An Elementary Description of the Oracle of Efficient Sparse Matrices

Benoît Valiron, Renaud Vilmart* and Adham Zekri†

Université Paris-Saclay, CentraleSupélec, CNRS, ENS Paris-Saclay, Inria, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Abstract

Hamiltonian Simulation and Block-Encoding are two ways to encode a given matrix inside a quantum circuit, to allow its processing by subsequent routines, and used in several algorithms such as HHL and QSVT. In this context, efficient sparse matrices served as a case study, where a black-box function is assumed and used to do such an implementation, while the complexity is analyzed relative to this black-box. In this paper, we propose an elementary description of this black-box function. This approach helps demystify the black-box by providing additional information. This demystification has led to the proposal of many applications. Using this approach, we provide a more explicit complexity analysis of the implementation of black-boxes assumed in the literature by introducing a new parameter, upon which the complexity depends, and comparing these implementation methods with a simple implementation method. We also propose an heuristic optimization for this description which minimizes implementation complexity using a clique cover technique, and we prove the hardness of finding such an optimal solution under the optimization framework.

Keywords

Quantum Circuit, Sparse Matrices, Oracle

1. Introduction

Quantum computation is a model of computation where information is stored on the state of quantum particles. The underlying logical representation makes use of Hilbert spaces: a quantum state over n quantum bits (qubits) is represented by a vector of dimension 2^n. This aspect brings many peculiar properties: quantum information is non-duplicable, reading is a probabilistic and destructive operation, and modifying quantum data can only be done through so-called quantum gates: unitary operations on the state space. General operations are realized using sequences of quantum gates, packaged in so-called quantum circuits [1].

The constraints on the manipulation of quantum registers are balanced by the exponential dimension of the state space. It has been shown possible to capitalize on the underlying parallelization allowed by quantum computation to offer speedup compared to classical techniques. Originally, the approach was theoretical, with a focus on the complexity analysis of algorithms [2]. Nowadays, the field has expanded towards industrial application, with the question of concrete resource estimation for specific problem instances [3].

One of the typical problems addressed by quantum computation is the resolution of linear systems of equations [4]. This problem is critical in many fields, ranging from machine learning [5] to chemical simulation [6]. Several competing quantum algorithms such as HHL [7] or QSVT [8] use techniques such as Hamiltonian simulations and Trotterization [9] or Block Encoding [10] to rephrase with quantum circuits the required linear algebraic operations. If the complexity of the algorithm depends on the cost of these subroutines, it also heavily depends on the structure of the linear system and on its presentation.

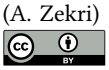
A linear system of equations is characterized by a vector and a matrix. The asymptotic complexity of a quantum algorithm is then governed by several parameters of said matrix: the number of rows and

ICTCS 2026: Italian Conference on Theoretical Computer Science, September 7–9, 2026, Udine, Italy

*Corresponding author.

† Main author.

benoit.valiron@universite-paris-saclay.fr (B. Valiron); renaud.vilmart@inria.fr (R. Vilmart); adham.zekri@inria.fr (A. Zekri)



columns, but also the condition number (i.e. the ratio between the largest and the smallest eigenvalue of the matrix), and the sparsity, that is, the maximum number of non-zero elements on a given row (or column).

The structure of the matrix is important. Typically, linear systems coming from the discretization of differential equations give sparse matrices, and they are very regular: we speak of row- (and column-) computable matrices [11]. In general, sparsity bounds the number of non-zero entries in each row and column. Row-computability requires an efficient procedure that, given a row index, returns the positions and values of the non-zero entries in that row. Their structure can therefore be represented in a compact manner, making it suitable for efficient quantum circuits. Finding such a suitable presentation is therefore critical for the efficiency of the procedure.

A particularly interesting class of matrices is the class of d-sparse [12, 13], efficiently row-computable Hermitian matrices: matrices for which the maximum number of non-trivial entries in each row and column is d, polynomial with respect to the quantum input size such that for any given row, all non-trivial columns can be classically computed in polynomial time with respect to the quantum input size. Many specific structured classes of efficient sparse matrices have been studied in the literature, whether banded circulant, Toeplitz, extended binary tree, *etc.* [14, 15]. For the general case, the state of the art for the compilation of this class of matrices into quantum circuits is presented in [16]. When the sparsity is d, the authors reduce the problem to the synthesis of oracles for 1-sparse Hermitian matrices, which are simpler to implement for they are the composition of a diagonal matrix and a unitary-permutation. The number of terms in the decomposition is crucial, for it affects the complexity of the final encoding of the matrix. The reduction they propose is very general and based on the local resolution of the coloring problem for associated degree-d bipartite graph, using $\leqslant d^2$ colors and resulting in a decomposition with d^2 terms. Another prominent approach, the Sparse-Access Input Model (SAIM) [10], using the synthesis introduced in [17], avoids decomposing the matrix into 1-sparse matrices altogether. It results in a better scaling in d (near-linear instead of quadratic) for the number of queries to the oracles, however the construction of said oracles proves more costly in practice than that of [12, 13].

If the procedure described in [16] is general, in some practical cases such as the one presented in [18], the natural definition is already trivially 1-sparse hermitian decomposable, and therefore it is not necessary to apply the d^2 transformation process. However, in cases such as [18], the presentation of the matrix is ad-hoc and not readily generalizable. If [19] attempts at giving a more direct construction than [16], it remains limited to matrices with entries ranging from a very small subset of fixed values.

In this paper, we propose a universal definition of efficient sparse matrices, that produces a more explicit description for efficient sparse matrices than the row-column-computable notion.

In particular, the main contributions of this paper are the following:

- We identify classes of sparse matrices for which we can give a natural, direct circuit implementation.
- We discuss the complexity of these implementations and compare it with the state of the art.
- We show that such sparse matrix descriptions are amenable to optimization via a reduction to the clique cover problem.

