

Depth-Optimal Addressing of 2D Qubit Array with 1D Controls Based on Exact Binary Matrix Factorization

Daniel Bochen Tan

University of California, Los Angeles
bochentan@ucla.edu

Shuohao Ping

University of California, Los Angeles
sp1831@ucla.edu

Jason Cong

University of California, Los Angeles
cong@cs.ucla.edu

Abstract—Reducing control complexity is essential for achieving large-scale quantum computing, particularly on platforms operating in cryogenic environments. Wiring each qubit to a room-temperature control poses a challenge, as this approach would surpass the thermal budget in the foreseeable future. An essential tradeoff becomes evident: reducing control knobs compromises the ability to independently address each qubit. Recent progress in neutral atom-based platforms suggests that rectangular addressing may strike a balance between control granularity and flexibility for 2D qubit arrays. This scheme allows addressing qubits on the intersections of a set of rows and columns each time. While quadratically reducing controls, it may necessitate more depth. We formulate the depth-optimal rectangular addressing problem as exact binary matrix factorization, an NP-hard problem also appearing in communication complexity and combinatorial optimization. We introduce a satisfiability modulo theories-based solver for this problem, and a heuristic, row packing, performing close to the optimal solver on various benchmarks. Furthermore, we discuss rectangular addressing in the context of fault-tolerant quantum computing, leveraging a natural two-level structure.

Index Terms—quantum, binary, rank, biclique, partition

I. INTRODUCTION

Quantum computing demands a thermal environment with significantly lower fluctuations than the energy level differences in qubits. Because of this, solid-state platforms, such as superconducting circuits or semiconductor quantum dots, can only operate in cryogenic environments. Controlling a quantum processor entirely from room temperature becomes impractical for large-scale quantum computing, where the requirement of millions of physical qubits exceeds the capacity of dilution refrigerators, supporting only hundreds of coaxial cables [1]. Given a certain degree of homogeneity in qubit fabrication, one viable strategy is manipulating multiple qubits with one signal. Hornibrook et al. [2] demonstrate a microarchitecture utilizing ‘switch matrices’ to selectively distribute analog signals. In this work, we extend it to 2D and aim to reduce the control complexity by exploring the idea of relaxing qubit addressability.

For a 2D array $X \times Y$, a (*combinatorial*) *rectangle* is a set of the form $X' \times Y'$, where $X' \subseteq X$ and $Y' \subseteq Y$. Specifying a rectangle requires $|X| + |Y|$ bits (one bit for each row and each column), a significant reduction compared to $|X| \cdot |Y|$ bits for all elements. In Figure 1a, we present a conceptual architecture where qubits can be addressed using rectangles formed by intersections of rows and columns. Compared to a single address line, there are both row and column address lines, and an additional switch for each row. This setting provides an advantage: in each clock cycle, the row and column address

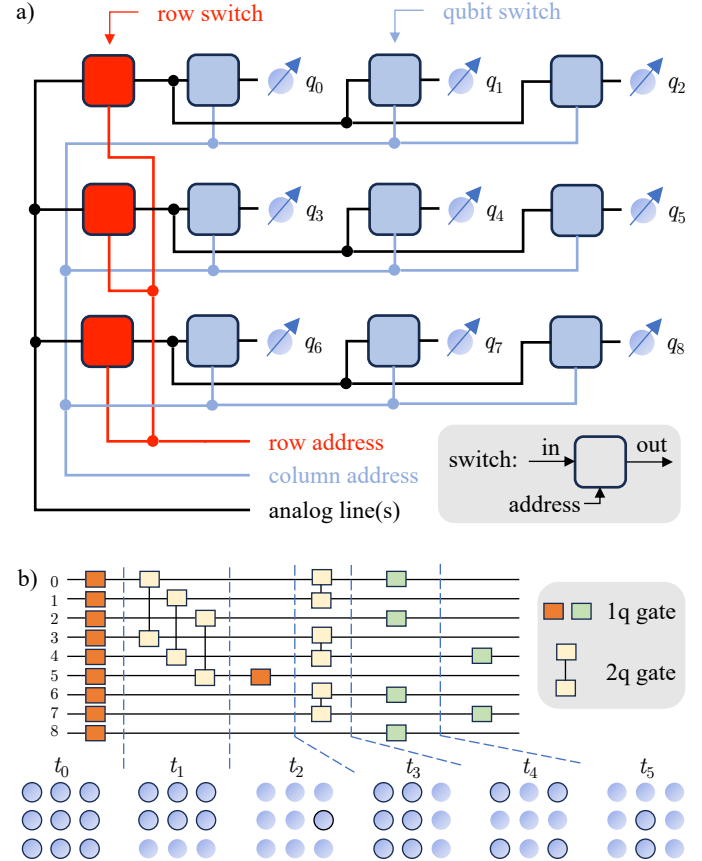


Figure 1. Quantum architecture employing the rectangular addressing for a 2D qubit array. a) Conceptual image of the architecture. Analog lines provide the signals to manipulate the qubits. They are first routed by the row switches using the row address signal, and later routed by the qubit switches using the column address signal. The row and column address signals transmit 6 bits per clock cycle, in contrast to 9 bits that would be used for individual addressing. In general, this is a quadratic difference. b) Addressed qubits at different stages of an example quantum circuit. At each stage, only a rectangle of qubits are addressed, which may necessitate more depth, e.g., t_4 and t_5 cannot be merged.

