Equivalence Checking of Quantum Circuits

Christian Schilling
christianms@cs.aau.dk

April 8, 2025

DQC Scientific Quantum Conference

DEPARTMENT OF COMPUTER SCIENCE

This talk is based on joint work[1]

Christian Bøgh Larsen

Simon Brun Olsen

Kim Guldstrand Larsen

INDEPENDENT RESEARCH FUND DENMARK

VILLUM FONDEN

DIREC
Digital Research Centre Denmark

[1] C. B. Larsen, S. B. Olsen, K. G. Larsen, and C. Schilling. "Contraction heuristics for tensor decision diagrams". *Entropy* (2024).

Overview

Equivalence checking of quantum circuits

Tensor networks and tensor decision diagrams

Equivalence checking based on tensor decision diagrams

Empirical evaluation

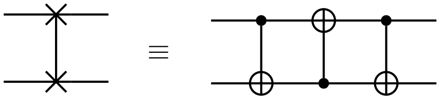
Conclusion and future work

Quantum circuit compilation

- When designing quantum algorithms, it is useful to have many types of operations available
- Real quantum computers only support a few types of operations
- A **compiler** translates a high-level circuit with many gate types to a low-level circuit with few gate types
- Important that the compiled circuits are **equivalent** (i.e., compute the same output for the same input)

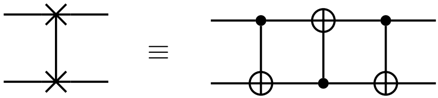
How to implement a SWAP gate?

- The SWAP gate is **equivalent** to three CNOT gates



How to implement a SWAP gate?

- The SWAP gate is **equivalent** to three CNOT gates



- We can easily prove this by comparing the matrices

$$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

Algorithm to check equivalence of quantum circuits

- Given: Two circuits C_1, C_2
- Question: Are the circuits equivalent ($C_1 \equiv C_2$)?

Algorithm to check equivalence of quantum circuits

- Given: Two circuits C_1, C_2
- Question: Are the circuits equivalent ($C_1 \equiv C_2$)?
- Simple algorithm
  1. Compute matrix representations U_1, U_2
  2. Check equality up to a factor (global phase $e^{i\theta}$)

$$U_1 \stackrel{?}{=} e^{i\theta} \cdot U_2$$

Algorithm to check equivalence of quantum circuits

- Given: Two circuits C_1, C_2
- Question: Are the circuits equivalent ($C_1 \equiv C_2$)?
- Simple algorithm
  1. Compute matrix representations U_1, U_2
  2. Check equality up to a factor (global phase $e^{i\theta}$)

$$U_1 \stackrel{?}{=} e^{i\theta} \cdot U_2$$

- Problem: Matrices are **exponentially large**

  For n qubits $\rightsquigarrow 2^n \times 2^n$

Equivalence checking is hard

- Checking exact equivalence is NQP-complete[1]
- Checking approximate equivalence is QMA-complete[2,3]
- Problems in these complexity classes are widely believed to require exponential computations in the worst case

[1] Y. Tanaka. "Exact non-identity check is NQP-complete". *Int. J. Quantum Inf.* (2010).
[2] D. Janzing, P. Wocjan, and T. Beth. ""Non-identity-check" is QMA-complete". *Int. J. Quantum Inf.* (2005).
[3] Z. Ji and X. Wu. *Non-identity check remains QMA-complete for short circuits*. 2009. arXiv: 0906.5416.

Approaches to equivalence checking

- ZX-calculus[1]
- Encoding with decision diagrams[2] ← relevant later
- Tensor network contraction[3] ← relevant later
- Simulation-based approach for the Clifford group[4]
- Weighted model counting[5]

[1] T. Peham, L. Burgholzer, and R. Wille. "Equivalence checking of quantum circuits with the ZX-calculus". *IEEE J. Emerg. Sel. Topics Circuits Syst.* (2022).

[2] L. Burgholzer and R. Wille. "Advanced equivalence checking for quantum circuits". *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* (2021).

[3] R. Orús. "Tensor networks for complex quantum systems". *Nature Reviews Physics* (2019).

[4] D. Thanos, T. Coopmans, and A. Laarman. "Fast equivalence checking of quantum circuits of Clifford gates". *ATVA*. 2023.

[5] J. Mei, T. Coopmans, M. M. Bonsangue, and A. Laarman. "Equivalence checking of quantum circuits by model counting". *IJCAR*. 2024.

