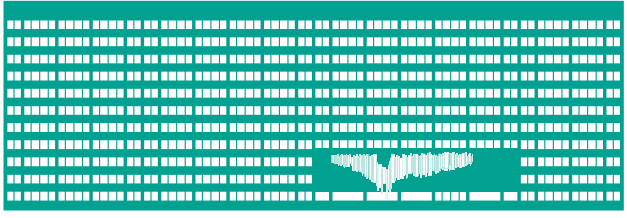
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Iterative warm-started QAOA for QUBO problems on noisy quantum hardware

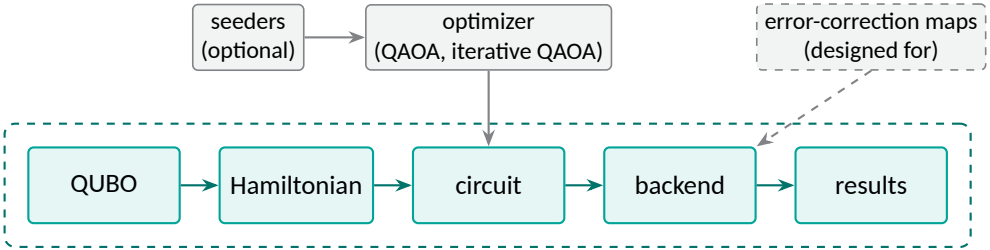
Ryszard Kukulski Adam Bílek

IT4Innovations, VSB – Technical University of Ostrava

Q-Neko webinar, 27 October 2026

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

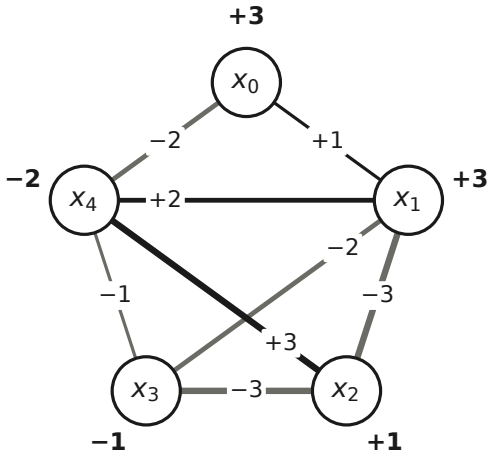
The framework



Everything today is built from this package's stages; the iterative loop in part 2 is one of its optimizers, and the controller in part 3 plugs into that loop.

Modular and backend-agnostic: swap any stage, plug in any backend.

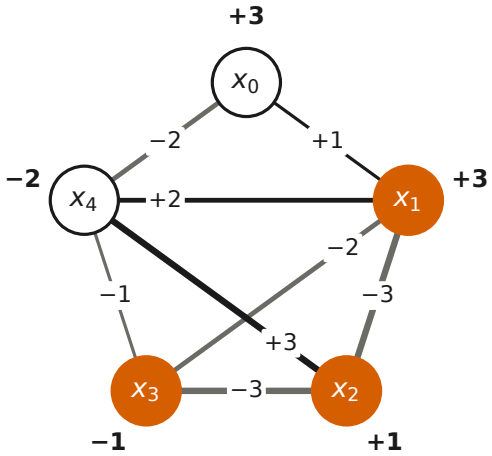
Our running example



cost $= x^{\mathsf{T}} Q x$, Q holds these weights

Cost of 01110 ($x_1 = x_2 = x_3 = 1$):

Our running example



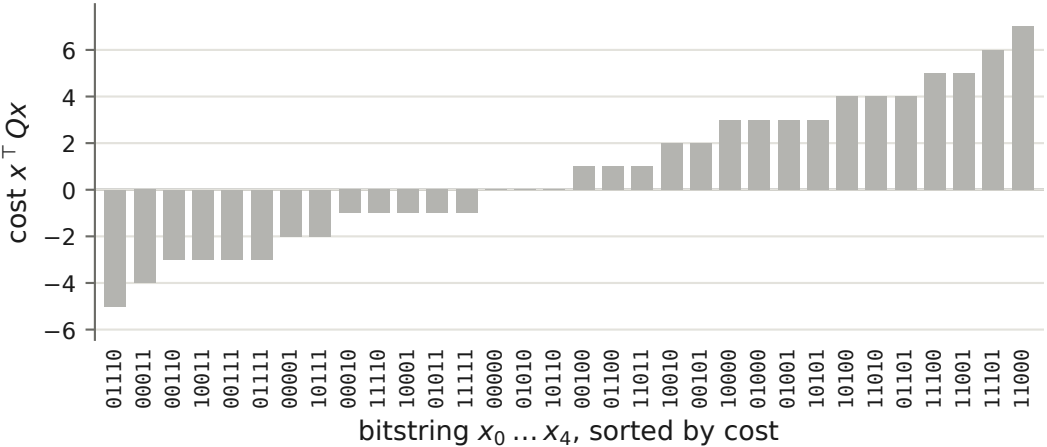
optimum 01110, cost −5

cost $= x^{\mathsf{T}} Q x$, Q holds these weights

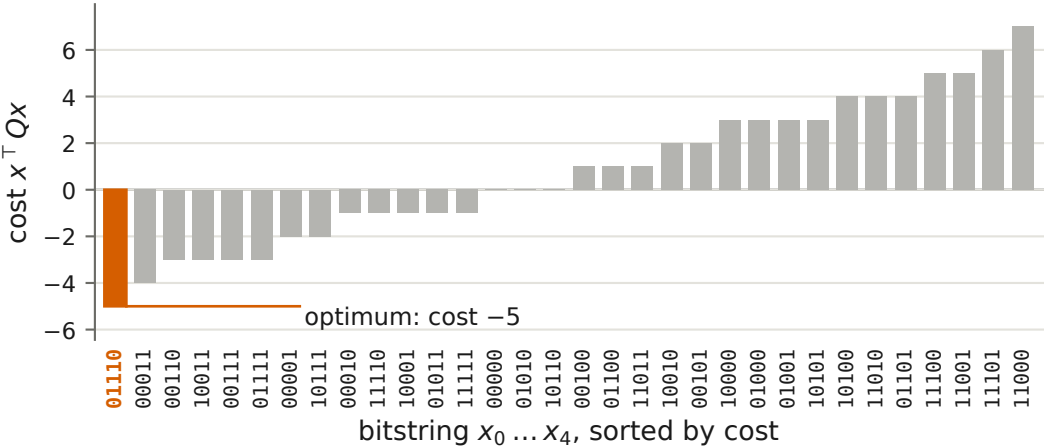
Cost of 01110 ($x_1 = x_2 = x_3 = 1$):
the three filled nodes and the three edges between them:

$$\underbrace{(3 + 1 - 1)}_{\text{nodes}} + \underbrace{(-3 - 2 - 3)}_{\text{edges}} = -5$$

