Simon’s Algorithm

Discover a hidden XOR period by sampling binary equations and solving them classically.

Undergraduate 21 min lesson Dr. Charith Jayasekara

By the end, you’ll be able to

- Identify the two-to-one XOR-pairing promise.
- Derive the constraint $y \cdot s = 0$ from a paired state.
- Recover a nonzero three-bit period from independent equations.

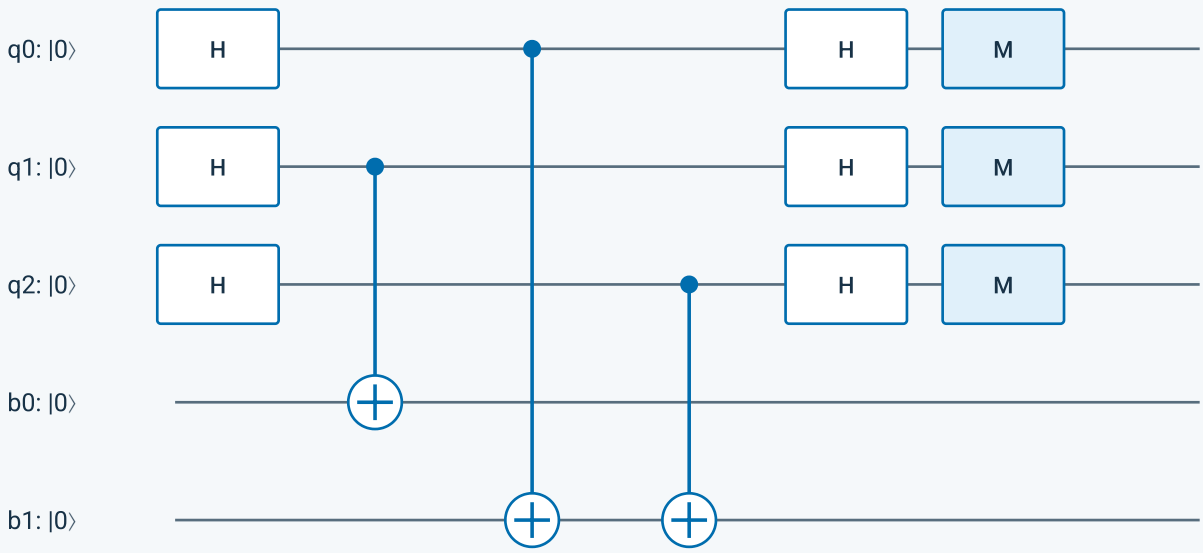
Find a hidden XOR pairing

Simon’s problem hides a bitstring s in the way a function repeats its outputs. Two inputs give the same output exactly when they are equal or differ by XOR with s. The algorithm learns this hidden pairing by collecting linear constraints, then solves those constraints classically. A single quantum run usually does not reveal s.

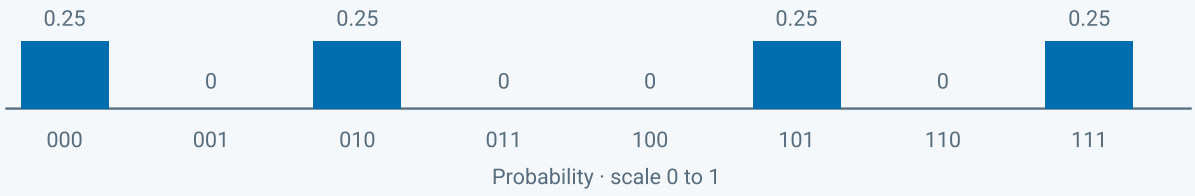
The prerequisites are tensor products, measurement and binary Fourier sampling. You also need equations modulo two: XOR plays the role of addition, so adding an equation to itself cancels it.

WORKED DIAGRAM · A05

Simon: paired inputs become binary equations



Allowed Fourier constraints



The three CNOTs form one reversible query with hidden XOR period 101. Ignore the output register. Input readout is uniform on 000, 010, 101, 111; collect independent equations y·s=0.