Overview

Equivalence checking of quantum circuits

Tensor networks and tensor decision diagrams

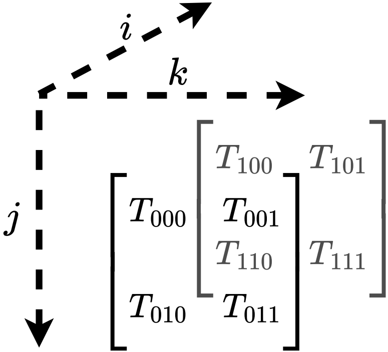
Equivalence checking based on tensor decision diagrams

Empirical evaluation

Conclusion and future work

Tensors

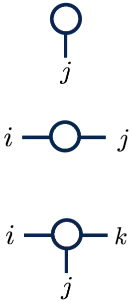
- Generalization of vectors / matrices to higher dimensions



Tensors

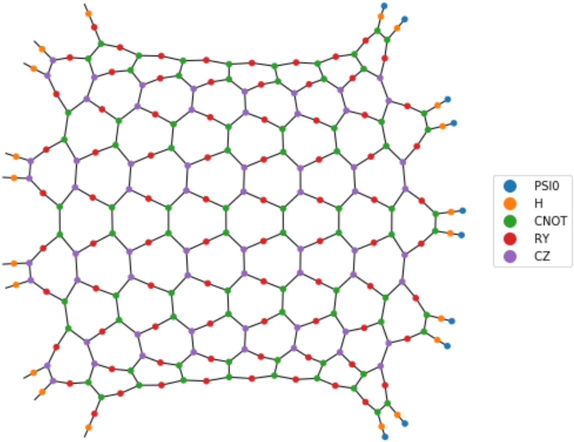
- Generalization of vectors / matrices to higher dimensions
- High-level graphical representation as a node with edges

vector	v_j
matrix	M_{ij}
3-index tensor	T_{ijk}



Tensor networks

- Tensors can be arranged in a graph

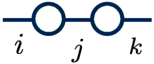


Tensor networks

- Tensors can be arranged in a graph
- Nodes with shared edges can be **contracted** (= merged)

  Corresponds to matrix-vector and matrix-matrix multiplication for special cases

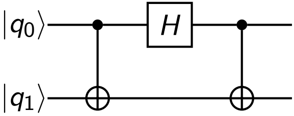
$$= \sum_j A_{ij} v_j$$

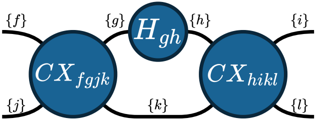


$$= \sum_j A_{ij} B_{jk} = AB$$

Example

1. Initial tensor network has one tensor for each gate
   Three choices for contraction $(\{g\}, \{h\}, \{k\})$

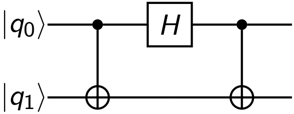


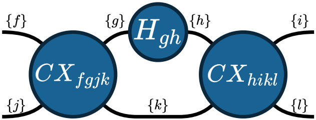


1.

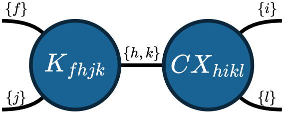
Example

1. Initial tensor network has one tensor for each gate
   Three choices for contraction $(\{g\}, \{h\}, \{k\})$
2. Contraction of CX_{fgjk} and H_{gh} via $\{g\}$





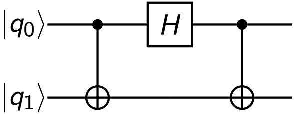
1.

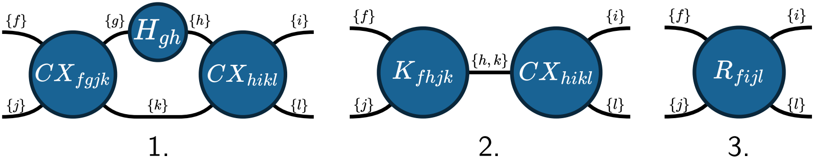


2.

Example

1. Initial tensor network has one tensor for each gate
   Three choices for contraction $(\{g\}, \{h\}, \{k\})$
2. Contraction of CX_{fgjk} and H_{gh} via $\{g\}$
3. Contraction of remaining two tensors





Application: Quantum simulation on classical computer[2]

- Contract tensors in smart orders
- Different contraction heuristics to minimize floating-point operations, size, etc.[1]

[1] J. Gray and S. Kourtis. "Hyper-optimized tensor network contraction". *Quantum* (2021).