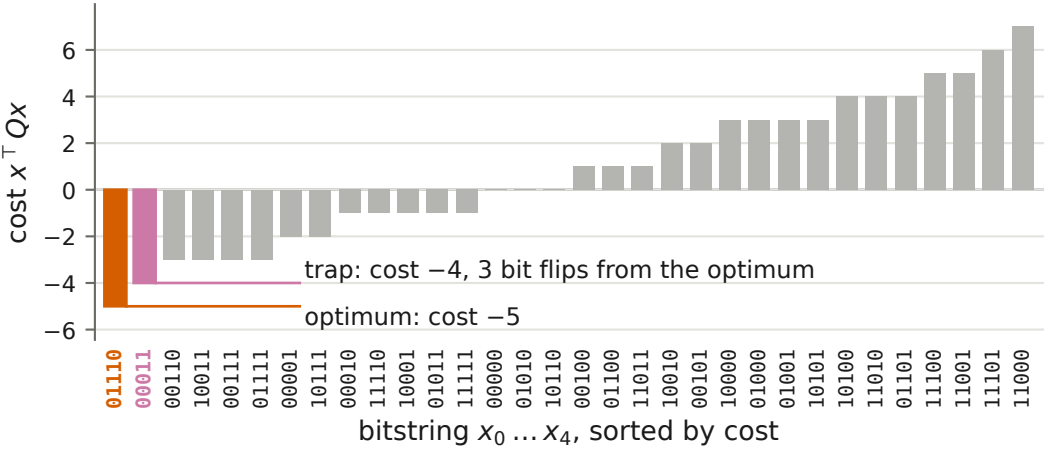
All 32 answers



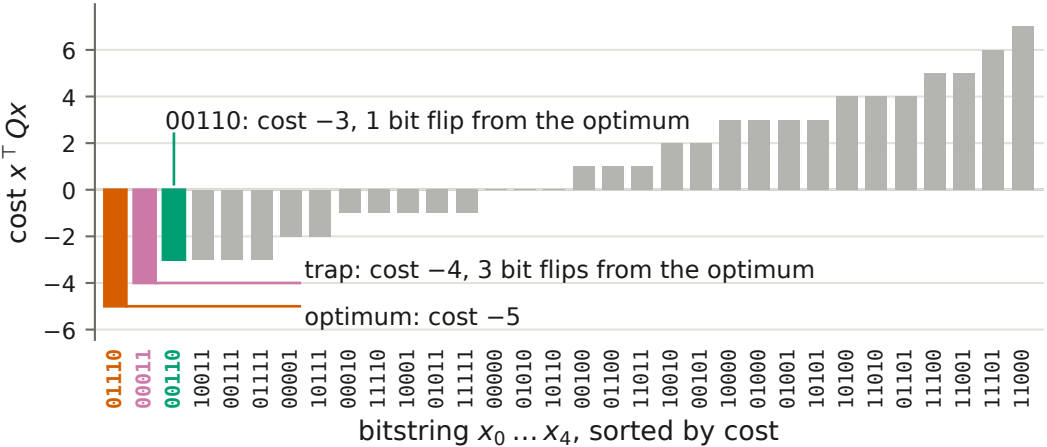
All 32 answers



All 32 answers



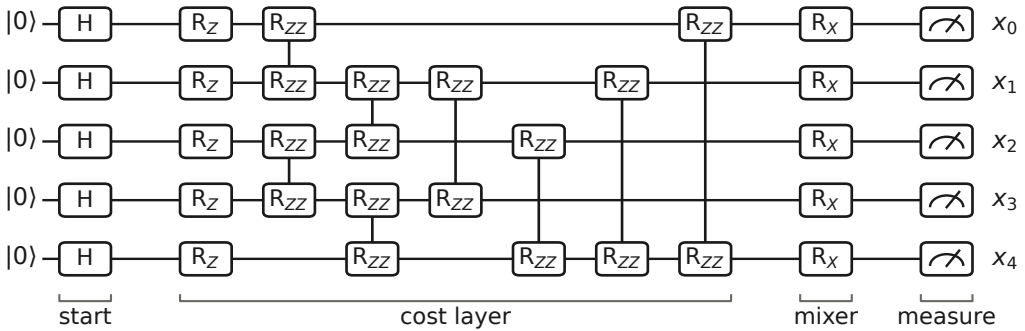
All 32 answers



Remember two answers: the trap (cost -4, 3 bit flips from the optimum) and 00110 (cost -3, 1 bit flip from the optimum).

From cost to circuit

$x = (1 - z)/2$ turns the QUBO into an Ising cost with fields h_i and couplings J_{ij}.

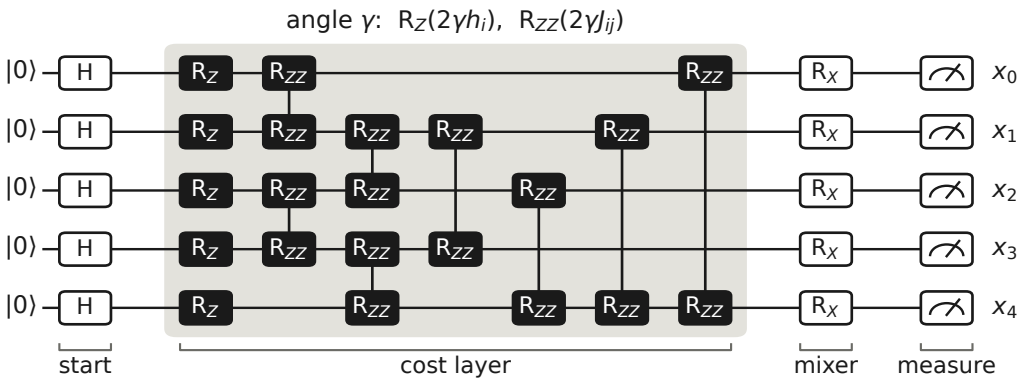


Run it once = one shot = one measured bitstring (also called a sample), scored exactly on a classical computer.

With enough layers QAOA can reach the optimum, provided it starts in the mixer's top eigenstate (here: the uniform superposition).

From cost to circuit

$x = (1 - z)/2$ turns the QUBO into an Ising cost with fields h_i and couplings J_{ij}.

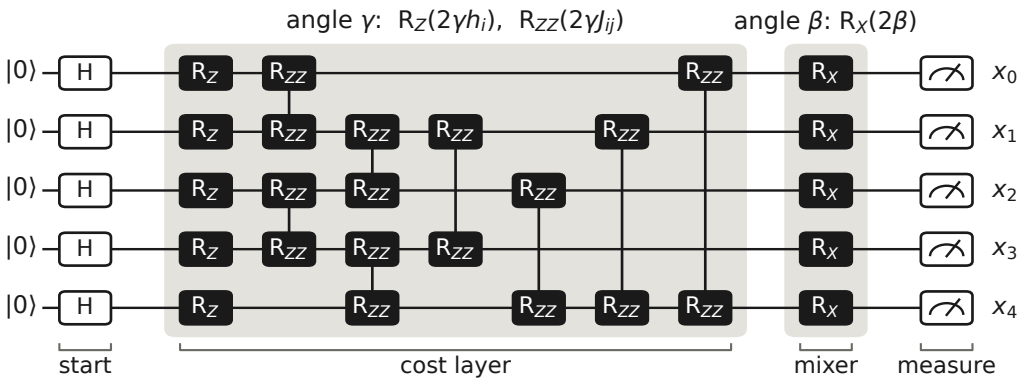


Run it once = one shot = one measured bitstring (also called a sample), scored exactly on a classical computer.

With enough layers QAOA can reach the optimum, provided it starts in the mixer's top eigenstate (here: the uniform superposition).

From cost to circuit

$x = (1 - z)/2$ turns the QUBO into an Ising cost with fields h_i and couplings J_{ij}.



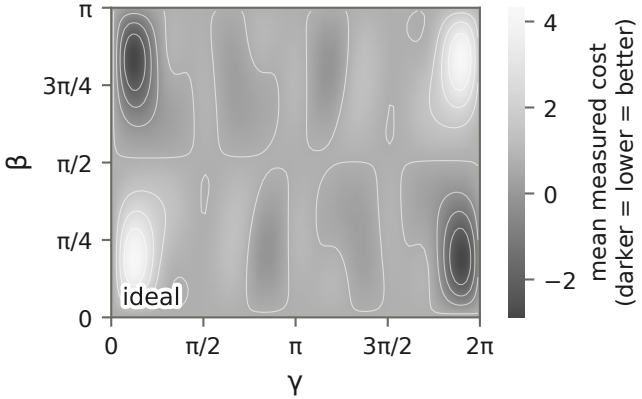
Run it once = one shot = one measured bitstring (also called a sample), scored exactly on a classical computer.

One layer ($p = 1$): a cost layer with angle γ and a mixer with angle β.

With enough layers QAOA can reach the optimum, provided it starts in the mixer's top eigenstate (here: the uniform superposition).

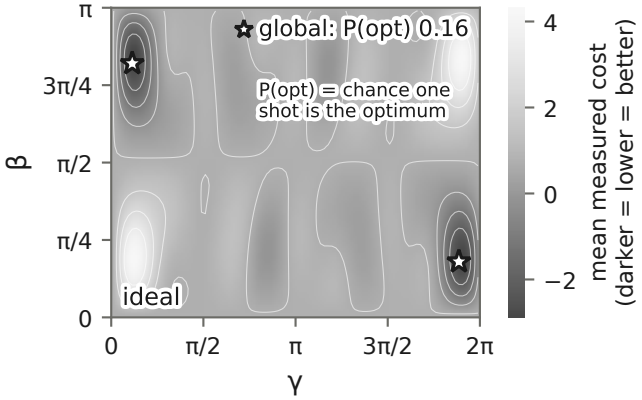
Choosing the angles

simulated VLQ = VLQ noise model: 24 qubits, star resonator, MOVE gates, per-qubit errors, T1/T2 relaxation



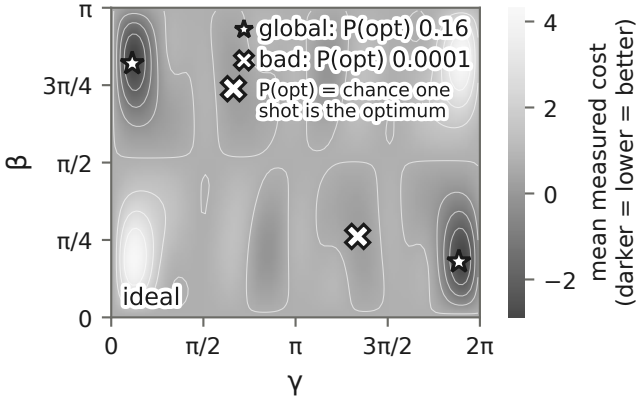
Choosing the angles

simulated VLQ = VLQ noise model: 24 qubits, star resonator, MOVE gates, per-qubit errors, T1/T2 relaxation



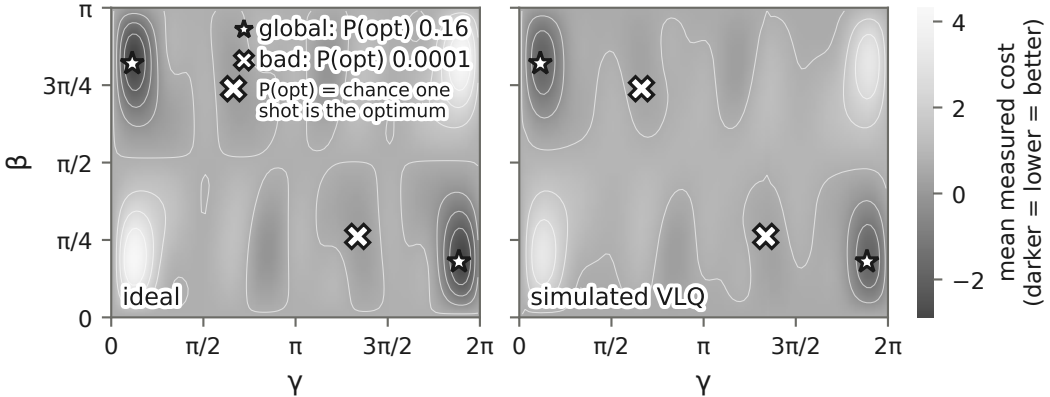
Choosing the angles

simulated VLQ = VLQ noise model: 24 qubits, star resonator, MOVE gates, per-qubit errors, T1/T2 relaxation



