Quantum Random Number Generation

Diogo Neiss, Henrique Assumpção

Universidade Federal de Minas Gerais

{*diogoneiss, henriquesoares*}*@dcc.ufmg.br*

November 3, 2023

Overview

Information Theory

Vamos inicialmente relembrar alguns conceitos úteis

Conteúdo de informação

Para uma variável X com distribuição de probabilidades p, para um evento $x \in X$ definimos o conteúdo de informação x como

$$h(x) = -log(p(x)) \tag{1}$$

Com essa definição então, vamos definir a Entropia de Shannon

Entropia: Shannon

Definição

Para uma dada variável X, com distribuição de probabilidade p, defininos a entropia de Shannon de X como

$$H(X) = \mathop{\mathbb{E}}_{x \in X}[h(x)] = \sum_{x \in X} -p(x) log(p(x)) \tag{2}$$

Quantifica bem a incerteza de uma variável aleatória, sendo então **o valor esperado do conteúdo de informação de todos os eventos**, contando a informação média presente em todos os eventos.

Entropia: min-entropy

Outra formulação útil de entropia de uma variável clássica aleatória é a seguinte

Definição

Seja X uma variável clássica aleatória. A entropia mínima (min-entropy) de X é definida como

$$H_{\min}(X) := h(\max_{x \in X} p(x)) \tag{3}$$

Isso significa que a min-entropy de X é definida como o **conteúdo de informação do evento mais provável em** X, o que, por sua vez, garante que tal quantidade seja sempre limitada superiormente pela Entropia de Shannon de X. Se estivermos lidando com um lançamento perfeito de moeda, a min-entropy é 1.

Aleatoriedade

O que é aleatoriedade?

Aleatoriedade

Diversas definições de probabilidade existem, como de Von Mises, Kolmogorov, Ville, Martin-Löf, resultando nas seguintes propriedades de sequências aleatórias:

1. Imprevisibilidade, i.e, nenhum fator influencia o próximo número
2. Convergência para uniformidade no infinito (grupos locais podem não ter essa propriedade)

Como aleatoriedade beneficia diversas aplicações

- **Criptografia**: Permite a geração de chaves seguras no algoritmo *RSA*
- **Métodos de Monte Carlo / Metropolis-Hastings**: Técnicas de amostragem aleatória evitam mínimos locais e garantem diversidade.
- **Sorteios**: Garantia de justiça para todos os participantes.
- **Pesquisa**: Garantia de imparcialidade da amostra nos resultados da pesquisa.

Números quase aleatórios - Quasi-random Number Generators

Nem sempre precisamos buscar números mais próximos da aleatoriedade perfeita.
Certas aplicações utilizam uma categoria de números aleatórios *intermediários*, isto é, não são aleatórios de fato mas também não são inocentemente gerados por procedimentos pseudoaleatórios.

Números quase aleatórios - Quasi-random Number Generators

Nem sempre precisamos buscar números mais próximos da aleatoriedade perfeita.
Certas aplicações utilizam uma categoria de números aleatórios *intermediários*, isto é, não são aleatórios de fato mas também não são inocentemente gerados por procedimentos pseudoaleatórios.

São os números **quase-aleatórios**, também conhecidos como **sequências de baixa discrepância**.

O que exatamente significa ser *quase aleatório?*

Números quase aleatórios

- São sequências numéricas geradas deterministicamente para preencher o espaço de uma maneira que reduza lacunas e agrupamentos, maximizando a uniformidade.

Números quase aleatórios

- São sequências numéricas geradas deterministicamente para preencher o espaço de uma maneira que reduza lacunas e agrupamentos, maximizando a uniformidade.
- Performam melhor em alguns métodos estocásticos, como integração numérica, algoritmos de ordenação, dentre outros problemas computacionais onde a cobertura uniforme e reprodutibilidade são importantes.

Como verificar aleatoriedade

- Não é possível provar aleatoriedade verdadeira.
- É possível criar sequências uniformes e imprevisíveis determinísticas

Como verificar aleatoriedade

- Não é possível provar aleatoriedade verdadeira.
- É possível criar sequências uniformes e imprevisíveis determinísticas

Logo, o princípio estatístico de testar números aleatórios é decidir se uma sequência de números é

1. Não aleatória
2. Possivelmente aleatória

Testes estatísticos

- Teste de Frequência (Monobit): Verifica se a quantidade de uns e zeros são aproximadamente iguais.
- Teste de Autocorrelação: Verifica a existência de correlações entre diferentes partes da sequência.
- Teste de Compressão: Dados verdadeiramente aleatórios são incompressíveis. (Kolmogorov)
- Testes NIST: Conjunto de testes estatísticos desenvolvidos pelo NIST para testar a aleatoriedade de sequências binárias.

Esses métodos oferecem apenas um nível razoável de confiança de que um RNG produz números verdadeiramente aleatórios, já que só detectam não-aleatoriedade.

Geradores clássicos de números aleatórios

Como computadores são inerentemente determinísticos, não são capazes de criar sequências verdadeiramente aleatórias, porém isso não os impede de serem bons o suficientes para a maioria das aplicações, principalmente quando é necessária velocidade de geração. Esses números são **pseudo-aleatórios**.

Exemplo: Linear Congruential Generator

De acordo com as constantes inseridas, conseguimos fazer ele *parecer* aleatório

$$X_{n+1} = (aX_n + c) \mod m \tag{4}$$

Nele, podemos aplicar uma fonte de entropia, ou constantes, ajustando os coeficientes e valor inicial de acordo, com a função servindo como uma espécie de *extrator* dessa origem.

Melhorando RNGs: fonte aleatória

Um método inteligente para projetar melhores RNGs é contrapor suas duas falhas mais críticas

- Determinismo: Semear usando entropia coletada de uma fonte física, como o decaimento radioativo, o seed será gerado a partir de um processo certificavelmente aleatório
- Periodicidade: Resemear frequentemente

Programadores utilizam algoritmos como *Math.random()* para gerar números aleatórios, escolhendo um bom seed e resemeando periodicamente. No entanto, ainda é possível descobrir o padrão do algoritmo e explorá-lo para fins maliciosos se o algoritmo é conhecido

Algoritmos populares e ataques de canal lateral

Um exemplo de algoritmo quebrável é o método aleatório do *javascript*, presente V8 Engine, base do Chrome, que usa o algoritmo *Xorshift*.

- Realiza operações de deslocamento para gerar novos números, disfarçando a periodicidade

Algoritmos populares e ataques de canal lateral

Um exemplo de algoritmo quebrável é o método aleatório do *javascript*, presente V8 Engine, base do Chrome, que usa o algoritmo *Xorshift*.