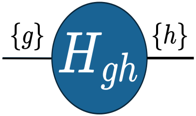
[2] I. L. Markov and Y. Shi. "Simulating quantum computation by contracting tensor networks". *SIAM J. Comput.* (2008).

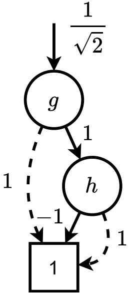
Tensor decision diagrams (TDDs)

- Alternative, unique representation of a tensor
- Informal introduction by example

Example: Hadamard gate with matrix, tensor, and TDD

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$





Overview

Equivalence checking of quantum circuits

Tensor networks and tensor decision diagrams

Equivalence checking based on tensor decision diagrams

Empirical evaluation

Conclusion and future work

Alternative "reverse scheme" for equivalence[1]

$$C_1 \equiv C_2 \overset{\text{def}}{\iff} \exists\theta\colon U_1 = e^{i\theta} \cdot U_2$$

$$\iff \exists\theta\colon U_1 \cdot U_2^{\dagger} = e^{i\theta} \cdot I \overset{\text{def}}{\iff} C_1 C_2^{-1} \equiv C_I$$

- C_2^{-1} is the inverted C_2 (reversed and each gate inverted)
- Allows to combine both circuits

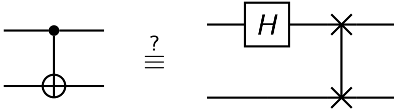
[1] G. F. Viamontes, I. L. Markov, and J. P. Hayes. "Checking equivalence of quantum circuits and states". *ICCAD*. 2007.

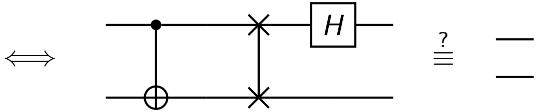
Alternative "reverse scheme" for equivalence[1]

$$C_1 \equiv C_2 \overset{\text{def}}{\iff} \exists\theta\colon U_1 = e^{i\theta} \cdot U_2$$

$$\iff \exists\theta\colon U_1 \cdot U_2^\dagger = e^{i\theta} \cdot I \overset{\text{def}}{\iff} C_1 C_2^{-1} \equiv C_I$$

- C_2^{-1} is the inverted C_2 (reversed and each gate inverted)
- Allows to combine both circuits





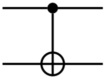
- (Coincidentally, the swap and Hadamard gates are self-inverse)

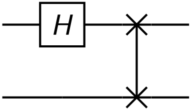
[1]G. F. Viamontes, I. L. Markov, and J. P. Hayes. "Checking equivalence of quantum circuits and states". *ICCAD*. 2007.