lines transmit only the square root of the total bits required in a single address line configuration, at the expense of possible layer execution time. This quadratic reduction is maintained while preserving individual addressability, as a single element can still be considered a rectangle. For example, in Figure 1b, we illustrate qubits being addressed at different stages of a quantum circuit. At t_2 , the column and row addresses are (0, 1, 0) and (0, 0, 1) respectively, ensuring only q_5 is addressed.

Our motivation stems from recent successful large-scale

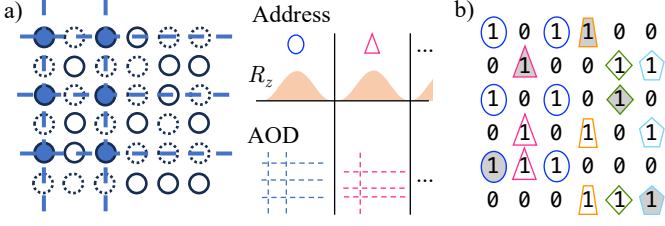


Figure 2. Rectangular addressing in neutral atom arrays. a) The experimental setup in Bluvstein et al. [3]: a 2D acousto-optic deflector (AOD, blue dashes) modulates another laser to realize R_z gates on qubits at the AOD crossing points (colored dots). Different qubits (uncolored dots) are addressed by changing the AOD signal. Qubits not in the pattern (dash circles) should not be addressed. b) Rectangular partition of a). Different markers distinguish 5 rectangles to partition the matrix. The 5 filled markers indicate a fooling set.

experiments on the neutral atom arrays platform [3] that highlights the effect of reducing control complexity by rectangular addressing. As depicted in Figure 2a, the acousto-optic deflector (AOD) illuminates a product of rows and columns. Quantum gates, induced by specific pulses modulated by the AOD, address qubits at the row and column intersections. AODs prove effective for implementing gates [3], [4] and qubit movements [5]–[8]. Moreover, certain key applications, such as the surface code [9], inherently feature rectangular structures. This code requires a 2D grid of data qubits, and another 2D grid of ancilla qubits to conduct error checks. Exploiting this structure, Versluis et al. [10] show that 8 sets of signals are sufficient for the decoding sequence on superconducting qubits. The qubits addressed by each set forms a rectangle, so, in principle, the sequence can be realized by addressing 8 control signals to all qubits within their respective rectangles.

The coarser granularity of rectangular addressing may reduce control complexity at the cost of increasing depth. Although t_4 and t_5 in Figure 1b involve single-qubit gates on different qubits, they cannot be combined into a single stage, because merging them would result in a non-rectangular arrangement of qubits to address. A more general problem is given by the matrix in Figure 2b, where the qubits to address are represented by the 1's. This matrix can be partitioned into five rectangles, each designated by distinct markers. Consecutively, each rectangle receives a modulated R_z pulse through specific AOD configurations. Minimizing the number of rectangles to partition arbitrary binary matrices becomes crucial.

In this paper, we consider the problem of achieving depth-optimal rectangular addressing, which we formulate as *exact binary matrix factorization*. We present a satisfiability modulo theories (SMT, extension of SAT) formulation for this problem and an effective heuristic dubbed *row packing*. The combined algorithm, SAP (SMT and packing), finds high-quality heuristic solutions quickly and then iteratively approaches to the optimal solution. To assess our methods, we generate three benchmark sets. The first set comprises random matrices, the second includes matrices with known optimal solutions, despite the problem generally being NP-hard. The third set is constructed to accentuate the gap between the partition number and the conventional matrix rank. We observe that for future fault-tolerant quantum computing (FTQC), the problem may exhibit a product structure. This implies that we can solve limited-size

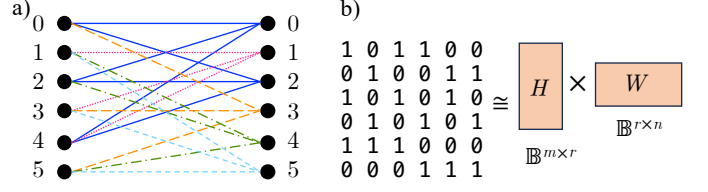


Figure 3. a) Interpreting the matrix as the adjacency matrix of a bipartite graph, the rectangular partition problem becomes *biclique partition* where the edges are partitioned to form complete bipartite subgraphs (different line types). b) Binary matrix factorization finds low-rank approximations HW of the original matrix where H and W are also required to be binary.

problems on multiple levels and then combine the solutions.

The paper is structured as follows. In Section II, we provide a review of mathematical concepts related to this problem across various contexts and applications. In Section III, we introduce our algorithms. The construction of benchmarks and the evaluation results are presented in Section IV. We discuss the problem in the context of FTQC in Section V. Finally, the conclusions and future directions are provided in Section VI.

