Quantum Circuit Design for Finding k-Cliques via Quantum Amplitude Amplification Strategies

Simone Perriello
simone.perriello@polimi.it
Department of Electronics, Information and Bioengineering (DEIB) — Politecnico di Milano
Milano, Italy

Abstract

The k-clique problem, which involves identifying complete subgraphs of size k within a graph, is a fundamental challenge in combinatorial optimization with applications in network analysis, bioinformatics, and cryptography. As the number of nodes n grows, classical algorithms become computationally intractable, motivating the exploration of quantum computing to address these limitations. This work introduces two quantum algorithms for solving the k-clique problem: one based on Quantum Amplitude Amplification (QAA) and another leveraging Quantum Walks (QW) over the Johnson graph $J(n,k)$. By exploiting the regular structure of the Johnson graph, the QW approach achieves the same quadratic speedup of the QAA approach over classical algorithms, while remaining applicable to arbitrary undirected, unweighted graphs. We provide detailed quantum circuit designs for both approaches, analyzing their performance in terms of qubit count, gate count, and circuit depth under the NOT-CNOT-Toffoli + arbitrary rotations and Clifford + T gate sets. Compared to state-of-the-art quantum algorithms, our methods achieve significant improvements in scaling, particularly for dense graphs, with speedups ranging from 2^5 to 2^{20}.

CCS Concepts

• **Theory of computation** → **Design and analysis of algorithms**; • **Mathematics of computing** → *Graph algorithms*; • **Computer systems organization** → **Quantum computing**.

Keywords

Quantum walks, Quantum amplitude amplification, Johnson graph, Grover algorithm, k-clique

ACM Reference Format:
Simone Perriello. 2025. Quantum Circuit Design for Finding k-Cliques via Quantum Amplitude Amplification Strategies . In *22nd ACM International Conference on Computing Frontiers (CF '25), May 28–30, 2025, Cagliari, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3719276.3725200

1 Introduction

The clique problem involves identifying cliques, that is, subsets of vertices that form complete subgraph, in a graph $G = (\mathcal{V}, \mathcal{E})$. This problem can take various forms, such as finding a maximum clique (a clique with the largest number of vertices), determining the existence of a clique of a given size k (k-clique), or enumerating all maximal cliques (cliques that cannot be extended by adding more vertices). The clique problem has wide-ranging applications across various domains. In bioinformatics, clique detection is employed to analyse protein-protein interaction networks [11]. In network analysis, cliques are used to identify tightly-knit communities or highly connected subgraphs [13]. In coding theory, cliques aid in constructing efficient error-correcting codes by identifying subsets of codewords having a minimum distance [12].

The decision version of the clique problem is computationally challenging, being NP-complete [15]. Notably, the k-clique problem was included in Karp's seminal list of 21 NP-complete problems [22]. Classical approaches to the clique problem face therefore exponential complexity for large graphs, making them infeasible for practical applications. This motivates the exploration of quantum algorithms as a potential path to accelerate the solution finding of such problem. In this respect, techniques based on *Quantum Amplitude Amplification (QAA)* [6] were shown to provide quadratic speedups for search problems. Additionally, when compared to the plain QAA approach, search approaches based on *Quantum Walks (QW)* can achieve better performances for specific categories of problems both in the asymptotic [1, 2, 33] and in the finite-regime [24].

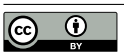
Related Works. Clique-related problems, including k-clique, triangle finding, and maximum clique, have been extensively studied in both classical and quantum computing contexts. In the quantum domain, two primary approaches have been explored: quantum annealing methods and gate-based circuit implementations.

Quantum annealing heuristics, such as those in [8, 9], solve clique problems by mapping them to QUBO formulations. While these methods show promise for certain graph families, their scalability remains constrained, with no guaranteed quantum speedups.

In the gate-based paradigm, [4] proposed a Grover-based algorithm for the maximum clique problem, achieving a theoretical complexity of $O\left(|\mathcal{V}|\sqrt{2^{|\mathcal{V}|}}\right)$. The work was further extended in [17], where a detailed quantum circuit with depth $O\left(|\mathcal{V}|^2\sqrt{2^{|\mathcal{V}|}}\right)$ and $O(|\mathcal{V}|^2)$ qubits was presented. The high-level circuit decomposition into primitive quantum gates was left for future work.

For k-clique problems, [27] proposed a quantum algorithm utilizing Dicke states and multi-controlled gates, achieving a depth of $O\left(\sqrt{\binom{n}{k}}(|\mathcal{V}|+|\mathcal{E}|)\right)$ and requiring $O\left(|\mathcal{V}|\log_2(k^2)\right)$ qubits.

Among fixed-size k-clique problems, triangle finding is particularly well-studied. Several works [7, 14, 26] have analysed the query complexity achieved when querying an oracle under a QAA scheme. The best-known result up to date [14] achieves a query complexity of $O(n^{5/4})$.



CF '25, Cagliari, Italy

ACM ISBN 979-8-4007-1528-0/2025/05
https://doi.org/10.1145/3719276.3725200

Classically, a naive brute-force approach for k-clique detection has a worst-case computational complexity of $O(n^k k^2)$, which becomes intractable for large k. Subsequent optimizations [35] reduced this complexity to $o(n^k)$ in time and $O(n^c)$ in space, where c is independent of k.

Our Contribution. This paper introduces two novel approaches to solving the k-clique problem, the first one relying on a pure QAA strategy, and the second one relying on a QW search strategy over a Johnson graph, which, to the best of our knowledge, has never been explored before. While exploiting the regular structure of the Johnson graph is essential for retaining the quadratic speedup of the quantum walk over classical methods, our approach is designed to detect k-cliques in arbitrary undirected, unweighted graphs. We provide detailed descriptions of the subcircuits, along with comprehensive analyses of their depth, qubit count (width), and gate count when relying on two distinct gate sets: the NOT-CNOT-Toffoli (NCT) reversible gate set, augmented with arbitrary rotation gates; and the Clifford + T, the most promising one for fault-tolerant computation. Our method achieves a circuit depth $\in O\left(\sqrt{\binom{n}{k}} n \log_2(n)\right)$ while requiring a number of qubits in $\in O\left(n \log_2(n)\right)$. This substantially improves scalability compared to the edge-based qubit scaling of [27], particularly for dense graphs. Finally, our circuit does not rely on Quantum Random-Access Memory (QRAM), whose theoretical feasibility remains controversial [20].

2 Background

Notation. We use calligraphic uppercase letters, such as $\mathcal{S}$, to represent sets; their cardinality is denoted as $|\mathcal{S}|$. $[x]$ represents the set of integers in the range $[0, x-1]$. Vectors are denoted by lowercase bold letters, such as $\boldsymbol{v}$. Matrices are denoted by uppercase bold letters, such as $\boldsymbol{M}$. For a vector $\boldsymbol{v}$, the element at index i is written as $\boldsymbol{v}[i]$, For a matrix $\boldsymbol{M}$, the element at row i and column j is denoted by $\boldsymbol{M}[i, j]$.

2.1 Quantum Computing

The fundamental unit of quantum information is the *qubit*, defined as a vector in the *Hilbert space* $\mathbb{C}^2$ [28]. Using Dirac notation, the state of a qubit is expressed as a column vector that forms a linear combination, or superposition, of the basis states $|0\rangle$ and $|1\rangle$. A system of n qubits is represented as a vector in a 2^n-dimensional Hilbert space. Each state is expressed as the linear combination: $|\psi\rangle = \sum_{i \in \{0,1\}^n} a_i |i\rangle$, where all the 2^n distinct $|i\rangle$ denote orthonormal basis vectors labelled by n-bit strings, and $a_i \in \mathbb{C}$ are complex amplitudes satisfying $\sum_i |a_i|^2 = 1$. A *measurement* collapses the superposition to a single basis state $|i\rangle$ with probability $|a_i|^2$.