This paper is organized as follows. Section 2 gives the background material to understand the rest. Section 3 defines the notion of efficient sparse matrices, the base notion we build upon. Section 4 presents our contribution: the notion of native computable matrices. Section 5 discusses the complexity of the construction and Section 6 discusses a class of optimizations based on a reduction to the clique covering problem. Finally, Section 7 concludes the paper.

2. Quantum Computation and Oracles for Matrices

In the following, we use the standard notions and notations for computations on qubit systems [1]. The classical states of an n-qubit register are denoted $|k\rangle$ (called "ket k") where $k \in \{0,1\}^n$. Mathematically,

$|k\rangle$ represents the elementary (column) vector of $\mathbb{C}^{2^n}$ with non-zero element at index k. Equivalently, we may use the binary expansion of k, i.e. consider that $k \in \{0,1\}^n$ (with the least significant bit on the right). An n-qubit quantum state is a normalised linear combination of the classical states: $|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle$, with $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$.

The global state of two registers in respective states $|\psi_1\rangle$ and $|\psi_2\rangle$ is the tensor product (or Kronecker product) of the two states: $|\psi_1\rangle \otimes |\psi_2\rangle$. For reference, the Kronecker product of two matrices $A = (a_{i,j})_{i,j}$ and B is the block matrix $A \otimes B = (a_{i,j}B)_{i,j}$.

For any linear map (matrix) A, we denote $A^\dagger$, its dagger, as the conjugate transpose of A. A linear map that is invariant by dagger (i.e. $H^\dagger = H$) is called Hermitian. The dagger of a quantum state $|\psi\rangle$ has a special notation, called "bra": $\langle\psi| = |\psi\rangle^\dagger$. Any linear map $\mathbb{C}^{2^n} \to \mathbb{C}^{2^m}$ can be written as a linear combination of compositions of bras and kets: $A = \sum_{k\in\{0,1\}^m, q\in\{0,1\}^n} \alpha_{k,q} |k\rangle\langle q|$ where $|k\rangle\langle q|$ is the outer product of the two states (or equivalently the composition $|k\rangle \circ \langle q|$).

When it is not interacting with its environment, a quantum register evolves unitarily, i.e. there is a linear map U verifying $U^\dagger U = UU^\dagger = I$ whose application on the state describes its evolution. In quantum computing, such a unitary is often described using quantum circuits, consisting of compositions of primitive gates. This representation can provide a fine-grained resource analysis of the computation, especially if the primitive gates match the ability of the hardware.

Many algorithms assume as input a classical matrix, which we need to encode into a unitary, so as to quantumly process it: this unitary is referred to as *oracle*. The first main approach for this is called Hamiltonian simulation. Given a Hermitian matrix H and number τ, we want to build $e^{iH\tau}$, which turns out to be unitary. When the input matrix A is not Hermitian, it is customary to work with the larger matrix $H = \left(\begin{smallmatrix} 0 & A^\dagger \\ A & 0 \end{smallmatrix}\right)$, which is Hermitian. The second approach is called Block-Encoding (see e.g. [14, 10]). There, the goal is to create a unitary $U = \left(\begin{smallmatrix} A/\alpha & * \\ * & * \end{smallmatrix}\right)$ whose top left block is a rescaled version of the input matrix A. Since U is unitary, $\|U\| = 1$, which implies $\alpha \geqslant \|A\|$. The value α is linked to the probability of success of the block encoding, and hence affects the complexity of the algorithms that use it. In practice, one wants α as close to $\|A\|$ as possible.

Several approaches to perform Hamiltonian simulation and Block Encoding have been considered in the literature, with one of the simplest being, if A is sparse, to decompose it as a linear combination of 1-sparse matrices (see definitions below). In that case, the way we can access the values of the matrix is crucial to get efficient decompositions, and hence matrix encodings. For completeness, we give in Fig. 1 an implementation of $e^{iB\tau}$ for B a 1-sparse hermitian matrix; and in Fig. 2 a block encoding of B. The former is different from the one found in [12], as we used a form that is more closely related to the block-encoding one.

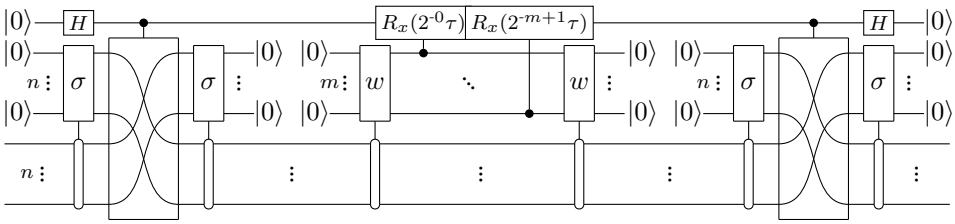


Figure 1: Implementation of $e^{iB\tau}$ when B is a n-qubit, 1-sparse hermitian and $\tau \geqslant 0$, using m bits of precision for the weights. σ is the (involutive) permutation induced by B and w is the weight function, i.e. such that $B|q\rangle = w(q)|\sigma(q)\rangle$. The block σ with controls represents the usual oracle $U_\sigma : |q\rangle|y\rangle \mapsto |q\rangle|y \oplus \sigma(q)\rangle$ where $\oplus$ is the bitwise XOR operation. If a boolean circuit is known for σ, then this oracle can be implemented as a reversible circuit with a comparable number of gates. The weights here are assumed to be in $[0,1]$ and use the unsigned fixed point format. This can easily be generalised to signed, or even complex values, provided oracles that implement respectively the absolute value and the argument functions.

