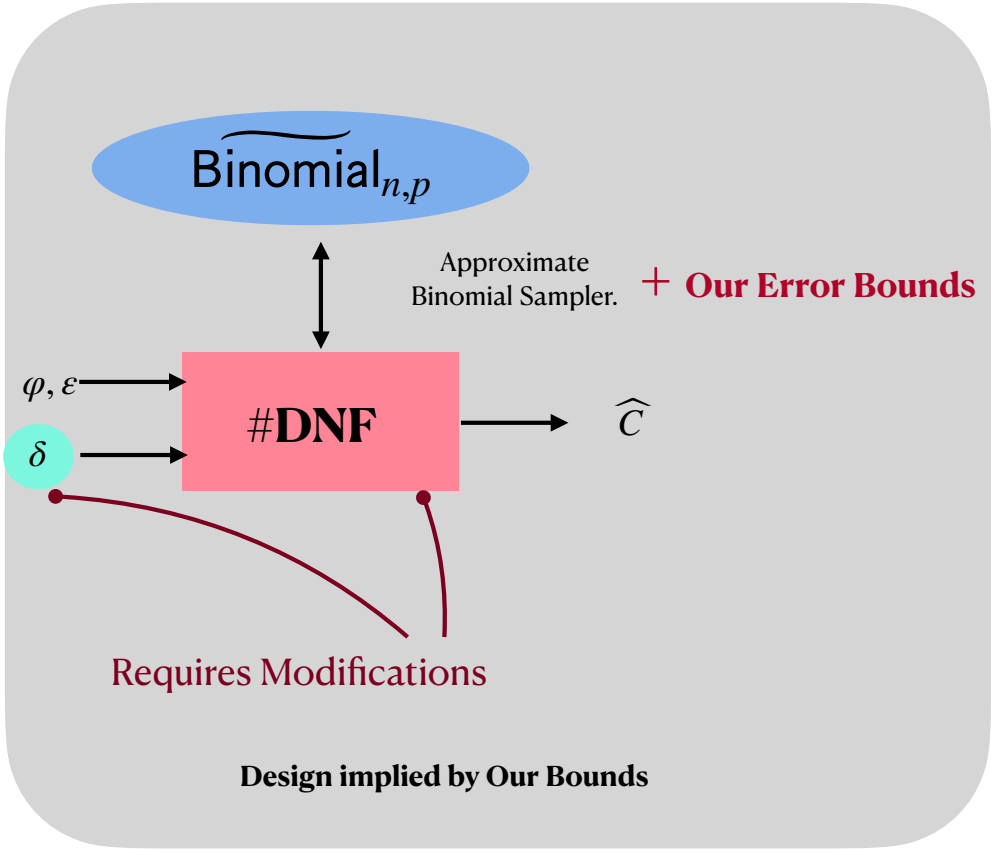
Then the **Statistical Distance** between $\widetilde{\text{Binomial}}_{n,p}$ and $\text{Binomial}_{n,p}$ is given by

$$\mathbf{D}\left(\text{Binomial}_{n,p}, \widetilde{\text{Binomial}}_{n,p}\right) \leq O\left(\frac{n}{2^{\beta}} + \zeta\right)$$

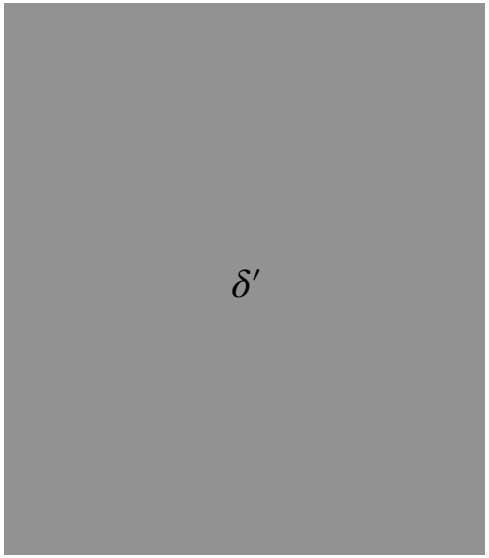
Where ζ denotes the error bound due to the factorial approximation.

