Effective Partitioning for Distributed Measurement-Based Quantum Computing

Raoul Heese[1], Matthias Heller[2,3], Kjell Fredrik Pettersen[4], and Giorgio Sartor*[4]

[1]NTT DATA, Munich, Germany
[2]Fraunhofer Institute for Computer Graphics Research IGD, Darmstadt, Germany
[3]Technical University of Darmstadt, Interactive Graphics Systems Group, Darmstadt, Germany
[4]SINTEF Digital, Oslo, Norway

December 5, 2025

Abstract

Measurement-Based Quantum Computing (MBQC) is naturally suited to distributed quantum architectures: once a resource state is prepared and distributed across the network, computation proceeds via local measurements coordinated by classical communication. However, since non-local gates acting on different Quantum Processing Units (QPUs) are a bottleneck, it is crucial to optimize the qubit assignment to minimize inter-node entanglement of the shared resource. For graph-state resources, this objective reduces to finding bi-partitions with minimal cut rank. We present an efficient incremental algorithm that updates the cut rank when two vertices swap sides across a bi-partition. Embedded in a local search framework, this update rule enables rapid exploration of the partition space and effectively identifies partitions with low cut rank. We show that the approach is highly effective for determining qubit assignments in distributed MBQC.

1 Introduction

Current quantum computing hardware platforms face significant challenges in scaling up the number of qubits on a single Quantum Processing Unit (QPU). As the qubit count increases, issues such as decoherence, gate fidelity, control complexity, and physical infrastructure limitations make it increasingly difficult to maintain performance and reliability. Unfortunately, these constraints limit the practical use of quantum computing, particularly when fault tolerance is required. Distributed Quantum Computing (DQC) has been recently presented as a viable option to scale up the number of available qubits in the future, for recent reviews see, e.g., [1, 2]. The basic idea of Distributed Quantum Computing (DQC) is to link together several QPUs giving rise to a quantum network. However, non-local gates implemented through such quantum links typically have significantly lower fidelity as well as longer execution time compared to the on-chip operations [3, 4]. Therefore, a new challenge arises in the setting of Distributed Quantum Computing (DQC): how to compile a quantum circuit such that the quantum communication between the nodes of the network is minimized?

Measurement-Based Quantum Computing (MBQC) is a an alternative model of quantum computing that consists of preparing an initial, highly-entangled quantum state (called resource state) and then drive its evolution through consecutive, adaptive single-qubit measurements [5]. What

*Corresponding author: giorgio.sartor@sintef.no makes MBQC particularly interesting in the context of DQC is that the initial resource state can be distributed in the beginning of the computation, such that the expensive non-local gates can be performed right in the beginning. After that, the protocol only uses classical communication to report the outcome of local measurements on the individual QPUs to implement the round of adaptive measurements. Distributed MBQC has been discussed and formalized in the literature [6].

Typically, the resource state chosen to perform MBQC is a graph state, which—as its name suggests—can be associated with an undirected graph in which vertices represent qubits and edges represent entanglement between pairs of qubits. Specifically, the graph state associated with an undirected graph $G = (V, E)$ with vertices V and edges E reads

$$|G\rangle := \prod_{e \in E} CZ_e \, |+\rangle^{\otimes |V|} , \tag{1}$$

where CZ_e represents a controlled-Z gate on two qubits connected by edge e, and $|+\rangle$ is the eigenstate of the Pauli-X operator with eigenvalue $+1$. Since controlled-Z gates are symmetric and commute, neither the qubit order in e nor the product order in Eq. (1) matter. The application of local Clifford gates to $|G\rangle$ can be understood as a transformation of the associated graph, where the changes to the graph can be formalized as a sequence of local complementations (and vertex relabelings) [7]. Since local gates do not change the bipartite entanglement, the associated graph states before and after the transformation are locally equivalent. Consequently, the corresponding graphs can also be considered locally equivalent, meaning they can be reconfigured via local complementations while preserving computational equivalence of the MBQC procedure on the respective graph states.