II. BACKGROUND

The term we have adopted, *rectangle*, is standard in communication complexity theory [11], where the matrix in Figure 2b represents a binary function g of two variables. Alice has some i , Bob has some j , and our interest is determining the number of bits the two need to communicate to compute $g(i, j)$. If $g = 1$ uniformly on a rectangle, it is a *1-monochromatic rectangle*. The number of 1-monochromatic and 0-monochromatic rectangles to partition the whole matrix serves as a crucial lower bound for the communication complexity. For an introduction to this topic, readers are referred to Kushilevitz & Nisan [12]. Two results they cover are worth mentioning for later discussions. First, there exists an alternative definition of a rectangle:

$$(i, j) \in R \text{ and } (i', j') \in R \Rightarrow (i, j') \in R. \quad (1)$$

In this paper, we only focus on 1-monochromatic rectangles and we will refer to these as ‘rectangles’ from now on. The second important fact is that the partition number is lower bounded by the size of *fooling sets*. In our case, a fooling set S consists of (i, j) such that $g(i, j) = 1$, and for any distinct pair (i, j) and (i', j') in S , $g(i', j) = 0$ or $g(i, j') = 0$. Indeed, the shaded markers in Figure 2b identify such a fooling set of size 5, implying that our partition into 5 rectangles is optimal. Fooling sets do not always guarantee a tight bound, e.g.,

$$\begin{aligned} & \text{3 rectangles are needed to partition} \\ & \text{but the size of any fooling set is } \leq 2 \end{aligned} \quad \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2)$$

The problem has a graph-theoretic interpretation when considering the matrix as the adjacency matrix of a bipartite graph, as illustrated in Figure 3a. The left vertices correspond to the rows, while the right vertices correspond to the columns. An edge exists between vertex i on the left and vertex j on the right if and only if element (i, j) is 1 in the matrix. Viewed in this way, a rectangle, seen as a set of edges, forms a *biclique*, i.e., a complete bipartite graph. For instance, the addressed sites in Figure 2a correspond to a complete (3,2)-bipartite subgraph

in Figure 3a, as denoted by the solid edges. Therefore, the rectangular partition is equivalent to a biclique partition of a bipartite graph. Reinterpreting the left vertices as sets and right vertices as objects, the biclique partition is finding a *normal set basis* to decompose each set. In our example, the basis is $\{\{0, 2\}, \{1\}, \{3\}, \{4\}, \{5\}\}$, with the first set on the left decomposed into $\{0, 2\} \sqcup \{3\}$. The decision problem is proven NP-complete [13]. Even approximating the problem is NP-hard [14]–[16]. Amilhastre et al. [17] have characterized certain graph families where the problem can be efficiently solved.

The third perspective regarding a rectangular partition is through matrix factorization, as each rectangle precisely corresponds to a rank-1 submatrix. In Figure 3b, within a *binary matrix factorization* (BMF), given a binary matrix $M \in \mathbb{B}^{m \times n}$ and an integer r , the objective is to minimize $\|M - HW\|$ where $H \in \mathbb{B}^{m \times r}$ and $W \in \mathbb{B}^{r \times n}$. Note that $HW = \sum_{i=1}^r P_i$ where P_i is the product of column i in H and row i in W . Each $P_i \in \mathbb{B}^{m \times n}$ is 1 on a combinatorial rectangle and 0 elsewhere, so it has rank 1. The minimum r for which $M - HW = 0$ is the *binary rank*, $r_{\mathbb{B}}$ of M . In this case, $\sum_{i=1}^r P_i$ is an *exact* binary matrix factorization (EBMF) of M . In contrast to SVD, which provides the rank in \mathbb{R} , EBMF additionally requires H and W to be binary. However, it is crucial to note that the additions in the matrix multiplication in EBMF is in \mathbb{R} , not in \mathbb{B} , e.g.,

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \stackrel{\text{EBMF}}{\neq} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}.$$

If the addition were in \mathbb{B} , the equality holds. But in \mathbb{R} , the top-left element appears in *both* rectangles on the r.h.s., violating the disjointedness requirement of rectangular partitioning. For binary matrices, we have a straightforward lower bound [18]:

$$\text{rank}_{\mathbb{R}}(M) \leq r_{\mathbb{B}}(M) \quad \forall M \in \mathbb{B}^{m \times n}. \quad (3)$$

Zhang et al. [19] develop a BMF optimizer which is integrated into a well-known package NIMFA [20]. However, since it is not designed for EBMF but to provide approximations given a fixed r , it does not perform well for our specific purposes.

III. ALGORITHM

Given a matrix $M \in \mathbb{B}^{m \times n}$, we are interested in its exact binary matrix factorization (EBMF) $M = \sum_{i=1}^{r_{\mathbb{B}}-1} P_i$ where each $P_i \in \mathbb{B}^{m \times n}$ is 1 on a rectangle and 0 elsewhere.