Choosing the angles

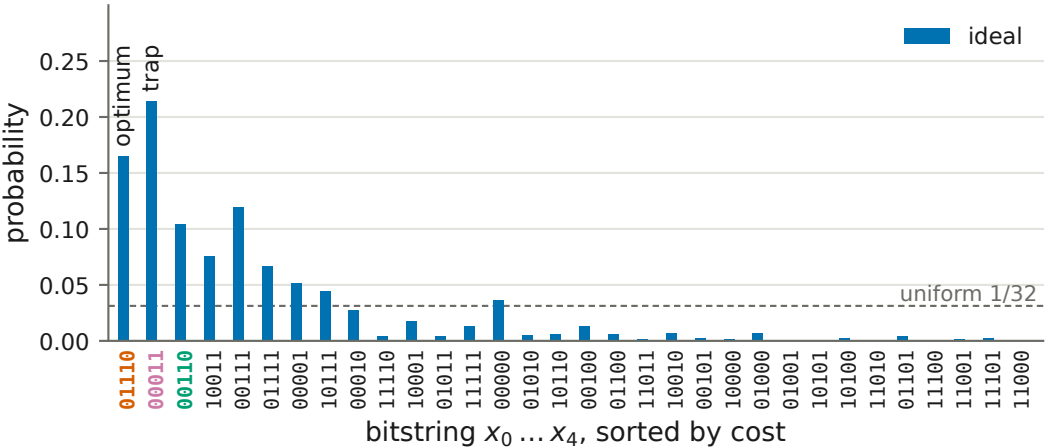
simulated VLQ = VLQ noise model: 24 qubits, star resonator, MOVE gates, per-qubit errors, T1/T2 relaxation



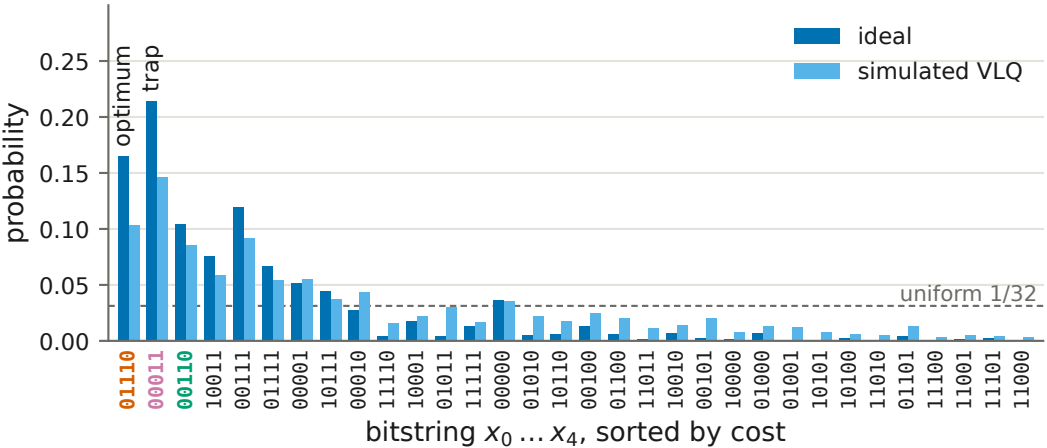
lowest mean cost: ideal -2.920 vs simulated VLQ -1.864

The angles matter (P(opt) about $1500\times$ higher ideally, $10\times$ on the simulated VLQ); noise flattens the landscape but barely moves the best valley, so we choose the angles offline, without spending shots.

What comes out

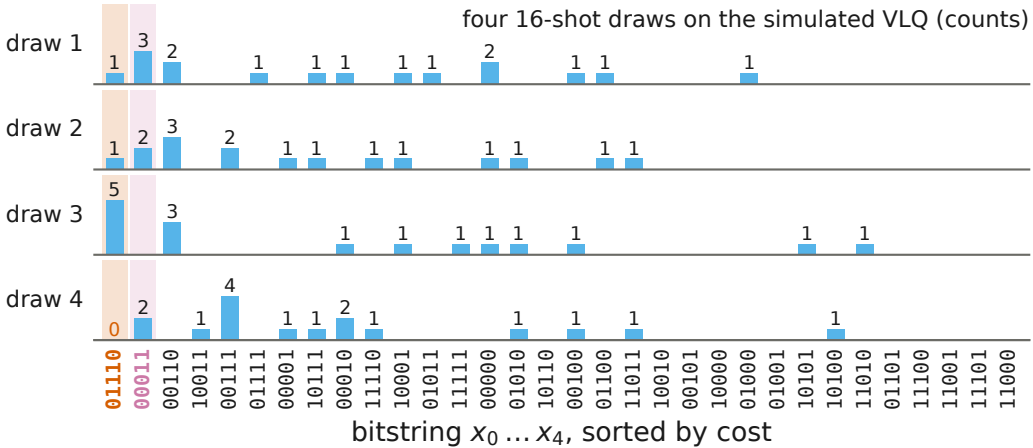


What comes out



QAOA prefers good answers, but its favourite is the TRAP (0.21 vs 0.16 for the optimum, ideal).

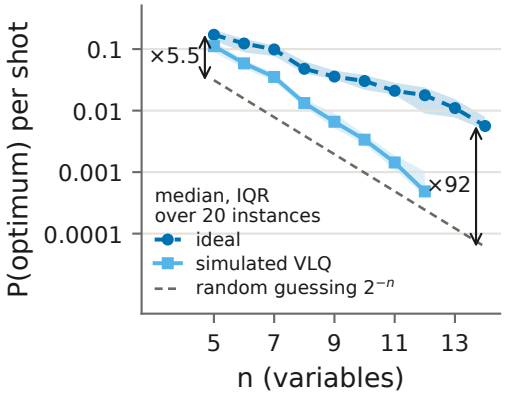
Few shots are a lottery



these 4 draws: the optimum 1, 1, 5 and 0 times; on average a 16-shot run misses it about 1 time in 6

With few shots one run is a lottery: the same circuit shows the optimum 5 times, once, or not at all.

Limits of a single QAOA run



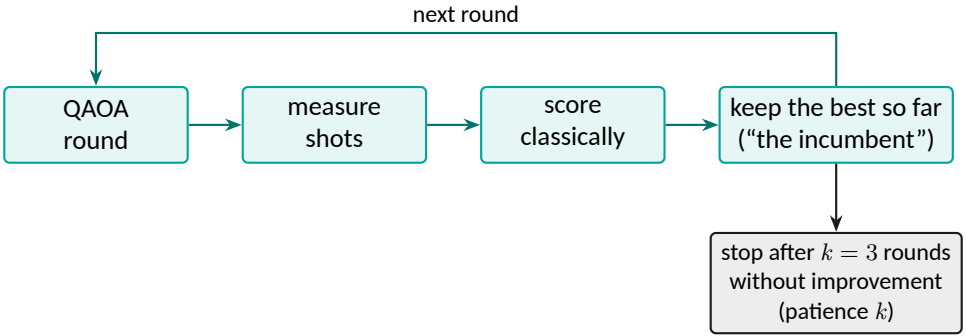
instances with several optimal answers: per optimal bitstring (so random guessing = 2^{-n} exactly)

theory: one layer gives no guarantee

per shot, P(optimum) falls with n
($0.17 \to 0.0056$)
though ideally it beats guessing by more and more
($\times 5.5 \to \times 92$)
noise eats a growing share
($\times 1.5$ at $n = 5$, $\times 37$ at $n = 12$)

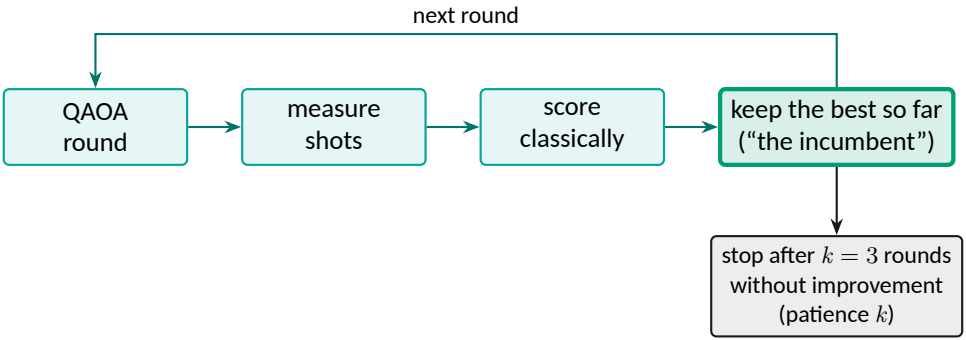
A single run hits the optimum less often as n grows, noise makes that much worse.

The loop



The answer can only get better. Next: make each round search near the incumbent.

The loop



The answer can only get better. Next: make each round search near the incumbent.

The method: ND-AWS, Noise-Directed Adaptive Warm-Starting (Maciejewski et al., arXiv:2607.09368). AWS = each round re-centres on the incumbent with a tilted warm start; ND = relabel the problem so the incumbent is 00000. Our own implementation.

Step 1: relabel so the incumbent reads 00000

answer	bits	the circuit sees	flips from 00000
incumbent	00110	00000	0
optimum	01110	01000	1
trap	00011	00101	2

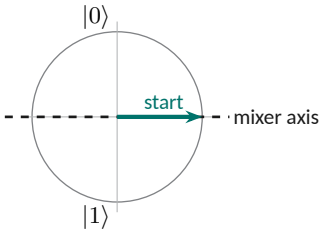
XOR every measured bitstring with the incumbent to read it back; in the circuit this only flips the sign of some cost gates — same energies, same answers.