New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs

Given: Quantum circuits C_1, C_2



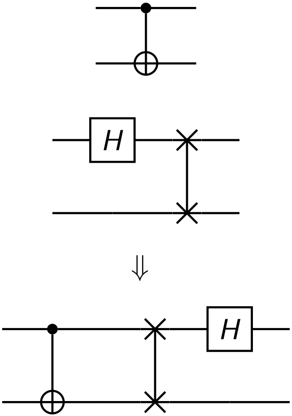


New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs

Given: Quantum circuits C_1, C_2

1. Construct circuit $C_1 C_2^{-1}$

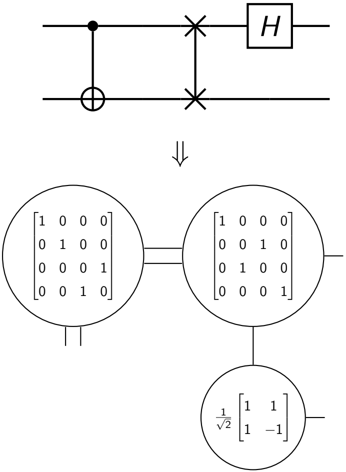


New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs

Given: Quantum circuits C_1, C_2

1. Construct circuit $C_1 C_2^{-1}$
2. Convert $C_1 C_2^{-1}$ to tensor network

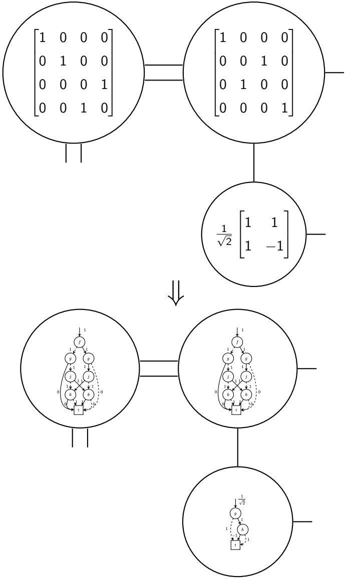


New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs

Given: Quantum circuits C_1, C_2

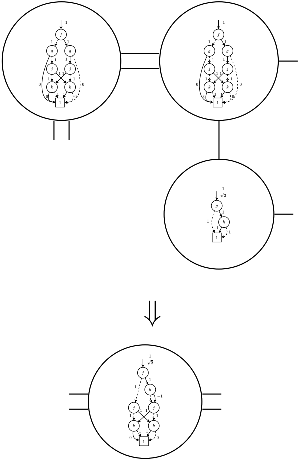
1. Construct circuit $C_1 C_2^{-1}$
2. Convert $C_1 C_2^{-1}$ to tensor network
3. Convert all tensors to TDDs



(TDDs on the right are only exemplary)

New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs



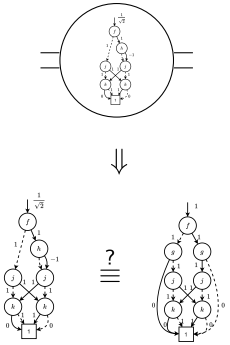
Given: Quantum circuits C_1, C_2

1. Construct circuit $C_1 C_2^{-1}$
2. Convert $C_1 C_2^{-1}$ to tensor network
3. Convert all tensors to TDDs
4. Contract TDD network

(TDDs on the right are only exemplary)

New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs



Given: Quantum circuits C_1, C_2

1. Construct circuit $C_1 C_2^{-1}$
2. Convert $C_1 C_2^{-1}$ to tensor network
3. Convert all tensors to TDDs
4. Contract TDD network
5. Compare TDD to identity TDD

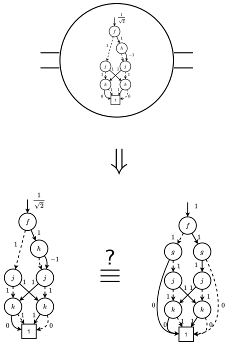
(TDDs on the right are only exemplary)

New algorithm for equivalence checking

Algorithm combines reverse scheme, tensor networks, and TDDs

Given: Quantum circuits C_1, C_2

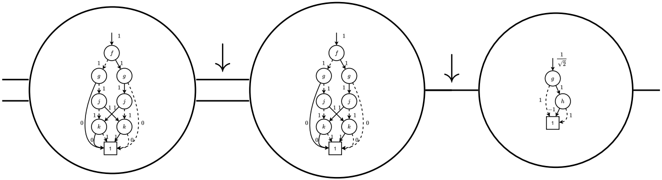
1. Construct circuit $C_1 C_2^{-1}$
2. Convert $C_1 C_2^{-1}$ to tensor network
3. Convert all tensors to TDDs
4. **Contract TDD network** $\leftarrow$ how?
5. Compare TDD to identity TDD



(TDDs on the right are only exemplary)

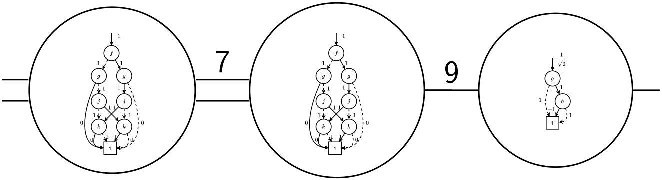
Lookahead heuristic for TDD contraction

- Greedy algorithm (finds a local optimum)
- In each contraction step:
  1. Evaluate all possible contractions



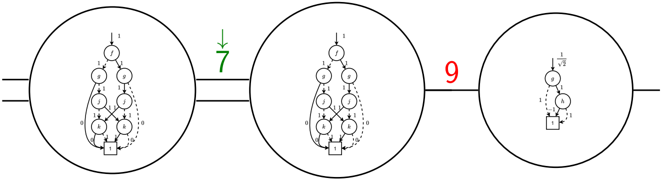
Lookahead heuristic for TDD contraction

- Greedy algorithm (finds a local optimum)
- In each contraction step:
  1. Evaluate all possible contractions
  2. Measure size of resulting TDDs