The evolution of an n-qubit quantum system is governed by quantum operators, modelled as unitary matrices in $\mathbb{C}^{2^n \times 2^n}$, that preserve the norm of the state vector. A matrix $\boldsymbol{U}$ is unitary iff $\boldsymbol{U}\boldsymbol{U}^\dagger = \boldsymbol{I}$, with $\boldsymbol{U}^\dagger$ being the conjugate transpose of $\boldsymbol{U}$. The transformations obtained after applying a quantum operator to a state vector can be represented as matrix-vector multiplications, such as $|\psi_1\rangle = \boldsymbol{U}|\psi_0\rangle$. If the n qubits are grouped into k disjoint, consecutive subsets, we can associate a distinct operator $\boldsymbol{U}_i$, $i = \{1, \ldots, k\}$ to each subset. The overall operator acting on the quantum state is $(\boldsymbol{U}_1 \otimes \ldots \otimes \boldsymbol{U}_k)|\alpha\rangle$, where $\otimes$ denotes the Kronecker product. On the other hand, when a sequence of d quantum operators is applied, they are executed in right precedence order as $\boldsymbol{U}_d \ldots \boldsymbol{U}_1 |\psi\rangle$.

Quantum computations can be represented using *quantum circuits*, where qubits are depicted as horizontal lines, and operations on these qubits are visualized as *quantum gates*. Key quantum gates include: the H gate, implementing the *Hadamard* operator, preparing the uniform superposition $1/\sqrt{2}(|0\rangle + |1\rangle)$; the NOT gate (also referred to as X gate, and often represented by a $\oplus$ symbol), analogous to the classical NOT gate, which flips the state of a qubit from $|0\rangle$ to $|1\rangle$ and vice versa; the Z gate, often called the phase gate, which introduces a phase shift by flipping the sign of the amplitude associated with $|1\rangle$ to $-|1\rangle$; and the $\mathsf{R_Y}(\theta)$ gate, which performs a *rotation* around the y-axis of the Bloch sphere by an angle θ[28].

Following the computational model described in [21], where the quantum unit functions as a memory peripheral managed by a classical controller, we adopt a pseudocode notation to integrate quantum and classical operations. In this direction, we define a collection of qubits as a *quantum register* and use a notation similar to classical arrays, with underlining to emphasize their quantum behaviour, such as $\underline{\boldsymbol{v}}$ for the register and $\underline{\boldsymbol{v}}[i]$ for an individual qubit.

The application of quantum gates to registers is expressed using function-style notation, like in $\mathtt{G}(\underline{\boldsymbol{a}})$, where gate G operates on the quantum register $\underline{\boldsymbol{a}}$. Conversely, classical routines are denoted using camelCase. For instance, $\mathtt{genCircuit}(\underline{\boldsymbol{a}}, a, \boldsymbol{A})$ refers to a classical routine that accepts a classical variable a and a classical vector $\boldsymbol{A}$ as inputs to generate a sequence of quantum gates to be applied to the register $\underline{\boldsymbol{a}}$.

Controlled gates, for example, perform conditional operations based on the states of *control qubits*. The CNOT gate applies an NOT operation to the target qubit only when the control qubit is in state $|1\rangle$. Similarly, the $\mathtt{C^2NOT}$ (also known as the *Toffoli* gate) acts on a target qubit conditioned on the states of two control qubits being $|11\rangle$. A generic gate G controlled by m qubits being in state $|1^m\rangle$ is represented as $\mathtt{C}^m\mathtt{G}$. In the circuit model, the control qubits are denoted by a black circle, while in the pseudocode, they are separated from the target qubits by the symbol $\big|$.

Fig. 1 illustrates a quantum evolution expressed through three representations: Fig. 1(a) the mathematical formulation using operators and bra-ket notation, Fig. 1(b) the equivalent quantum circuit, and Fig. 1(c) the corresponding pseudocode implementation.

Boolean masking. In our proposal, we extend the classical concept of Boolean masking to the quantum circuit model. Consider two quantum registers: a control register containing n qubits and a data register containing n elements, in which each element can be encoded using one or more qubits. Let G be a quantum operator that acts on a single data element. The task is to conditionally apply G to the i-th element of the data register based on the state of the i-th qubit in the control register. Specifically, if the i-th qubit of the control register is in the state $|1\rangle$, then G is applied to the i-th element of the data register. This operation can be expressed as a controlled gate CG, where each qubit of the control register acts as the control for the corresponding element in the data register.

Reflection operators. *Reflection operators* are quantum operators that perform a reflection of a quantum state through a specific subspace. We denote by $\boldsymbol{R}_\alpha$ the operator that reflects a quantum state around the subspace spanned by the state $|\alpha\rangle$. Its action is defined

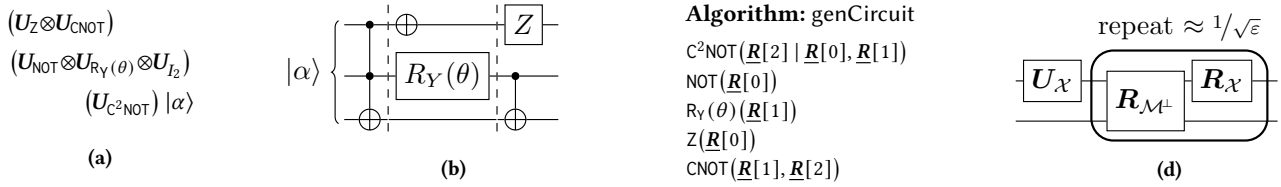


Figure 1: (a)–(c). Three conventions to represent a sequence of operations applied to a quantum state: (a) matrix-vector multiplications between the quantum operators (the unitaries $U_\cdot$, with I_4 denoting the identity matrix of size 4×4) and the quantum state ($|\alpha\rangle$); (b) gate-based circuit, with the dashed lines dividing layers composed of gates that can be applied in parallel; (c) pseudocode description. (d) High-level representation of the circuit associated to the Quantum Amplitude Amplification framework, with the $U_{\mathcal{X}}$ operator generating the input superposition of the basis states labelled as the bitstrings of the domain $\mathcal{X}$, the reflection operator $R_{\mathcal{M}^\perp}$ changing the amplitude of the bitstrings belonging to the marked set $\mathcal{M}$, and the reflection operator $R_{\mathcal{X}}$ performing a reflection around the input superposition. ε denotes the ratio $\mathcal{M}/\mathcal{X}$.

as: $\boldsymbol{R}_\alpha\,|\alpha\rangle = |\alpha\rangle$ and $\boldsymbol{R}_\alpha\,|\alpha^\perp\rangle = -\,|\alpha^\perp\rangle$, where $|\alpha^\perp\rangle$ represents the state orthogonal to $|\alpha\rangle$. Similarly, $\boldsymbol{R}_{\alpha^\perp}$ denotes the operator that reflects around the subspace spanned by $|\alpha^\perp\rangle$.

Two reflection operators are of particular interest in the circuit model. The $\boldsymbol{R}_{1^\perp}$ operator inverts the sign of the amplitude associated with the basis state labelled by the all-ones bitstring $|1^m\rangle$. It corresponds to a $\mathtt{C}^m\mathtt{Z}$ gate, where all m qubits participate in a controlled-Z operation targeting the state $|1^m\rangle$. The $\boldsymbol{R}_{0^\perp}$ operator inverts the sign of the amplitude associated with the basis state labelled by the all-zeros bitstring $|0^m\rangle$. Its implementation involves a layer of NOT gates applied to each of the m qubits, followed by a $\mathtt{C}^m\mathtt{Z}$ gate, and finally another layer of NOT gates.