- Realiza operações de deslocamento para gerar novos números, disfarçando a periodicidade
- Podemos utilizar um solver SMT para encontrar a semente com base em restrições do funcionamento do algoritmo

O melhor que sistemas clássicos podem fornecer

Existe outra categoria de RNGs, mais resiliente a tentativas de engenharia reversa: Geradores de Números Pseudoaleatórios Criptograficamente Seguros (CSPRNG), que satisfazem:

- Passam no "teste do próximo bit". Se conhecermos os primeiros 'k' bits de uma sequência, não deve existir um método prático para adivinhar o próximo bit com uma taxa de sucesso superior 50%.
- Resistem a "extensões de comprometimento de estado". Mesmo com vazamentos, é impossível reconstruir os números aleatórios gerados anteriormente e não permitir a previsão dos estados futuros do CSPRNG.

Uma implementação popular é o *Fortuna*, usado no Linux.

QRNG

O que são QRNGs

Um Gerador de Números Aleatórios Quânticos (QRNG) é o padrão ouro de geração de números verdadeiramente aleatórios, já que consegue grandes fontes de entropia baseando-se nas propriedades quânticas do sistema para produzir números.
O princípio fundamental por trás dos QRNGs de hardware é a suposição de que não há uma maneira determinística de determinar para qual estado um dado estado quântico irá colapsar. Dependem de duas etapas:

1. Configuração: A preparação do estado inicial, que requer confiança do usuário de que o sistema está realmente no estado de superposição.
2. Geração: O processo de medição de uma série de estados no momento desejado, gerando os números aleatórios desejados.

O que são QRNGs

Um Gerador de Números Aleatórios Quânticos (QRNG) é o padrão ouro de geração de números verdadeiramente aleatórios, já que consegue grandes fontes de entropia baseando-se nas propriedades quânticas do sistema para produzir números.
O princípio fundamental por trás dos QRNGs de hardware é a suposição de que não há uma maneira determinística de determinar para qual estado um dado estado quântico irá colapsar. Dependem de duas etapas:

1. Configuração: A preparação do estado inicial, que requer confiança do usuário de que o sistema está realmente no estado de superposição.
2. Geração: O processo de medição de uma série de estados no momento desejado, gerando os números aleatórios desejados.

Categorias de QRNG

1. **Geradores Práticos de Números Aleatórios Quânticos:** Arquitetura canônica, rápidos e baratos
2. **Geradores Auto Verificáveis de Números Aleatórios Quânticos:** Muito lentos e geralmente requerem dispositivos adicionais do receptor para a validação
3. **Geradores Semi Verificáveis de Números Aleatórios Quânticos:** *Tradeoff* de velocidade e segurança

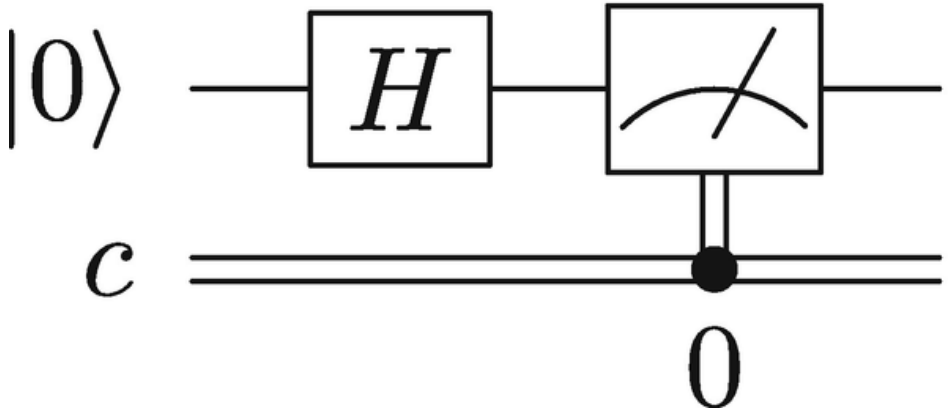
Categorias de QRNG

1. **Geradores Práticos de Números Aleatórios Quânticos:** Arquitetura canônica, rápidos e baratos
2. **Geradores Auto Verificáveis de Números Aleatórios Quânticos:** Muito lentos e geralmente requerem dispositivos adicionais do receptor para a validação
3. **Geradores Semi Verificáveis de Números Aleatórios Quânticos:** *Tradeoff* de velocidade e segurança

Depois da geração de números aleatórios com a fonte de entropia quântica, geralmente é aplicado um extrator de aleatoriedade para uniformizar os números aleatórios "crus" gerados, em muitas implementações é fator limitante do QRNG.

Geradores Práticos

Implementação clássica, como a medição de um qubit em superposição, e sua aleatoriedade depende da fidelidade e robustez da implementação modelo teórico. São rápidos e baratos, **mas é necessário confiar** no dispositivo e seus componentes para a segurança desses geradores.



Gerador quântico simples: Gate H para gerar $|+\rangle$ e medição

Geradores Auto Verificáveis

Realizam testes na sequência gerada devido à confiança limitada no processo físico implementado, seja por falhas físicas ou a presença de adversários.

A verificação pode depender de testes clássicos ou quânticos, como através das **desigualdades de Bell**. Aleatoriedade intrínseca pode ser modelada como um atributo do sistema quântico, e violações da desigualdade seriam usados para prová-la.

Geradores Auto Verificáveis

Realizam testes na sequência gerada devido à confiança limitada no processo físico implementado, seja por falhas físicas ou a presença de adversários.

A verificação pode depender de testes clássicos ou quânticos, como através das **desigualdades de Bell**. Aleatoriedade intrínseca pode ser modelada como um atributo do sistema quântico, e violações da desigualdade seriam usados para prová-la.

Estes geradores são impraticavelmente lentos e muitas vezes requerem aparelhos de teste adicionais.

Geradores Semi Verificáveis

Compromisso entre as duas categorias anteriores, em que algum nível de confiança na implementação reduz a necessidade de testes extensos, otimizando os parâmetros de velocidade ao custo da confiança na aleatoriedade gerada.

Geradores Semi Verificáveis

Compromisso entre as duas categorias anteriores, em que algum nível de confiança na implementação reduz a necessidade de testes extensos, otimizando os parâmetros de velocidade ao custo da confiança na aleatoriedade gerada.

Há dois tipos principais, baseando se em qual componente nesses dispositivos é considerado seguro

1. independente de fonte - medição é segura
2. independente de medição - fonte é segura

1. QRNG independente de fonte

A ideia essencial para este tipo de esquema é usar a medição para monitorar a fonte em tempo real, já que os **dispositivos de medição** são assumidos como confiáveis.

- Normalmente é necessário alternar entre diferentes configurações de medição complementares, de modo que a fonte não possa prever a medição com antecedência.
- Necessitam de uma boa caracterização dos dispositivos de medição. Por exemplo, os limites superior e inferior sobre as eficiências dos detectores precisam ser conhecidos para evitar potenciais ataques induzidos por discrepâncias de eficiência dos detectores.

1. QRNG independente de fonte

A ideia essencial para este tipo de esquema é usar a medição para monitorar a fonte em tempo real, já que os **dispositivos de medição** são assumidos como confiáveis.

- Normalmente é necessário alternar entre diferentes configurações de medição complementares, de modo que a fonte não possa prever a medição com antecedência.
- Necessitam de uma boa caracterização dos dispositivos de medição. Por exemplo, os limites superior e inferior sobre as eficiências dos detectores precisam ser conhecidos para evitar potenciais ataques induzidos por discrepâncias de eficiência dos detectores.