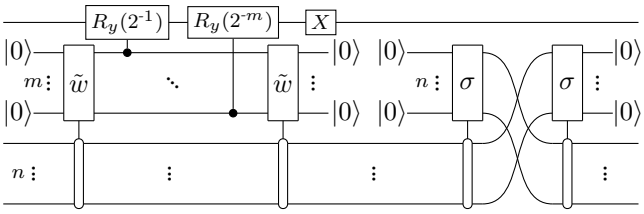


Figure 2: Implementation of a Block-Encoding for a n-qubit, 1-sparse matrix B, using m bits of precision for the weights. σ is the (involutive) permutation induced by B and w is the weight function, i.e. such that $B|q\rangle = w(q)|\sigma(q)\rangle$. We use here an oracle for $\tilde{w} : q \mapsto 2\arcsin(w(q))$, which is efficiently implementable if w is (see e.g. [20]). All initialised qubits are returned to the $|0\rangle$ state. The first qubit is to be pre- and post-selected in the $|0\rangle$ state to recover (a scaled) B. The weights here are assumed to use the unsigned fixed point format. Again, this can easily be generalised to signed, or even complex values, provided oracles that implement respectively the absolute value and the argument functions.

3. Efficient Sparse Matrices

In this paper, we are interested in the oracle implementation of matrices. We want these oracles to be efficient in some way, and, as discussed in Section 2, it turns out that often, efficiency of such oracles is related to the sparsity of the matrices. We say that a matrix $A \in \mathbb{C}^{2^n \times 2^n}$ is *sparse* if each row and column of A has at most $\text{poly}(n)$ non-zero entries, it is d-sparse if each row and column has at most d non-zero values. On the other hand, a function $f : \{0,1\}^n \to \{0,1\}^{n'}$ is *efficient* if f is $\text{poly}(n)$-time. In particular, a permutation $f : \{0,1\}^n \to \{0,1\}^n$ is *two-way efficient* if both the function and its inverse are $\text{poly}(n)$-time. Note that when we say that a function is efficient or that a matrix is sparse, we are implicitly referring to a *uniform family* of fixed instances.

The standard definition of sparse matrices presented above is however not enough to ensure we have efficient access to its non-zero entries. The literature [12, 16] often considers the following refinement:

Definition 3.1. A sparse matrix A is *d-row-computable* if there exists an efficient function $R_A : \{0,1\}^n \to (\{0,1\}^n \times \mathbb{C})^d$ such that:

1. R_A is a row-function of A, meaning that $\langle k| A = \sum_{(q,w)\in R_A(k)} w \langle q|$.
2. $R_A(k)[s] = (k, 0)$ if $s + 1$ exceeds the number of non-zero entries on row k.

We define similarly *d-column-computable* matrices. A matrix that is both d-row-computable and d'-column-computable is *(d, d')-row-column-computable*.

In other words, a d-row-computable matrix is one where there exists a function that associates each k with a list of the column indices of the non-zero entries of row k of A, together with the value itself. Moreover, although the list is always of size d, if a row k only has $d_k < d$ non-zero entries, the second condition ensures that the indices of the non-zero entries appear exactly once in the first d_k entries of the list. Notice that d and the precision m of the complex values are functions of n.

In this paper, we consider only row-column-computable sparse matrices.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 2 & 1 & 2 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 1 & 2 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & 2 & 1 & 2 & 0 \\ 0 & \cdots & 0 & 0 & 2 & 1 & 2 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(a) 3-sparse, constant values

$$\begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 & \cdots \\ 1 & 0 & 4 & 0 & 0 & 0 & \cdots \\ 0 & 3 & 0 & 6 & 0 & 0 & \cdots \\ 0 & 0 & 5 & 0 & 8 & 0 & \cdots \\ 0 & 0 & 0 & 7 & 0 & 10 & \cdots \\ 0 & 0 & 0 & 0 & 9 & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

(b) 2-sparse, all distinct

Table 1
Examples of n-qubit matrices

Example 3.2. Consider the examples in Table 1. If A is defined as in Table 1a, a possible R_A is as follows (successors and predecessors for index computation are taken modulo 2^n):

$$R_A(k) = \begin{cases} ((k,1),(k+1,0),(k-1,0)) & \text{if } k = 0 \text{ or } k = 2^n - 1 \\ ((k,1),(k+1,2),(k-1,2)) & \text{else} \end{cases}$$

There is no unicity: one can shuffle the tuples and split the cases at wish. For instance, another possible R_A is

$$R_A(k) = \begin{cases} ((k,1),(k+1,0),(k-1,0)) & \text{if } k = 0 \\ ((k-1,0),(k,1),(k+1,0)) & \text{if } k = 2^n - 1 \\ ((k,1),(k+1,2),(k-1,2)) & \text{if } 0 < k < 2^n - 1 \text{ and } k \text{ is even} \\ ((k+1,2),(k-1,2),(k,1)) & \text{else} \end{cases}$$

Consider now the matrix B defined in Table 1b. Unlike for the matrix of Table 1a, the coefficients on the bands are not all equal. A possible R_B is for instance

$$R_B(k) = \begin{cases} ((k+1,(2k+2)(2^n-1-k)/(2^n-1)),(k-1,(2k-1)(k/(2^n-1)) \\ \qquad \text{if } k = 0 \text{ or } k = 2^n - 1 \\ ((k-1,2k-1),(k+1,2k+2)) \\ \qquad \text{else} \end{cases}$$

The first approach proposed for the implementation was presented in the context of Hamiltonian simulation [12, 13]. The idea consists in decomposing an efficient Hermitian matrix into a sum of efficient 1-sparse Hermitian matrices using its row-function. This decomposition uses an edge-coloring technique to compute the row-function of each 1-sparse matrix. The exact form resulting from such a decomposition can be defined as follows:

Definition 3.3. A sparse matrix A is *d-row-ready-computable* if there exist efficient functions $R_A\colon \{0,1\}^n \to (\{0,1\}^n \times \mathbb{C})^d$ and $c_A\colon \{0,1\}^n \to (\{0,1\}^n)^d$ where

1. R_A is a row-function of A: $\langle k| A = \sum_{(q,w)\in R_A(k)} w \langle q|$.
2. $R_A(.)[s][0]$ is a permutation.
3. $c_A(.)[s]$ is the inverse of $R_A(.)[s][0]$, meaning that $c_A(R_A(k)[s][0])[s] = k$.
4. Indices in $R_A(k)$ are paired with non-zero weights at most once, meaning that if $R_A(k)[s][0] = R_A(k)[s'][0]$ and $R_A(k)[s][1] \neq 0$ and $R_A(k)[s'][1] \neq 0$ then we have $s = s'$.