Quantum Phase Estimation (QPE) algorithm The algorithm is used to estimate the phase φ of an eigenvalue associated with a unitary operator $\boldsymbol{U}$. Given an eigenstate $|\psi\rangle$ of $\boldsymbol{U}$, where $\boldsymbol{U}\,|\psi\rangle = e^{2i\pi\varphi}\,|\psi\rangle$, the algorithm determines φ with high precision. QPE works by preparing a superposition of states using t qubits as phase registers and applying controlled $\boldsymbol{U}^{2^k}$ operations for $k = 0, \ldots, t-1$. A Quantum Fourier Transform [28] is then performed on the phase register to extract a binary representation of φ to t bits of precision.

2.2 Quantum Amplitude Amplification (QAA)

Quantum Amplitude Amplification (QAA), proposed in [6], is a generalization of Grover's algorithm [16]. Let $f : \mathcal{X} \mapsto \{0,1\}$ be a Boolean function, where $\mathcal{X} \subseteq \{0,1\}^n$, and define the set of marked elements as $\mathcal{M} = \{x \in \mathcal{X} \mid f(x)=1\}$. A classical probabilistic algorithm requires, on average, $1/\varepsilon$ evaluations of f to identify an element of $\mathcal{M}$, where $\varepsilon = |\mathcal{M}|/|\mathcal{X}|$ denotes the fraction of marked elements.

In contrast, when provided with two quantum operators, namely $\boldsymbol{U}_{\mathcal{X}}$, generating a uniform superposition of all the basis states labelled as bitstrings of $\mathcal{X}$, and $\boldsymbol{U}_f$, implementing f, QAA identifies a solution with probability close to 1 using approximately $1/\sqrt{\varepsilon}$ applications of $\boldsymbol{U}_f$, providing a quadratic improvement over classical approaches. In [5] the authors proved that the same complexity can be achieved even if the size of $\mathcal{M}$ is not known in advance by iteratively increasing the number of $\boldsymbol{U}_f$ applications and observing the outcome. The QAA algorithm, schematically illustrated in Fig. 1(d), is based on three stages.

1) Input Preparation ($\boldsymbol{U}_{\mathcal{X}}$). This operator initializes the quantum system into a uniform superposition of all basis states corresponding to the bitstrings in the function domain $\mathcal{X}$. The resulting state can be expressed as $|\chi\rangle = \frac{1}{\sqrt{|\mathcal{X}|}} \sum_{i \in \mathcal{X}} |i\rangle$.

2) Oracle ($\boldsymbol{R}_{\mathcal{M}^\perp}$). The oracle is a reflection operator that inverts the sign of the amplitudes associated with all the basis states labelled as bitstrings belonging to $\mathcal{M}$. When given the quantum operator $\boldsymbol{U}_f$, the oracle operator can be expressed as $\boldsymbol{U}_f^\dagger \boldsymbol{R}_{1^\perp} \boldsymbol{U}_f$.

3) Diffusion ($\boldsymbol{R}_{\mathcal{X}}$). This operator performs a reflection around the initial superposition state prepared by $\boldsymbol{U}_{\mathcal{X}}$. It can be expressed as $\boldsymbol{R}_{\mathcal{X}} = \boldsymbol{U}_{\mathcal{X}} \boldsymbol{R}_{0^\perp} \boldsymbol{U}_{\mathcal{X}}{}^\dagger$.

2.3 Quantum Walk (QW) search over Johnson Graphs

Quantum Walk (QW) search generalizes the QAA scheme by adapting the random walk search strategy to the quantum computing paradigm. A random walk on a graph $G = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ the set of vertices and $\mathcal{E}$ the set of edges, is a stochastic process characterized by probabilistic transitions between adjacent vertices. These probabilities are represented by a stochastic matrix $\boldsymbol{P}$ of size $|\mathcal{V}| \times |\mathcal{V}|$, where $\boldsymbol{P}[i,j]$ gives the probability of transitioning from vertex v_i to vertex v_j. The state of the walk at time step t is described by a probability vector $\boldsymbol{s}_t$ of size $|\mathcal{V}|$, where each entry represents the probability of being at a specific vertex.

When the next transition, or step, of the walk depends only on the current state, the process forms a *Markov chain* [29]. For stationary Markov chains, starting from any initial state, the system converges to a stationary distribution $\boldsymbol{s}_\pi$ after approximately $1/\delta$ steps, where δ is known as *eigenvalue gap* or *spectral gap* — the difference between the two largest eigenvalues of $\boldsymbol{P}$.

In random walk-based search algorithms, the domain of a Boolean function f is modeled as the vertices of G; that is, $f : \mathcal{V} \mapsto \{0,1\}$. A step on this graph corresponds to transitioning between elements of the domain. The structure of G and the resulting transition rules determine the efficiency of the walk. A promising choice is the use of a Johnson graph $J(n,k)$, which is an undirected and regular graph where each vertex represents a k-subset $\mathcal{S}$ of $\{1, \ldots, n\}$.

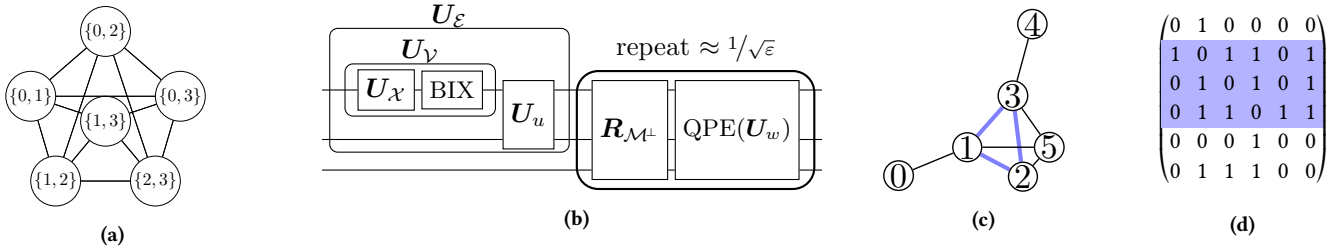


Figure 2: (a) An example of a Johnson graph $J(n = 4, k = 2)$. (b) High-level representation of the circuit associated to the QW search framework of [25], as implemented in [24]. With respect to Fig. 1(d), it relies on the additional BIX routine, used to generate a superposition of all the nodes, the update operator U_u, used to generate a superposition of edges, and the QPE routine applied to the walk operator U_w, replacing the QAA diffusion operator. (c) An example of an undirected, unweighted graph and (d) its associated adjacency matrix. The edges in blue belong to the 3-clique formed by the nodes $\{1, 2, 3\}$.

In this graph, two vertices are connected by an edge if their corresponding subsets differ by exactly one element. The Johnson graph has $|\mathcal{V}| = \binom{n}{k}$ vertices and a spectral gap $\delta = \frac{n}{k(n-k)} = \Theta(1/k)$ [32], which is fixed and sufficiently small to enable efficient mixing. Fig. 2(a) shows an example of a Johnson graph.

Using a uniform Markov chain—one in which the stationary distribution corresponds to the uniform distribution—the search algorithm begins with a uniform distribution $\boldsymbol{s}_\pi$ over the vertices. The algorithm samples one element v_i from it and checks if $f(v_i) = 1$. If not, the algorithm performs $1/\delta$ steps of the random walk to reestablish the uniform distribution and repeats the process.

The best quantum adaptation of this approach for Johnson graphs, known as the MNRS framework from the authors' initials, was introduced in [25] and implemented in the quantum circuit model in [24]. This framework achieves the same asymptotic speedup as QAA while outperforming QAA in the finite regime for specific problems. Below, we outline the key distinctions between the MNRS quantum walk framework and QAA. Fig. 2(b) illustrates the high-level circuit for this approach.