É bem eficiente, já que a implementação física da fonte que usualmente é a causa dos problemas

2. QRNG independente de medição

Complementarmente, podemos considerar o cenário em que a **fonte de entrada** é bem caracterizada, enquanto o dispositivo de medição não é confiável.

- A fonte confiável envia ocasionalmente estados quânticos auxiliares para verificar se a medição está na base correta.
- Elimina a maioria dos canais laterais do detector, mas podem estar sujeitos a imperfeições na modelagem da fonte e medição prematura.

Testando um gerador prático em um computador quântico real

Temos em [AS21] uma descrição muito interessante da implementação de gerador prático de números aleatórios em um computador quântico real da IBM, que obteve resultados interessantes

- IBM 20Q Poughkeepsie - 20 Qubits
- 8192 execuções do circuito - máximo número de execuções ininterruptas
- 579 amostras retiradas

Cada um dos 20 qubit's produziu 579 amostras de 8192 bits, o suficiente para a maioria dos testes estatísticos.

Configuração do computador para o experimento

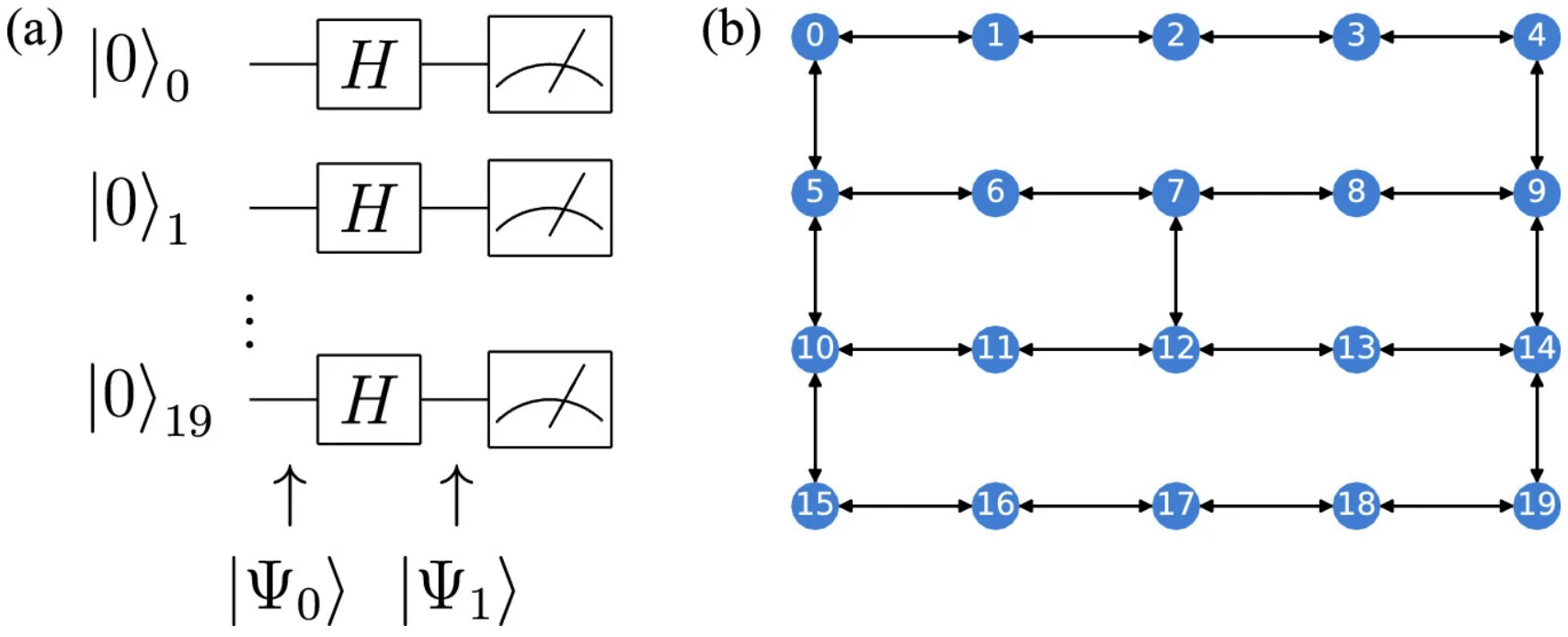


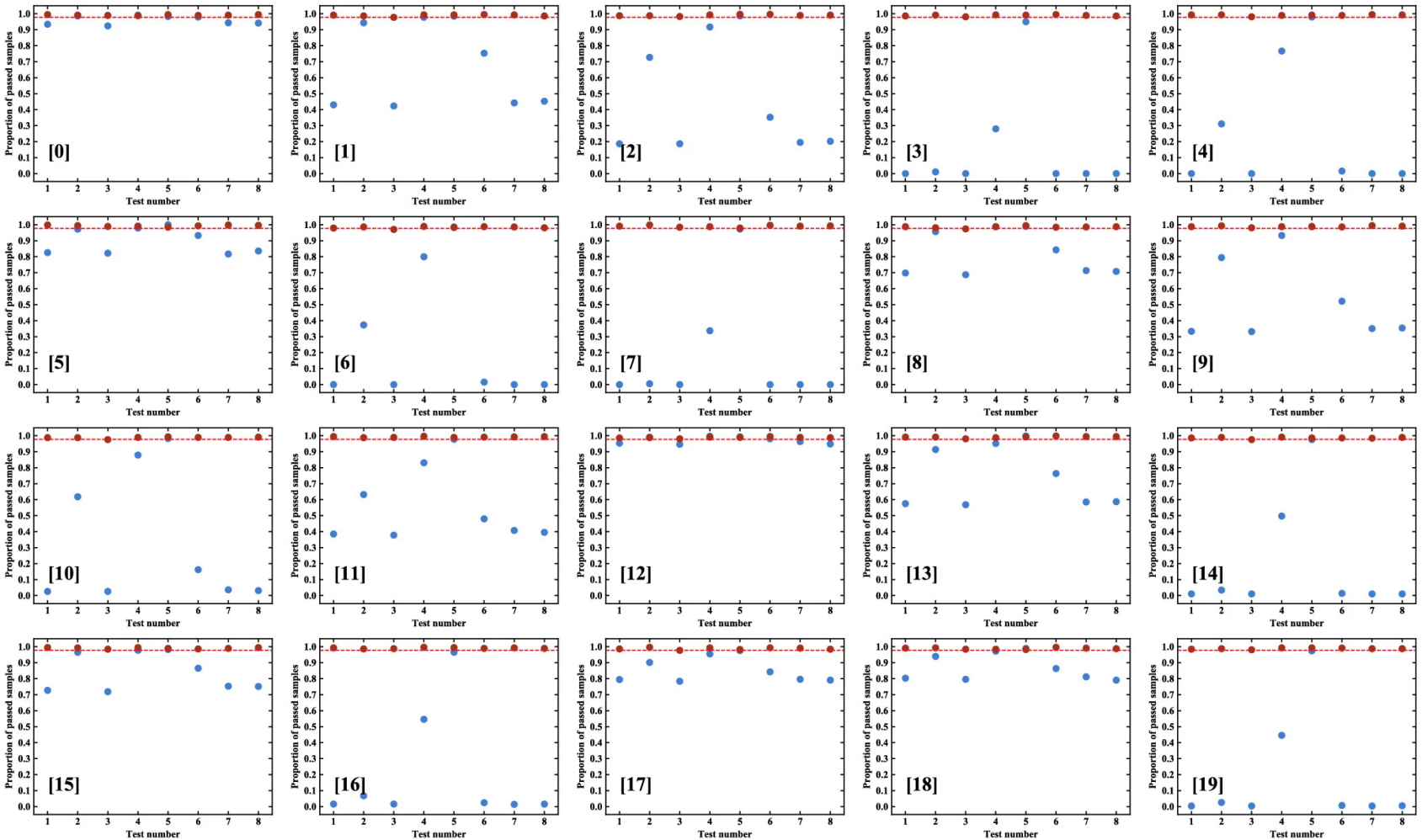
Diagrama: a) PQRNG e b) arquitetura do computador IBM 20Q Poughkeepsie