State the promise

For the main version here, s is a nonzero n-bit string. The function has the property

$$f(x) = f(y) \quad \Longleftrightarrow \quad y = x \text{ or } y = x \oplus s.$$

Every output therefore corresponds to exactly two inputs. The oracle reversibly XORs f(x) into a sufficiently large output register. The desired classical output is s. The promise is stronger than merely saying the function has some collisions: the collisions must follow one consistent XOR pairing across the whole domain.

A common extended version permits s equal to zero, corresponding to an injective function. That version also needs a rule for distinguishing the injective case. Start with the nonzero promise so the pairing and constraint mechanism remain visible.

The procedure

Prepare the input uniformly with H gates and initialise the output register to zero. Query the function. Measure the output register, or discard it in a way that produces the same input statistics. Apply H to the input and measure it, obtaining a string y. Repeat on fresh registers to collect equations. Each y satisfies s dot y equal to zero modulo two.

Once the equations have rank n minus one, their solution space has two elements: zero and the promised nonzero s. Solve by Gaussian elimination over binary arithmetic, then return the nonzero solution. Repeated or all-zero samples do not add independent information, so count the rank rather than the raw number of samples.

One oracle evaluation correlates each input with its function value. Conditioning on one output label leaves its promised pair.

$$\frac{1}{\sqrt{2^n}}\sum_x |x\rangle|0^m\rangle \xrightarrow{U_f} \frac{1}{\sqrt{2^n}}\sum_x |x\rangle|f(x)\rangle \quad \rightsquigarrow \quad \frac{|x\rangle + |x\oplus s\rangle}{\sqrt{2}}.$$

The last state is the conditional input state after selecting an output value, for the nonzero-period case. m is the output-register width. Measuring that register is optional; ignoring it gives the same input measurement statistics later.

Checkpoint 2

PAUSE & THINK

A Simon run returns 000. What has this run learned about a nonzero s?

- ☐ That s must be 000
- ☐ The complete hidden string
- ☐ No additional linear constraint

Answer: No additional linear constraint. The zero sample is valid but uninformative. Further independent samples are required.

Worked three-bit example

Let s be 101. A valid function pairs 000 with 101, 001 with 100, 010 with 111, and 011 with 110. Assign a different output label to each pair. For example, use 00, 01, 10 and 11 respectively. The labels themselves do not reveal the period; their consistent pairing does.

Suppose the output measurement selects the first pair. The input becomes the equal superposition of 000 and 101. After H on all three input qubits, only y values with first and last bits equal can occur. The possible strings are 000, 010, 101 and 111, each with equal probability. Selecting another pair gives the same allowed set of y values, though some amplitudes can have different signs.

Imagine the first useful samples are 010 and 101. The first gives the equation that the middle bit of s is zero. The second says the first and last bits are equal. The solutions are 000 and 101. Under the nonzero promise, the answer is 101. A third sample 111 gives no new independent equation because it is the XOR of the earlier two.

For s=101, the allowed samples have equal first and last bits. Two independent samples determine the nonzero hidden string.

$$\begin{aligned} y = 010 &\implies s_1 = 0, \\ y = 101 &\implies s_0 \oplus s_2 = 0, \\ s \neq 000 &\implies s = 101. \end{aligned}$$

The possible outputs are $000, 010, 101, 111$, each with probability $1/4$. A repeated equation, or 000, adds no new constraint.

Checkpoint 1

PAUSE & THINK

For hidden s=101, which label is forbidden by the ideal circuit?

- [ ] 010
- [ ] 100
- [ ] 111

Answer: 100. The allowed labels satisfy y0⊕y2=0. Label 100 violates that equation.

Why the unwanted outcomes cancel

After the output is measured, the input has the form of an equal superposition of x and x XOR s. The amplitude of y after the Hadamard transform is proportional to

$$(-1)^{x\cdot y}\left[1+(-1)^{s\cdot y}\right].$$

If s dot y is one, the bracket is zero and the outcome cannot occur. If it is zero, the two contributions reinforce. The unknown x affects a sign but not the allowed support. Thus measuring an arbitrary output label does not prevent the input register from revealing a useful period constraint.

This is a major conceptual change from Bernstein–Vazirani. There, one transform returns the hidden string. Here, the transform samples the subspace orthogonal to that string. Classical postprocessing combines several partial pieces. Quantum algorithms often output useful constraints rather than an immediately readable complete answer.