1) Input Preparation ($\boldsymbol{U}_{\mathcal{E}}$). While QAA initializes a uniform superposition $|\chi\rangle$ over all bitstrings $\in \mathcal{X}$, the MNRS framework generates a uniform superposition $|\epsilon\rangle$ over all the edges $\in \mathcal{E}$. In [24], this approach is realized by combining the operator $\boldsymbol{U}_{\mathcal{X}}$ used in QAA with two additional quantum circuits: a Binary Index Extractor (BIX), generating a superposition of all the nodes in $\mathcal{V}$ starting from $|\chi\rangle$; and the *update* operator $\boldsymbol{U}_u$, creating a uniform superposition over all edges. At the end of this stage, the state is:

$$|\epsilon\rangle = \frac{1}{\sqrt{|\mathcal{E}|}} \sum_{(v_L, v_R) \in \mathcal{E}} |v_L\rangle\,|v_R\rangle\,.$$

in which the left side (v_L) represents the starting vertex of an edge, while the right side (v_R) represents the target vertex.

2) Oracle ($\boldsymbol{R}_{\mathcal{M}^\perp}$). The oracle operator $\boldsymbol{R}_{\mathcal{M}^\perp}$ in the MNRS framework is almost identical to that in QAA. It operates on the left side of the state to invert the amplitude of each state labelled as a bitstring of the marked set $\mathcal{M}$.

3) Reflection ($\boldsymbol{R}_{\mathcal{E}}$). The primary innovation of the MNRS framework lies in the amplitude amplification process, traditionally realized as the diffusion operator in QAA. The MNRS framework introduces the *walk operator* $\boldsymbol{U}_w$, which corresponds to a unitary representation of $\boldsymbol{P}$. In [25] the operator $\boldsymbol{U}_w$ is realized by two applications of $\boldsymbol{U}_u \boldsymbol{R}_{0^\perp} \boldsymbol{U}_u^\dagger$. To perform the reflection $\boldsymbol{R}_{\mathcal{E}}$, the framework employs a QPE circuit, applying $\frac{1}{\sqrt{\delta}}$ times the controlled version of $\boldsymbol{U}_w$. Indeed, if the phase is 0, the corresponding eigenvalue is 1, which corresponds to the quantum state representing the stationary (and uniform) distribution $\boldsymbol{s}_\pi$.

2.4 k-clique problem

A graph is denoted as $G = (\mathcal{V}_G, \mathcal{E}_G)$, with $\mathcal{V}_G = \{1, 2, \ldots, n\}$ denoting the set of vertices, and $\mathcal{E}_G \subseteq \mathcal{V}_G \times \mathcal{V}_G$ the set of edges. We consider finite, undirected, and unweighted graphs, with no self-loops. A graph is called *complete* if every pair of vertices is connected by an edge. Formally, a complete graph satisfies $(i, j) \in \mathcal{E}_G$ for all $i, j \in \mathcal{V}_G$ with $i \neq j$. A *clique* in G is a subset $\mathcal{V}_{G_1} \subseteq \mathcal{V}$ such that $G_1 = \left(\mathcal{V}_{G_1}, \mathcal{E}_G \cap \mathcal{V}_{G_1} \times \mathcal{V}_{G_1}\right)$ is a complete subgraph.

Definition 2.1 (k-clique problem (search version)). Given a graph $G = (\mathcal{V}_G, \mathcal{E}_G)$ and a value k, find a clique of k nodes in G.

k-clique through adjacency matrix. A graph G can be represented by an adjacency matrix $\boldsymbol{A}_G$, where $a_{ij} = a_{ji} = 1$ if $(i, j) \in \mathcal{E}_G$ and $a_{ij} = a_{ji} = 0$ otherwise. As an example, Fig. 2(c) shows a graph, and its adjacency matrix is depicted in Fig. 2(d).

Given k indices corresponding to the rows of the adjacency matrix, the sum of these rows yields a vector where the value at the k indices equals $k-1$ if and only if the selected k rows correspond to a k-clique in the graph. For the running example, summing rows $\{1, 2, 3\}$ of the adjacency matrix results in the vector $[1, 2, 2, 2, 1, 3]$. At indices $\{1, 2, 3\}$ of this vector, the value is exactly 2, confirming that these three nodes form a 3-clique.

3 Implementation

This section outlines the subcircuits required to implement the QAA and QW search strategies for identifying a k-clique in an n-node undirected, unweighted graph $G = (\mathcal{V}_G, \mathcal{E}_G)$. As the QAA strategy utilizes a subset of the subcircuits needed for the QW strategy, we focus on describing the QW implementation, highlighting in the text and algorithms the components that differ between the two approaches.

Given the unknown structure of G, exploiting the graph directly in a QW poses non-trivial challenges, including but not limited to the eigenvalue gap δ, for which a small value may reduce the efficiency of the strategy, and the implementation of a resource

Algorithm 1: $U_{\mathcal{V}}$

```
dickeState(D, n, k)                          // generates U_X
binaryIndexExtractor(D, L, L̄)
```

Algorithm 2: U_u

```
copyState(L, R)                              copyState(L̄, R̄)
wState(W, k, 1)                              wState(W̄, n−k, 1)
removeElement(R, T | W)                      removeElement(R̄, T̄ | W̄)
binaryToOneHot(T, T_1)                       binaryToOneHot(T̄, T̄_1)
insertElement(T̄, R)                          insertAtIdx(T, R̄)
updateDicke(D, T_1, T̄_1)
```

efficient update operator $\boldsymbol{U}_u$, preparing the uniform superposition of all the edges in $\mathcal{E}_G$, which may require a high number of gates.

To overcome these challenges, our strategy exploits the regular structure of the Johnson graph $J(n,k)$ to perform a QW search. The core idea underlying the functioning of the quantum circuit is to use the k integers encoded in a single node of $J(n,k)$ as indices of the row of the $n\times n$ adjacency matrix $\boldsymbol{A}_G$ of G to sum together. If such rows form a k-clique, the resulting vector will contain, on the k indices, a value equal to $k-1$, as explained in Sec. 2.4. The uniform superposition of all the nodes of the Johnson graph can then be exploited to assess the k-clique property for all the distinct $\binom{n}{k}$ combinations of k rows out of n of $\boldsymbol{A}_G$ in superposition.

In the rest of this section, we focus on the design of the operators $\boldsymbol{U}_{\mathcal{V}}$, $\boldsymbol{U}_u$, and $\boldsymbol{U}_f$ introduced in Sec. 2.3, adapting them to address the specific requirements of the k-clique problem. Following standard conventions in quantum computing, we assume that the quantum state is initialized to the all-zero basis state $|0\rangle$.

3.1 Superposition of $J(n,k)$ nodes ($U_{\mathcal{V}}$)

This stage prepares a uniform superposition of all the nodes belonging to the Johnson graph $J(n,k)$. While in a pure QAA strategy the first stage is limited to the $\boldsymbol{U}_{\mathcal{X}}$ operator, producing a uniform superposition of all the $\binom{n}{k}$ binary strings of length n with Hamming weight k, our QW approach extends this process by preparing a superposition over all the $\binom{n}{k}$ nodes in $\mathcal{V}$. Alg. 1 outlines the functions used to construct the quantum circuit for the QW strategy.

1) Dicke State generation circuit. The first quantum circuit, generated by the `dickeState` function, prepares the Dicke state D_k^n on a quantum register $\underline{\boldsymbol{D}}$ of size n. The Dicke state is defined as the uniform superposition:

$$|\chi\rangle = \frac{1}{\sqrt{\binom{n}{k}}} \sum_{i\in\mathcal{B}_k^n} |i\rangle\,,$$

in which $\mathcal{B}_k^n$ represent the set of all the $\binom{n}{k}$ length-n binary vectors that have a Hamming weight of k.