We similarly define d-column-ready-computable matrices.

Remark 3.4. Notice that, by definition, a d-row-ready-computable form is trivially decomposable as a sum of d 1-sparse matrices, where each 1-sparse matrix is trivially decomposable as DP, where P an efficient unitary-permutation, and D an efficient diagonal.

Row- and column-ready-computable matrices are quasi-trivially the same:

Lemma 3.5. *Any d-row-ready-computable form of any sparse A can be reduced efficiently to a d-column-ready-computable form, and vice-versa.*

Proof. From (R_A, c_A, d), we construct efficiently for $s < d$, $C_A(q)[s] = (c_A(q)[s], R_A(c_A(q)[s])[s][1])$ and $r_A(k)[s] = R_A(k)[s][0]$ and vice versa. □

Lemma 3.6. *Any d-row-ready-computable form of any sparse A can be reduced efficiently to a (d,d)-row-column-computable one.*

Proof. After taking $R_A(k)[s] = R_A(k)[s']$ the $s+1$ non-zero entry in $R_A(k)$ else $(k,0)$ in $O(d)$ queries, this form is reduced efficiently to a d-row-computable one, and from Theorem 3.5, it can be reduced efficiently to a d-column-ready-computable one, after to a d-column-computable one, hence, we have efficiently a (d,d)-row-column-computable one. □

Definition 3.7. We call a form *Hermitian* if it represents a Hermitian matrix, and *hblock* if it represents a matrix in the form $\left(\begin{smallmatrix} 0 & A^\dagger \\ A & 0 \end{smallmatrix}\right)$, and *bipartite* if it represents a Hermitian matrix whose adjacency graph (i.e. $G_H = (\{0,1\}^n, E)$, where $(k,k') \in E$ iff $\langle k| H |k'\rangle \neq 0$) is bipartite.

Remark 3.8. An hblock form is bipartite where the first partite consists of vertices smaller than 2^{n-1}, and the second partite consists of the remaining elements.

A reduction to ready-computable forms has been presented in [12, 13], followed by an improvement for the bipartite case by [16].

Theorem 3.9 ([12, 13])**.** *Any d-row-computable hermitian form of any sparse Hermitian H can be reduced efficiently to a $d^2 \log_2^*(n)$-row-ready-computable one.*

Remark 3.10. The number of terms in [13] is given as $6d^2$, by arguing that for any reasonable size, $\log_2^*(n) \leq 6$. Taking the actual complexity into account gives the statement above.

Theorem 3.11 ([16])**.** *Any d-row-computable bipartite form of any sparse Hermitian H with an efficient 2-vertex-coloring $\delta_H : \{0,1\}^n \to \{0,1\}$ can be reduced efficiently to a d^2-row-ready-computable one.*

Remark 3.12. Any Hermitian matrix H can be simulated using Theorem 3.11 even if it is not bipartite. Indeed, $X \otimes H$ is a bipartite form and can easily be simulated since $e^{iX\otimes H\tau} |+\rangle |\psi\rangle = |+\rangle e^{iH\tau} |\psi\rangle$, with $|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}$.

Corollary 3.13. *Any (d,d')-row-column-computable form of any sparse A can be reduced efficiently to a $(d+d')^2 \log_2^*(n)$-row-ready-computable one.*

4. Native Computable Matrices

In this section, we consider another natural way to define a sparse matrix with efficient access, for whom we will study the decomposition and synthesis problem. Consider the form presented in Definition 3.1. In many cases, as was shown for instance in Example 3.2, the matrix A is very regular and the function R_A can be defined using a simple case-distinction analysis.

The definition of this class of matrices is based on the notion of efficiently computable. A set $X \subseteq \{0,1\}^n$ is *efficient* if there exists an efficient membership function $f_X : \{0,1\}^n \to \{0,1\}$ such that: $f_X(k) = 1$ if $k \in X$, and 0 otherwise.

Definition 4.1. A sparse matrix A is (d,t)*-row-native-computable* if there exists a list $\langle \mathbf{R}_s \rangle_{0\leqslant s<d}$ of lists $\mathbf{R}_s = \langle (X_{s,v}, p_{s,v}, w_{s,v}) \rangle_{0\leqslant v<t}$, where $X_{s,v} \subseteq \{0,1\}^n$ an efficient set, $p_{s,v} : \{0,1\}^n \to \{0,1\}^n$ a two-way efficient permutation, and $w_{s,v} : \{0,1\}^n \to \mathbb{C}$ an efficient weight-function, verifying:

1. For all s, the list $\langle X_{s,v} \rangle_v$ form a partition of $\{0,1\}^n$ (allowing empty parts). That is, for all s, $\biguplus_v X_{s,v} = \{0,1\}^n$.
2. No k can both belong to different indexed sets and be mapped to the same index with non-zero weight through the corresponding permutations and weight-functions: meaning that if $k \in X_{s,v} \cap X_{s',v'}$ and $p_{s,v}(k) = p_{s',v'}(k)$ and $w_{s,v}(k) \neq 0$ and $w_{s',v'}(k) \neq 0$ then we have $s = s'$.

such that:

$$R_A(k)[s] = \begin{cases} (p_{s,0}(k), w_{s,0}(k)) & \text{if } k \in X_{s,0} \\ \vdots & \\ (p_{s,t-1}(k), w_{s,t-1}(k)) & \text{else} \end{cases}$$

is an efficient row-function of A.

We say that it is *ready* if moreover, $\forall s : \biguplus_v p_{s,v}(X_{s,v}) = \{0,1\}^n$, and we similarly define (ready) d-column-native-computable matrices.

Example 4.2. Any poly(n)-sized dictionary as described in [19] is $(\text{poly}(n), 2)$-native-computable form with constant weight-functions.

Remark 4.3. A poly(n)-sized dictionary framework is more restrictive than the usual row-column-computable one. For example, if the matrix in Table 1a (when in form $(3,2)$-row-native-computable) can be captured with a poly(n)-sized dictionary, the matrix in Table 1b cannot (although it is $(2,2)$-row-native-computable).