Include the normalisation to see both which outcomes survive and their probabilities.

$$\alpha_y = \frac{(-1)^{x\cdot y}}{\sqrt{2^{n+1}}}\left(1+(-1)^{s\cdot y}\right), \qquad P(y) = \begin{cases} 2^{-(n-1)}, & s\cdot y = 0, \\ 0, & s\cdot y = 1. \end{cases}$$

Each allowed y supplies one linear equation about s, over binary arithmetic. The conditional phase depends on the selected pair, while this probability distribution does not.

Complexity and assumptions

The ideal algorithm needs order n samples for a constant probability of collecting sufficient independent equations, with additional repetitions to reduce failure probability. Each sample uses one coherent function query and order n H gates outside the oracle. Straightforward binary Gaussian elimination has polynomial classical cost. In the usual black-box model, classical collision-finding requires exponentially many queries in n for bounded success, on the square-root-of-domain scale.

That separation depends on the promised access model. If a particular implementation reveals the hidden string through its source code or additional metadata, the classical task changes. A practical implementation must also include output-register size, reversible function construction and noise. The algorithm is an important theoretical example and a precursor to period-finding techniques, not a claim that every function with repeated outputs becomes easy.

A useful hand experiment

Write the four pairs for a chosen three-bit s. Pick one pair and calculate the eight post-H amplitudes using the two-term formula. Check that exactly the orthogonal y strings survive. Then generate several allowed y values and solve their equations. This separates the quantum sampling proof from the classical reconstruction step.

If a noisy run yields a y with nonzero dot product against the true s, the exact linear equations become inconsistent. Handling noisy constraints is a different inference problem; the ideal proof cannot simply be reused unchanged. State the noise model and recovery method before claiming robustness.

Explore the mechanism

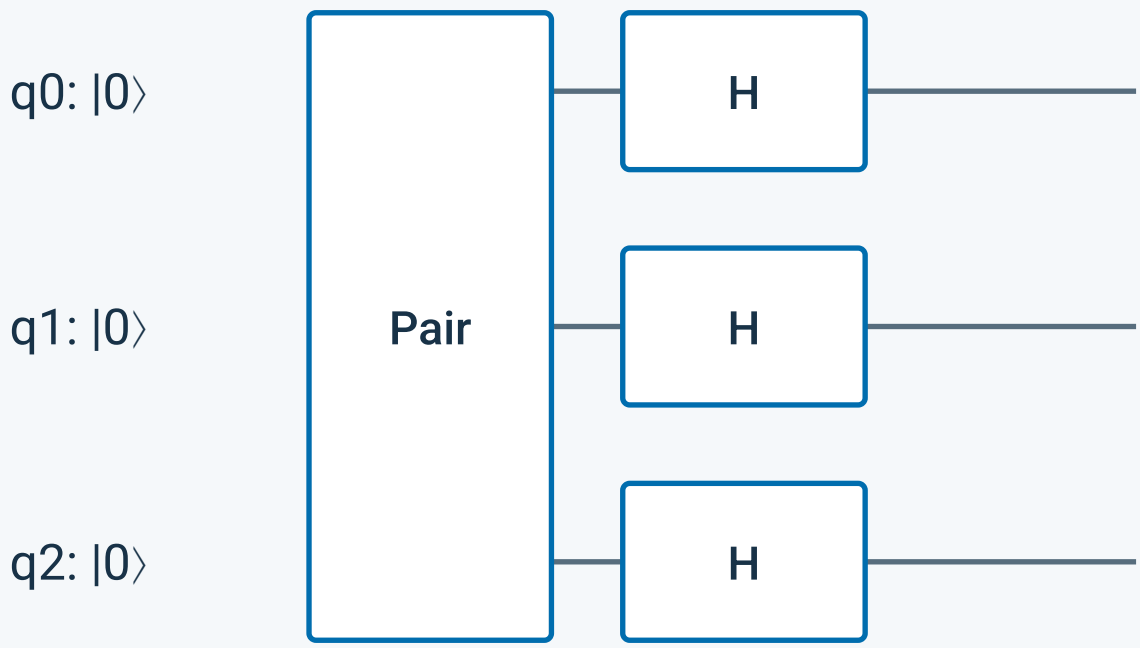
Use Explore freely to compare amplitudes and measurement probabilities. Choose Predict & test for a quick prediction, then run measurements. Change one setting at a time and explain the change.