The most efficient algorithm to generate such a state, as described in [3], starts by initializing the quantum state to $|1^k 0^{n-k}\rangle$ and then leverages the recursive nature of the Dicke state to apply a sequence of controlled rotations that incrementally build the balanced superposition. This strategy achieves a circuit depth of $O(k)$, using $O(nk)$ `CNOT` and $\mathsf{R_Y}$ gates without auxiliary qubits.

2) Binary Index Extractor. The second step, needed only in the QW technique, while not necessary for the QAA strategy, creates a uniform superposition of all nodes in the Johnson graph, encoding them into a quantum register $\underline{\boldsymbol{L}}$ of size $k\log_2(n)$. For this purpose, the `binaryIndexExtractor` function, as proposed in [24], processes the Dicke state superposition encoded in $\underline{\boldsymbol{D}}$. Specifically, this function generates a quantum circuit composed by only constant additions and controlled shifts, both performed on $\underline{\boldsymbol{L}}$, to extract the binary representation of the indices corresponding to the k qubits in $\underline{\boldsymbol{D}}$ equal to $|1\rangle$, encoding these indices into $\underline{\boldsymbol{L}}$. Moreover, such indices are stored in ascending order, allowing to exploit a history-independent data structure [21] that is used to perform non-random accesses to quantum register locations.

At the conclusion of this step, $\underline{\boldsymbol{L}}$ encodes the state:

$$|v\rangle = \frac{1}{\sqrt{|\mathcal{V}|}} \sum_{v_L\in\mathcal{V}} |v_L\rangle\,,$$

where each $|v_L\rangle$ represents the binary encoding of a node of the Johnson graph $J(n,k)$. The subscript indicates that the node will correspond to the left part of an edge.

Simultaneously, the indices of the $n-k$ qubits in $\underline{\boldsymbol{D}}$ equal to $|0\rangle$ are stored in a register $\underline{\bar{\boldsymbol{L}}}$ of size $(n-k)\log_2(n)$. The quantum state of $\underline{\bar{\boldsymbol{L}}}$, which encodes the complement of $\underline{\boldsymbol{L}}$ with respect to the total set $[n]$, is crucial for generating a uniform superposition of all edges in $\mathcal{E}$ during subsequent steps.

The circuit implementing `binaryIndexExtractor` has a depth of $O\big(n^2\log_2(n)\big)$, as outlined in [24]. Its gate complexity includes $O\big(n^2\log_2(n)\big)$ $\mathsf{C^2NOT}$ and `CNOT`gates, alongside $O\big(n\log_2(n)\big)$ `NOT`.

3.2 Superposition of $J(n,k)$ edges (U_u)

The operator $\boldsymbol{U}_u$ is only required by the QW strategy to build a superposition of edges adjacent to those encoded in $\underline{\boldsymbol{L}}$, encoding them into another register $\underline{\boldsymbol{R}}$ having the same size. Since in $J(n,k)$ two nodes are adjacent if they differ by only a single element, the core idea of this stage is to first copy the content of $\underline{\boldsymbol{L}}$ and $\underline{\bar{\boldsymbol{L}}}$ into $\underline{\boldsymbol{R}}$ and $\underline{\bar{\boldsymbol{R}}}$ respectively, and then to perform a uniform sample of an element of $\underline{\boldsymbol{R}}$, and replace it with a uniform sample of an element of $\underline{\bar{\boldsymbol{R}}}$.

Alg. 2 provides an overview of this process, with steps detailed below. The structure of the algorithm highlights the parallelism and independence between the two sides, as they operate on complementary registers (e.g., $\underline{\boldsymbol{R}}$ and $\underline{\bar{\boldsymbol{R}}}$).

1–2) Initialization of adjacent nodes. Since adjacent vertices in the Johnson graph differ by only one element, the algorithm begins by copying the labels of the basis states from $\underline{\boldsymbol{L}}$ into an auxiliary register $\underline{\boldsymbol{R}}$ of the same size. This operation involves $k\log_2(n)$ `CNOT` gates, all of which can be applied in parallel. Similarly, the labels in $\underline{\bar{\boldsymbol{L}}}$ are copied into another auxiliary register $\underline{\bar{\boldsymbol{R}}}$ through $(n-k)\log_2(n)$ independent `CNOT` gates.

3–4) W State generation circuit. In this step, a W state — a Dicke state superposition where exactly one qubit is in the state $|1\rangle$ — is generated for two quantum registers: $\underline{\boldsymbol{W}}$, of size k, and $\underline{\bar{\boldsymbol{W}}}$, of

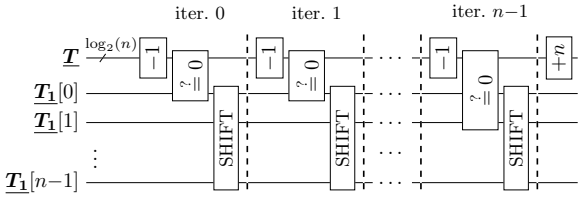


Figure 3: Circuit encoding the binary representation of the integer encoded in $\underline{T}$ into a one-hot representation in $\underline{T_1}$.

size $n-k$. The registers are used to drive a uniform sample of the elements of $\underline{R}$ and $\overline{\underline{R}}$ respectively in the next stage. For this stage, we use the implementation of [10], that achieves a logarithmic depth using a linear number of $\mathsf{R_Y}$ gates and does not require auxiliary qubits.

5–6) Removing an element from $\underline{R}$ and $\overline{\underline{R}}$. The register $\underline{W}$ (resp., $\overline{\underline{W}}$) is used as a control in a Boolean masking operation to remove one element from $\underline{R}$ (resp., $\overline{\underline{R}}$), and store it in a temporary quantum register $\underline{T}$ (resp., $\overline{\underline{T}}$) of size $\log_2(n)$ (resp., $\log_2(n)$). The removal relies on the Johnson vertex deletion circuit presented in [21], requiring: a depth of $O(k \log n)$ (resp., $O(n-k)\log_2(n)$) for $\underline{T}$ (resp., $\overline{\underline{T}}$). Most importantly, such deletion keeps the elements of $\underline{R}$ (resp., $\overline{\underline{R}}$) ordered, leaving an empty element only in the last position of the register.

7–8) Binary to one-hot encoding. This circuit, generated by the `binaryToOneHot` function, converts the binary value stored in $\log_2(n)$ qubits $\underline{T}$ (resp., $\overline{\underline{T}}$) into a one-hot encoding in an additional quantum register $\underline{T_1}$ (resp., $\overline{\underline{T_1}}$) of size n. Specifically, if $\underline{T}$ holds the binary representation of an integer $x \in [n]$, the register $\underline{T_1}$ will contain the state $|1\rangle$ only at the qubit corresponding to index x, with all other qubits in state $|0\rangle$.

The circuit for this operation is depicted in Fig. 3 for the registers $\underline{T}$ and $\underline{T_1}$. It comprises n identical steps. In each step, the binary value in $\underline{T}$ is decremented by 1. The circuit then checks if $\underline{T}$ equals 0; if so, it sets the first qubit of $\underline{T_1}$ to 1. This is followed by a cyclic shift of all qubits in $\underline{T_1}$. After completing all n iterations, $\underline{T_1}$ contains the one-hot encoding corresponding to the original value stored in $\underline{T}$. Finally, the circuit restores $\underline{T}$ to its initial value by adding $+n$.

The circuit requires $n+1$ adders and n equality checks, while the shifts correspond to a logical relabelling of the qubits and requires no quantum gate. The adders can be realized using the proposal of [34], giving a depth and a number of gates of $\log_2(n)$ for each of them, and a total depth and total number of gates of $n\log_2(n)$. The equality check, on the other hand, corresponds to a multi-controlled version of a `NOT` gate, in which all the control qubits should be in state $|0\rangle$ to apply the `NOT` gate on the target qubit $\underline{T}[0]$. This can be implemented by employing a $\mathsf{C}^{\log_2(n)}\mathsf{NOT}$ gate, using a layer of `NOT` gates on the control qubits right before and right after it.