From the MBQC viewpoint, distributing a computation across multiple QPUs reduces to a graph partitioning problem: choose a partition of V such that each part fits on a QPU and the *cut rank*—which equals the Schmidt rank across the partition for graph states [8, 9]—is minimized. Here and in the following, we limit ourselves to a DQC scenario with exactly two QPUs. The cut rank of a graph $G = (V, E)$, given the partition into $X \subseteq V$ and $Y = V \setminus X$, is defined as [10]

$$\rho_{X,Y}(G) := \operatorname{rank}\bigl(A[X, Y]\bigr), \tag{2}$$

where $A[X, Y]$ denotes the adjacency matrix of G with only the rows X and columns Y defined over the binary field $GF(2)$. In short, finding a partition (X, Y) of minimal cut rank corresponds to an optimal partition of the qubits in the resource state.

The cut rank of a bi-partition measures how many rows (or columns) are linearly independent, so it captures the diversity of connections, not just the quantity. Indeed, in the case when the amount of edges connecting two partitions is larger than the cut rank, one can reduce the number by embedding the graph into a slightly larger one, which is equivalent up to local complementations and vertex deletions. In the MBQC picture, this corresponds to local measurements and local Clifford gates. Since bipartite Schmidt rank is invariant under local unitaries (in particular local Cliffords), the cut rank is invariant under local complementations [11]. An example is sketched in Fig. 1.

In this paper, we develop a heuristic algorithm to find the bi-partition of a graph with the smallest cut rank using a simualted annealing approach. Given a graph $G = (V, E)$ and a partition $X \subseteq V$ and $V \setminus X$, we develop an algorithm that computes the change in cut rank for a fixed vertex for every possible swap with vertices from the other set in $O(n^2)$ and then performs the swaps that reduced the cut rank, if any. We use this algorithm to accelerate simulated annealing-based optimization, for which computing the change in cut rank would be the computational bottleneck. Here, we propose a simple analytical and efficient method that cuts down the computation time by up to two-orders of magnitude for graphs with up to 400 nodes.

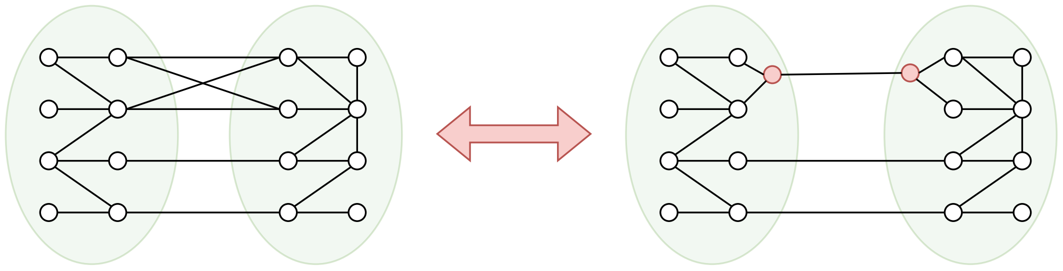

Figure 1: Reducing the number of edges between a bi-partition of a graph. While the graph on the left-hand-side has 6 edges connecting both partitions, we can introduce ancillary nodes (qubits, marked in red) to reduce the number of edges (CZ gates) to 3, which is equivalent to the cut rank.

2 An incremental algorithm for cut rank

The aim of this section is to develop an algorithm that takes a simple graph $G = (E, V)$ and partition size n as input and finds a partition $X \subseteq V$ and $Y = V \setminus X$ with minimal cut rank and such that $|X| = n$. The basic idea is to use simulated annealing [12] to solve this problem as defined in Algorithm 1.

Algorithm 1 Simulated annealing for fixed-size partitioning

```
Require: Graph $G = (V, E)$, partition size $n$, temperature schedule $\mathcal{T} = T_1, T_2, \dots T_N$
 1: Initialize: Select a random initial partition $X$ with $|X| = n$.
 2: $c = \text{rank}(A[X, V \setminus X])$
 3: for $T_i \in \mathcal{T}$ do
 4:     for $i \in X$ do
 5:         for $j \in V \setminus X$ do
 6:             $\tilde{X} = \big(X \setminus \{i\}\big) \cup \{j\}$
 7:             $\Delta c = \text{rank}(A[\tilde{X}, V \setminus \tilde{X}]) - c$
 8:             if $\exp(-\Delta c / T_i) > \text{rand}(0, 1)$ then
 9:                 $X \leftarrow \tilde{X}$
10:                 $c \leftarrow c + \Delta c$
11:             end if
12:         end for
13:     end for
14: end for
```

The most computationally demanding step in this algorithm is in line 7, where we have to calculate the change of cut rank after performing one swap of vertices in the two partitions. Naively, one would first calculate the new cut rank corresponding to $\tilde{X}$ and then calculate Δc, which would require complexity $O(N^3)$ due to the rank calculation. However, Δc can be calculated more efficiently by using information from the previous calculations. This can be done by defining some key matrices defined from the current partition from which one can calculate the cut rank changes for a fixed i simultaneously for all j with time complexity $O(N^2)$. The complexity of maintaining the necessary key matrices after applying a swap also has complexity $O(N^2)$.