Seleção de testes por comprimento mínimo

Test number	Test name	Minimum length
1	Frequency	$n \geq 100$
2	Frequency within a block	$n \geq 100$
3	Runs	$n \geq 100$
4	Longest run of ones	$n \geq 128$
	Binary matrix rank	$n \geq 38912$
5	DFT	$n \geq 1000$
	Non-overlapping T. M.	$n \geq 8m - 8$
	Overlapping T. M.	$n \geq 10^6$
	Maurer's Universal	$n > 387840$
	Linear complexity	$n \geq 10^6$
	Serial	$n > 16$
6	Approximate entropy	$n > 64$
7	Cumulative sums (forward)	$n \geq 100$
8	Cumulative sums (backward)	$n \geq 100$
	Random excursions	$n \geq 10^6$
	Random excursions variant	$n \geq 10^6$

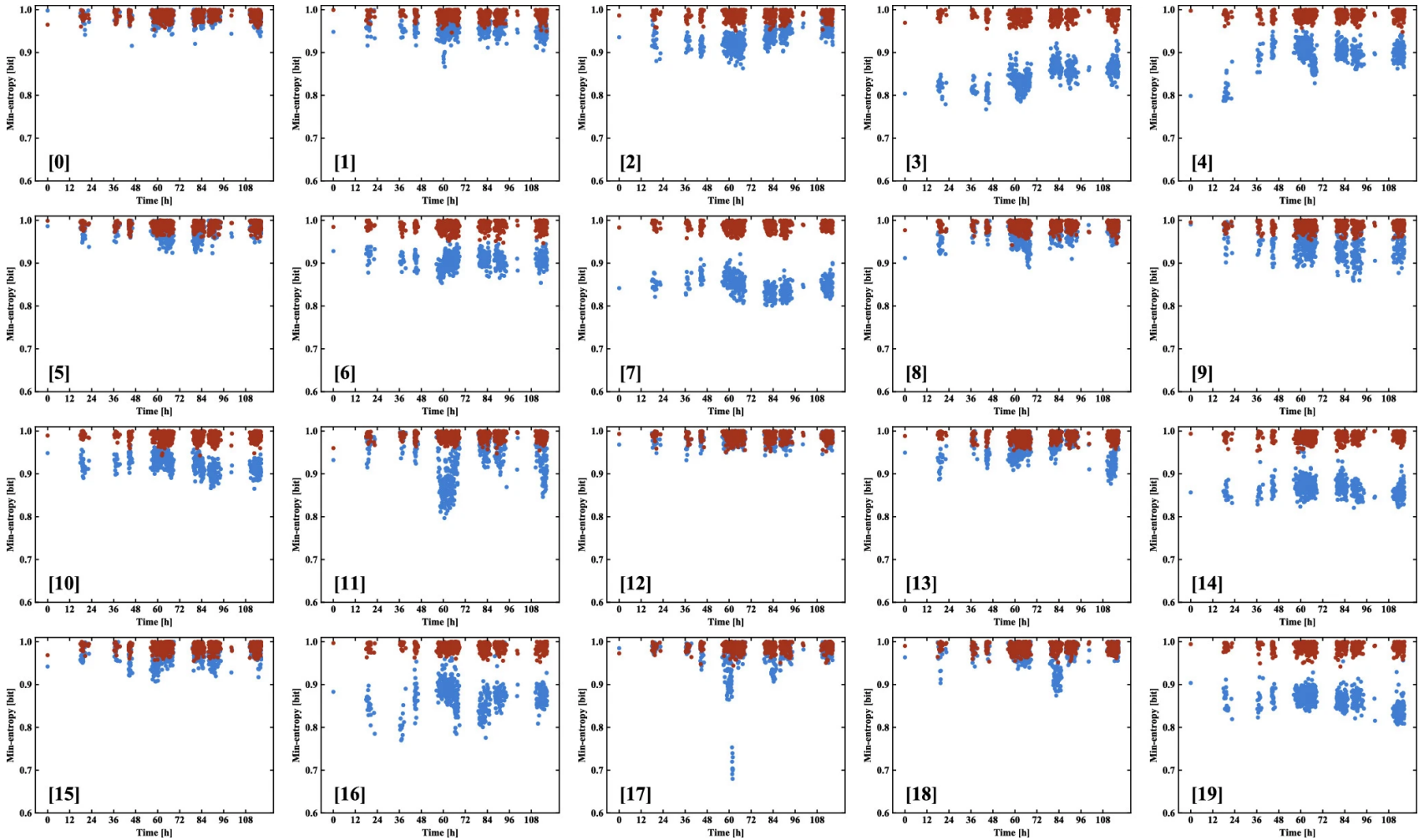
Testes utilizados no estudo: Testes em cinza não foram feitos