Lemma 4.4. *Any (d,t)-row-native-computable form of any sparse A can be reduced efficiently to a $(dt,2)$-column-native-computable one, and vice-versa.* □

We can easily obtain a row-ready form from a row-native one:

Lemma 4.5. *Any (d,t)-row-native-computable form of any sparse A can be reduced efficiently to a dt-row-ready-computable one.*

Proof. From $(\langle \mathbf{R}_s \rangle, d, t)$, we construct efficiently for $s < d, v < t$: $R_A(k)[st+v] = (p_{s,v}(k), w_{s,v}(k))$ if $k \in X_{s,v}$ else $(p_{s,v}(k), 0)$, and $c_A(q)[st+v] = p_{s,v}^{-1}(q)$. □

One may always turn a row-column-computable form into a native one. This shows that the notion of native-computability is not more restrictive than the usual row-column-computable one:

Lemma 4.6. *Any (d,d')-row-column-computable form of any sparse A can be reduced efficiently to a $((d+d')^2 \log_2^*(n), 1)$-row-native-computable one.*

Proof. From (R_A, d, C_A, d') and using Theorem 3.13, we can efficiently construct a $(d+d')^2 \log_2^*(n)$-row-ready-computable form for A. Since row-ready-computable forms are trivially row-native-computable, we have then efficiently constructed a $((d+d')^2 \log_2^*(n), 1)$-row-native-computable form for A. □

In the case of a bipartite form with an efficient 2-vertex-coloring, we may even get better bounds:

Lemma 4.7. *Any d-row-computable bipartite form of any sparse H with an efficient 2-vertex-coloring $\delta_H : \{0,1\}^n \to \{0,1\}$ can be reduced efficiently to a $(d, 2d+1)$-row-native-computable one.*

Proof. From (R_H, d), we construct efficiently $\langle \mathbf{R}_s \rangle_{0 \leqslant s < d}$. For $s < d, v < 2d$:

$$X_{s,v} = \left\{ k \in \{0,1\}^n \;\middle|\; \begin{array}{l} \delta_H(k) = (v < d) \\ R_H(R_H(k)[s][0])[v \bmod d][0] = k \\ R_H(k)[s][1] \neq 0 \end{array} \right\}$$

i.e. k is accepted if it lies in the first, second partite resp. makes a round trip to the second, first partite resp. via s, and returns via $v \pmod d$. By this definition, all sets indexed by the same s are pairwise disjoint.

$$p_{s,v}(k) = \begin{cases} R_H(k)[s][0] & \text{if } k \in X_{s,v} \\ R_H(k)[v \bmod d][0] & \text{else if } k \in X_{v \bmod d, s+(v<d)d} \\ k & \text{else} \end{cases}$$

And, $w_{s,v}(k) = R_H(k)[s][1]$. Finally, we add a set to complete the partition for $s < d$: $X_{s,2d} = \{0,1\}^n - \bigcup_v^{2d-1} X_{s,v}$, $p_{s,2d} = id$, $w_{s,2d} = 0$. □

Given the relationship between the proofs of Theorem 4.7 and Theorem 3.11, we can apply the similar idea from Theorem 3.9 to further reduce the bound in Theorem 4.6. The idea behind these proofs is to show that any A admits at least one form in the native-computable form. However, in practice, it is more interesting to consider cases where the form of A satisfies the native-computable form trivially.

5. Oracle Complexity

In this section, we compare the complexity of implementing a (d, t)-column-native-computable hblock form of H using three different approaches:

- Reducing it to a ready form using Theorem 4.5.
- Reducing it to a ready form using Theorems 3.9 and 3.11 of [13, 16].
- Implementing it using the Sparse-Access Input Model (SAIM) oracle of [10].

The oracles require unitary subroutines for p, p^{-1}, w, and of the membership test for X. We write ctrl-$U_{G_{\max,\max}}$ for the cost of the (controlled version of the) subroutine G, and identify Toffoli gates and Rotations with their costs.

So given this, the final complexity in the Block-Encoding context is:

	Theorem 3.11	SAIM oracle [10]	Theorem 4.5
total qubits	$O(n+m+\log(d))$	$O(d(n+m))$	$O(n+m+\log(d))$
circuit complexity	$O(\underline{d^2}d^2tU_*)$	$O(\underline{d}dtU_*)$	$O(\underline{dt}dtU_*)$

where n and m are the sizes of the registers respectively for the indices of the matrix and for the precision of the weights, and where U_* is:

$$O\left(\begin{array}{c}\log(d)\text{ Toffoli} + \text{ctrl-}U_{\in X_{\max,\max}} + \text{ctrl-}U_{p_{\max,\max}} \\ + \text{ctrl-}U_{w_{\max,\max}} + \frac{1}{t}\text{ ctrl-Rotation}\end{array}\right).$$

The part that is **not** underlined in the complexities above represents the cost of a block encoding without accounting for the probability of success. Taking this into account introduces an overhead, underlined in the complexities.

Corollary 5.1. *Implementing a $(d, O(1))$-column-native-computable hblock form using Theorem 4.5 uses fewer qubits than [10], and has the same efficiency.* □

The final complexity for Hamiltonian simulation using the Product Formula (i.e. Trotterization [9]) is as follows:

	Theorem 3.11	Theorem 4.5
total qubits	$O(n+m)$	$O(n+m)$
circuit complexity	$O(\underline{d^4}d^2tU'_*)$	$O(\underline{d^2t^2}dtU'_*)$

where $U'_* = O(\text{ctrl-}U_{\in X_{\max,\max}} + \text{ctrl-}U_{p_{\max,\max}} + \text{ctrl-}U_{w_{\max,\max}} + \frac{1}{t}\text{ ctrl-Rotation})$. The part that is not underlined represents the cost of a single step in the formula, while the number of iterations required by the full Product Formula is underlined.