2.1 Key matrices in the cut rank calculations

The cut rank for the current partition is given as $\rho_{X,Y}(G) = \text{rank}(A[X, Y])$. This implies there are subsets $X^B \subseteq X$ and $Y^B \subseteq Y$ where $|X^B| = |Y^B| = \rho_{X,Y}(G)$ such that the submatrix

$$C := A[X^B, Y^B] \tag{3}$$

of $A[X,Y]$ is invertible. The sets X^B and Y^B are assumed to be known, as well as the inverse C^{-1} which is naturally indexed with Y^B as rows and X^B as columns. The rows (resp. columns) indexed by X^B (Y^B) define a basis for the row (column) vectors of $A[X,Y]$, therefore

$$A[X^F, Y^B] \cdot C^{-1} \cdot A[X^B, Y^F] = A[X^F, Y^F], \tag{4}$$

where $X^F = X \setminus X^B$ and $Y^F = Y \setminus Y^B$ are the remaining vertices in the partition sets.

To perform the cut rank calculation we also need the matrices

$$D_X := A[V(G), Y^B] \cdot C^{-1}, \tag{5}$$

$$D_Y := C^{-1} \cdot A[X^B, V(G)], \tag{6}$$

and

$$F := A[V(G), Y^B] \cdot C^{-1} \cdot A[X^B, V(G)] + A[V(G), V(G)], \tag{7}$$

where the columns of D_X are indexed by X^B and the rows of D_Y are indexed by Y^B.

2.2 The cut rank calculation process

A swap that updates the partition sets X and Y by swapping the vertices $i \in X$ and $j \in Y$ may have an effect on the cut rank and basis sets X^B and Y^B. The process for finding these updates has two steps, reduction and extension.

- **Reduction:** The basis sets X^B and Y^B are reduced by equally many vertices so i and j are no longer in the basis sets, and the reduced $A[X^B, Y^B]$ matrix is still invertible. This is only necessary if $i \in X^B$ or $j \in Y^B$. Maximum 2 vertices will be removed from each set.
- **Extension:** The vertices are swapped, and the new X^B and Y^B sets are extended if necessary with equally many vertices until (4) is satisfied. The sizes of X^B and Y^B give the new cut rank after the swap is applied. Maximum 4 vertices will be added to each basis set, and the net change of the cut rank from the reduction and extension combined will be limited by ± 2.

The vertices to be removed and added in the reduction and extension steps are found by matrix investigations and attempts to clean the matrix $A[X,Y]$ with the basis rows and columns. The process will depend on wether certain properties are satisfied. Some of them only depend on one of the swapped vertices, and may be settled for all i and j in a preprocessing phase. Two of these properties are

$$\begin{aligned} P_1^X(i) &:= (i \in X^B) \wedge (\exists k_1 \in X^F : D_X[k_1, i] = 1) \\ P_1^Y(j) &:= (j \in X^B) \wedge (\exists l_1 \in Y^F : D_Y[l_1, j] = 1) \end{aligned} \tag{8}$$

If $P_1^X(i)$ (resp. $P_1^Y(j)$) is true we assume k_1 (l_1) is implicitly given.

The formulation of the next properties depend on the truth values of the first ones:

$$P_2^X(i) := \exists k_2 \in X^F : \begin{cases} (k_2 \neq k_1) \wedge (F[k_2, i] \\ \qquad\quad + F[k_1, i] D_X[k_2, i] = 1) & \text{if } P_1^X(i) \\ (k_2 \neq i) \wedge (F[k_2, i] = 1) & \text{if not } P_1^X(i) \end{cases} \tag{9}$$

$$P_2^Y(j) := \exists l_2 \in Y^F : \begin{cases} (l_2 \neq l_1) \wedge (F[j, l_2] \\ \qquad\quad + F[j, l_1] D_Y[j, l_2] = 1) & \text{if } P_1^Y(j) \\ (l_2 \neq j) \wedge (F[j, l_2] = 1) & \text{if not } P_1^Y(j) \end{cases} \tag{10}$$

Again, if $P_2^X(i)$ (resp. $P_2^Y(j)$) is true we take k_2 (l_2) implicitly as given.