9–10) Adding an element to $\underline{R}$ and $\overline{\underline{R}}$. This step inserts the element encoded in $\overline{\underline{T}}$ into $\underline{R}$, and analogously the element encoded in $\underline{T}$ into $\overline{\underline{R}}$. After this stage, the register $\underline{R}$ encodes a superposition of basis states, each one representing the node adjacent to the one in $\underline{L}$. That is, $\underline{L}$ and $\underline{R}$ contain the quantum state

$$|\epsilon\rangle = \sum_{(v_L, v_R)\in\mathcal{E}} |v_L\rangle\,|v_R\rangle\,.$$

The first part of the quantum circuit generated through the `insertElement` function is identical to the one generated by the `deleteElement` function, in which the gate are applied in reverse order. Additionally, the quantum circuit performs a final sequence of operations to reset the data present into the quantum register $\underline{T}$ and $\overline{\underline{T}}$ to the all-0's bitstrings.

11) Update Dicke state. The final step updates the label encoded in $\underline{D}$ with the one from $\underline{T_1}$ and $\overline{\underline{T_1}}$ by performing `CNOT` gates between $\underline{T_1}$ and $\underline{D}$, and from $\overline{\underline{T_1}}$ and $\underline{D}$. At the end of this stage, $\underline{D}$ contains the one-hot encoding of the integer encoded in $\underline{R}$. Such a state is required to drive a Boolean masking in the next steps.

Algorithm 3: U_f

```
addRows(A_G, S | D)
isEqual(S, E, k−1)
isClique(E, D)
```

3.3 Checking the k-clique condition (U_f)

The oracle, a critical component for both the QW and the QAA strategy, is responsible for determining whether the original graph contains a k-clique. This step leverages the k integers encoded in $\underline{R}$, corresponding to a node of the Johnson graph $J(n,k)$, as indices for the rows of the adjacency matrix $\boldsymbol{A}_G$ encoded in $\underline{A}$. By summing these rows and checking if the resulting vector contains the value $k-1$ at the specified indices (as detailed in Sec. 2.4), the oracle can verify the k-clique condition.

Alg. 3 outlines the steps involved in implementing the $\boldsymbol{U}_f$ operator within the oracle framework. As described in Sec. 2.2, the oracle operator $\boldsymbol{R}_{\mathcal{M}^\perp}$ is defined as $\boldsymbol{U}_f^\dagger \boldsymbol{R}_{1^\perp} \boldsymbol{U}_f$, where $\boldsymbol{U}_f^\dagger$ reverses the order of gates in $\boldsymbol{U}_f$. Below, we detail the quantum operations that constitute $\boldsymbol{U}_f$.

1) Adding Rows of $\boldsymbol{A}_G$. The `addRows` procedure generates a quantum circuit that conditionally sums the rows of the adjacency matrix $\boldsymbol{A}_G$ based on the labels encoded in the quantum register $\underline{D}$. The results are stored in an auxiliary register $\underline{S}$ of size $n\log_2(n)$.

Specifically, for each qubit $\underline{D}[i]$, with $0 \le i < n$, if $\boldsymbol{A}[i,j] = 1$, we perform a controlled +1 addition to $\underline{S}[j]$, conditioned on $\underline{D}[i]$ being in the $|1\rangle$ state. Since $\underline{D}$ encodes basis states with a Hamming weight of k, this step effectively computes all $\binom{n}{k}$ row sums in superposition.

Each addition involves summing the value +1 in $\underline{S}[j]$, represented using $\log_2(n)$ qubits, whenever $\underline{D}[i] = |1\rangle$ and $\boldsymbol{A}[i,j] = 1$. In the worst-case scenario, when $\boldsymbol{A}_G$ is dense, there are $\approx \binom{n}{2}$ additions performed, each involving two $\log_2(n)$-qubit registers and controlled by n qubits in $\underline{D}$. To optimize this process, we interlace the control operations across different qubits and targets, enabling up to n additions to be performed in parallel.

The quantum adder circuit from [34] is employed for these additions. This adder circuit requires $O(\log_2(n))$ $\mathsf{C^2NOT}$ and `CNOT` gates, achieving a depth of $O(\log_2(n))$. As a result, the overall gate complexity and circuit depth for this step are $O(n\log_2(n))$.

Table 1: Complexity metrics as a function of n and k for the quantum subcircuits described in Sec. 3, showing the dominant terms for each cell. Lower-order terms are omitted for clarity. Shaded columns indicate subcircuits exclusive to the QW approach, while the QAA approach relies solely on the non-shaded columns.

	Cost metrics	U_X (Alg. 1, Sec. 3.1)		U_u (Alg. 2, Sec. 3.2)			U_f (Alg. 3, Sec. 3.3)	
		1)	2)	1–6, 9–10)	7–8)	11)	1)	2–3)
NCT + R_y	R_Y	$4nk - 4k^2$	0	$2n$	0	0	0	0
	NOT	k	n	2	$\log_2(n)$	0	0	$2n$
	CNOT	$6nk - 5k^2$	$2n^2 \log_2(n)$	$29n \log_2(n) - 20n$	$5n \log_2(n)$	$2n$	$5n^2 \log_2(n)$	n
	C^2NOT	0	$n^2 \log_2(n)$	$9n \log_2(n) - 4n$	$2n \log_2(n)$	0	$2n^2 \log_2(n)$	0
	Depth	$k \log_2(n/k)$	$n^2 \log_2(n)$	$n \log_2(n) - k \log_2(n)$	$2n \log_2(n)$	1	$2n \log_2(n)$	1
	Qubits	n	$n \log_2(n)$	$n \log_2(n)$	$2n$	0	$n \log_2(n)$	0

2) Computing k-Clique Condition. This step verifies in superposition whether the selected rows of A_G form a k-clique. The `isEqual` procedure compares each element of $\underline{S}$ with the value $k-1$. If a comparison succeeds, the corresponding qubit in the quantum register $\underline{E}$ (of size n) is set to $|1\rangle$. For a fixed k, as it is the case for our algorithm, each comparison employs multi-controlled NOT gates having $\log_2(n)$ control qubits. At the conclusion of this stage, the register $\underline{E}$ contains $|1\rangle$ at index i if and only if the sum of the selected k rows of A_G equals $k-1$ at the specified indices.

3) Checking the k-Clique Condition The final step involves verifying that the indices in $\underline{D}$ match those in $\underline{E}$, which indicates that the selected rows satisfy the k-clique condition. This is achieved using a bitwise XOR operation, where $\underline{D}$ is XORed with $\underline{E}$. If $\underline{D}$ and $\underline{E}$ align (i.e., they both contain $|1\rangle$ at the same positions), the resulting state in $\underline{E}$ will be the all-zero bitstring.

To complete the oracle operation, a layer of NOT gates is applied to all qubits in $\underline{E}$, marking the indices that satisfy the k-clique condition. The output of $\underline{E}$ is then fed into the $R_{1^\perp}$ operator, finalizing the oracle implementation.

4 Performance Evaluation

In this section, we analyze the performance of the proposed quantum algorithms for solving the k-clique problem by comparing our QW and QAA strategies with state-of-the-art approaches. Our analysis is conducted at the logical level in a hardware-agnostic manner, focusing on algorithmic complexity and expected quantum speedup while disregarding issues related to error-correction, decoherence, and qubit connectivity. Although we tested and validated all the individual components, simulating the full circuit is infeasible even for small graphs due to the exponential overhead in classical simulation.