Application of Our Bound

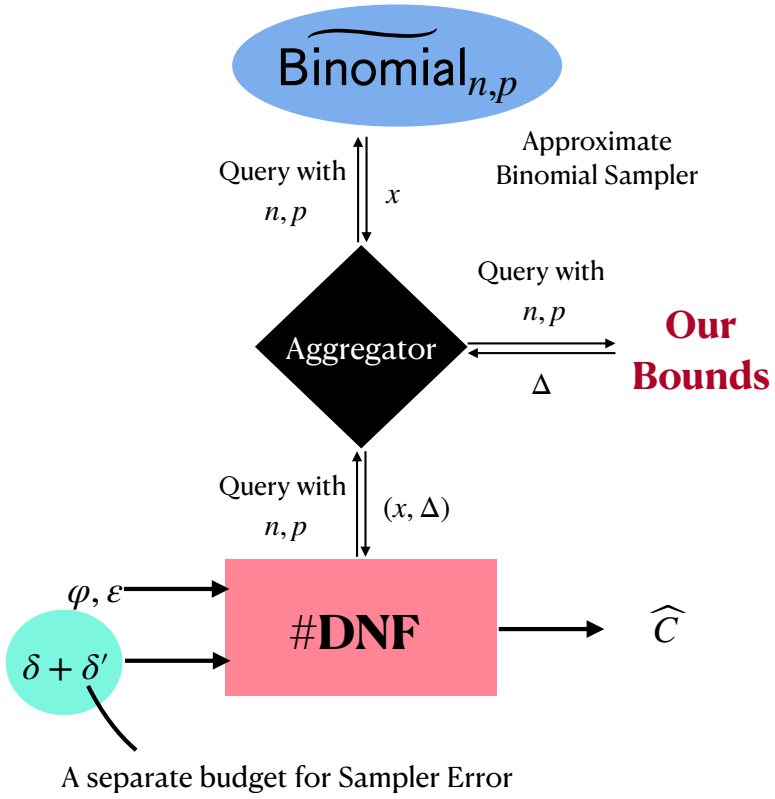




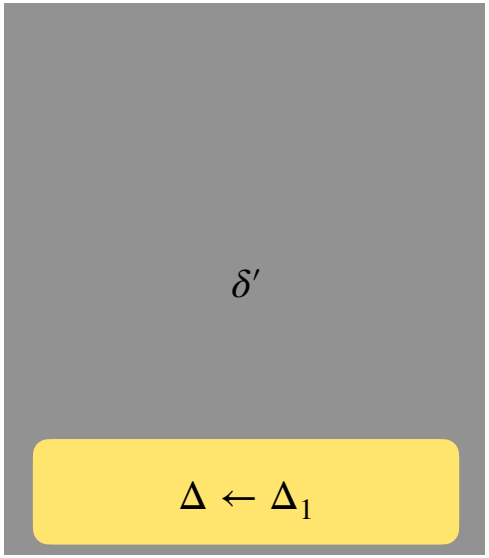
Application of Our Bound



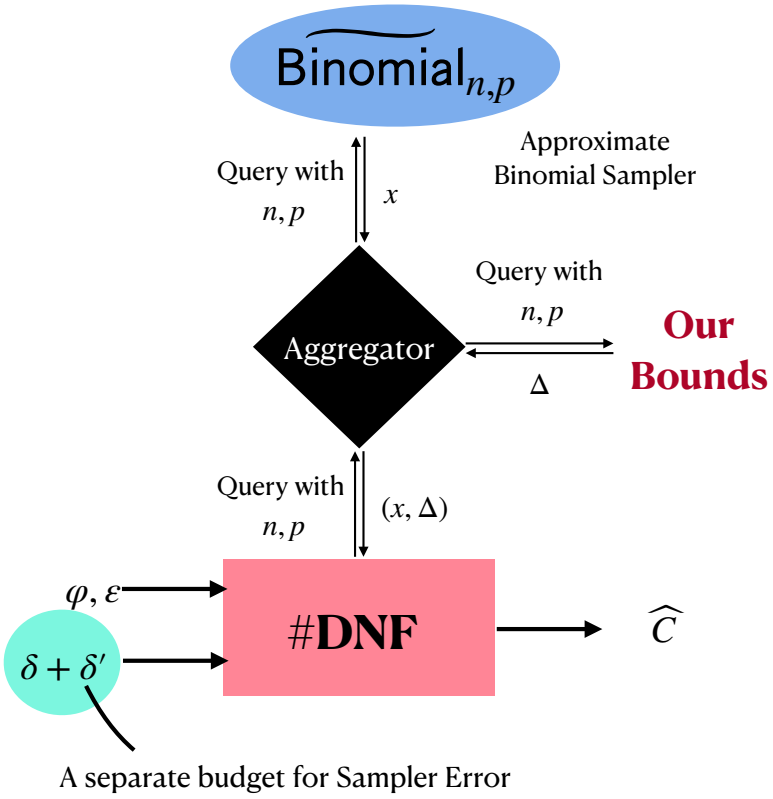
Budget of Sampler Errors



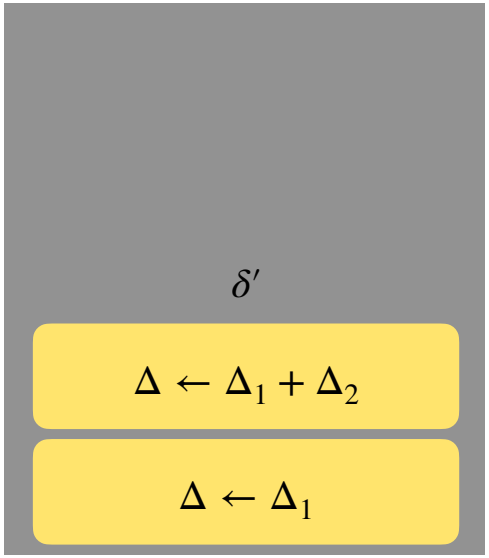
Application of Our Bound



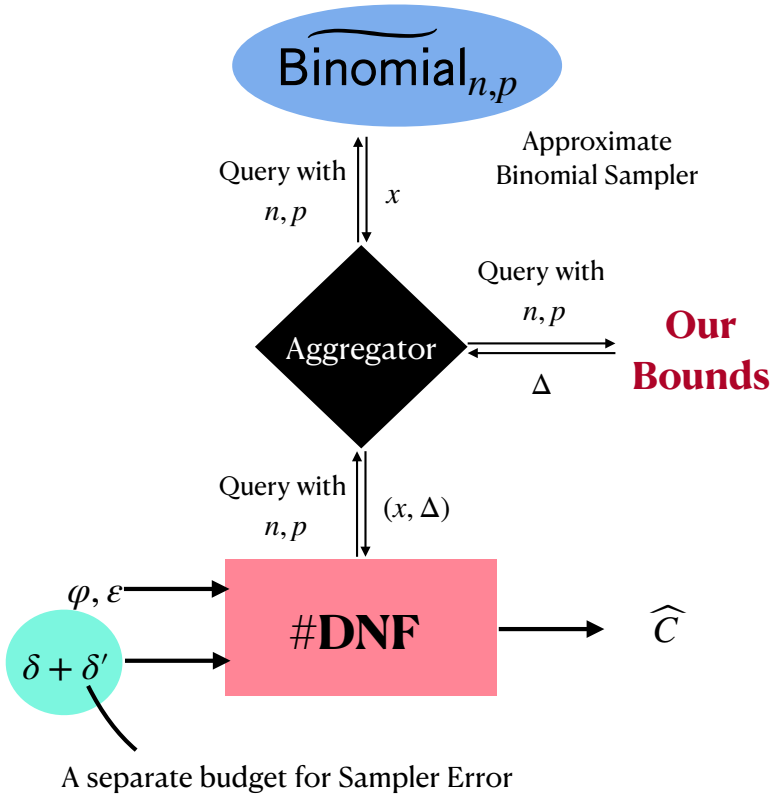
Budget of Sampler Errors



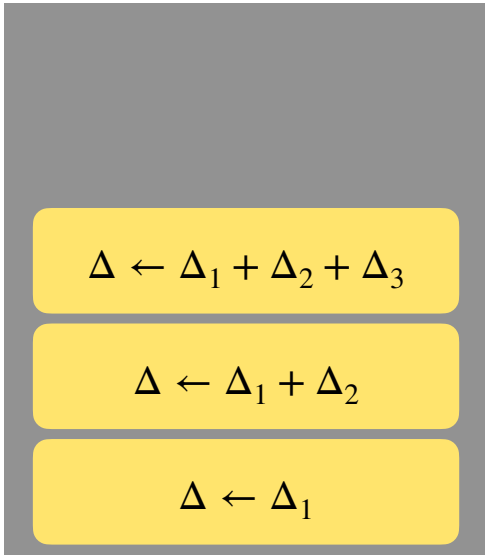
Application of Our Bound



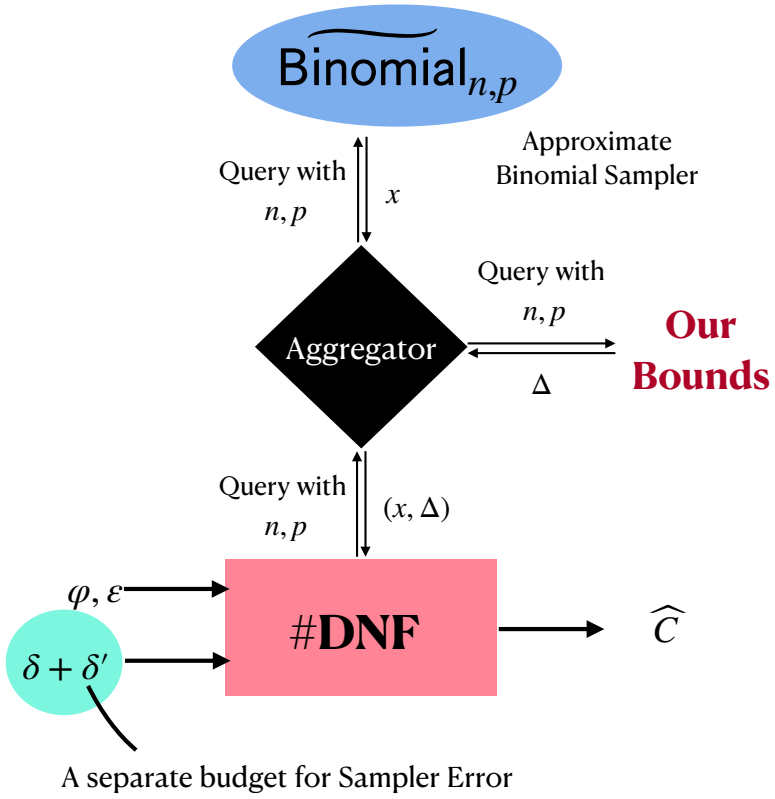
Budget of Sampler Errors