Resultado dos testes



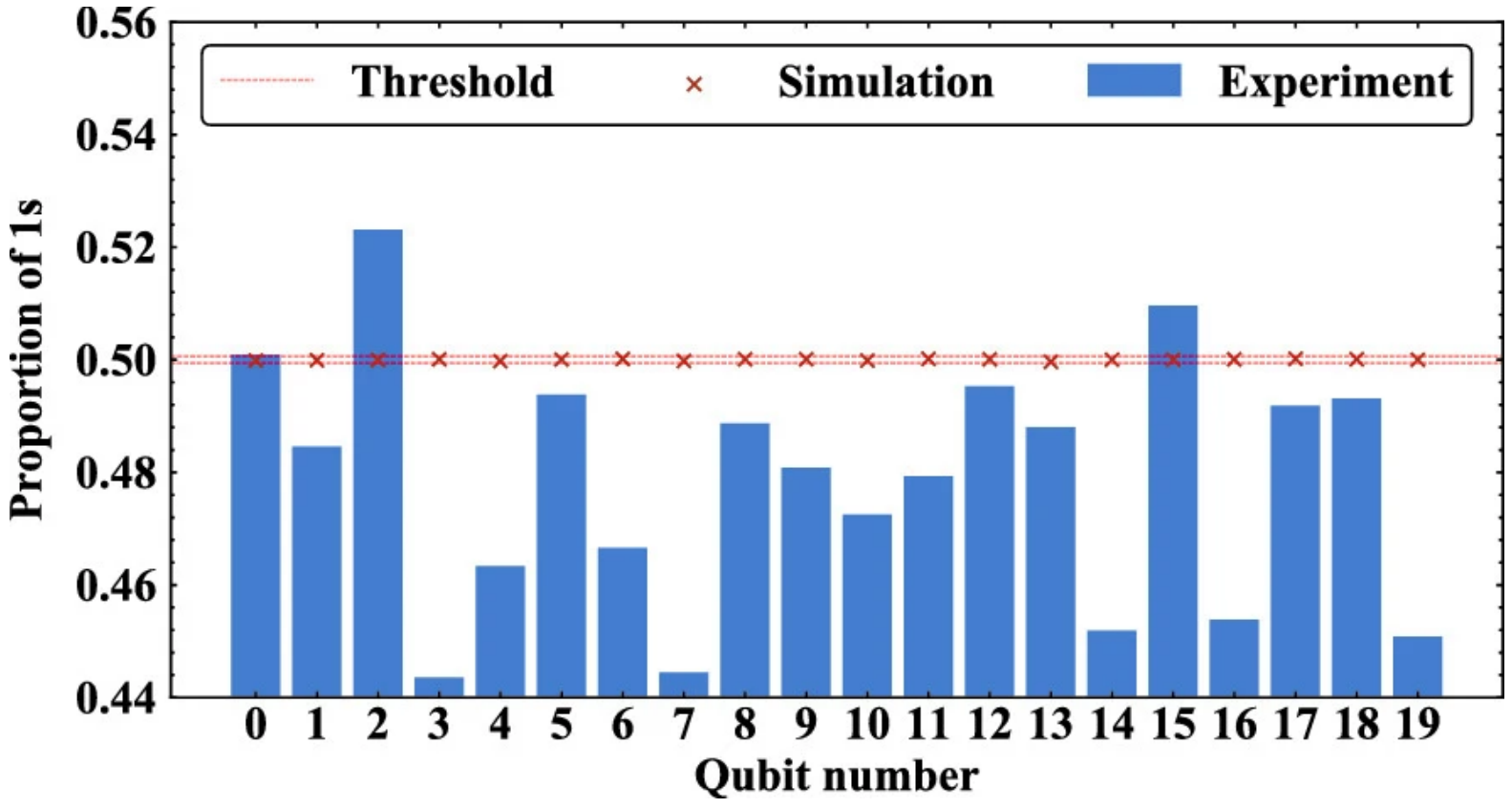
Índice do teste vs proporção de amostras que passaram: experimento realizado e simulação utilizando ruído

Min-Entropy dos Qubits deve aproximar y = 1



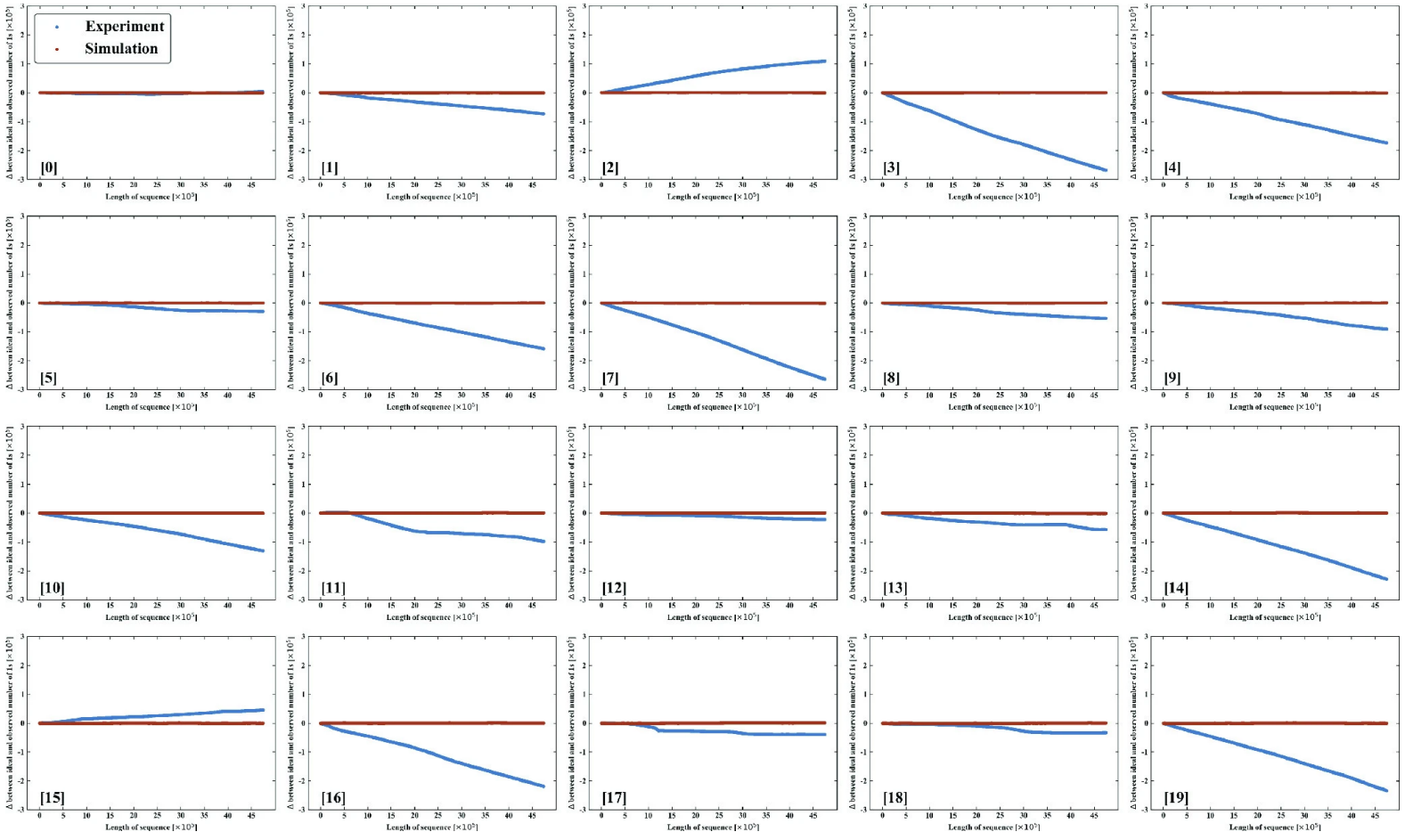
MinEntropy a cada hora do experimento: experimento realizado e simulação utilizando ruído

Uniformidade por bit



Faixa vermelha: Apenas qubits 0 e 12 aproximaram a proporção uniforme necessária

Uniformidade cumulativa na sequência por bit



Deveria aproximar a linha constante: Mantendo nossa constatação, apenas qubits 0 e 12 aproximaram a proporção uniforme necessária

Conclusão do experimento

- Queda abrupta da min-entropy no qubit 17

Conclusão do experimento

- Queda abrupta da min-entropy no qubit 17
- Nenhum dos qubits produziu proporções aceitáveis de 1s.