In the circuit's frame, "near 00000" means "near the incumbent".

Step 2: tilt every qubit towards 0

one qubit: $\sqrt{1-c}\,|0\rangle + \sqrt{c}\,|1\rangle$

c = probability that a qubit starts flipped, i.e. away from the incumbent

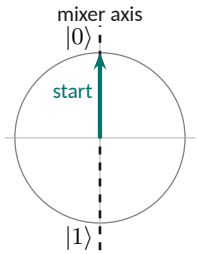


$c = 0.5$: start on the equator
(stock QAOA)

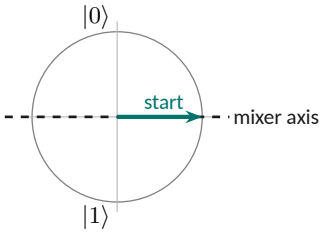
Step 2: tilt every qubit towards 0

one qubit: $\sqrt{1-c}\,|0\rangle + \sqrt{c}\,|1\rangle$

c = probability that a qubit starts flipped, i.e. away from the incumbent



$c = 0$: the mixer becomes Z —
it can only add phases,
not flip bits: frozen

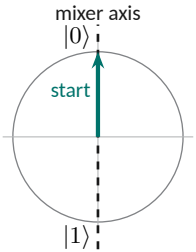


$c = 0.5$: start on the equator
(stock QAOA)

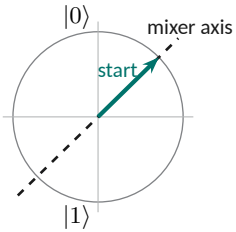
Step 2: tilt every qubit towards 0

one qubit: $\sqrt{1-c}\,|0\rangle + \sqrt{c}\,|1\rangle$

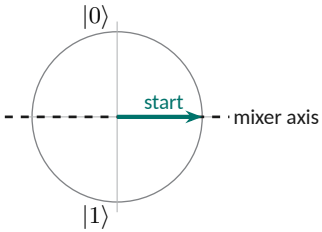
c = probability that a qubit starts flipped, i.e. away from the incumbent



$c = 0$: the mixer becomes Z — it can only add phases, not flip bits: frozen



in between: start tilted by c, mixer axis tilted with it

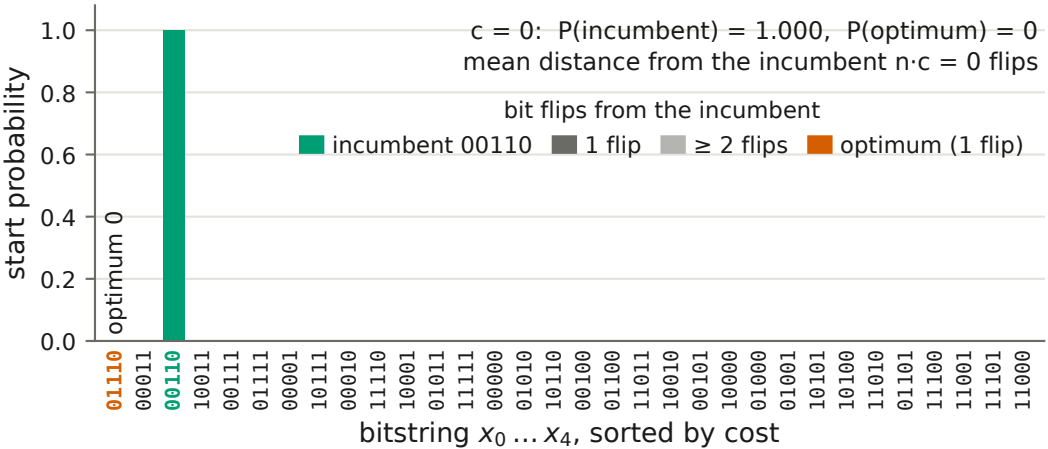


$c = 0.5$: start on the equator (stock QAOA)

Tilt the start AND the mixer together: $c = 0$ freezes, $c = 0.5$ is stock QAOA, in between is a focused search.