The matrix C^{-1} is invertible. It has therefore no row or column with 0 only. This implies

$$i \in X^B \Rightarrow \exists \alpha \in Y^B : C^{-1}[\alpha, i] = 1 \tag{11}$$

$$j \in Y^B \Rightarrow \exists \beta \in X^B : C^{-1}[j, \beta] = 1 \tag{12}$$

If $i \in X^B$ (resp. $j \in Y^B$), we take α (β) as given. None of the vertices k_2, l_2, α and β are needed to only determine the value of the cut rank, but may occur among the removed or added vertices.

The process is split into five different cases, depending on whether i and j are in the basis sets, and for the case when both are, the value of $C^{-1}[j, i]$. The vertices to remove in the reduction step are the same within each of these cases, while there will be several sub-cases for the extension step.

First case: $i \in X^F$ and $j \in Y^F$ No reduction is needed since the C matrix does not contain row i or column j. The different cases for the extension step depend on the properties $P_2^X(i)$ and $P_2^Y(j)$, and the value of $F[j, i]$, this is summarized by

$P_2^X(i)$	$P_2^Y(j)$	$F[j,i]$	Δc	$+X^B$	$+Y^B$
T	T		+2	j, k_2	i, l_2
T	F		+1	k_2	i
F	T		+1	j	l_2
F	F	1	+1	j	i
F	F	0	0		

where T means true, F means false, Δc is the combined cut rank change from the reduction and extension, and $+X^B$ (resp. $+Y^B$) are the vertices added to X^B (Y^B).

Second case: $i \in X^B$ and $j \in Y^F$ The reduction step will need to remove i from X^B and some column from Y^B. If C' is the submatrix of C with row i and column α removed, then

$$\det(C') = \text{Adj}(C)[\alpha, i] = C^{-1}[\alpha, i] = 1 \tag{13}$$

so C' is invertible. The reduction step can therefore be set to removes i from X^B and α from Y^B. The extension step can be split into two cases depending on the truth value of $P_1^X(i)$:

$P_1^X(i)$ is true

$P_2^X(i)$	$P_2^Y(j)$	Q	Δc	$+X^B$	$+Y^B$
T	T		+2	j, k_1, k_2	i, α, l_2
T	F		+1	k_1, k_2	i, α
F	T		+1	j, k_1	α, l_2
F	F	1	+1	j, k_1	i, α
F	F	0	0	k_1	α

where $Q := F[j, i] + D_X[j, i]F[k_1, i]$.

$P_1^X(i)$ is false

$D_X[j,i]$	$P_2^X(i)$	$P_2^Y(j)$	$F[j,i]$	Δc	$+X^B$	$+Y^B$
1	T			+1	j, k_2	i, α
1	F			0	j	α
0	T	T		+1	j, k_2	i, l_2
0	T	F		0	k_2	i
0	F	T		0	j	l_2
0	F	F	1	0	j	i
0	F	F	0	-1		

Third case: $i \in X^F$ and $j \in Y^B$ This is symmetric to the previous case. The reduction step will remove β from X^B and j from Y^B. The extension step depends on $P_1^Y(j)$:

$P_1^Y(j)$ is true

$P_2^Y(j)$	$P_2^X(i)$	Q	Δc	$+X^B$	$+Y^B$
T	T		$+2$	j, β, k_2	i, l_1, l_2
T	F		$+1$	j, β	l_1, l_2
F	T		$+1$	β, k_2	i, l_1
F	F	1	$+1$	j, β	i, l_1
F	F	0	0	β	l_1

where $Q := F[j,i] + D_Y[j,i]F[j,l_1]$.