Our SMT formulation encodes the problem: given M and a number b , determine if $r_{\mathbb{B}}(M) \leq b$. When $r_{\mathbb{B}}$ is unknown, we query an SMT solver with decreasing values of b to compute it. Given the problem's complexity, the worst-case runtime is exponential to the size of M . Hence, the key lies in establishing relatively tight bounds for b to minimize SMT invocations.

Our approach, SAP (SMT and packing), is presented in Algorithm 1. First, our heuristic, *row packing*, provides a valid EBMF, P . Since $|P|$ is an upper bound of $r_{\mathbb{B}}(M)$, the SMT solving initiates with $b = |P| - 1$ and terminates when the SMT formula is unsatisfiable or when b falls below $\text{rank}_{\mathbb{R}}(M)$, a lower bound as per Equation 3. P is updated each time the SMT formula is satisfiable so that it retains to the best solution found thus far even if the process is prematurely interrupted.

Algorithm 1: SAP (SMT and packing) EBMF

Data: $M \in \mathbb{B}^{m \times n}$
Result: P , an EBMF of M consisting of rectangles

```

1  $P \leftarrow \text{row\_packing\_EBMF}(M);$           /* Algorithm 2 */
2  $b \leftarrow |P| - 1;$ 
3  $\text{formula} \leftarrow \text{construct\_SMT\_formula}(M, b);$ 
4 while  $b \geq \text{real rank of } M$  do
5   if  $\text{formula}$  is satisfiable then
6      $P \leftarrow \text{readout\_solution}(\text{formula});$ 
7      $b \leftarrow b - 1;$ 
8      $\text{formula} \leftarrow \text{narrow\_down\_depth}(\text{formula}, b);$ 
9   else
10    break;
```

A. SMT Formulation

Fundamentally, we want to compute a function $f : E \rightarrow P$, where E comprises the 1's in the matrix, and P contains the rectangles. This definition offers the convenience of inherently ensuring the disjointedness of the rectangles. Furthermore, the constraints needed to enforce the validity of f in specifying rectangles can be expressed using first-order logic and equality between function values. This closely aligns with the uninterpreted function, a major addition in SMT compared to SAT [21], [22]. Concretely, the only set of constraints follows from Equation 1: for every pair of distinct 1's at (i, j) and (i', j') ,

$$\begin{cases} f_{i,j} \neq f_{i',j'} & \text{if } M_{i,j'} = 0, \\ (f_{i,j} = f_{i',j'}) \Rightarrow (f_{i,j} = f_{i,j'}) & \text{if } M_{i,j'} = 1. \end{cases} \quad (4)$$

Another SMT feature we leverage are bit-vectors. In fact, both the domain and range of f are bit-vectors: $f_{i,j}$ above means $f(e(i, j))$ where e is an index function of the 1's in M , and the value of f is the index of a rectangle. To narrow down the solution space as in line 8 of Algorithm 1, we just add new constraints $f_{i,j} \neq b$ for every $M_{i,j} = 1$ to the SMT formula.

B. Heuristics

A trivial upper bound of $r_{\mathbb{B}}(M)$ is the width or height of M , whichever smaller, after removing empty and duplicated rows and columns. This corresponds to partitioning the matrix into single rows or columns and consolidating duplicated ones.

The normal set basis viewpoint inspires our second heuristic. We process matrix M row by row, with the goal of forming a *basis* – each basis vector corresponds to one rectangle, as outlined in Algorithm 2. For each row r_i , as in lines 4-7, if an existing basis vector v_j is found within this row, we append i to the rectangle P_j associated with v_j . Subsequently, we remove the 1's in v_j from r_i and continue this process. The outcome is the decomposition of r_i into a disjoint union of existing basis vectors, potentially leaving a *residue* of 1's. An example is displayed in Figure 4a where the first 4 rows cannot be decomposed, so the residues are just the rows themselves, and they are added to the basis, i.e., $v_i = r_i$, $i \in \{0, 1, 2, 3\}$. When it comes to r_4 , we note it contains v_0 (circles) and v_1 (triangles), so the residue is $(0, 0, 0, 0, 1)$ denoted by the pentagon. Based on this decomposition, the rectangles P_0 and P_1 , corresponding to v_0 and v_1 , vertically grow to include row 4.

Algorithm 2: Row-Packing EBMF

Data: $M \in \mathbb{B}^{m \times n}$

Result: P , an EBMF of M consisting of rectangles

```

1  $M' \leftarrow \text{shuffle\_rows}(M)$ ;
2  $\text{basis} \leftarrow []$ ;  $P \leftarrow []$ ;
3 for  $r_i \in M'$   $i = 0, 1, \dots, m-1$  do
4   for  $v_j \in \text{basis}$  do
5     if  $\{1\text{'s in } v_j\} \subseteq \{1\text{'s in } r_i\}$  then
6        $P_j \leftarrow \text{vertical\_grow}(P_j, i)$ ;
7        $r_i \leftarrow r_i - v_j$ ;
8   if  $r_i \neq \vec{0}$  then
9      $c \leftarrow \text{one\_hot\_column\_vec}(i)$ ;
10    for  $v_k \in \text{basis}$  do
11      if  $\{1\text{'s in } r_i\} \subseteq \{1\text{'s in } v_k\}$  then
12         $P_k \leftarrow \text{horizontal\_shrink}(P_k, r_i)$ ;
13         $v_k \leftarrow v_k - r_i$ ;
14         $c_k \leftarrow 1$ ;
15     $\text{basis.append}(r_i)$ ;
16     $P.append(c \times r_i)$ ;
17  $P \leftarrow \text{undo\_shuffle}(P, M, M')$ ;
```