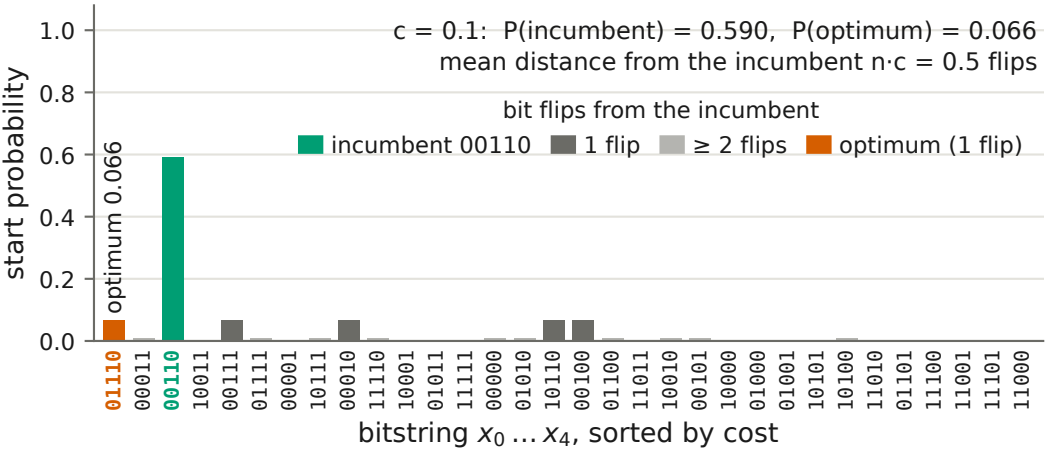
The start state is a cloud

incumbent 00110, 1 bit flip from the optimum



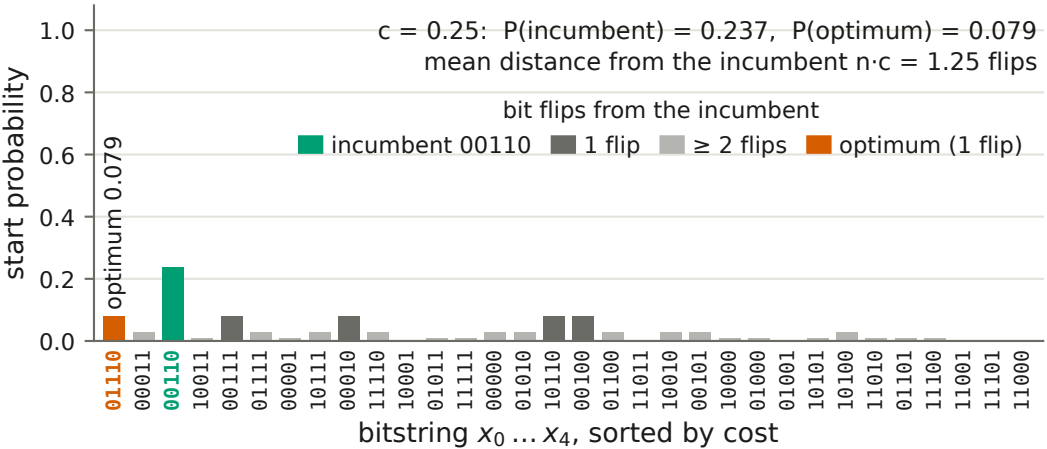
The start state is a cloud

incumbent 00110, 1 bit flip from the optimum



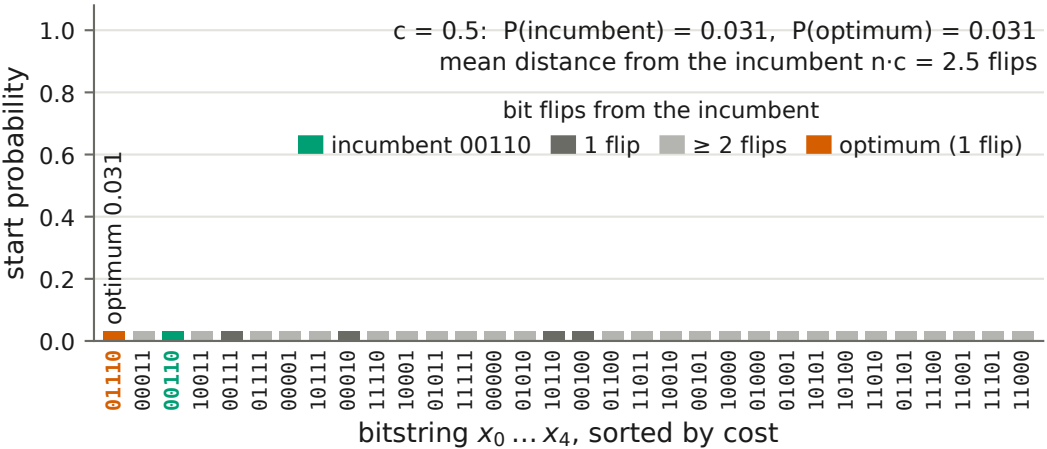
The start state is a cloud

incumbent 00110, 1 bit flip from the optimum



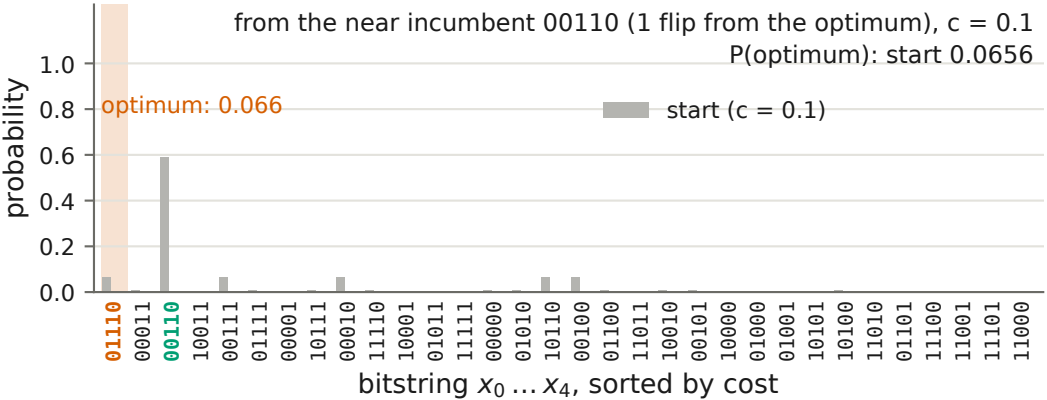
The start state is a cloud

incumbent 00110, 1 bit flip from the optimum

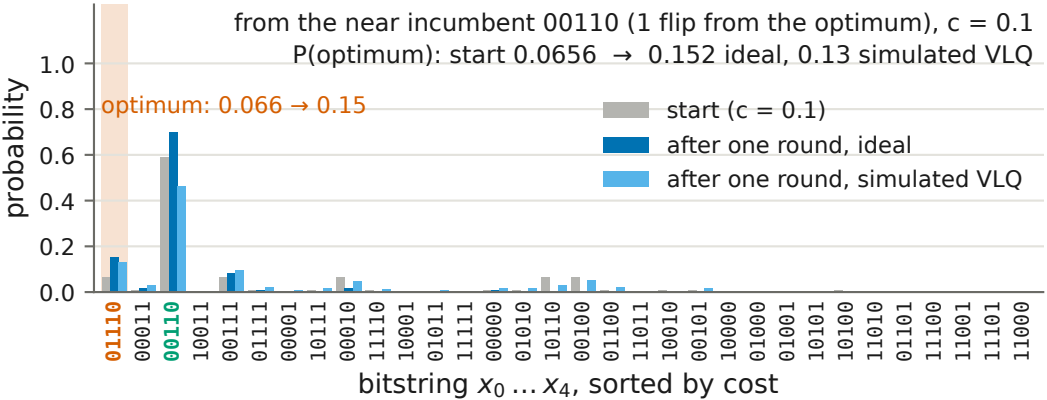


Each bit flips with probability c, so the cloud sits about $n \cdot c$ flips from the incumbent.

One round reshapes the cloud

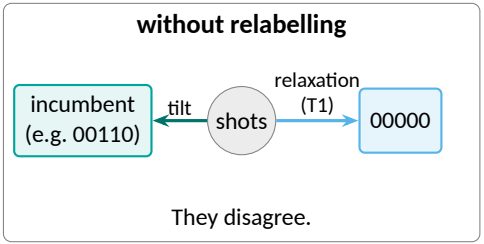


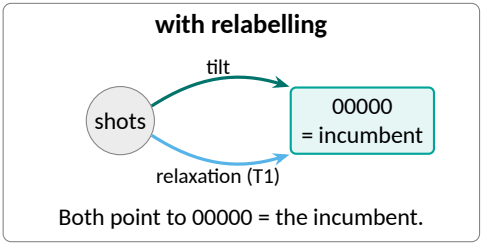
One round reshapes the cloud



The circuit pushes the cloud's weight towards lower cost: P(optimum) 0.066 → 0.152 (ideal; simulated VLQ 0.130).

Why relabel: let noise help





Relaxation pulls qubits towards 0; after the relabelling that means towards the incumbent. In our model it keeps slightly more weight on the incumbent (0.55 vs 0.53) but does not raise P(optimum) at $n = 5$; the paper reports the benefit in $n = 20$ simulations with amplitude damping (Fig. 2) and on 100 qubits of ibm_boston (Fig. 3).

One round, step by step



1 incumbent | 2 relabel | 3 angles & tilt | 4 measure | 5 XOR back | 6 keep best

incumbent

00110 cost −3

One round, step by step

1 incumbent | **2 relabel** | 3 angles & tilt | 4 measure | 5 XOR back | 6 keep best

incumbent

00110 cost −3

relabel: flip x_2, x_3

`00110 reads 00000`

One round, step by step

1 incumbent | 2 relabel | 3 angles & tilt | 4 measure | 5 XOR back | 6 keep best

incumbent

00110 cost −3

relabel: flip x_2, x_3

`00110 reads 00000`

γ = 0.33, β = 2.64 (offline, 0 shots)

tilt c = 0.1

One round, step by step



1 incumbent | 2 relabel | 3 angles & tilt | **4 measure** | 5 XOR back | 6 keep best

incumbent

00110 cost −3

relabel: flip x_2, x_3

00110 reads 00000

γ = 0.33, β = 2.64 (offline, 0 shots)

tilt c = 0.1

circuit sees	shots
00000	8
00010	2
00001	1
00101	1
00110	1
01000	1
other (2 strings)	2

One round, step by step



1 incumbent | 2 relabel | 3 angles & tilt | 4 measure | **5 XOR back** | 6 keep best

incumbent

00110 cost −3

relabel: flip x_2, x_3

`00110 reads 00000`

$\gamma = 0.33$, $\beta = 2.64$ (offline, 0 shots)

tilt $c = 0.1$

circuit sees	shots	⊕ 00110	cost
00000	8	00110	−3
00010	2	00100	+1
00001	1	00111	−3
00101	1	**00011**	−4
00110	1	00000	0
01000	1	**01110**	−5
other (2 strings)	2		≥ +1

One round, step by step

1 incumbent | 2 relabel | 3 angles & tilt | 4 measure | 5 XOR back | **6 keep best**

incumbent

00110 cost −3

relabel: flip x_2, x_3

`00110 reads 00000`

$\gamma = 0.33$, $\beta = 2.64$ (offline, 0 shots)

tilt $c = 0.1$

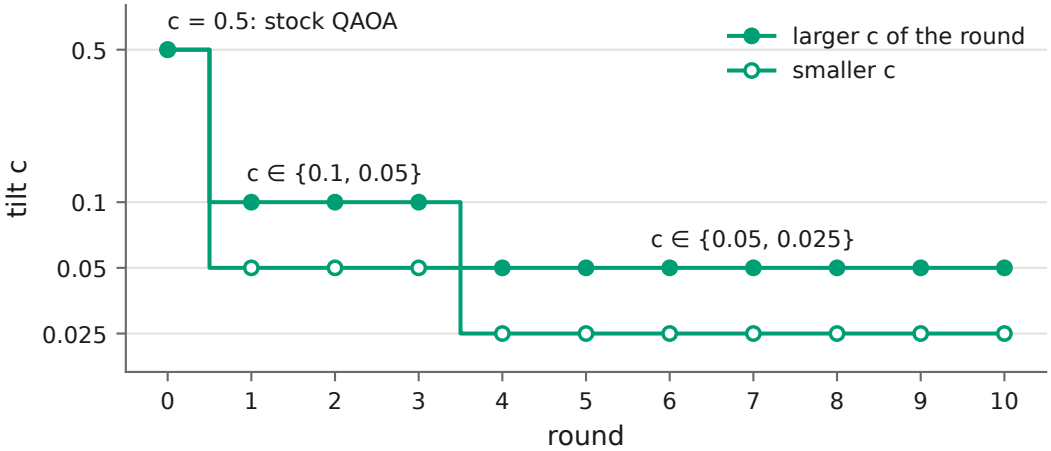
new incumbent

01110 cost −5 (optimum)

circuit sees	shots	⊕ 00110	cost
00000	8	00110	−3
00010	2	00100	+1
00001	1	00111	−3
00101	1	**00011**	−4
00110	1	00000	0
01000	1	**01110**	−5
other (2 strings)	2		≥ +1

The quantum computer only proposes; the classical side checks every proposal exactly and keeps the best.

The published schedule

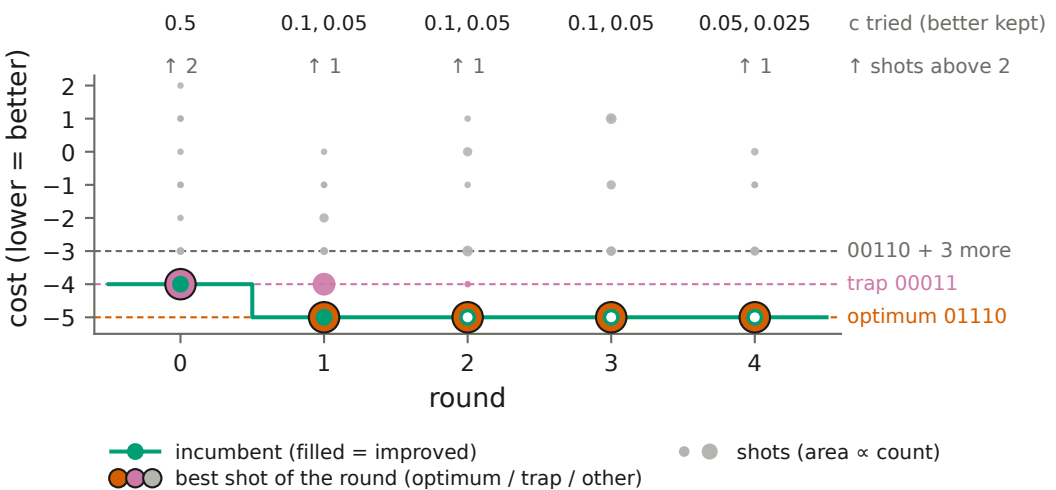


Paper: 10 000 shots per circuit + local search; here: 8–16 shots, none, so a fixed schedule can lock in. (Maciejewski et al., Sec. III.A)

Narrow the focus round by round: two c values per round, the better one kept.