$P_1^Y(j)$ is false

$D_Y[j,i]$	$P_2^Y(j)$	$P_2^X(i)$	$F[j,i]$	Δc	$+X^B$	$+Y^B$
1	T			$+1$	j, β	i, l_2
1	F			0	β	i
0	T	T		$+1$	j, k_2	i, l_2
0	T	F		0	j	l_2
0	F	T		0	k_2	i
0	F	F	1	0	j	i
0	F	F	0	-1		

Fourth case: $i \in X^B$, $j \in Y^B$ and $C^{-1}[j,i] = 1$ As before, C with row i and column j removed is invertible since $C^{-1}[j,i] = 1$. Therefore, the reduction step will remove i from X^B and j from Y^B. The extension step splits into two cases:

$P_1^X(i)$ and $P_1^Y(j)$ are true

$P_2^X(i)$	$P_2^Y(j)$	Q	Δc	$+X^B$	$+Y^B$
T	T		$+2$	j, k_1, k_2	i, l_1, l_2
T	F		$+1$	k_1, k_2	i, l_1
F	T		$+1$	j, k_1	l_1, l_2
F	F	1	$+1$	j, k_1	i, l_1
F	F	0	0	k_1	l_1

where $Q := F[j,i] + D_X[j,i]F[k_1,i] + D_Y[j,i]F[j,l_1] + F[k_1,i]F[j,l_1]$.

$P_1^X(i)$ or $P_1^Y(j)$ is false Two more properties are needed

$$\begin{aligned} P_3^X(i,j) &= \exists k_3 \in X^F : F[k_3,i] + D_X[k_3,i]D_Y[j,i] = 1 \\ P_3^Y(i,j) &= \exists l_3 \in Y^F : F[j,l_3] + D_X[j,i]D_Y[j,l_3] = 1 \end{aligned} \tag{14}$$

where we take k_3 (resp. l_3) implicitly for given if $P_3^X(i,j)$ ($P_3^Y(i,j)$) is true. The extension step splits into the following combinations:

$P_3^X(i,j)$	$P_3^Y(i,j)$	Q	Δc	$+X^B$	$+Y^B$
T	T		$+1$	j, k_3	i, l_3
T	F		0	k_3	i
F	T		0	j	l_3
F	F	1	0	j	i
F	F	0	-1		

where $Q := F[j,i] + D_X[j,i]D_Y[j,i]$.

Fifth case: $i \in X^B$, $j \in Y^B$ and $C^{-1}[j,i] = 0$ Removing i and j from the basis will give a singular submatrix of C since $C^{-1}[j,i] = 0$, so this is not enough for the reduction step. However the submatrix

$$C^{-1}[\{j,\alpha\},\{i,\beta\}] := \begin{bmatrix} 0 & 1 \\ 1 & C^{-1}[\alpha,\beta] \end{bmatrix} \tag{15}$$

is invertible, then so is the submatrix of C with the same vertices removed from the basis. Therefore the reduction step will remove i, β from X^B and j, α from Y^B. The extension step then depends on the truth values of both $P_1^X(i)$ and $P_1^Y(j)$.

$P_1^X(i)$ and $P_1^Y(j)$ are both true

$P_2^X(i)$	$P_2^Y(j)$	Q	Δc	$+X^B$	$+Y^B$
T	T		$+2$	j,β,k_1,k_2	i,α,l_1,l_2
T	F		$+1$	β,k_1,k_2	i,α,l_1
F	T		$+1$	j,β,k_1	α,l_1,l_2
F	F	1	$+1$	j,β,k_1	i,α,l_1
F	F	0	0	β,k_1	α,l_1

where $Q := F[j,i] + D_X[j,i]F[k_1,i] + D_Y[j,i]F[j,l_1]$.

$P_1^X(i)$ is true, $P_1^Y(j)$ is false

$P_2^Y(j)$	$D_Y[j,i]$	$P_2^X(i)$	Q	Δc	$+X^B$	$+Y^B$
T	1			$+1$	j,β,k_1	i,α,l_2
T	0	T		$+1$	j,k_1,k_2	i,α,l_2
T	0	F		0	j,k_1	α,l_2
F	1			0	β,k_1	i,α
F	0	T		0	k_1,k_2	i,α
F	0	F	1	0	j,k_1	i,α
F	0	F	0	-1	k_1	α

where $Q := F[j,i] + D_X[j,i]F[k_1,i]$.

$P_1^X(i)$ is false, $P_1^Y(j)$ is true

$P_2^X(i)$	$D_X[j,i]$	$P_2^Y(j)$	Q	Δc	$+X^B$	$+Y^B$
T	1			$+1$	j,β,k_2	i,α,l_1
T	0	T		$+1$	j,β,k_2	i,l_1,l_2
T	0	F		0	β,k_2	i,l_1
F	1			0	j,β	α,l_1
F	0	T		0	j,β	l_1,l_2
F	0	F	1	0	j,β	i,l_1
F	0	F	0	-1	β	l_1

where $Q := F[j,i] + D_Y[j,i]F[j,l_1]$.