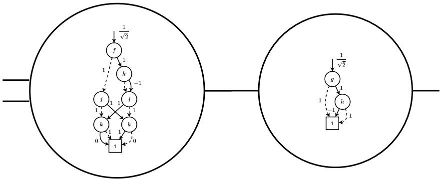
Lookahead heuristic for TDD contraction

- Greedy algorithm (finds a local optimum)
- In each contraction step:
  1. Evaluate all possible contractions
  2. Measure size of resulting TDDs
  3. Execute a contraction with smallest output



Lookahead heuristic for TDD contraction

- Greedy algorithm (finds a local optimum)
- In each contraction step:
  1. Evaluate all possible contractions
  2. Measure size of resulting TDDs
  3. Execute a contraction with smallest output



Lookahead heuristic for TDD contraction

- Greedy algorithm (finds a local optimum)
- In each contraction step:
  1. Evaluate all possible contractions
  2. Measure size of resulting TDDs
  3. Execute a contraction with smallest output
- Why should this scale?
  - Network is sparsely connected
  - Initial contractions (with many tensors) are cheap
  - Results can be stored for later iterations
- Still, step 1. is quite expensive

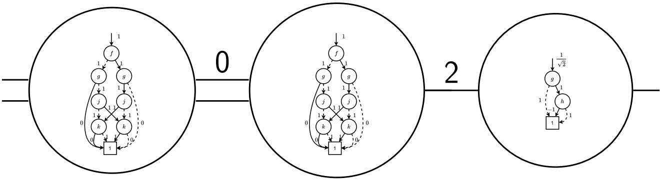
Counting heuristic for TDD contraction

- Goal: Imitate lookahead heuristic without expensive step 1
- Empirical observation: Lookahead heuristic prefers to distribute the contractions over the tensor network

[1] We set one edge to "2" to get an interesting example

Counting heuristic for TDD contraction

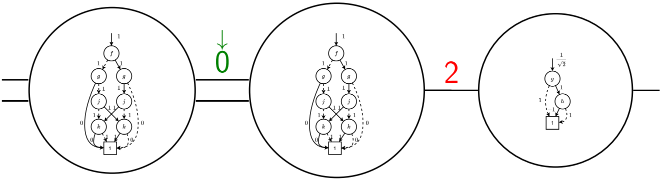
- Goal: Imitate lookahead heuristic without expensive step 1
- Empirical observation: Lookahead heuristic prefers to distribute the contractions over the tensor network
- In each contraction step:
    1. Select nodes with oldest participation in contraction[1]



[1] We set one edge to "2" to get an interesting example

Counting heuristic for TDD contraction

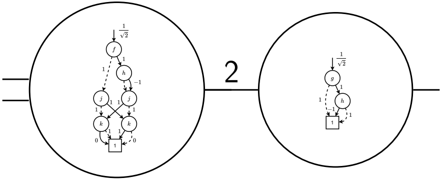
- Goal: Imitate lookahead heuristic without expensive step 1
- Empirical observation: Lookahead heuristic prefers to distribute the contractions over the tensor network
- In each contraction step:
  1. Select nodes with oldest participation in contraction[1]



[1] We set one edge to "2" to get an interesting example

Counting heuristic for TDD contraction

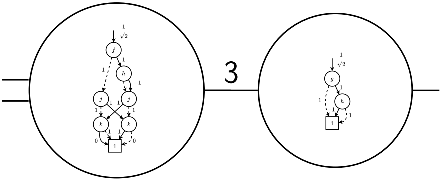
- Goal: Imitate lookahead heuristic without expensive step 1
- Empirical observation: Lookahead heuristic prefers to distribute the contractions over the tensor network
- In each contraction step:
  1. Select nodes with oldest participation in contraction[1]



[1] We set one edge to "2" to get an interesting example

Counting heuristic for TDD contraction

- Goal: Imitate lookahead heuristic without expensive step 1
- Empirical observation: Lookahead heuristic prefers to distribute the contractions over the tensor network
- In each contraction step:
  1. Select nodes with oldest participation in contraction[1]
  2. Update usage statistics of neighboring edges



