清華大學 交叉信息研究院
Institute for Interdisciplinary Information Sciences, Tsinghua University

Quantum Error-Correcting Codes in QuantumClifford.jl

鄢语轩 @ IIIS, Tsinghua
GSoC mentor: Stefan Krastanov

Outline

- Stabilizers
- Quantum error correction
- My contribution: random circuit codes, lifted product codes

Stabilizers

Quantum states are generally not classical simulatable (unless BPP=BQP).

Yet, not all of them.

Stabilizer states: +1 or -1 eigenstates of some Pauli strings, called its stabilizer group.

The stabilizer group of a pure state has n generators.

Describe the state by those generators.

Generators (Pauli strings) are Bool vectors.

X -> 1|0, Y -> 1|1, Z -> 0|1

```
[4]: using QuantumClifford

[6]: st = ghz(3)

[6]: + XXX
     + ZZ_
     + _ZZ

[8]: stab_to_gf2(st)

[8]: 3×6 Matrix{Bool}:
      1  1  1  0  0  0
      0  0  0  1  1  0
      0  0  0  0  1  1
```

Clifford

Clifford operations transform a Pauli string always to another Pauli string.

-> Transform a stabilizer state to another stabilizer state.

By transforming each generator.

“Stabilizer states + Clifford” are classically simulatable.

-> QuantumClifford.jl

```
[7]: cl = random_clifford(3)
```

```
[7]: X₁ ⟼ - Y__
     X₂ ⟼ - YZ_
     X₃ ⟼ + __X
     Z₁ ⟼ - XX_
     Z₂ ⟼ + _X_
     Z₃ ⟼ + __Y
```

```
[8]: cl * st
```

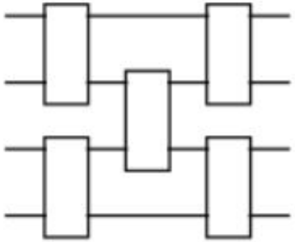
```
[8]: + _ZX
     - X__
     + _XY
```

Clifford circuit

Sequence of sparse Clifford operations (gates).

[17]:
```
circ = random_brickwork_clifford_circuit((4,),3)
```

[17]:



[24]:
```
circ[1].indices
```

[24]:
```
2-element Vector{Int64}:
 1
 2
```

[23]:
```
circ[1].cliff.tab
```

[23]:
```
+ YY
- XX
- _X
- ZX
```

[34]:
```
st = ghz(4)
```

[34]:
```
+ XXXX
+ ZZ__
+ _ZZ_
+ __ZZ
```

[35]:
```
for op in circ
    apply!(st, op)
end
```

[36]:
```
st
```

[36]:
```
- YY_X
- YY__
- XZ__
+ __XX
```

Quantum error correction

Errors happen and destroy information.

Solution: quantum error correction.

Idea: encode information into a subspace.

Encode: k logical qubits -> subspace of n qubits

Decode: -> k logical qubits

(Similar to classical error correction, where qubits -> bits, space are binary.)

Stabilizer codes

Stabilizer codes: the subspace is stabilized by (n-k) Pauli generators.

Syndrome, parity checks: measure some of stabilizer group elements.

The left k generators are taken as logical operators.

Parameters [[n,k,d]]:

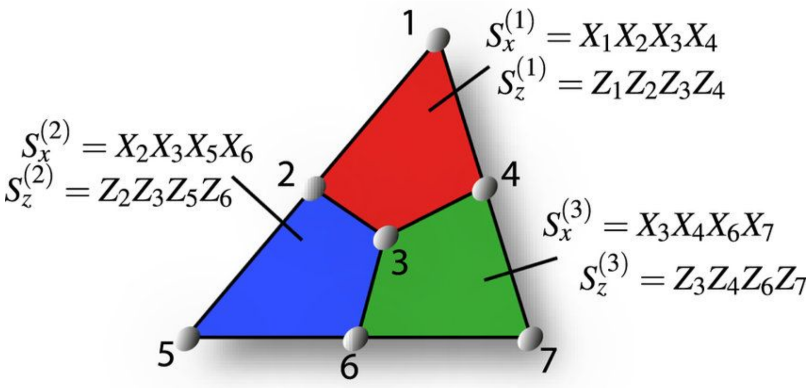
- n: size
- k: dimension
- d: distance (related to maximal correctable errors)

```
[38]: code = Steane7()

[38]: Steane7()

[39]: code_n(code)

[39]: 7

[40]: code_k(code)

[40]: 1
```

Parity checks

Choose (n-k) elements from stabilizer group as parity checks.