Corollary 5.2. *Theorem 4.5 is more efficient than Theorem 3.11 for a single Product Formula step.* □

Indeed, for a single step, Theorem 4.5 has complexity $O(dtU'_*)$ as compared to $O(d^2tU'_*)$. The number of steps that is required is however different ($O(d^2t^2)$ vs $O(d^4)$). It is however clear that the asymptotic complexity of Theorem 4.5 is better when $t = o(d\sqrt{d})$.

This complexity analysis shows when the simple implementation may be preferable in practice to the algorithm from the literature.

6. Clique Cover Approximation for Optimization

Reducing efficiently a d-column-ready-computable form of A to a $\tilde{d}$-column-ready-computable one such that: $\tilde{d} < d$, may be viewed as an optimization step, since the upper-bound sparsity d plays a role in the complexity of the quantum implementation [13, 10]. For example, reducing d in the context of Block-Encoding reduces the scaling factor on which the complexity depends.

Definition 6.1. A (d,t)-row-native-computable form is *completed* if for all s: $w_{s,t-1} = 0$, and $0 \notin w_{s,v}(X_{s,v})$ for all $v < t-1$.

Remark 6.2. Any (d,t)-row-native-computable form of any sparse matrix A can be reduced efficiently to a completed $(d,t+1)$-row-native-computable one by splitting each set into a subset with non-zero weights and another with zero weights, and then combining all non-zero weight sets into a new set with a zero weight function, and an identity permutation.

In this paper, we consider a completed (d,t)-column-native-computable form and aim to reduce it efficiently to a column-ready-computable one for implementation. From Theorem 4.5, we already have efficiently a dt-column-ready-computable one. We call this one the trivial case. The question now is whether we can push this bound further, to a $\tilde{d} < dt$.

Before proceeding, we may wonder about the hardness of obtaining such a form with the optimal upper-bound.

A d-column-ready-computable form of A is optimal if A has no d'-column-ready-computable form with $d' < d$.

Lemma 6.3. *Reducing efficiently any d-column-ready-computable form of any sparse matrix A to an optimal d_+-column-ready-computable one implies P = NP.*

Proof. Let X be an efficient set. Let's construct efficiently this 1-column-ready-computable form (C_A, r_A): $C_A(k) = (k,1)$ if $k \in X$ else $(k,0)$, and $r_A = id$.
Finding the optimal form here means: deciding if this form or the trivial form (i.e. no function required) is the optimal one. Thus, if we can reduce efficiently any d-column-ready-computable form of any sparse A to an optimal d_+-column-ready-computable one, we can decide efficiently if $X \neq \emptyset$, which is known to be NP-hard (Circuit-SAT Problem). □

Remark 6.4. The counter-example in this proof can also be used to prove hardness for all other forms.

Given the general hardness of this problem, heuristic optimization methods may be used to efficiently derive a better form than the trivial one in Theorem 4.5.

6.1. The Permutation Packing Problem

One way to achieve this is to stack together permutations, when possible. Indeed, notice that if we have $s, s' < d$ and $v, v' < t-1$, with $X_{s,v} \cap X_{s',v'} = \emptyset$ and $p_{s,v}(X_{s,v}) \cap p_{s',v'}(X_{s',v'}) = \emptyset$, such that $p_{s,v}$ and $p_{s',v'}$ are two efficient permutations on $\{0,1\}^n$, then we can build an efficient permutation $p_{s,v} \oplus p_{s',v'}$ on $\{0,1\}^{n+1}$ as described by:

$$q \mapsto \begin{cases} p_{s,v}(q) + 2^n & \text{if } q < 2^n \wedge q \in X_{s,c} \\ p_{s',v'}(q) + 2^n & \text{else if } q < 2^n \wedge q \in X_{s',c'} \\ p^{-1}_{s,v}(q - 2^n) & \text{else if } q \geqslant 2^n \wedge q \in X_{s,c} \\ p^{-1}_{s',v'}(q - 2^n) & \text{else if } q \geqslant 2^n \wedge q \in X_{s',c'} \\ q & \text{else} \end{cases}$$

Importantly, this generalises to more than 2 permutations with the same domain and codomain $\{0,1\}^{n+1}$, as long as the $X_{s,v}$ are all pairwise disjoint, as well as their image through the $p_{s,v}$. We call these permutations compatible. In the native form, we can then replace compatible permutations by this larger one.

In the ideal case, we can verify the disjointness of the above sets efficiently. We call this case *well-behaved*. Notice how the last constraint comes from the definition of native-computable forms.

Definition 6.5. A list $\langle (X_s, p_s) \rangle_{0 \leqslant s < n}$ of pairs of efficient sets $X_s \subseteq \{0,1\}^n$ and permutations $p_s : \{0,1\}^n \to \{0,1\}^n$ is *well-behaved* if $X_s \cap X_{s'} = \emptyset$ and $p_s(X_s) \cap p_{s'}(X_{s'}) = \emptyset$ can be decided efficiently, and if $k \in X_s \cap X_{s'}$ and $p_s(k) = p_{s'}(k)$ then we have $s = s'$.

In that case, we call the Permutation Packing Problem, the problem of finding the minimum number of larger permutations required to subsume the permutations p_s. Notice that this problem only tries to group certain permutations together. While reducing d may not in general reduce the total quantum-circuit complexity, since the other terms may increase, in the specific case of permutation packing, we incur no additional cost when performing the reduction. The larger permutation built with the technique above will be done in postprocessing step. It turns out that we can relate this problem to a well-known graph problem, the Clique Cover Problem in [21].

6.2. Optimization by Clique Cover

We now exhibit a link between the problem presented in Section 6.1 and the Clique Cover Problem, consisting in finding a minimum clique cover for a graph $G = (V, E)$. A clique is a subset of its vertices $c \subseteq V$ in which all vertices are connected, i.e. $\forall u, v \in c,\ u \neq v \implies (u, v) \in E$, and a clique cover of G is a partition of the vertices V of G, into cliques. It is well known that this problem is **NP-hard** [21].