[1] We set one edge to "2" to get an interesting example

Overview

Equivalence checking of quantum circuits

Tensor networks and tensor decision diagrams

Equivalence checking based on tensor decision diagrams

Empirical evaluation

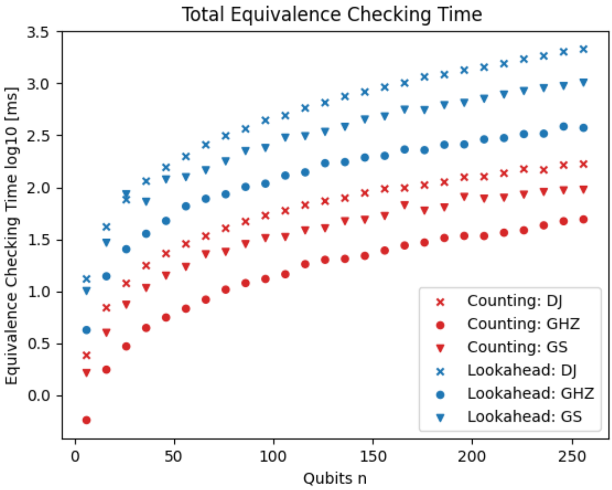
Conclusion and future work

Quantum circuits in evaluation

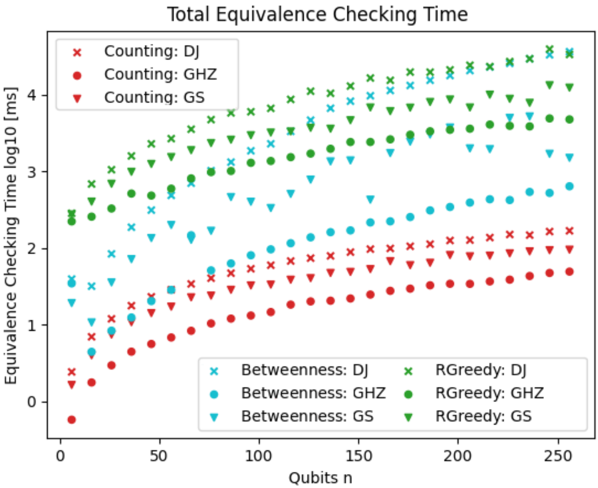
- Circuits from MQT Bench[1] with varying number of qubits at two compilation levels (level 1 and 3 (out of 4)) with significantly different gate sets and layouts
  - Deutsch-Jozsa algorithm (DJ)
  - Greenberger-Horne-Zeilinger state preparation (GHZ)
  - Graph state preparation (GS)

[1] N. Quetschlich, L. Burgholzer, and R. Wille. "MQT Bench: Benchmarking software and design automation tools for quantum computing". *Quantum* (2023).

Comparison of own heuristics

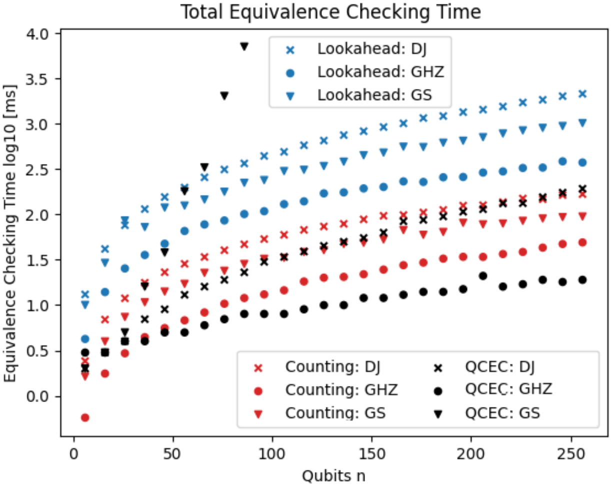


Comparison to cotengra[1]



[1] J. Gray and S. Kourtis. "Hyper-optimized tensor network contraction". *Quantum* (2021).

Comparison to QCEC[1]



[1] L. Burgholzer and R. Wille. "QCEC: A JKQ tool for quantum circuit equivalence checking". *Softw. Impacts* (2021).

Overview

Equivalence checking of quantum circuits

Tensor networks and tensor decision diagrams

Equivalence checking based on tensor decision diagrams

Empirical evaluation

Conclusion and future work

Conclusion

- Integration of "reverse scheme" and TDD networks
- Lookahead heuristic (greedy)
- Counting heuristic (cheap approximation)
- Evaluation:
  - Outperforms cotengra's heuristics
  - Often keeps up with QCEC

Future work & FMQC 2025

- Exploit parallelization (CPU, GPU)
- Find other heuristics
  - Generalize tensor network heuristics to TDD networks
  - Identify equivalent subcomponents (modularity)
  - Employ machine learning
- New PhD project since December 2024

DeiC

Workshop on Formal Methods in Quantum Computing

co-located with CONFEST 2025 in Aarhus, August 25

https://fmqc-workshop.github.io/2025/