Since we adhere to the order of basis vectors, the decomposition can be suboptimal. For instance, we overlook the possibility of $r_4 = v_2 + v_3$, and the residue could have been 0. To mitigate this, we run the heuristic multiple times, shuffling the rows in each trial, e.g., another trial with a different row ordering is exhibited in Figure 4b. This is a compromise to the complexity of the problem. Formally, we are trying to find a *packing* or *exact cover* of r_i by the basis vectors, and it is an NP-complete problem [23] to decide whether one exists.

In the presence of a residue, we perform an update to the basis in lines 9-16. The intuition behind this is that smaller basis vectors enhance the likelihood of a successful row packing. If an existing basis vector v_k contains the residue, we remove the residue from the corresponding rectangle P_k and update v_k . In step 2 of Figure 4b, r_1 itself (triangles) is the residue. We find existing basis vector $v_0 = r_0$ (circles) containing r_1 , so v_0 is updated to $r_0 - r_1$, and then r_1 is added to the basis as v_1 . Because of the updates, some existing rectangles shrink to remove the columns of 1's in the new basis vector, which we record with a column vector c . In step 3 of the example, c notes that v_0 gets updated and v_1 gets added, so the new rectangle $P_1 = c \cdot v_1$ spans rows 0 and 1, and columns 0 and 1. And the existing rectangle P_0 shrinks by removing columns 0 and 1, which leads to the successful packing of r_2 in step 4.

Finally, in line 17, we reverse the initial shuffling to derive the correct EBMF. It is worth noting that the algorithm introduces at most one rectangle for each non-repeating row, ensuring that the result is no worse than the trivial heuristic. The overall time complexity is $O(n^3k)$, where k represents the number of trials, and n denotes the larger of matrix width and height. This complexity is due to nested loops in lines (3,10) and (3,4), with the innermost loop involving vector operations. Additionally, note that we run the heuristic on both the original matrix and its transpose, retaining the better result.

In scenarios with limited time budgets, two further compro-

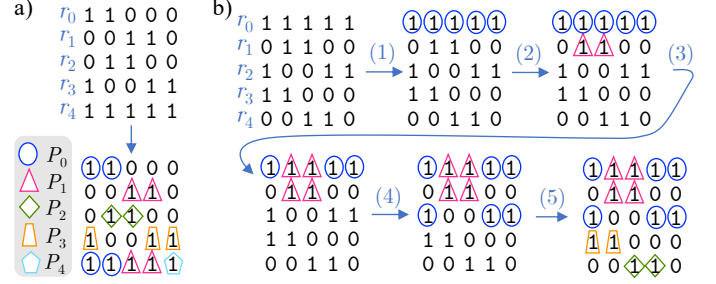


Figure 4. Two trials of running the row packing heuristic. Rectangles found are represented by different markers. a) needs 5 rectangles but b) needs 4.

mises can be considered. The first one is removing the basis update in lines 10-14. The second one is arranging rows with a smaller number of 1's at the beginning instead of random shuffling, and invoking fewer runs. Based on our experience, both of these tend to result in more suboptimal 'local minima' compared to the current setting, so we have not adopted them.

IV. EVALUATION

We implement the above approach which is open-source under the MIT License¹. The software relies on numpy 1.26.3 and z3-solver 4.12.1.0 [21]. The evaluation is conducted on a server with an AMD EPYC 7V13 CPU and 512 GB RAM.

A. Benchmark Construction

We provide benchmarks in two sizes: 1) limiting the number of rows by 10 so that we can reliably prove the optimality of the solutions using SMT, and 2) 100×100 , which is considered to be the current limit of atom array technology [3].

The first benchmark set consists of random matrices. We generate 10 matrices with varying occupancies of 1's (10%, 20%, ..., 90%) for sizes 10×10 , 10×20 , and 10×30 . For the 100×100 size, we choose occupancies of 1%, 2%, 5%, 10%, and 20%, because higher occupancies almost always result in full rank, which is trivial for our evaluation.

The second benchmark set is comprised of matrices with known optimal solutions. According to Equation 3, if a matrix has a k -rectangle partition and the real rank is also k , the partition is optimal. We create pairs of disjoint rows r_i and linearly independent columns c_i , leading to matrices $M = \sum_{i=1}^k c_i \cdot r_i$. We enforce disjointedness among the rows to ensure that the outcome matrices only contain 0's and 1's, and the rectangles cannot merge. For each rank $k = 1, 2, \dots, 10$, we generate 10 benchmarks of size 10×10 with known optimal solutions.