Quantum circuit complexity is commonly evaluated using three fundamental metrics: *depth*, which measures the length of the longest sequence of gates acting on any qubit; *width*, indicating the number of qubits required; and *depth-times-width*, used to capture trade-offs relevant to physical implementations [30]. While practical implementations are constrained by factors such as qubit connectivity, error rates, and coherence times, this work focuses on the logical description of the quantum circuits, and therefore assume a quantum architecture with all-to-all qubit connectivity and logical qubits with perfect reliability.

4.1 Performance Analysis using NCT+R_Y gates

To align with the constraints of realistic quantum hardware, typically limited to 1- and 2-qubit gates, we evaluate the algorithms using the NOT, CNOT, Toffoli (NCT) gate set, comprising {NOT, CNOT, C^2NOT} gates, augmented with the R_Y gate. In this framework, gates involving more than two qubits are decomposed into simpler gates, except for C^2NOT gates, which are treated as primitives. Multi-controlled gates requiring decomposition appear in subcircuits generated by Alg. 2 (line 7–8) and Alg. 3 (line 2). These gates, controlled by $\log_2(n)$ qubits, can be decomposed into $O(\log_2(n))$ C^2NOT gates using $O(\log_2(n))$ auxiliary qubits, achieving a depth of $O(\log_2(\log_2(n)))$ [18, 30]. Tab. 1 summarizes the resource requirements of the key algorithmic components discussed in Sec. 3.

Total Resource Estimates. As described in Sec. 2.2 and Sec. 2.3, the full implementation of both the QAA and QW approach requires $O(1/\sqrt{\varepsilon})$ applications of the $R_{\mathcal{M}^\perp}$ operator, whose implementation is provided in Sec. 3.3. The QAA approach further requires $O(1/\sqrt{\varepsilon})$ applications of the R_X operator, requiring the Dicke state generation circuit explained in Sec. 3.1. The QW approach, on the other hand, includes $O(1/\sqrt{\varepsilon})$ applications of the Quantum Phase Estimation (QPE) circuit, each one involving $O\left(1/\sqrt{\delta}\right)$ applications of the U_u operator detailed in Sec. 3.2.

The parameter $\varepsilon = |\mathcal{M}|/|\mathcal{X}|$ depends on the number of k-cliques, $|\mathcal{M}|$, present in the graph G. As $|\mathcal{M}|$ is unknown at circuit generation time, this introduces uncertainty into the circuit complexity estimates. Amplitude amplification schemes rely on iterating an operator a precise number of times to maximize solution probability. Deviation from the optimal iteration count reduces the likelihood of detecting a k-clique. To address this challenge, [5] proposes an iterative strategy in which amplitude amplification is repeated multiple times, incrementally increasing the number of iterations until a solution is identified. The strategy requires around three times the number of iterations required if the number of solutions is known in advance.

4.2 State-of-the-Art comparison using Clifford + T gates

The only state-of-the-art proposal explicitly addressing the k-clique problem is presented in [27], which employs a QAA-based approach to solve the problem. In the work, the quantum circuit uses n qubits,

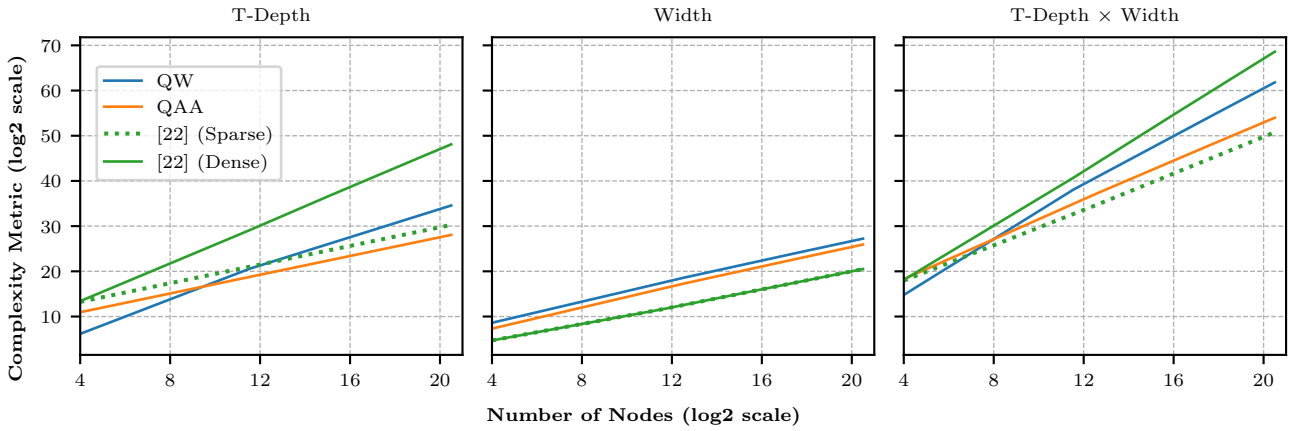


Figure 4: Comparison of resource requirements for solving the k-clique problem using our proposed approaches (QAA and QW) and the state-of-the-art method from [27]. The analysis evaluates T-depth and width (number of qubits) for the Clifford + T gate set, targeting the worst-case scenario with $k = 0.5n$ and a single solution. For [27], results are presented for both dense graphs ($|\mathcal{E}_G| = \binom{n}{2}$) and sparse graphs ($|\mathcal{E}_G| = n$).

each representing a node in the graph G, and a Dicke state generation circuit D_k^n is used to create a uniform superposition of length-n, weight-k bitstrings. The oracle of this approach operates by materializing all edges in the edge set $\mathcal{E}_G$ using multi-controlled NOT gates. Specifically, each edge, involving two qubits, acts as a control to increment an edge counter register. Similarly, a node counter register is incremented using again multi-controlled NOT gates. The k-clique condition is confirmed when the edge and node counters reach the values $\binom{k}{2}$ and k, respectively.

We assess our proposal and the one of [27] using the Clifford + T gate set, considered to be the most promising one for fault-tolerant quantum computing. This gate set includes {H, S, CNOT, T} gates. Since T gates are expected to dominate resource consumption in fault-tolerant implementations, minimizing T-depth is a priority. In the Clifford + T gate set, C^2NOT and $\mathrm{R_Y}(g)$ates require decomposition. A C^2NOT gate can be implemented with T-depth 1 and T-size 4 using one auxiliary qubit [19]. On the other hand, arbitrary $\mathrm{R_Y}$ rotations, used in Dicke and W state generation, require angle-specific synthesis. These gates can be approximated using the algorithm in [23], with an average T-depth of 149 [31].

Fig. 4 presents a comparative analysis of resource requirements for our proposals and the approach from [27], specifically focusing on T-depth and width in the Clifford + T gate set. The comparison assumes $k = 0.5n$ and $|\mathcal{M}| = 1$, representing the worst-case scenarios. For the state-of-the-art approach, we evaluate its complexity for both dense graphs ($|\mathcal{E}_G| = \binom{n}{2}$) and sparse graphs ($|\mathcal{E}_G| = n$), as its resource requirements scale with the number of edges $|\mathcal{E}_G|$.

The results highlight that both of our proposed approaches require significantly fewer resources compared to [27], particularly for dense graphs. Specifically, the QAA-based approach achieves a substantial reduction in T-depth, with gains proportional to the number of nodes n, albeit at the expense of requiring a larger number of qubits. In contrast, the QW-based approach provides more modest improvements in T-depth with respect to [27], and becomes less efficient then our QAA-based approach when the number of nodes in G is greater than $\geq 2^{10}$. Finally, considering the aggregate metric of T-depth×width, both of our proposed approaches outperform [27], with the gap growing exponentially as the number of nodes increases.