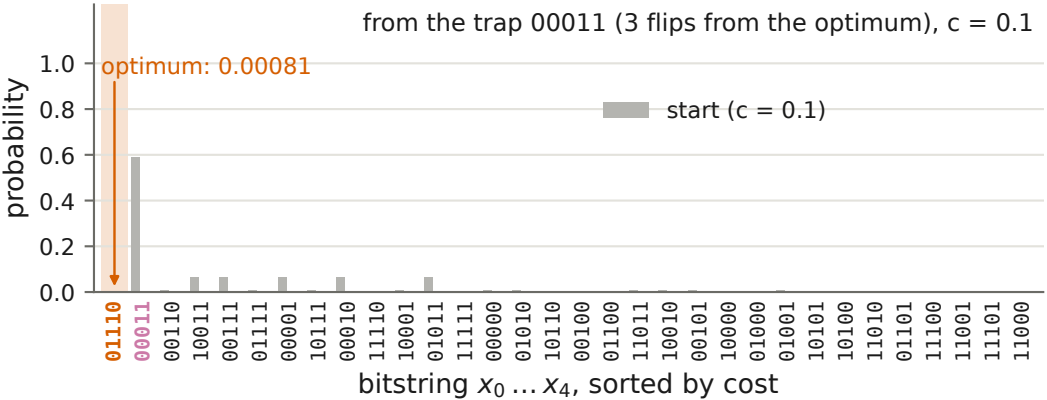
Many rounds

simulated VLQ, running example: a run that starts at the trap and escapes in round 1

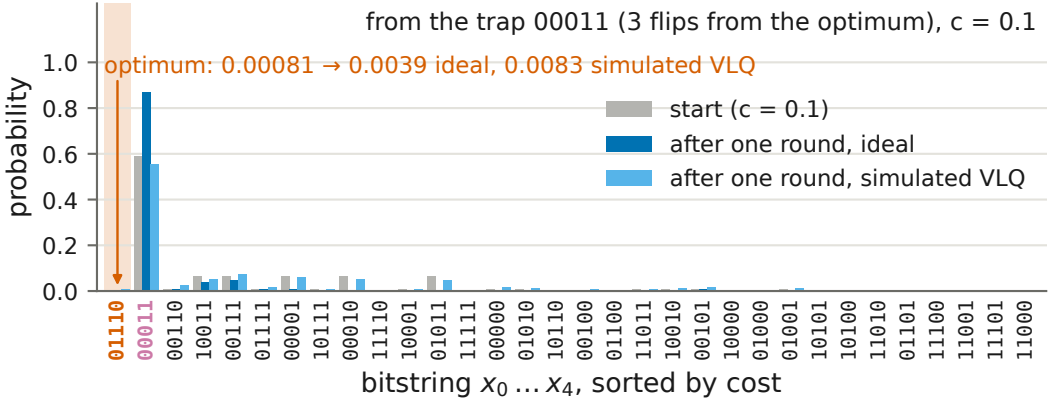


The incumbent improves round by round until it reaches the optimum, or patience $k = 3$ runs out.

When the incumbent is the trap



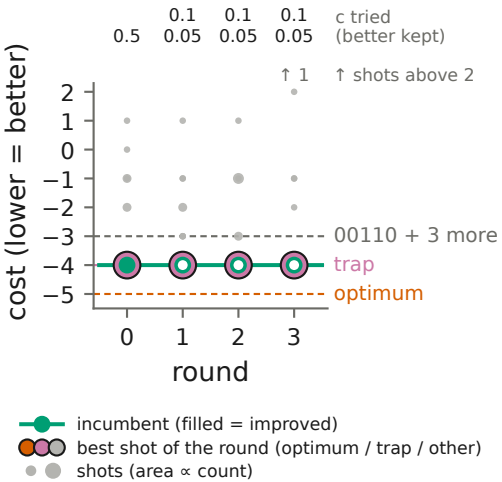
When the incumbent is the trap



From the trap a narrow focus rarely reaches the optimum: one round escapes about 1 time in 5 (simulated VLQ), 1 in 14 ideally.

Lock-in

simulated VLQ, running example, 100 runs: the same loop with 8 shots per c instead of 16

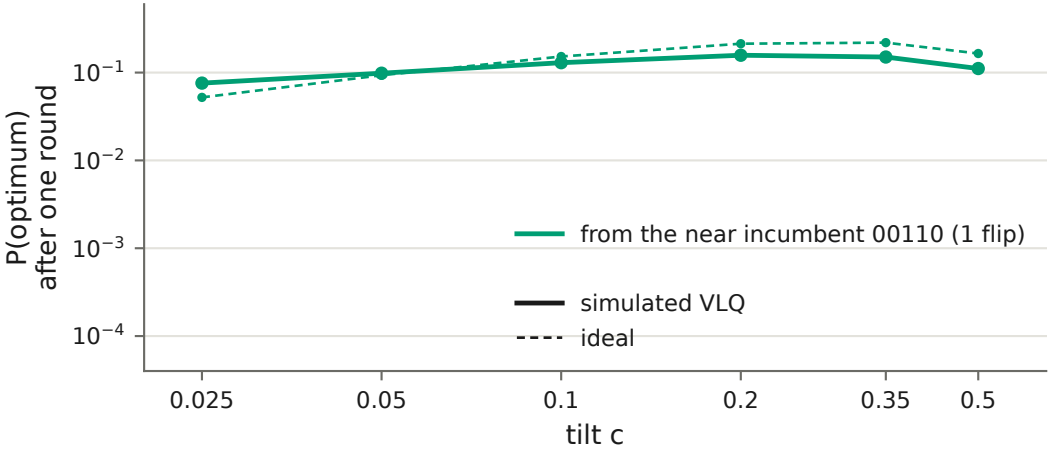


runs that end on the trap	
shots per c	of 100 runs
16	6
8	25

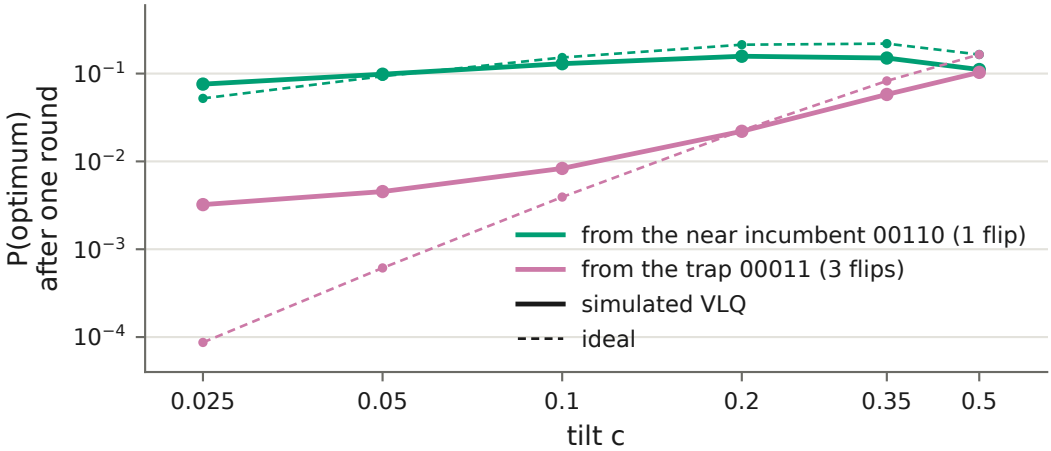
The paper uses 10 000 shots per circuit plus a classical local search; we use 8–16 shots and none, where a fixed schedule can lock in.

With few shots the run can stay stuck on the trap until patience runs out: 25 of 100 runs at 8 shots per c.