The third benchmark set is designed to create a gap between the real rank and the binary rank. We begin by sampling a random row r and then randomly decompose it into disjoint row pairs $r = r' + r''$. The parameter for this family of benchmarks is the number of row pairs, k , which is limited to $\lfloor m/2 \rfloor$. The real rank of these $2k$ rows should be $k+1$ because any pair can recover the original row, $r = r_0 + r_1$. Each pair then should provide an independent basis vector, e.g., r_{2i} for $i = 0, \dots, k-1$. Note that decompositions like $r_3 = r_0 + r_1 - r_2$ require the use of negative numbers, which are not allowed in an EBMF.

¹<https://github.com/UCLA-VAST/EBMF>

Table I
PERCENTAGE OF CASES FINDING AN OPTIMAL SOLUTION

benchmark	rank [†]	trivial	row packing, number of trials			
			1	10	100	1000
10×10, rand	98%	80%	91%	99%	100%	100%
10×20, rand	100%	100%	100%	100%	100%	100%
10×30, rand	100%	100%	100%	100%	100%	100%
100×100, rand	100% [‡]	62%	92%	96%	98%	100%
10×10, opt	100%	100%	100%	100%	100%	100%
10×10, gap, 2	74%	29%	88%	100%	100%	100%
10×10, gap, 3	63%	16%	91%	100%	100%	100%
10×10, gap, 4	47%	40%	94%	98%	99%	99%
10×10, gap, 5	42%	84%	90%	94%	96%	96%

[†]Percentage of cases where real and binary ranks are the same.

[‡]Since the heuristics managed to find optimal solutions, the real and binary ranks are the same for this set. The SMT for these cases are too large to solve.

Consequently, the binary rank of the matrix should be larger than $k + 1$. The remaining $m - 2k$ rows are completed with random rows having a 50% occupancy, resulting in a total real rank equal to or slightly lower than $m - k + 1$. We generate 100 benchmarks of size 10×10 with 2, 3, 4, and 5 row pairs.

B. Results

The SMT solver allows us to compute optimal solutions. The percentage of cases achieving optimal solutions with the heuristics is presented in Table I. The ‘rank’ column indicates the percentage of cases where the binary rank equals the real rank. Although the 100×100 benchmarks are too large for SMT to find solutions, the heuristics find solutions with the number of rectangles equal to the real rank. Consequently, these solutions are known to be optimal, and the real and binary ranks are in agreement. Several observations can be made.

Observation 1: the real and binary ranks are equal with high probability for random matrices. This can be attributed in part to the near-full real rank of random matrices. We observe that almost all 10×20 matrices with an occupancy of 20% and higher, all 10×30 matrices, and 100×100 matrices with an occupancy of 5% and higher are full rank, necessitating full binary rank, resulting in equality between the two.

Observation 2: the constructed benchmarks with known optimal are easy. Due to the mechanism of row packing, it can always find the optimal solutions for these benchmarks. Surprisingly, the trivial heuristic also manages to find the optimal solutions on all cases, because even though the row space cannot be reduced by construction, e.g.,

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

the columns may be reduced by recognizing duplication.

Observation 3: row packing is an effective heuristic. On benchmarks with gaps and the large random benchmarks, there is a big gap between the trivial heuristic and even one trial of row packing, indicating row packing is highly non-trivial. As expected, the performance of row packing improves with more trials. On most of the benchmarks, it saturates at 100 trials and finds optimal solutions on a remarkable percentage of cases.

Observation 4: edge cases for row packing needs more general search. We look into the cases where row packing

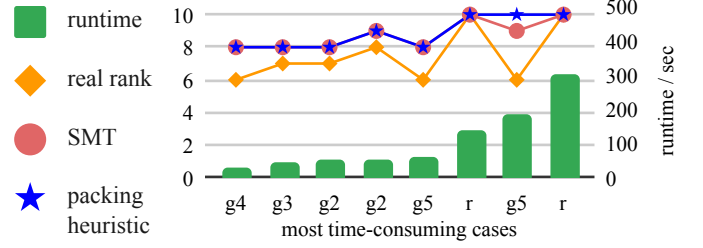


Figure 5. The most time-consuming cases. ‘r’ means it is a random benchmark, ‘g2’ means it comes from benchmarks with gap using 2 row pairs, etc.

fails to find the optimal solution. Going through Algorithm 2, we find these cases necessitates introducing more than one basis at some rows, whereas the row packing heuristic at most introduces one new basis per row in order for efficiency.

Observation 5: the most time consuming cases are proving UNSAT. We collect the most time-consuming cases in Figure 5. In the majority of these cases, the SMT solver can only find solutions with the same number of rectangles as row packing. Then, the solver goes on decreasing the bound by 1 and proves the formula to be UNSAT. This is the most time consuming task. Note that in Algorithm 1, when we terminate at any time, we can return P , the best solution found so far.