$P_1^X(i)$ and $P_1^Y(j)$ are both false

$D_X[j,i]$	$D_Y[j,i]$	$P_2^X(i)$	$P_2^Y(j)$	$F[j,i]$	Δc	$+X^B$	$+Y^B$
1	1				0	j, β	i, α
1	0	T			0	j, k_2	i, α
1	0	F			-1	j	α
0	1		T		0	j, β	i, l_2
0	1		F		-1	β	i
0	0	T	T		0	j, k_2	i, l_2
0	0	T	F		-1	k_2	i
0	0	F	T		-1	j	l_2
0	0	F	F	1	-1	j	i
0	0	F	F	0	-2		

3 Partition swaps applied in the annealing algorithm

From a simple graph $G = (V, E)$ and an initial partition size, we can run the annealing algorithm [12] on vertex swaps to search for a partition with minimal cut rank. This has been implemented using cut-rank calculations from inspections on the key matrices for the partition. In all the result presented here, the temperature schedule $\mathcal{T}$ is chosen linearly in the range from 1.0 to 0.1 with a reduction of 0.1 between each step.

3.1 Results on square grids

The annealing algorithm has been applied to $n \times n$ grid graphs to compare the two ways to calculate the cut rank changes, either by ordinary Gauss-Jordan elimination (using an in-house python implementation), or by using the key matrices for the partition. It can also be used to indicate how often the annealing algorithm finds a partition with minimum possible cut rank, since this minimum is known to be n if $n \geq 3$. The execution times for a random balanced partition of the $n \times n$ grid graphs for different cut rank calculation methods are shown in Fig. 2. Furterhmore, the average deviations of the cut rank from the known minimum n after running the annealing algorithm on 100 $n \times n$ grid graphs with random initial partitions are shown in Fig. 3.

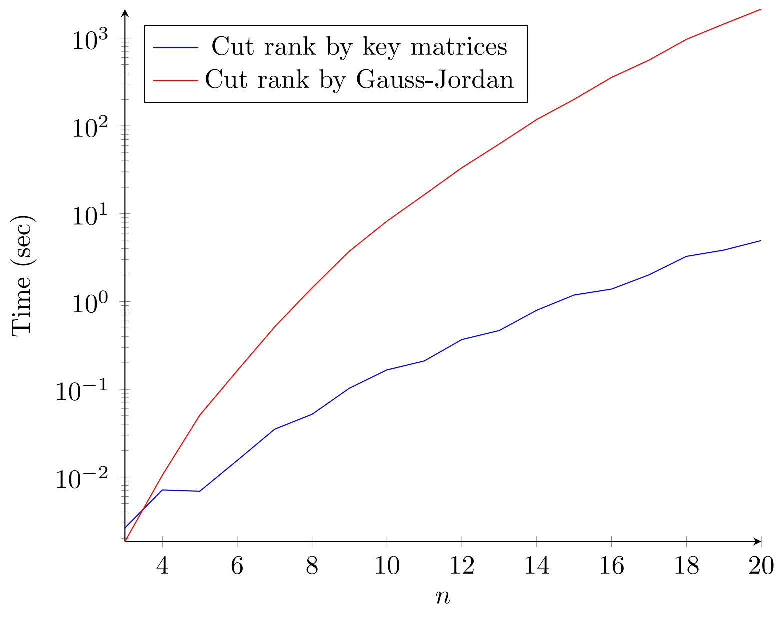

Figure 2: Execution times of the simulated annealing algorithm for a random balanced partition of the $n \times n$ grid graphs using naive Gauss-Jordan elimination for calculating the rank in each iteration, vs. our proposed algorithm using update rules. While Gauss-Jordan scales with n^3, our algorithm achieves a scaling of n^2.