Proposition 6.6. *The Permutation Packing Problem is polynomially equivalent to the Clique Cover Problem, it is hence NP-hard.*

Proof. We show the result by exhibiting a polynomial reduction for each problem to the other.

• Permutation Packing reduces to Clique Cover: Let's build a graph $G = (V, E)$ from $S = \langle (X_s, p_s) \rangle_{0 \leqslant s < n}$. Each vertex corresponds to a permutation p_s, and each pair of vertices, representing p_s and $p_{s'}$, share an edge iff $X_s \cap X_{s'} = \emptyset$ and $p_s(X_s) \cap p_{s'}(X_{s'}) = \emptyset$. This construction is polynomial, for there are n permutations, and the set is well-behaved. A clique in G then corresponds to a compatible subset of S. Conversely, any compatible subset of S yields a clique in G. It is then direct to see that a solution to the Clique Cover problem corresponds to a solution to the Permutation Packing problem.

• Clique Cover reduces to Permutation Packing: We give a construction for $n \geqslant 12$ so that $\binom{n}{2} \leqslant 2^n$ and $3\lceil \log_2(n) \rceil \leqslant n$. Let $G = (V, E)$ be a simple graph with n vertices. Let's build a well-behaved set of partial injections out of it. Consider $\bar{G} := (V, \bar{E})$ the complement of G (i.e. the graph with the same vertices and exactly the edges that are not in the original graph). Give a distinct number between 1 and $\binom{n}{2}$ to all the edges of $\bar{G}$. Assign to each vertex v in V the set X_v of numbers associated to the edges it is connected to in $\bar{G}$. Notice that for every pair of connected vertices (u, v) in G, their associated sets X_u and X_v are disjoint. Let $\ell = \lceil \log_2 \binom{n}{2} \rceil$, i.e. the number of bits to write all edge values. For each vertex v, we build a permutation p_v that satisfies the last constraint of well-behaved lists, as follows: $p_v(k) = 2^\ell v + k$ if $k \in X_v$, and k' else if $\exists k' \in X_v : k = 2^\ell v + k'$, and k otherwise. Notice that $p_v(X_v) = 2^\ell v + X_v \subseteq [\![2^\ell v, 2^\ell (v+1) - 1]\!]$. Hence if $u \neq v$, then $p_v(X_v) \cap p_u(X_u) = \emptyset$. It is then clear that if $u \neq v$, then there is no x such that $x \in X_u \cap X_v$ and $p_u(x) = p_v(x)$. We hence have a well-behaved set. Again, a solution to the Permutation Packing problem exactly corresponds to a solution to the Clique Cover problem. □

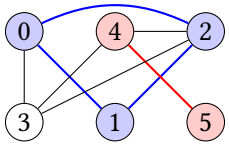
While the theorem establishes an equivalence between the two problems, we are only interested in the first reduction, for optimization purposes.

The above result shows that even in the well-behaved case, the optimization problem is a hard one. However, we can now use this correspondence, and potential heuristics for clique cover finding, to optimize our permutations.

Example 6.7. Suppose we have the following list for $n > 4$:

$$\left\langle \begin{array}{c} ([\![0, 2^{n-1} - 1]\!], q \mapsto q + 1 \bmod 2^n)\ ,\ ([\![2^{n-1}, 2^n - 4]\!], q \mapsto q + 2 \bmod 2^n) \\ (\{2^n - 3\}, q \mapsto q + 3 \bmod 2^n)\ ,\ (\{100, 110\} \times \{0,1\}^{n-3}, \oplus_2) \\ (\{001, 011\} \times \{0,1\}^{n-3}, \oplus_0 \oplus_2)\ ,\ ([\![2^{n-1}+1, 2^n - 1]\!], q \mapsto -q \bmod 2^n) \end{array} \right\rangle$$

where $\oplus_i$ it the function that flips the bit at index i (with the leftmost bit being at index 0). The associated graph is built as follows:



with a numbering that follows the order of the permutations in the set. In this example, we can cover the graph with three cliques represented with three different colors. Hence, we can pack permutations 0, 1 and 2 together, permutations 4 and 5 together, and leave permutation 3 alone.

Example 6.8. To illustrate the relevance of the clique-based approximation, we may look at the following large scale, artificial example.

We consider a $(n, 2)$-column-native-computable form, defined by an irrational number $\sqrt{\eta}$ where $\eta \in \mathbb{N}$, approximated to $n^2 + n$ bits. Let's denote $\sqrt{\eta}[a, b]$ the bits from position a to b after the dot in $\sqrt{\eta}$, $[\![a, b[\![$ the usual number interval when $a < b$, and $[\![a, 2^n[\![\cup [\![0, b[\![$ else. We then define for $s < n$:

$$X_{s,0} = [\![\sqrt{\eta}[sn, (s+1)n - 1], \sqrt{\eta}[(s+1)n, (s+2)n - 1][\![,$$

$p_{s,0}(q) = q + s \pmod{2^n}$, $w_{s,0} = 1$, and $X_{s,1}$ the set completing the partition.

We built the sets and permutations using $\eta = 2026$, built the corresponding graph as explained above, and ran a simple heuristic clique finding algorithm to optimize these permutations. We ran the algorithm for various values of n, in each case giving $d = n$ permutations. The resulting numbers of compatible sets $\tilde{d}$ found for each value n are shown in the table:

$d = n$	4	64	256	512	$\cdots$	2048	4096	$\cdots$
$\tilde{d}$	3	34	138	275	$\cdots$	1101	2151	$\cdots$

In the general case, we may not have a well-behaved set to start with. In that case we may still use the above result, by simply omitting edges between vertices for which we can't establish efficiently the disjointness of the corresponding sets. A clique cover in the resulting graph then still produces a valid optimization. This is only a heuristic optimization technique: finding the optimal clique cover does not imply finding the optimal form.