Conclusão do experimento

- Queda abrupta da min-entropy no qubit 17
- Nenhum dos qubits produziu proporções aceitáveis de 1s.
- Qubits 0 e 12 eram relativamente estáveis

Conclusão do experimento

- Queda abrupta da min-entropy no qubit 17
- Nenhum dos qubits produziu proporções aceitáveis de 1s.
- Qubits 0 e 12 eram relativamente estáveis

Logo, nenhum dos qubits atendeu aos padrões da suíte de testes, porém vale ressaltar que

- Modelo QRNG prático, sem verificação/ajuste
- Autor utilizou um computador de 20 qubits com bastante erro e viés entre os qubits

Outros artigos na literatura utilizaram menos qubits, compensação de erro, independência da fonte e extrator de aleatoriedade, resultando em números aleatórios que passaram no teste.

QRNG & Quantum Supremacy

Preliminares

A partir de agora, iremos considerar circuitos quânticos com n qubits, onde $N := 2^n$. Iremos começar com algumas definições:

Preliminares

A partir de agora, iremos considerar circuitos quânticos com n qubits, onde $N := 2^n$. Iremos começar com algumas definições:

- $\mathbb{U}(N) := \{f : \mathbb{C}^N \mapsto \mathbb{C}^N | f \text{ é linear}, f^{-1} = f^*\}$

Preliminares

A partir de agora, iremos considerar circuitos quânticos com n qubits, onde $N := 2^n$. Iremos começar com algumas definições:

- $\mathbb{U}(N) := \{f : \mathbb{C}^N \mapsto \mathbb{C}^N | f \text{ é linear}, f^{-1} = f^*\}$
- $(\mathbb{U}(N), \circ)$ é um *grupo*, e também é um *grupo topológico* (é conexo e compacto)

Preliminares

A partir de agora, iremos considerar circuitos quânticos com n qubits, onde $N := 2^n$. Iremos começar com algumas definições:

- $\mathbb{U}(N) := \{f : \mathbb{C}^N \mapsto \mathbb{C}^N | f \text{ é linear}, f^{-1} = f^*\}$
- $(\mathbb{U}(N), \circ)$ é um *grupo*, e também é um *grupo topológico* (é conexo e compacto)
- **Fato**: existe uma única medida de probabilidade μ em $\mathbb{U}(N)$ que é invariante em relação à operação do grupo, e μ é dita a medida de *Haar* em $\mathbb{U}(N)$.

Preliminares

A partir de agora, iremos considerar circuitos quânticos com n qubits, onde $N := 2^n$. Iremos começar com algumas definições:

- $\mathbb{U}(N) := \{f : \mathbb{C}^N \mapsto \mathbb{C}^N | f \text{ é linear}, f^{-1} = f^*\}$
- $(\mathbb{U}(N), \circ)$ é um *grupo*, e também é um *grupo topológico* (é conexo e compacto)
- **Fato**: existe uma única medida de probabilidade μ em $\mathbb{U}(N)$ que é invariante em relação à operação do grupo, e μ é dita a medida de *Haar* em $\mathbb{U}(N)$.
- Podemos definir $\mathcal{D}$ como uma distribuição de probabilidade em $\mathbb{U}(N) \Rightarrow$ isso **define uma distribuição de probabilidade sobre o conjunto de circuitos quânticos de n-qubits**!

Preliminares

Lemma

Seja $C \in \mathbb{U}(N)$ uma matriz unitária representando um circuito quântico, e seja p_C uma distribuição de probabilidade definida por $p_C(s) = |\langle s|C|0\rangle^{\otimes n}|^2$. Se $C \sim Haar(N)$, então

$$\mathop{\mathbb{E}}_{C} \mathop{\mathbb{E}}_{s\sim p_C} [p_C(s)] = \frac{2}{N+1} \tag{5}$$

■

Random Circuit Sampling (RCS)

RCS foi a tarefa utilizada nos experimentos de *Quantum Supremacy* [AAB+19] realizados pela Google em 2019. Na RCS, nós possuímos dois agentes: o **Usuário** (Arthur) e o **Servidor** (Merlin). O protocolo é descrito a seguir:

[1] Tempo polinomial em n.

Random Circuit Sampling (RCS)

RCS foi a tarefa utilizada nos experimentos de *Quantum Supremacy* [AAB+19] realizados pela Google em 2019. Na RCS, nós possuímos dois agentes: o **Usuário** (Arthur) e o **Servidor** (Merlin). O protocolo é descrito a seguir:

1. O **Usuário** tem uma distribuição $\mathcal{D}$ sobre $\mathbb{U}(N)$. Ele amostra um circuito $C \sim \mathcal{D}$, e então o envio para o **Servidor**.

[1] Tempo polinomial em n.

Random Circuit Sampling (RCS)

RCS foi a tarefa utilizada nos experimentos de *Quantum Supremacy* [AAB+19] realizados pela Google em 2019. Na RCS, nós possuímos dois agentes: o **Usuário** (Arthur) e o **Servidor** (Merlin). O protocolo é descrito a seguir:

1. O **Usuário** tem uma distribuição $\mathcal{D}$ sobre $\mathbb{U}(N)$. Ele amostra um circuito $C \sim \mathcal{D}$, e então o envio para o **Servidor**.
2. O **Servidor** *rapidamente*[1] retorna uma string de n bits s_i para o **Usuário**. Um **Servidor** *honesto* iria usar um computador quântico para aplicar C no estado $|0\rangle^{\otimes n}$, medir o resultado na base computacional, e então retornar a saída ao **Usuário**.

[1] Tempo polinomial em n.

Random Circuit Sampling (RCS)

RCS foi a tarefa utilizada nos experimentos de *Quantum Supremacy* [AAB+19] realizados pela Google em 2019. Na RCS, nós possuímos dois agentes: o **Usuário** (Arthur) e o **Servidor** (Merlin). O protocolo é descrito a seguir:

1. O **Usuário** tem uma distribuição $\mathcal{D}$ sobre $\mathbb{U}(N)$. Ele amostra um circuito $C \sim \mathcal{D}$, e então o envio para o **Servidor**.
2. O **Servidor** *rapidamente*[1] retorna uma string de n bits s_i para o **Usuário**. Um **Servidor** *honesto* iria usar um computador quântico para aplicar C no estado $|0\rangle^{\otimes n}$, medir o resultado na base computacional, e então retornar a saída ao **Usuário**.
3. O **Usuário** repete o processo diversas vezes e obtém saídas $s_1, ..., s_k$, e então aplica um **teste estatístico** em um *computador clássico* para determinar se o **Servidor** foi honesto.

[1] Tempo polinomial em n.