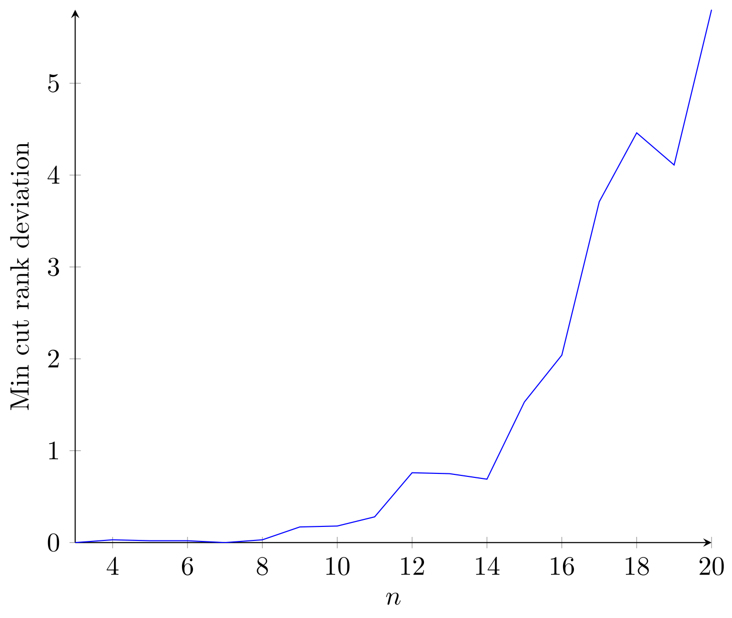

Figure 3: Average deviation from known minimal cut rank n after running the annealing algorithm on 100 $n \times n$ grid graphs with random initial partitions.

3.2 Results on sparse graphs

In this section, we show results for random graphs. We evaluate the average runtimes and final cut-rank for the annealing algorithm on 100 sparse Erdős-Rényi random graphs $G(n, p)$ for each n, where $p = c/n$ and the first partition set has size $P_1 n$. The results are shown in Figs. 4 and 5, respectively.

The rank-width for random graphs is known asymptotically [13]. If we would allow flexible partition sizes with the constraint that the sizes of both sets are between $n/3$ and $2n/3$, the cut-rank is bounded by rank-width [13]. However, here we use fixed-sized partitioning and thus observe a scaling that is different from the rank-width scaling.

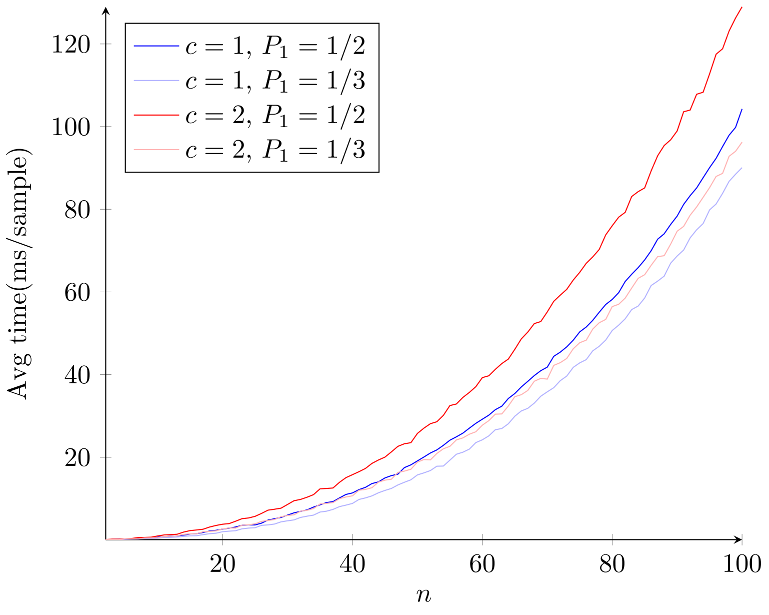

Figure 4: Average runtimes for the annealing algorithm on 100 sparse Erdős-Rényi random graphs $G(n, p)$ for each n, where $p = c/n$ and the first partition set has size $P_1 n$.