For a compatible set of permutations $\langle (X_s, p_s) \rangle_s$ with corresponding weights $\langle w_s \rangle_s$ (corresponding to a clique in the corresponding graph), we may then build the 1-sparse matrix A with function:

$$C_A(q) = \begin{cases} (p_s(q) + 2^n, w_s(q)) & \text{if } q < 2^n \text{ and } q \in X_s \\ (p_s^{-1}(q - 2^n), w_s^\dagger(p_s^{-1}(q - 2^n))) & \text{else if } q \geqslant 2^n \text{ and } q \in p_s(X_s) \\ (q, 0) & \text{else} \end{cases}$$

7. Conclusion

In this paper, we presented the notion of native-computable sparse matrices: sparse matrices defined with a simple case-distinction analysis. We presented their structure and properties, showed that they are as expressive as row-column-computable ones (Lemmas 4.6 and 4.7), and discussed how they can give circuit implementations competitive with the state of the art. We finally analyzed how they are amenable to optimization, focusing on the relationship with the clique cover problem.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

Acknowledgments

This work has been partially funded by the European Union through the MSCA SE project QCOMICAL (Grant Agreement ID: 101182520), by the French National Research Agency (ANR): project TaQC ANR-22-CE47-0012, within the framework of "Plan France 2030", under the research projects EPIQ ANR-22-PETQ-0007 and HQI-R&D ANR-22-PNCQ-0002, and by BPI France I-Demo with the project AeroQat, contract 202400143.

References

[1] M. A. Nielsen, I. L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press, 2002.

[2] J. Watrous, Quantum computational complexity, in: Encyclopedia of Complexity and Systems Science, Springer New York, 2009, p. 7174–7201. doi:10.1007/978-0-387-30440-3_428.

[3] V. Gheorghiu, M. Mosca, Quantum resource estimation for large scale quantum algorithms, Future Generation Computer Systems 162 (2025) 107480. doi:10.1016/j.future.2024.107480.

[4] M. E. Morales, L. Pira, P. Schleich, K. Koor, P. C. Costa, D. An, A. Aspuru-Guzik, L. Lin, P. Rebentrost, D. W. Berry, Quantum linear system solvers: A survey of algorithms and applications, 2026. doi:10.1103/x6gh-d8gh.

[5] A. Prakash, Quantum Algorithms for Linear Algebra and Machine Learning, Ph.D. thesis, EECS Department, University of California, Berkeley, 2014. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-211.html, technical Report No. UCB/EECS-2014-211.

[6] I. Kassal, J. D. Whitfield, A. Perdomo-Ortiz, M.-H. Yung, A. Aspuru-Guzik, Simulating chemistry using quantum computers, Annual Review of Physical Chemistry 62 (2011) 185–207. doi:10.1146/annurev-physchem-032210-103512.

[7] A. W. Harrow, A. Hassidim, S. Lloyd, Quantum algorithm for linear systems of equations, Physical Review Letters 103 (2009) 150502. doi:10.1103/PhysRevLett.103.150502.

[8] J. M. Martyn, Z. M. Rossi, A. K. Tan, I. L. Chuang, Grand unification of quantum algorithms, PRX Quantum 2 (2021). doi:10.1103/prxquantum.2.040203.

[9] N. Hatano, M. Suzuki, Finding exponential product formulas of higher orders, in: Quantum Annealing and Other Optimization Methods, Springer Berlin Heidelberg, 2005, p. 37–68. doi:10.1007/11526216_2.

[10] A. Gilyén, Y. Su, G. H. Low, N. Wiebe, Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics, in: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC '19, ACM, 2019, p. 193–204. doi:10.1145/3313276.3316366.

[11] D. Aharonov, A. Ta-Shma, Adiabatic quantum state generation and statistical zero knowledge, in: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, STOC'03, ACM, 2003, p. 20–29. doi:10.1145/780542.780546.

[12] G. Ahokas, Improved Algorithms for Approximate Quantum Fourier Transforms and Sparse Hamiltonian Simulations, Master's thesis, University of Calgary, 2004. URL: http://hdl.handle.net/1880/41417.

[13] D. Berry, G. Ahokas, R. Cleve, B. Sanders, Efficient quantum algorithms for simulating sparse Hamiltonians, Communications in Mathematical Physics 270 (2006) 359–371. doi:10.1007/s00220-006-0150-x.

[14] D. Camps, L. Lin, R. Van Beeumen, C. Yang, Explicit quantum circuits for block encodings of certain sparse matrices, SIAM Journal on Matrix Analysis and Applications 45 (2024) 801–827. doi:10.1137/22m1484298.

[15] C. Sünderhauf, E. Campbell, J. Camps, Block-encoding structured matrices for data input in quantum computing, Quantum 8 (2024) 1226. doi:10.22331/q-2024-01-11-1226.

[16] D. Berry, A. Childs, R. Cleve, R. Kothari, R. Somma, Exponential improvement in precision for

simulating sparse Hamiltonians, in: Proceedings of the forty-sixth annual ACM symposium on Theory of computing, STOC'14, ACM, 2014, p. 283–292. doi:10.1145/2591796.2591854.

[17] D. Berry, A. Childs, R. Kothari, Hamiltonian simulation with nearly optimal dependence on all parameters, in: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, IEEE, 2015, p. 792–809. doi:10.1109/focs.2015.54.

[18] A. Suau, G. Staffelbach, H. Calandra, Practical quantum computing: Solving the wave equation using a quantum approach, ACM Transactions on Quantum Computing 2 (2021) 1–35. doi:10.1145/3430030.

[19] C. Yang, Z. Li, H. Yao, Z. Fan, G. Zhang, J. Liu, Dictionary-based block encoding of sparse matrices with low subnormalization and circuit depth, Quantum 9 (2025) 1805. doi:10.22331/q-2025-07-22-1805.

[20] I. Burge, M. Barbeau, J. Garcia-Alfaro, Quantum cordic–arcsin on a budget, arXiv preprint arXiv:2411.14434 (2024).

[21] R. M. Karp, Reducibility among combinatorial problems, in: Complexity of Computer Computations, Springer US, 1972, p. 85–103. doi:10.1007/978-1-4684-2001-2_9.