CSS code: if parity checks can be either “all X” or “all Z.”




```
[43]: parity_checks(code)

[43]: + ___XXXX
      + _XX__XX
      + X_X_X_X
      + ___ZZZZ
      + _ZZ__ZZ
      + Z_Z_Z_Z

[41]: parity_checks_x(code)

[41]: 3×7 Matrix{Bool}:
       0  0  0  1  1  1  1
       0  1  1  0  0  1  1
       1  0  1  0  1  0  1

[42]: parity_checks_z(code)

[42]: 3×7 Matrix{Bool}:
       0  0  0  1  1  1  1
       0  1  1  0  0  1  1
       1  0  1  0  1  0  1
```

Simulating codes

Pipeline: encoding

-> errors

-> syndrome (parity checks)

-> decoding

-> correction

Everything in Clifford and QuantumClifford.jl!

```
@testset "belief prop decoders, good for sparse codes" begin
    codes = vcat(LP04, LP118, test_gb_codes, other_lifted_product_codes)

    noise = 0.001

    setups = [
        CommutationCheckECCSetup(noise),
        NaiveSyndromeECCSetup(noise, 0),
        ShorSyndromeECCSetup(noise, 0),
    ]

    for c in codes
        for s in setups
            for d in [c -> PyBeliefPropOSDecoder(c, maxiter=2)]
                nsamples = 10000
                if true
                    @test_broken false # TODO these are too slow to test in CI
                    continue
                end
                e = evaluate_decoder(d(c), s, nsamples)
                # @show c
                # @show s
                # @show e
                @test max(e...) <= noise
            end
```

Better (and larger) codes?

New code construction (GSoC 2024)

Skip it because too textbook

Concatenated quantum code #289

Merged Krastanov merged 6 commits into `QuantumSavory:master` from `royess:concat` on Jun 14

add random Clifford circuit codes #298

Merged Krastanov merged 15 commits into `QuantumSavory:master` from `royess:randomcircuit` on Jul 5

Lifted and lifted product codes via Hecke's GroupAlgebra #356

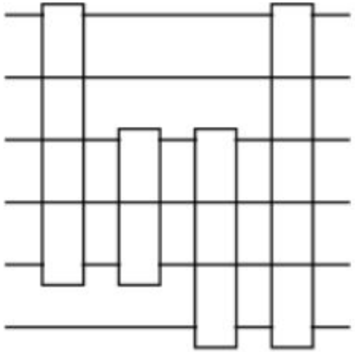
Merged Krastanov merged 70 commits into `QuantumSavory:master` from `royess:lift-dev-hecke` on Sep 27

Random Clifford circuit codes

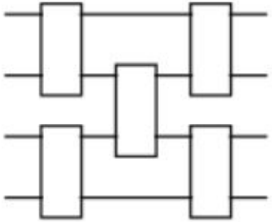
Idea: to encode a logical qubit, we need to spread it to the whole system.

Use random circuit as encoding circuit.

```
[59]: code = random_all_to_all_circuit_code(6, 4, [1]);

[60]: code.circ

[60]:
```



```
[49]: code = random_brickwork_circuit_code((4,), 3, [1]);

[47]: code.circ

[47]:
```



```
[48]: code.encode_qubits

[48]: 1-element Vector{Int64}:
 1
```

Lifted product codes

Quantum LDPC (QLDPC) code.

Idea: combine two classical codes into a quantum code.

Hypergraph product.

The hypergraph product yields two classical codes $C_{X,Z}$ with parity-check matrices

$$H_X = \begin{pmatrix} H_1 \otimes I_{n_2} & I_{r_1} \otimes H_2^T \end{pmatrix} \tag{1}$$

$$H_Z = \begin{pmatrix} I_{n_1} \otimes H_2 & H_1^T \otimes I_{r_2} \end{pmatrix} , \tag{2}$$

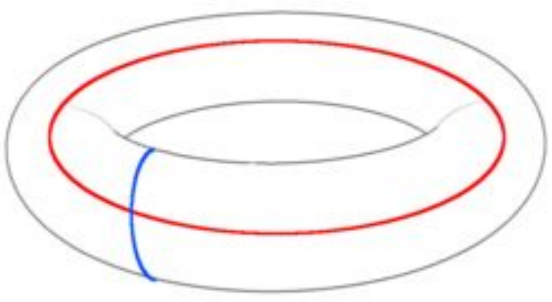
where I_m is the m-dimensional identity matrix. These two codes then yield a hypergraph product code via the CSS construction.