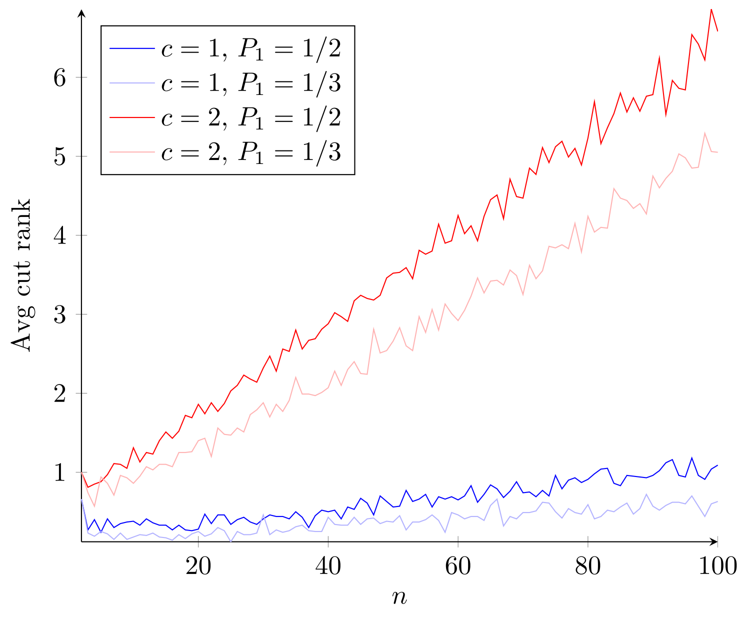

Figure 5: Average final cut rank after running the annealing algorithm on 100 sparse Erdős-Rényi random graphs $G(n, p)$ for each n, where $p = c/n$ and the first partition set has size $P_1 n$.

4 Conclusions and future work

We show that there exists an efficient algorithm for the simple task of computing incremental changes in the cut rank of graph when two nodes are swapped across two partitions. This can be used in a simulated annealing-based optimization to find fixed-sized partitions of graph with minimum cut rank This is a first step towards efficient algorithms for graph state partitioning in distributed Measurement-Based Quantum Computing. Future work includes extensions to multiple partitions and more sophisticated metaheuristics.

Acknowledgements

This work was partially supported by the research project *Zentrum für Angewandtes Quantencomputing* funded by the Hessian Ministry for Digital Strategy and Innovation and the Hessian Ministry of Higher Education, Research and the Arts (M.H.), and by the project Quantum Extension of the Norwegian Research Infrastructure (Q-NRI) funded by the The Research Council of Norway (G.S and K.F.P). We thank Sang-il Oum and Tony Huynh for helpful discussions.

References

[1] Marcello Caleffi, Michele Amoretti, Davide Ferrari, Jessica Illiano, Antonio Manzalini, and Angela Sara Cacciapuoti. Distributed quantum computing: A survey. *Computer Networks*, 254:110672, 2024.

[2] Juan C Boschero, Niels MP Neumann, Ward van der Schoot, Thom Sijpesteijn, and Robert Wezeman. Distributed quantum computing: Applications and challenges. In *Intelligent Computing-Proceedings of the Computing Conference*, pages 100–116. Springer, 2025.

[3] Naomi H. Nickerson, Ying Li, and Simon C. Benjamin. Topological quantum computing with a very noisy network and local error rates approaching one percent. *Nature Communications*, 4(1):1756, Apr 2013.

[4] Shobhit Gupta, Nikolay Sheshko, Daniel J. Dilley, Alvin Gonzales, Manish K. Singh, and Zain H. Saleem. Gate teleportation vs circuit cutting in distributed quantum computing, 2025.

[5] Robert Raussendorf and Hans J Briegel. A one-way quantum computer. *Physical review letters*, 86(22):5188, 2001.

[6] Vincent Danos, Ellie D'Hondt, Elham Kashefi, and Prakash Panangaden. Distributed measurement-based quantum computation. *Electronic Notes in Theoretical Computer Science*, 170:73–94, 2007. Proceedings of the 3rd International Workshop on Quantum Programming Languages (QPL 2005).

[7] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Graphical description of the action of local clifford transformations on graph states. *Physical Review A*, 69(2):022316, 2004.

[8] Marc Hein, Jens Eisert, and Hans J Briegel. Multiparty entanglement in graph states. *Physical Review AAtomic, Molecular, and Optical Physics*, 69(6):062311, 2004.

[9] M. Van den Nest, W. Dür, G. Vidal, and H. J. Briegel. Classical simulation versus universality in measurement based quantum computation. *Phys. Rev. A*, 75:012337, 2007.

[10] Huy-Tung Nguyen and Sang-il Oum. The average cut-rank of graphs. *European Journal of Combinatorics*, 90:103183, December 2020.

[11] Sang-il Oum. Rank-width: Algorithmic and structural results. *Discrete Applied Mathematics*, 231:1524, 11 2017.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[13] Choongbum Lee, Joonkyung Lee, and Sang-il Oum. Rank-width of random graphs. *Journal of Graph Theory*, 70(3):339–347, 2012.