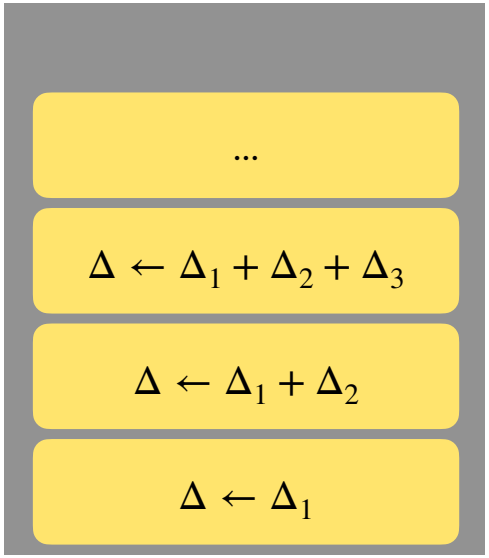
Application of Our Bound



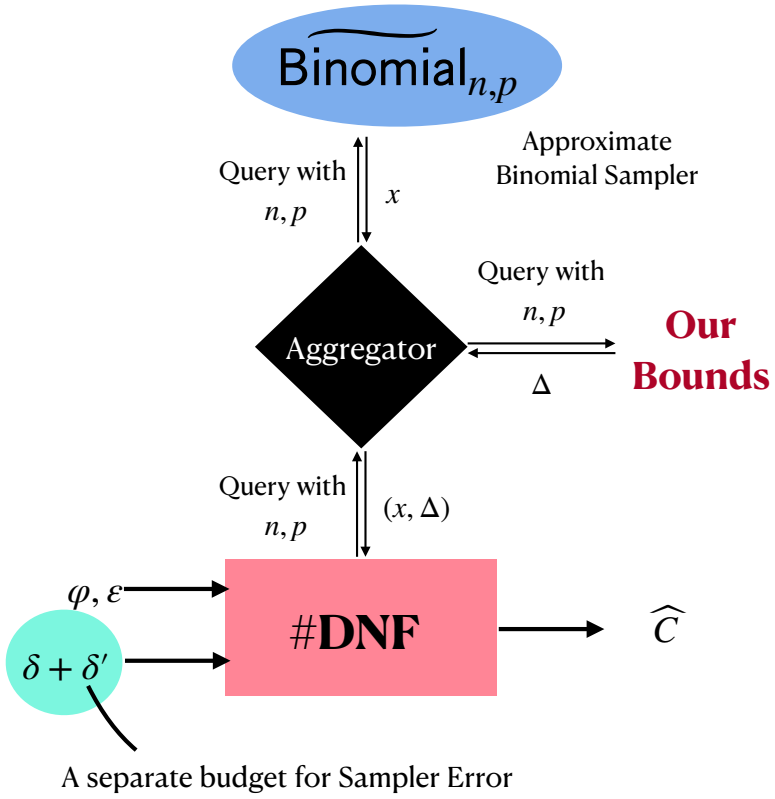
Budget of Sampler Errors



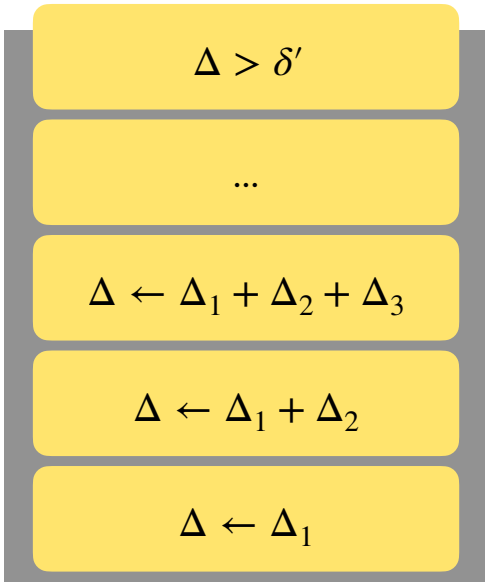
Application of Our Bound



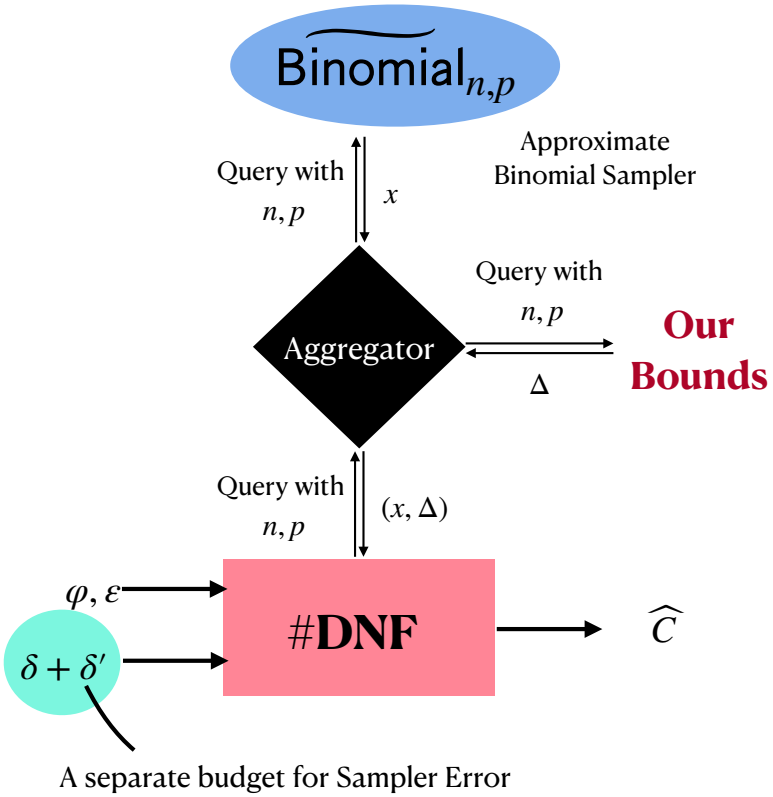
Budget of Sampler Errors



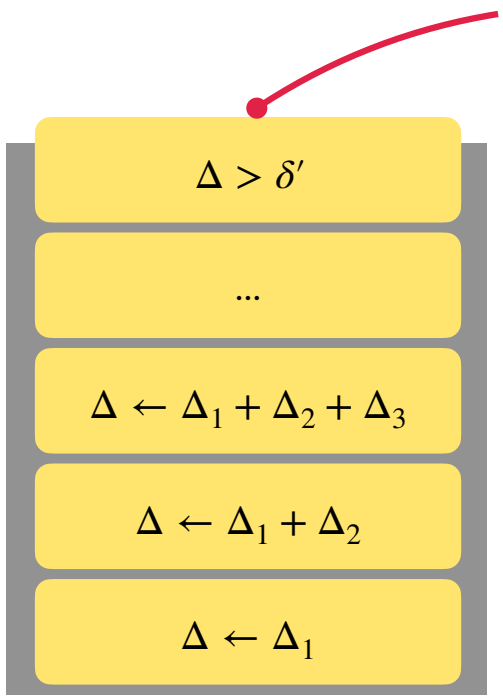
Application of Our Bound



Budget of Sampler Errors

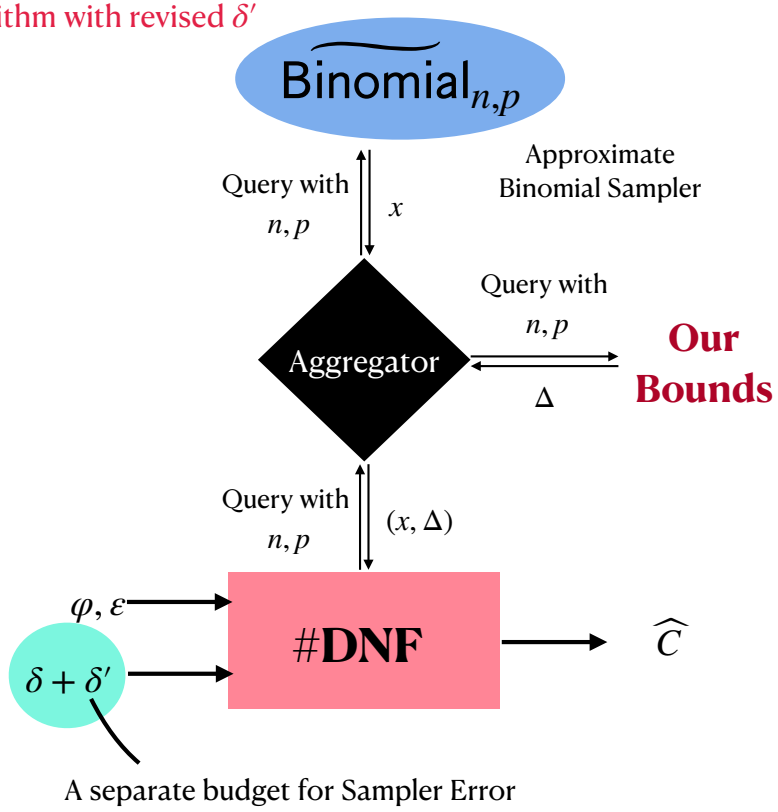


Application of Our Bound



Budget of Sampler Errors

Returns Failure!
User has to restart the algorithm with revised δ'



Conclusion

1. We realize that efficient Binomial samplers CANNOT be exact.

2. The error is small enough that a few calls don't matter. But a lot of calls can be disastrous.

3.Our work is the first to quantify the amount of error and enable library functions for designing randomized algorithms that use Binomial Samplers.

4. While we begin with Binomial Samplers, this work lays the foundation for refining bounds for other distributions like Poisson, Geometric etc.

Conclusion

1. How can verification of randomized programs work without a quantitative measure of the quality of samplers?

2. Do we need to change the way samplers work in programming languages? May be output an error bound and let the programmer decide how to handle it.

3. It opens the door for introducing new samplers that offer sound δ-guarantees despite being approximate.

4. The situation is even more crazy for continuous distributions.

Conclusion

1.How can verification of randomized programs work without a quantitative measure of the quality of samplers?

2.Do we need to change the way samplers work in programming languages? May be output an error bound and let the programmer decide how to handle it.

3. It opens the door for introducing new samplers that offer sound δ-guarantees despite being approximate.

4. The situation is even more crazy for continuous distributions.

Thank You!