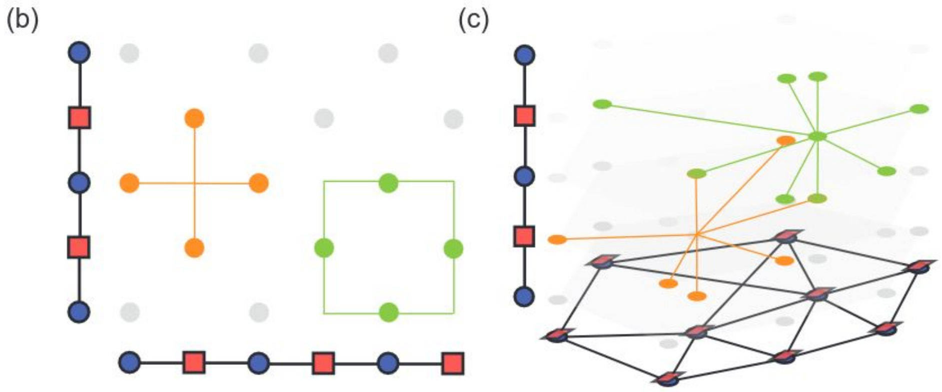
https://errorcorrectionzoo.org/c/hypergraph_product

Geometrical view of hypergraph product

Canonical example: repetition code

$\otimes_{HG}$ repetition code -> toric code





https://arxiv.org/abs/2312.08462

Lifted product

Motivation: save qubits to make better codes. (Asymptotically good QLDPC codes.)

1. Use matrices of group algebra elements in hypergraph product.

$$
\begin{aligned}
H_X &= \begin{pmatrix} H_1 \otimes I_{n_2} & I_{r_1} \otimes H_2^T \end{pmatrix} \\
H_Z &= \begin{pmatrix} I_{n_1} \otimes H_2 & H_1^T \otimes I_{r_2} \end{pmatrix} ,
\end{aligned}
$$

2. Then, lift each group algebra element to its binary matrix representation (e.g., permutation group representation).

Group algebra, group ring: group elements with coefficients, e.g., $g_1 + g_2$.

$(g_1 + g_2) * g_3 = g_1 * g_3 + g_2 * g_3$

Construction by group algebra (Hecke)

B1) $[[882, 24, d]]$ code, $18 \leqslant d \leqslant 24$. The matri
H_X and H_Z are (3,6)-regular ($\ell = 63$).

$$A = \begin{pmatrix} x^{27} & 0 & 0 & 0 & 0 & 1 & x^{54} \\ x^{54} & x^{27} & 0 & 0 & 0 & 0 & 1 \\ 1 & x^{54} & x^{27} & 0 & 0 & 0 & 0 \\ 0 & 1 & x^{54} & x^{27} & 0 & 0 & 0 \\ 0 & 0 & 1 & x^{54} & x^{27} & 0 & 0 \\ 0 & 0 & 0 & 1 & x^{54} & x^{27} & 0 \\ 0 & 0 & 0 & 0 & 1 & x^{54} & x^{27} \end{pmatrix},$$

$$B = (1 + x + x^6)I_7.$$

https://doi.org/10.22331/q-2021-11-22-585

Cyclic group of order l=63

```
[69]: import Hecke: group_algebra, GF, abelian_group, gens
      import LinearAlgebra: diagind

[70]: l = 63
      GA = group_algebra(GF(2), abelian_group(l))
      x = gens(GA)[]

      Element of Group algebra of group of order 63 over GF(2) with

[71]: A = zeros(GA, 7, 7);
      A[diagind(A)] .= x^27;
      A[diagind(A, -1)] .= x^54;
      A[diagind(A, 6)] .= x^54;
      A[diagind(A, -2)] .= GA(1);
      A[diagind(A, 5)] .= GA(1);
      B = reshape([1 + x + x^6], (1, 1));

[72]: c1 = LPCode(A, B);

[73]: code_n(c1), code_k(c1)

[73]: (882, 24)
```

Lesson: reinvent the wheel (group algebra)

Lifted and lifted product codes #312

Closed royess wants to merge 28 commits into `QuantumSavory:master` from `royess:lift`

QuantumClifford.jl / src / ecc / group_ring.jl

royess fix a type ambiguity

Code Blame 293 lines (218 loc) · 8.27 KB

```
import Base: *, +, -, ==, deepcopy_internal, show, isone, iszero, one, zero, adjoint

import Nemo: Ring, RingElem, RingElement, NCRing, NCRingElem, NCRingElement, Perm, CacheDictType,
    @attributes, base_ring, base_ring_type, elem_type, get_cached!,
    is_domain_type, is_exact_type, parent, parent_type


"""
Permutation [group ring](https://en.wikipedia.org/wiki/Group_ring) over a base ring, also known as group algebra.

```

https://github.com/royess/QuantumClifford.jl/blob/53322b587c3e4c94175b42c38578cbb2b429e697/src/ecc/group_ring.jl

Important but not covered in this talk

QuantumClifford.jl

- Decoders, e.g., BP-OSD.
- Entanglement.

- Logical gates.
- T gates and magic.
- Mixed states, destabilizers.

- More development experiences.