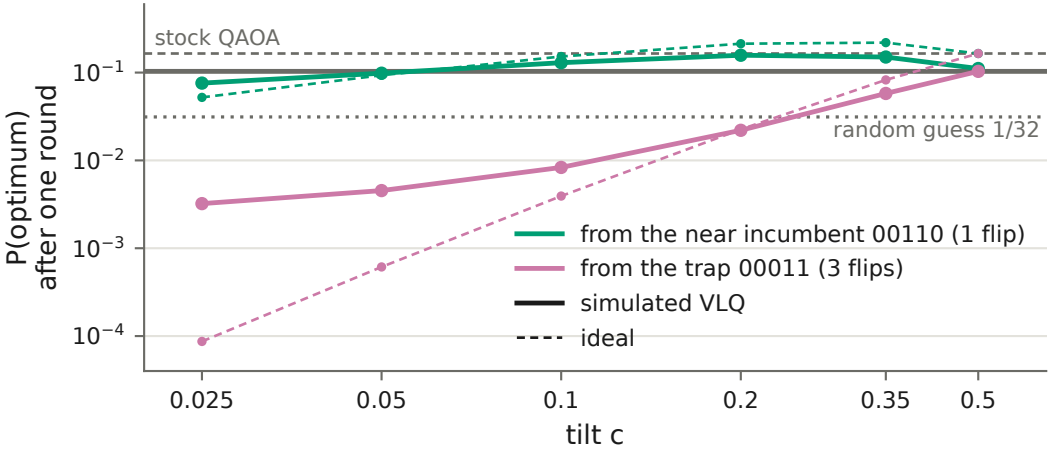
Focus or reach: the c a fixed schedule has to guess



Focus or reach: the c a fixed schedule has to guess



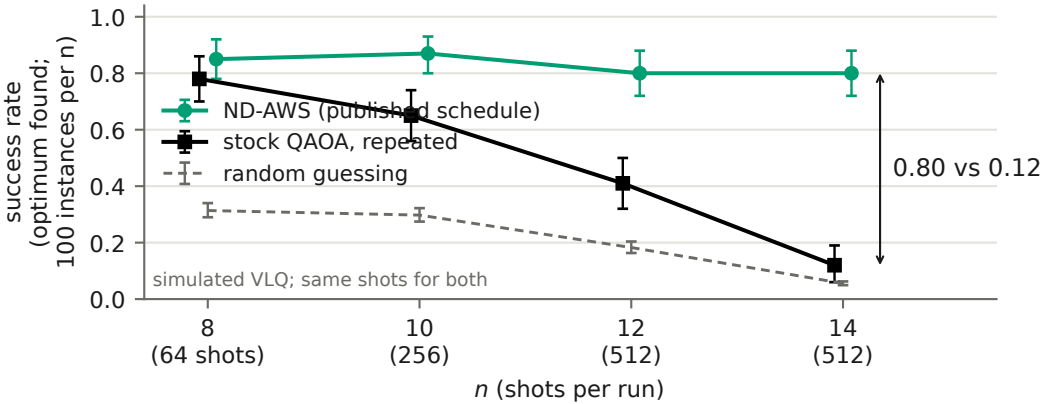
Focus or reach: the c a fixed schedule has to guess



Small c needs the optimum near (and even then $c \leq 0.05$ is below stock here); large c throws the focus away; the right c depends on where we are, which we don't know.

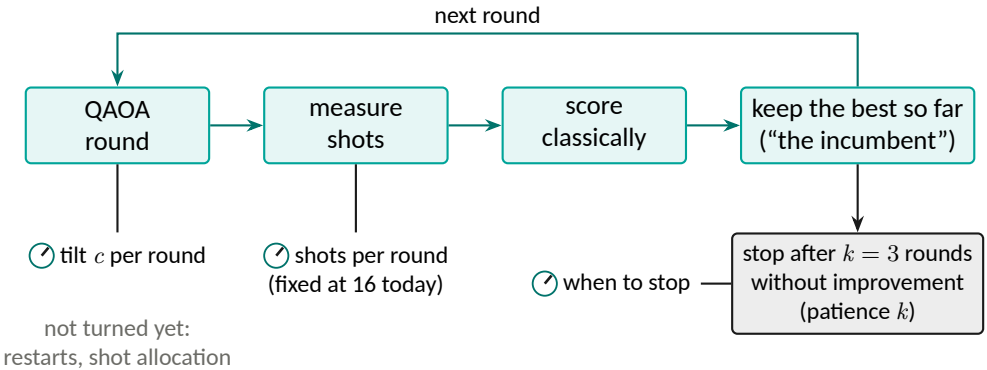
It holds up at scale

success rate = fraction of 100 instances where the optimum was found; both methods get the same number of shots



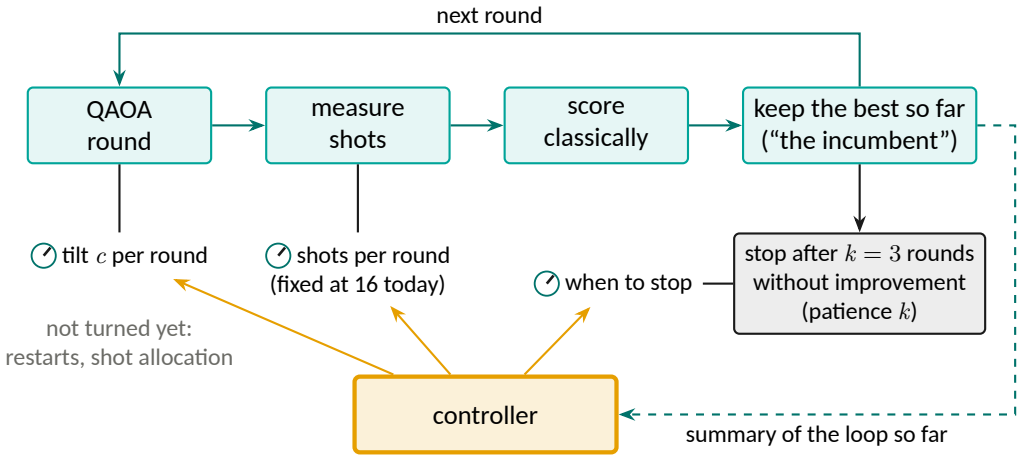
Same shots: ND-AWS finds the optimum more often from $n = 10$ on ($n = 14$: 80 % vs 12 %); at $n = 8$ the difference is within noise. Preliminary, simulated VLQ.

From schedule to controller



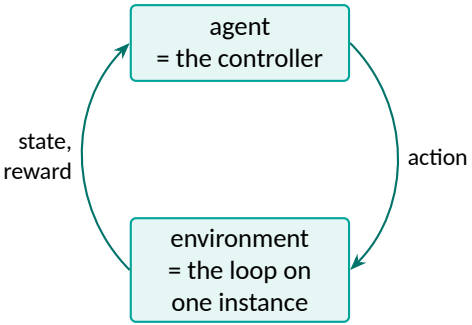
A fixed schedule turns these dials blindly.

From schedule to controller



A fixed schedule turns these dials blindly; a controller watches the loop and decides.

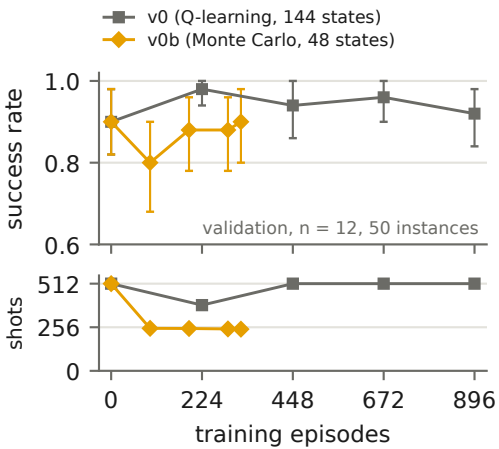
RL in one slide



- **state** = a summary of the loop so far
- **action** = next round's c, shots, or stop
- **reward** = improvement $-$ a small price per shot
- **episode** = one solve

Reinforcement learning = learning those decisions from many practice solves.

First attempt (v0): nothing to learn



	v0	**v0b**
practised on	$n = 10$, 12	$n = 12$, 14
table size	144 states	48 states
actions	13	6
scored by	round by round	whole solve
validation	0.92, all 512 shots	0.90, ≈245 shots
stops?	never	always

Why v0 failed: at $n = 10$–12 almost every rule already finds the optimum 9 times in 10, so there was little to learn.

Lesson: practise where the choices matter, with a table small enough to fill.

What v0b sees and does

What it sees (the state)

- rounds without improvement: 0, 1, 2, 3+
- budget used: in quarters
- last c: ≤ 0.1, 0.2, ≥ 0.35

What it does (the actions)

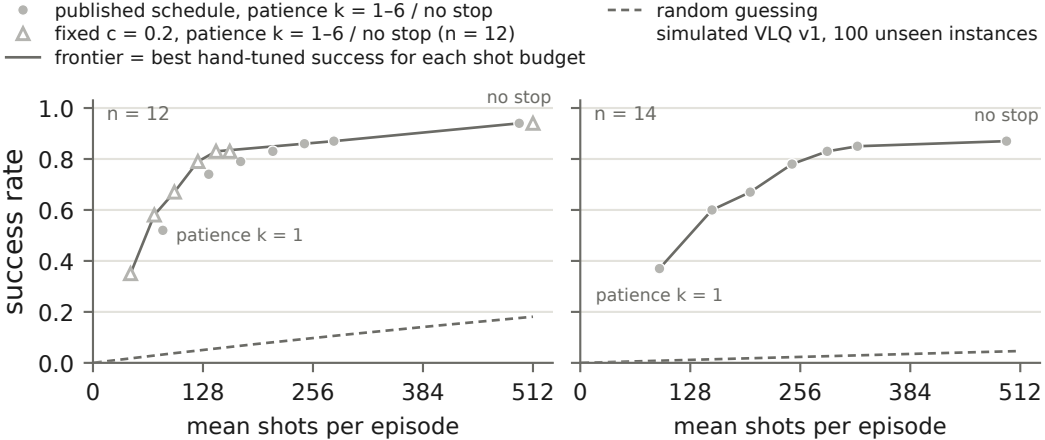
- $c \in \{0.05, 0.1, 0.2, 0.35, 0.5\}$ at 16 shots
- or stop

Budget = 512 shots per solve. It never sees the answer; the same for any n.

Small and readable on purpose: a first version we can check.