V. FAULT-TOLERANT QUANTUM COMPUTING

Fault-tolerant quantum computing performs on top of quantum error correction codes that encode each logical qubit using quantum states distributed across multiple physical qubits. A promising approach is exemplified by the surface code [9], where a logical qubit manifests as a patch of physical qubits, as depicted in Figure 6a. For simplicity, only the data qubits are illustrated, and check qubits are not shown. A single-logical-qubit operation, designated as U , corresponds to a 2D pattern (M) of physical gates, as highlighted in the callout. On the logical level, the quantum circuit may necessitate another 2D pattern (\hat{M}) of logical operations. Consequently, the overall physical operation is expressed as the tensor product $\hat{M} \otimes M$. This two-level structure allows for the independent computation of the rectangular partition of \hat{M} and M . Subsequently, taking the tensor product of the partitions produces the solution.

However, is this solution optimal? The real rank is multiplicative under a tensor product, as elementary row operations can be employed to make both M and \hat{M} upper triangular, resulting in an upper-triangular tensor product. In contrast, whether the binary rank is multiplicative under a tensor product remains an open question. Our aforementioned solution (tensor product of partitions) provides an upper bound: $r_{\mathbb{B}}(\hat{M} \otimes M) \leq r_{\mathbb{B}}(\hat{M}) \cdot r_{\mathbb{B}}(M)$. For lower bounds, Watson [18] notes that

$$\max \left(r_{\mathbb{B}}(\hat{M}) \cdot \phi(M), r_{\mathbb{B}}(M) \cdot \phi(\hat{M}) \right) \leq r_{\mathbb{B}}(\hat{M} \otimes M) \quad (5)$$

where ϕ denotes the maximum fooling set size. However, as per Equation 2, ϕ is not always equal to $r_{\mathbb{B}}$. In practice, the majority of M is simple, such as applying X , Z , or H to all the physical qubits in one patch. In this case, all the elements of M are 1, and indeed we have $\phi(M) = r_{\mathbb{B}}(M) = 1$, so the rectangular partition of \hat{M} leads to an optimal solution.

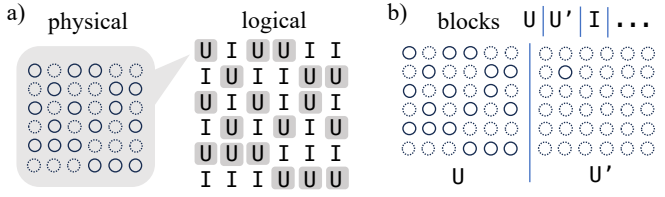


Figure 6. Rectangular addressing in fault-tolerant quantum computing. a) An operation U on 2D patterns of logical qubits can be realized by the tensor product of partitions on the logical and physical levels. b) For logical blocks in 1D layout and with different operations, addressing by row is usually enough.

Another family of quantum error correction codes gaining popularity is quantum low-density parity-check codes, which can take advantage of the mobility of atom arrays [24]. In this code, logical qubits are more globalized, with multiple logical qubits stored in one *logical block* instead of one qubit per block. These blocks are usually arranged in a 1D fashion, as shown in Figure 6b, because they only serve as memory, and logical qubits need to be read out to a computing zone. Considering logical operations that can be realized with single-qubit gates in this setting, the pattern on each block can be quite different, depending on the offset of logical qubits inside the blocks. We conjecture that addressing qubits row by row is usually sufficient in this case, as in our evaluation, we find that given the same occupancy, the 10×20 and 10×30 random matrices are much easier to be full rank than the 10×10 matrices.

VI. CONCLUSION AND DISCUSSION

In this paper, we consider the depth-optimal rectangular addressing for 2D qubit arrays, which turns out to be equivalent to the *exact binary matrix factorization* problem that has applications in various fields. We introduce an SMT-based solver for it along with an effective heuristic, *row packing*, which scales to the current limits of atom array technology. Several future directions can be explored. The packing procedure in our implementation might benefit from ideas in existing works such as Knuth’s Algorithm X for exact cover [25] instead of purely relying on shuffling. Another avenue is the introduction of vacancies in the atom arrays. Since there are no qubits, it is irrelevant how many times we address these sites. They can be represented as *don’t cares* in a matrix, which may be leveraged to reduce rectangles. This task is binary matrix *completion* [26], [27] instead of factorization. Additionally, the SMT tool could aid in investigating the behavior of binary rank under a tensor product. Furthermore, optimal rectangular partitions generated by this tool can provide insight for designing cryogenic control architecture of qubit arrays for various applications.

ACKNOWLEDGEMENT

This work is funded by NSF grant 442511-CJ-22291. The authors would like to thank Y. Song for discussions on superconducting circuits, D. Bluvstein and H. Zhou for conversations on neutral atom arrays, and a post on TCS Stack Exchange about binary rank by D. Issac, R. Kothari, and S. Nikolov.

REFERENCES

[1] C. G. Almudever, L. Lao, X. Fu *et al.*, “The engineering challenges in quantum computing,” in *Design, Automation & Test in Europe Conference & Exhibition*, Mar. 2017, pp. 836–845.