LXEB: testando a resposta do Servidor

Definition (Linear Cross-Entropy Benchmarking)

Seja $\mathcal{D}$ uma distribuição de probabilidade sobre $\mathbb{U}(N)$, C amostrado de $\mathcal{D}$ e $p_C(s) = |\langle s|C|0\rangle^{\otimes n}|^2$. Para um conjunto de amostras $s_1, s_2, ..., s_k, s_i \in \{0,1\}^n$. O problema do LXEB$(b,k)$ consiste em gerar amostras tais que

$$\frac{1}{k}\sum_{i=1}^{k} p_C(s_i) \geq \frac{b}{N} \tag{6}$$

Por quê um computador clássico deveria falhar o LXEB?

- Atualmente, não se conhece nenhum algoritmo clássico para simular um CQ de forma eficiente.

Por quê um computador clássico deveria falhar o LXEB?

- Atualmente, não se conhece nenhum algoritmo clássico para simular um CQ de forma eficiente.
- Se assumirmos que o **Servidor** está utilizando um computador clássico, então ele não consegue inferir p_C para um $C \sim \mathcal{D}$ arbitrário em tempo polinomial em n. Portanto, o **Servidor** tem pouco a fazer além de amostrar uniformemente strings de $\{0,1\}^n$.

Por quê um computador clássico deveria falhar o LXEB?

- Atualmente, não se conhece nenhum algoritmo clássico para simular um CQ de forma eficiente.
- Se assumirmos que o **Servidor** está utilizando um computador clássico, então ele não consegue inferir p_C para um $C \sim \mathcal{D}$ arbitrário em tempo polinomial em n. Portanto, o **Servidor** tem pouco a fazer além de amostrar uniformemente strings de $\{0,1\}^n$.
- Nós então esperaríamos que $\frac{1}{k}\sum_{i=1}^{k} p(s_i) = \frac{1}{N} \Rightarrow$ essa estratégia clássica tem alta probabilidade de passar no LXEB$(1,k)$. Logo, se garantirmos que $b > 1$, podemos esperar que **nenhum algoritmo clássico conseguirá passar no LXEB com alta probabilidade**.

Por quê um CQ deveria passar no LXEB?

- Agora, assuma que o **Servidor** é honesto, i.e., ele usa um CQ para aplicar C no estado $|0\rangle^{\otimes n}$. Pelo Lema 1, sabemos que um CQ perfeito satisfaz

$$\frac{1}{k}\sum_{i=1}^{k} p_C(z_i) = \frac{2}{N+1}$$

com probabilidade $1 - 1/\exp(k)$, logo queremos garantir que

$$\frac{2}{N+1} \geq \frac{b}{N} \Rightarrow 2(\frac{N}{N+1}) - b \geq 0$$

e então, para valores de k suficientemente grandes, e para $b < 2$, espera-se que um CQ perfeito passe no LXEB com alta probabilidade.

Quantum Supremacy

- O RCS então usa o LXEB$(b, k), b \in (1, 2)$, como o teste estatístico do usuário, com milhares ou milhões de amostras.

Quantum Supremacy

- O RCS então usa o LXEB$(b, k), b \in (1, 2)$, como o teste estatístico do usuário, com milhares ou milhões de amostras.
- Os experimentos de 2019 da Google utilizaram um valor de $b \approx 1.002$, e 54 qubits no seu processador Sycamore. Por quê um valor de b tão baixo? Pois os CQs atuais ainda são extremamente ruidosos, e portanto para valores de b mais altos, tais CQs também falhariam o teste LXEB.

Quantum Supremacy

- O RCS então usa o LXEB$(b, k), b \in (1, 2)$, como o teste estatístico do usuário, com milhares ou milhões de amostras.
- Os experimentos de 2019 da Google utilizaram um valor de $b \approx 1.002$, e 54 qubits no seu processador Sycamore. Por quê um valor de b tão baixo? Pois os CQs atuais ainda são extremamente ruidosos, e portanto para valores de b mais altos, tais CQs também falhariam o teste LXEB.
- Por quê esse resultado experimental é conhecido como **Quantum Supremacy**? Pois resolver o problema de RCS signifca que **o CQ da Google é capaz de resolver tarefas que nenhum computador clássico atual consegue**, mesmo apesar de tais tarefas possuírem poucas aplicações práticas.

Quantum Supremacy

- O RCS então usa o LXEB$(b, k), b \in (1, 2)$, como o teste estatístico do usuário, com milhares ou milhões de amostras.
- Os experimentos de 2019 da Google utilizaram um valor de $b \approx 1.002$, e 54 qubits no seu processador Sycamore. Por quê um valor de b tão baixo? Pois os CQs atuais ainda são extremamente ruidosos, e portanto para valores de b mais altos, tais CQs também falhariam o teste LXEB.
- Por quê esse resultado experimental é conhecido como **Quantum Supremacy**? Pois resolver o problema de RCS signifca que **o CQ da Google é capaz de resolver tarefas que nenhum computador clássico atual consegue**, mesmo apesar de tais tarefas possuírem poucas aplicações práticas.
- Podemos portanto nos perguntar: **Será que é possível utilizar o RCS para criar uma aplicação mais prática para CQs?**

Certified Randomness from Quantum Supremacy

- Um artigo recente escrito por Scott Aaronson e Shih-Han Hung [AH23] descreve **uma forma de utilizar o RCS para criar um sistema de QRNG!**

Certified Randomness from Quantum Supremacy

- Um artigo recente escrito por Scott Aaronson e Shih-Han Hung [AH23] descreve **uma forma de utilizar o RCS para criar um sistema de QRNG!**
- A ideia principal é: usar a saída do **Servidor** para criar bits aleatórios que possuem uma cota inferior para sua entropia mínima, permitindo que o **Usuário** seja capaz de obter um valor próximo de n bits verdadeiramente aleatórios.

Certified Randomness from Quantum Supremacy

- Um artigo recente escrito por Scott Aaronson e Shih-Han Hung [AH23] descreve **uma forma de utilizar o RCS para criar um sistema de QRNG!**
- A ideia principal é: usar a saída do **Servidor** para criar bits aleatórios que possuem uma cota inferior para sua entropia mínima, permitindo que o **Usuário** seja capaz de obter um valor próximo de n bits verdadeiramente aleatórios.
- Entretando, os autores primeiro discutem uma questão que até agora deixamos sem resposta: **e se o Servidos estiver usando um CQ, mas ainda estiver sendo desonesto?** Isto é, e se o Servidor não estiver resolvendo o problema de LXEB por meio de amostragem da distribuição de saída dos circuitos?