ALGORITHM LAB

Ideal local model

Simon: collect XOR constraints

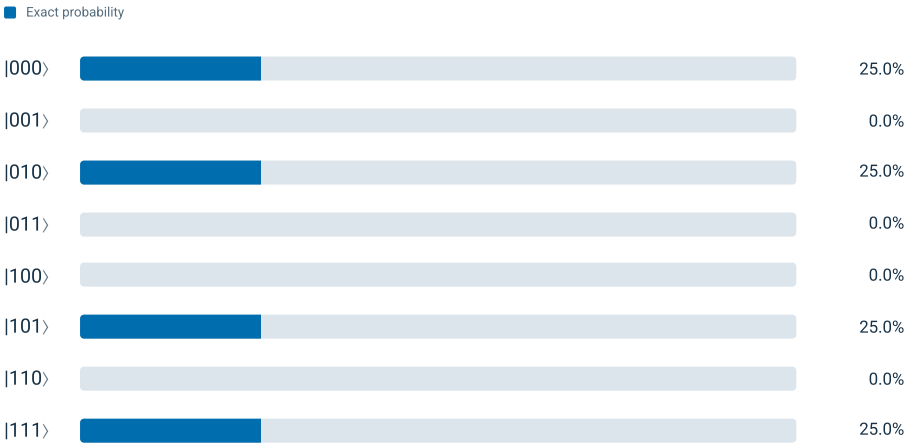
The input is conditioned on measuring a function value: its two preimages are x and x XOR s. H produces linear constraints on the hidden period.



Selected pair: 000 and 101. The full function-query circuit appears in the chapter.

The table is the conditional input-register state after the final H gates. Collect independent constraints to solve for s over binary arithmetic.

Initial model reference



Statevector before readout · ideal simulation

Basis	Amplitude a + bi	Magnitude	Phase (rad)	Probability
|000⟩	`0.5 + 0i`	0.500000	0.000000	25.0%
|001⟩	`0 + 0i`	0.000000	undefined	0.0%
|010⟩	`0.5 + 0i`	0.500000	0.000000	25.0%
|011⟩	`0 + 0i`	0.000000	undefined	0.0%
|100⟩	`0 + 0i`	0.000000	undefined	0.0%
|101⟩	`0.5 + 0i`	0.500000	0.000000	25.0%
|110⟩	`0 + 0i`	0.000000	undefined	0.0%
|111⟩	`0.5 + 0i`	0.500000	0.000000	25.0%

Signs and complex phases are part of the state. Phase is undefined for a zero amplitude. A shared global phase changes no probabilities; relative phases can affect a later operation. Values are rounded to six decimal places.

Chapter summary

- The oracle's equal-output pairs differ by the same hidden XOR string.
- Hadamards turn a pair into random linear constraints y·s=0.
- Collect independent constraints and solve them using binary linear algebra.

A common misconception

One Simon measurement directly returns the hidden string. It returns a random string orthogonal to the hidden string. Several independent equations are needed; a sample can coincidentally equal s in some examples.

Code practice

Download the tested Qiskit notebook and follow the pinned environment instructions. Inspect the circuit, predict a result and run its independent assertions.

Exercises

1. For s equal to 110, which of 001, 010, 100 and 111 satisfy the allowed constraint?
2. In the worked s-equals-101 example, do samples 010 and 111 determine the nonzero solution? Solve them explicitly.

Worked answers

1. The constraint is that the first two bits of y agree. Strings 001 and 111 satisfy it; 010 and 100 do not.
2. The 010 equation makes the middle hidden bit zero. The 111 equation then makes the first and last hidden bits equal. The solutions are 000 and 101, so the nonzero promise again selects 101. The identity of the independent samples can vary; their rank is what matters.

Check your understanding

PAUSE & THINK

With hidden period 101, which sample is forbidden in the ideal algorithm?

- ☐ 010
- ☐ 111
- ☐ 100

Answer: 100. 100 has binary dot product one with 101, so its paired Fourier contributions cancel. The other two strings have dot product zero.

Sources

IBM's Simon algorithm lesson states the promise and derives the orthogonal-subspace sampling rule.