[2] J. M. Hornibrook, J. I. Colless, I. D. Conway Lamb *et al.*, “Cryogenic control architecture for large-scale quantum computing,” *Physical Review Applied*, vol. 3, no. 2, p. 024010, Feb. 2015.

[3] D. Bluvstein, S. J. Evered, A. A. Geim *et al.*, “Logical quantum processor based on reconfigurable atom arrays,” *Nature*, 2023.

[4] T. M. Graham, Y. Song, J. Scott *et al.*, “Multi-qubit entanglement and algorithms on a neutral-atom quantum computer,” *Nature*, vol. 604, no. 7906, pp. 457–462, Apr. 2022.

[5] S. Ebadi, T. T. Wang, H. Levine *et al.*, “Quantum phases of matter on a 256-atom programmable quantum simulator,” *Nature*, vol. 595, no. 7866, pp. 227–232, Jul. 2021.

[6] D. Bluvstein, H. Levine, G. Semeghini *et al.*, “A quantum processor based on coherent transport of entangled atom arrays,” *Nature*, vol. 604, no. 7906, pp. 451–456, Apr. 2022.

[7] B. Tan, D. Bluvstein, M. D. Lukin, and J. Cong, “Qubit mapping for reconfigurable atom arrays,” in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, Oct. 2022.

[8] D. B. Tan, D. Bluvstein, M. D. Lukin, and J. Cong, “Compiling quantum circuits for dynamically field-programmable neutral atoms array processors,” Jun. 2023, arXiv:2306.03487 [quant-ph].

[9] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, “Surface codes: Towards practical large-scale quantum computation,” *Phys. Rev. A*, vol. 86, p. 032324, Sep. 2012.

[10] R. Versluis, S. Poletto, N. Khammassi *et al.*, “Scalable quantum circuit and control for a superconducting surface code,” *Physical Review Applied*, vol. 8, no. 3, p. 034021, Sep. 2017.

[11] A. C.-C. Yao, “Some complexity questions related to distributive computing,” in *Proceedings of the eleventh annual ACM symposium on Theory of computing*, New York, NY, USA, Apr. 1979, pp. 209–213.

[12] E. Kushilevitz and N. Nisan, *Communication Complexity*. Cambridge University Press, 1997, ch. 1.1.

[13] T. Jiang and B. Ravikumar, “Minimal NFA problems are hard,” *SIAM Journal on Computing*, vol. 22, no. 6, pp. 1117–1141, Dec. 1993.

[14] D. Bein, L. Morales, W. Bein *et al.*, “Clustering and the biclique partition problem,” in *Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, Jul. 2008.

[15] P. Chalermsook, S. Heydrich, E. Holm, and A. Karrenbauer, “Nearly tight approximability results for minimum biclique cover and partition,” in *Algorithms - European Symposium on Algorithms 2014*, 2014, pp. 235–246.

[16] S. Chandran, D. Issac, and A. Karrenbauer, “On the parameterized complexity of biclique cover and partition,” in *11th International Symposium on Parameterized and Exact Computation*, 2016.

[17] J. Amilhastre, M. Vilarem, and P. Janssen, “Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs,” *Discrete Applied Mathematics*, vol. 86, no. 2-3, pp. 125–144, Sep. 1998.

[18] T. Watson, “Nonnegative rank vs. binary rank,” *Chicago Journal of Theoretical Computer Science*, vol. 22, no. 1, 2016.

[19] Z. Zhang, T. Li, C. Ding, and X. Zhang, “Binary matrix factorization with applications,” in *Seventh IEEE International Conference on Data Mining*, Oct. 2007, pp. 391–400.

[20] M. Zitnik and B. Zupan, “NIMFA: A Python library for nonnegative matrix factorization,” *Journal of Machine Learning Research*, vol. 13, pp. 849–853, 2012.

[21] L. de Moura and N. Björner, “Z3: An efficient SMT solver,” in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., 2008, pp. 337–340.

[22] W.-H. Lin, J. Kimko, B. Tan, N. Björner, and J. Cong, “Scalable optimal layout synthesis for NISQ quantum processors,” in *2023 60th ACM/IEEE Design Automation Conference*, Jul. 2023.

[23] R. M. Karp, “Reducibility among combinatorial problems,” in *Complexity of Computer Computations*, ser. The IBM Research Symposia Series, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, Eds. Boston, MA: Springer, 1972, pp. 85–103.

[24] Q. Xu, J. P. B. Ataiades, C. A. Pattison *et al.*, “Constant-overhead fault-tolerant quantum computation with reconfigurable atom arrays,” Aug. 2023, arXiv:2308.08648 [quant-ph].

[25] D. E. Knuth, “Dancing links,” *Millennial Perspectives in Computer Science*, pp. 187–214, Nov. 2000, arXiv:cs/0011047.

[26] M. Beckerleg and A. Thompson, “A divide-and-conquer algorithm for binary matrix completion,” *Linear Algebra and its Applications*, vol. 601, pp. 113–133, Sep. 2020.

[27] P. Yadava, “Boolean matrix factorization with missing values,” Master’s thesis, Universität des Saarlandes, 2012.