LLQSV: formalizando o RCS

Definition (Long List Quantum Supremacy Verification)

Seja $M = O(N^3)$. Suponha que temos acesso a circuitos quânticos de n-qubits $C_1, ..., C_M$ amostrados de maneira independente de $\mathcal{D}$, e a strings $s_1, ..., s_M, s_i \in \{0,1\}^n$, e denote por p_{C_i} a distribuição de saída de C_i, $p_{C_i}(s) = |\langle s|C_i|0^n\rangle|^2$. O problema de LLQSV($\mathcal{D}$) consiste em distinguir dentre os seguintes casos:

- **Yes-case**: cada s_i foi amostrado do respectivo C_i, i.e., $\forall i \in \{1, 2, ..., M\} : s_i \sim p_{C_i}$
- **No-case**: cada s_i foi amostrado de uma distribuição uniforme sobre $\{0,1\}^n$

Entendendo o LLQSV

- O LLQSV pode ser utilizado para modelar o protocolo RCS descrito anteriormente: podemos considerar que o **Usuário** envia circuitos $C_1, ..., C_M$, e que o **Servidor** retorna strings $s_1, ..., s_M$. O LLQSV então pergunta: é possível que o **Servidor** crie um algorítmo que consiga convencer o **Usuário** que ele está no *Yes-case*? Em outras palavras, **existe alguma estratégia eficiente em que o Servidor consiga enganar o Usuário?**

Entendendo o LLQSV

- O LLQSV pode ser utilizado para modelar o protocolo RCS descrito anteriormente: podemos considerar que o **Usuário** envia circuitos $C_1, ..., C_M$, e que o **Servidor** retorna strings $s_1, ..., s_M$. O LLQSV então pergunta: é possível que o **Servidor** crie um algorítmo que consiga convencer o **Usuário** que ele está no *Yes-case*? Em outras palavras, **existe alguma estratégia eficiente em que o Servidor consiga enganar o Usuário?**
- Como discutido previamente, espera-se que nenhuma estratégia clássica consiga enganar um usuário utilizando o LXEB, **mas será que existe alguma estratégia quântica**?

LLHA: existe CQ desonesto?

A pergunta anterior ainda está em aberto, entretanto, para criar um sistema QRNG que garante as propriedades desejadas, Aaronson and Hung **assumem que a resposta é não**.

LLHA: existe CQ desonesto?

A pergunta anterior ainda está em aberto, entretanto, para criar um sistema QRNG que garante as propriedades desejadas, Aaronson and Hung **assumem que a resposta é não**.

Assumption (Long List Hardness Assumption (LLHA))

Não existe nenhum protocolo Quantum-Classical Arthur-Merlin no qual Merlin consegue enganar Arthur e o convencer de que ele está no Yes-case do LLQSV em tempo $n^{O(1)}$, dado que Arthur utilize o LXEB como teste estatístico.

Em seu artigo, os autores mostram que a LLHA implica que qualquer algoritmo quântico que passa no LXEB deve retornar aproximadamente n bits aleatórios em sua saída. A demonstração é feita por contradição: os autores mostram que, se existe um algoritmo com baixa entropia que passa no LXEB, então existe um protocolo QCAM que resolve o LLQSV em tempo polinomial em n, violando então a LLHA.

Certified Randomness Protocol

O protocolo descrito por Aaronson e Hung pode ser sumarizado da seguinte forma:

Certified Randomness Protocol

O protocolo descrito por Aaronson e Hung pode ser sumarizado da seguinte forma:

1. O **Usuário** gera uma sequência de circuitos quânticos de n-qubits $C_1, C_2, ..., C_M$ de forma pseudoaleatória, utilizando uma seed aleatória, e então os envia para o **Servidor**.

Certified Randomness Protocol

O protocolo descrito por Aaronson e Hung pode ser sumarizado da seguinte forma:

1. O **Usuário** gera uma sequência de circuitos quânticos de n-qubits $C_1, C_2, ..., C_M$ de forma pseudoaleatória, utilizando uma seed aleatória, e então os envia para o **Servidor**.
2. Para cada C_i, o **Servidor** rapidamente retorna k strings de n bits. Se o **Servidor** é de fato honesto, então as strings são amostradas da distribuição de saída de $C_i|0\rangle^{\otimes n}$.

Certified Randomness Protocol

O protocolo descrito por Aaronson e Hung pode ser sumarizado da seguinte forma:

1. O **Usuário** gera uma sequência de circuitos quânticos de n-qubits $C_1, C_2, ..., C_M$ de forma pseudoaleatória, utilizando uma seed aleatória, e então os envia para o **Servidor**.
2. Para cada C_i, o **Servidor** rapidamente retorna k strings de n bits. Se o **Servidor** é de fato honesto, então as strings são amostradas da distribuição de saída de $C_i|0\rangle^{\otimes n}$.
3. O **Usuário** usa a sua seed para escolher um subconjunto dos circuitos, e então, para cada um deles, checa se as k amostras retornadas pelo CQ passam no LXEB$(b, k), b \in (1, 2)$.

Certified Randomness Protocol

O protocolo descrito por Aaronson e Hung pode ser sumarizado da seguinte forma:

1. O **Usuário** gera uma sequência de circuitos quânticos de n-qubits $C_1, C_2, ..., C_M$ de forma pseudoaleatória, utilizando uma seed aleatória, e então os envia para o **Servidor**.
2. Para cada C_i, o **Servidor** rapidamente retorna k strings de n bits. Se o **Servidor** é de fato honesto, então as strings são amostradas da distribuição de saída de $C_i|0\rangle^{\otimes n}$.
3. O **Usuário** usa a sua seed para escolher um subconjunto dos circuitos, e então, para cada um deles, checa se as k amostras retornadas pelo CQ passam no LXEB$(b, k), b \in (1, 2)$.
4. Se as amostras passam nos testes, então o **Usuário** fornece as strings para um *randomness extractor* clássico para obter aproximadamente n bits verdadeiramente aleatórios.

Alguns detalhes

- As matriz unitárias que representam os circuitos C_i possuem tamanho $2^n \times 2^n$, e portanto computar o LXEB ainda requer tempo exponencial na quantidade de qubits n, i.e., o **Usuário** já precisaria de um supercomputador para valores relativamente pequenos de n.

Alguns detalhes

- As matriz unitárias que representam os circuitos C_i possuem tamanho $2^n \times 2^n$, e portanto computar o LXEB ainda requer tempo exponencial na quantidade de qubits n, i.e., o **Usuário** já precisaria de um supercomputador para valores relativamente pequenos de n.
- Por outro lado, o protocolo de Aaronson e Hung *inerentemente requer um CQ*, e também foi pensado dentro de um contexto de um problema que consegue ser resolvido por CQ atuais.

Referências

Scott Aaronson and Shih-Han Hung.

Certified randomness from quantum supremacy.

Arxiv (2023).

Arute, F., Arya, K., Babbush, R. et al.

Quantum supremacy using a programmable superconducting processor.

Nature 574, 505–510 (2019).

Xiongfeng Ma, Xiao Yuan, Zhu Cao, Bing Qi and Zhen Zhang.

Quantum random number generation.

npj Quantum Information 2, 16021 (2016).

Amura, K. and Shikano, Y.

Quantum Random Numbers Generated by a Cloud Superconducting Quantum Computer.

In: Takagi, T., Wakayama, M., Tanaka, K., Kunihiro, N., Kimoto, K., Ikematsu, Y. (eds) International Symposium on Mathematics, Quantum Theory, and Cryptography. Mathematics for Industry, vol 33. Springer, Singapore (2021).

Perguntas?