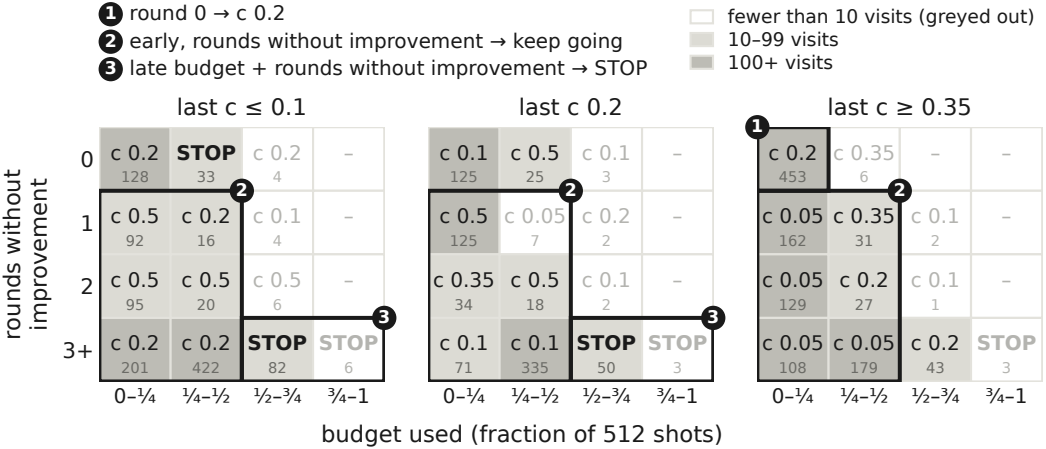
The fair baseline

each grey point = one hand-tuned rule: a schedule + when to stop



Beating one schedule is easy to fake. A fair controller must beat the frontier of the hand-tuned rules run on the test set.

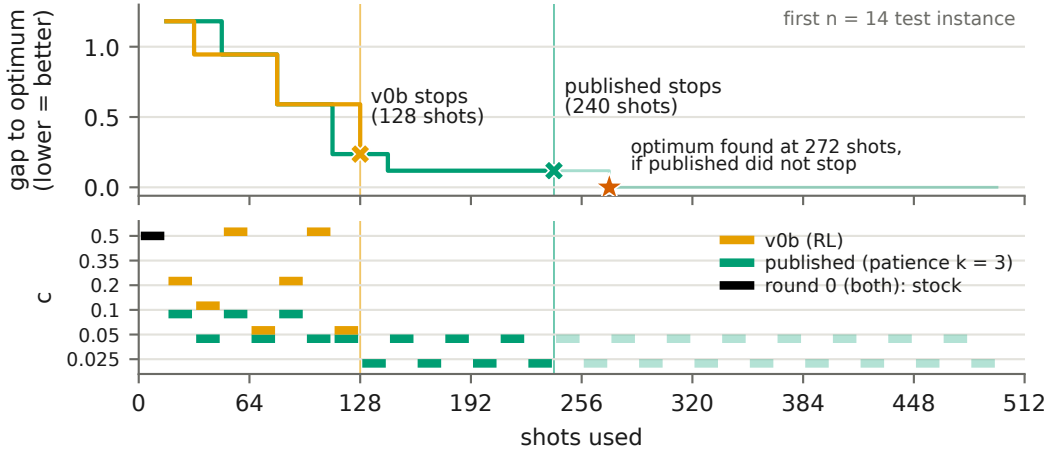
What v0b learned



$c = 0.2$ **early, keep going while less than half the budget is used, after that mostly stop at 3+ rounds without improvement.**

One episode

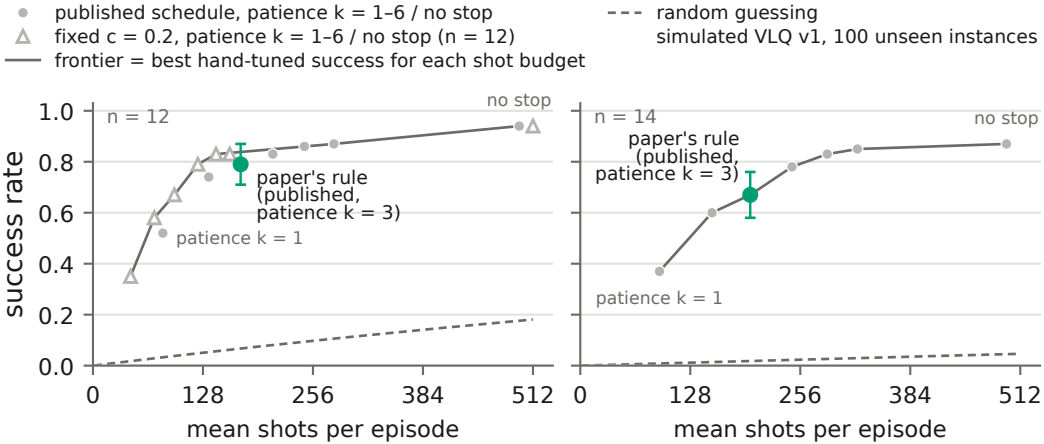
published rule = published schedule + stop after 3 rounds without improvement



Same instance: v0b stops at 128 shots, the published rule at 240; going on would have found the optimum at 272. Stopping early saves shots, and sometimes costs the answer.

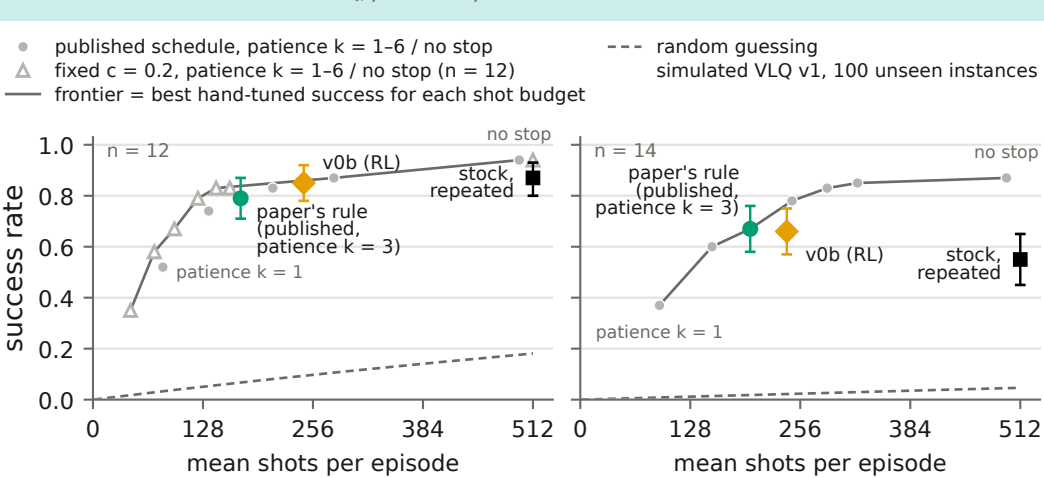
The honest test

simulated VLQ, preliminary



The honest test

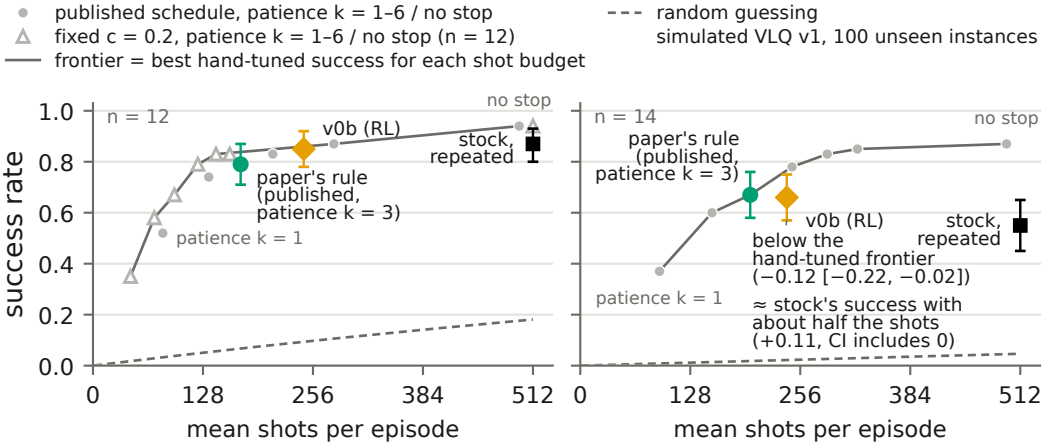
simulated VLQ, preliminary



On unseen instances v0b is on par with the published schedule in success, at somewhat more shots, and below the best hand-tuned rule — not better than hand-tuning yet.

The honest test

simulated VLQ, preliminary



On unseen instances v0b is on par with the published schedule in success, at somewhat more shots, and below the best hand-tuned rule — not better than hand-tuning yet.

What we learned, and where we go

- It learned to stop early (as good as the paper's stop rule, not better), not yet which c.
- We suspect a few summary numbers are not enough information.
- Next: a GNN (a graph neural network that reads the problem itself) choosing a c per qubit or the mixer; richer shot statistics; larger n; the real VLQ.
- Better-structured experiments and more powerful RL methods.

Simulated VLQ, $n = 12$–14; preliminary.

Not better than hand-tuning yet — next we test which dials and which information a controller needs.

Summary

You take home

1. Re-centre each round on the best answer: relabel it to 00000 and tilt the start (c).
2. With few shots a fixed c schedule can lock in when round 0 hands it the trap; at $n = 14$ it still beats repeated stock QAOA at the same shots (0.80 vs 0.12).
3. A first learned controller learned to stop early, not yet where to search.

Code and data: Adam Bílek

Acknowledgments

This work was supported by the project Q-Neko that has received funding from the European Union's Horizon Europe Framework Programme (HORIZON) under Grant Agreement no. 101241875. The project is supported by the EuroHPC Joint Undertaking and Japan's Strategic Innovation Promotion Program (SIP).
This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254)