5 Conclusion

We presented two novel quantum circuit designs for solving the k-clique problem, leveraging the adjacency matrix structure of the graph. The first approach is a tailored Quantum Amplitude Amplification algorithm, while the second exploits the Johnson graph's regularity within the Quantum Walk search framework [24, 25], preserving its quadratic speedup. A notable contribution of this work is demonstrating how the Johnson graph's regularity can be leveraged to address problems on irregular graphs. Notably, our approach is independent of any specific graph structure, making it suitable for a wide range of graph instances without requiring structural constraints or preprocessing assumptions. Our proposed algorithms scale with the number of nodes n, in contrast to state-of-the-art approaches that depend on the number of edges. This makes our techniques particularly effective for dense graphs, where the number of edges approaches $\binom{n}{2}$. Notably, our methods achieve significant speedups for graphs with up to 2^{20} nodes, improving both the number of gates and circuit depth across a range from 2^5 to 2^{20} nodes. This comes at the cost of requiring approximately 2^3 to 2^7 additional qubits within the same range, while still achieving an improvement of up to 2^{15} in the depth × width metric.

Future work will explore extensions to weighted clique problems, maximum clique finding, and other combinatorial problems.

Acknowledgments

The activity was partially funded by the Italian *Centro Nazionale di Ricerca in HPC, Big Data e Quantum computing - SPOKE 10*

References

[1] Scott Aaronson and Andris Ambainis. 2005. Quantum Search of Spatial Regions. *Theory of Computing* 1, 1 (2005), 47–79. doi:10.4086/TOC.2005.V001A004

[2] Andris Ambainis. 2007. Quantum Walk Algorithm for Element Distinctness. *Siam Journal On Computing* 37, 1 (2007), 210–239. doi:10.1137/S0097539705447311

[3] Andreas Bärtschi and Stephan J. Eidenbenz. 2022. Short-Depth Circuits for Dicke State Preparation. In *IEEE International Conference on Quantum Computing and Engineering, QCE 2022, Broomfield, CO, USA, September 18-23, 2022*. IEEE, 87–96. doi:10.1109/QCE53715.2022.00027

[4] Alan Bojić. 2012. Quantum Algorithm for Finding a Maximum Clique in an Undirected Graph. *Journal of Information and Organizational Sciences* 36, 2 (Dec. 2012), 91–98.

[5] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. 1998. Tight Bounds on Quantum Searching. *Fortschritte der Physik: Progress of Physics* 46, 4–5 (1998), 493–505. doi:10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P

[6] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. 2002. Quantum Amplitude Amplification and Estimation. *Quantum Computation and Information* 305 (2002), 53–74. doi:10.1090/conm/305/05215

[7] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. 2005. Quantum Algorithms for Element Distinctness. *Siam Journal On Computing* 34, 6 (2005), 1324–1330. doi:10.1137/S0097539702402780

[8] Guillaume Chapuis, Hristo N. Djidjev, Georg Hahn, and Guillaume Rizk. 2017. Finding Maximum Cliques on a Quantum Annealer. In *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*. ACM, 63–70. doi:10.1145/3075564.3075575

[9] Andrew M. Childs, Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2002. Finding Cliques by Quantum Adiabatic Evolution. *Quantum Information & Computation* 2, 3 (2002), 181–191. doi:10.26421/QIC2.3-1

[10] Diogo Cruz, Romain Fournier, Fabien Gremion, Alix Jeannerot, Kenichi Komagata, Tara Tosic, Jarla Thiesbrummel, Chun Lam Chan, Nicolas Macris, Marc-André Dupertuis, et al. 2019. Efficient Quantum Algorithms for GHZ and W States, and Implementation on the IBM Quantum Computer. *Advanced Quantum Technologies* 2, 5–6 (2019), 1900015. doi:10.1002/qute.201900015

[11] John D. Eblen, Charles A. Phillips, Gary L. Rogers, and Michael A. Langston. 2012. The Maximum Clique Enumeration Problem: Algorithms, Applications, and Implementations. *BMC Bioinformatics* 13, 10 (June 2012), S5. doi:10.1186/1471-2105-13-S10-S5

[12] Tuvi Etzion and Patric RJ Ostergard. 1998. Greedy and Heuristic Algorithms for Codes and Colorings. *IEEE Transactions on Information Theory* 44, 1 (1998), 382–388.

[13] Santo Fortunato. 2010. Community Detection in Graphs. *Physics reports* 486, 3–5 (2010), 75–174. doi:10.1016/j.physrep.2009.11.002

[14] François Le Gall. 2014. Improved Quantum Algorithm for Triangle Finding via Combinatorial Arguments. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. IEEE Computer Society, 216–225. doi:10.1109/FOCS.2014.31

[15] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman.

[16] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual {ACM} Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, Gary L. Miller (Ed.). ACM, 212–219. doi:10.1145/237814.237866

[17] Andrew Haverly and Sonia López. 2021. Implementation of Grover's Algorithm to Solve the Maximum Clique Problem. In *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2021, Tampa, FL, USA, July 7-9, 2021*. IEEE, 441–446. doi:10.1109/ISVLSI51109.2021.00087

[18] Yong He, Ming-Xing Luo, E Zhang, Hong-Ke Wang, and Xiao-Feng Wang. 2017. Decompositions of N-Qubit Toffoli Gates with Linear Circuit Complexity. *International Journal of Theoretical Physics* 56 (2017), 2350–2361.

[19] Samuel Jaques, Michael Naehrig, Martin Roetteler, and Fernando Virdia. 2020. Implementing Grover Oracles for Quantum Key Search on AES and LowMC. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12106)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 280–310. doi:10.1007/978-3-030-45724-2_10

[20] Samuel Jaques and Arthur G. Rattew. 2023. QRAM: A Survey and Critique. arXiv:2305.10310 [quant-ph]

[21] Samuel Jaques and John M. Schanck. 2019. Quantum Cryptanalysis in the RAM Model: Claw-finding Attacks on SIKE. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11692)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 32–61. doi:10.1007/978-3-030-26948-7_2

[22] Richard M. Karp. 1972. Reducibility among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations, Held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (The IBM Research Symposia Series)*, Raymond E. Miller and James W. Thatcher (Eds.). Plenum Press, New York, 85–103. doi:10.1007/978-1-4684-2001-2_9

[23] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. 2016. Practical Approximation of Single-Qubit Unitaries by Single-Qubit Quantum Clifford and T Circuits. *IEEE Trans. Comput.* 65, 1 (2016), 161–172. doi:10.1109/TC.2015.2409842

[24] Giacomo Lancellotti, Simone Perriello, Alessandro Barenghi, and Gerardo Pelosi. 2024. Design of a Quantum Walk Circuit to Solve the Subset-Sum Problem. In *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC 2024, San Francisco, CA, USA, June 23-27, 2024*, Vivek De (Ed.). ACM, 298:1–298:6. doi:10.1145/3649329.3657337

[25] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. 2011. Search via Quantum Walk. *SIAM J. Comput.* 40, 1 (2011), 142–164. doi:10.1137/090745854

[26] Frédéric Magniez, Miklos Santha, and Mario Szegedy. 2007. Quantum Algorithms for the Triangle Problem. *Siam Journal On Computing* 37, 2 (2007), 413–424. doi:10.1137/050643684

[27] Sara Ayman Metwalli, François Le Gall, and Rodney Van Meter. 2020. Finding Small and Large k -Clique Instances on a Quantum Computer. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–11. doi:10.1109/TQE.2020.3045692

[28] Michael A Nielsen and Isaac L Chuang. 2011. *Quantum Computation and Quantum Information*. Cambridge University Press.

[29] James R. Norris. 1998. *Markov Chains*. Cambridge University Press.

[30] Simone Perriello, Alessandro Barenghi, and Gerardo Pelosi. 2023. Improving the Efficiency of Quantum Circuits for Information Set Decoding. *ACM Transactions on Quantum Computing* 4, 4 (Aug. 2023), 1–40. doi:10.1145/3607256

[31] Simone Perriello, Alessandro Barenghi, and Gerardo Pelosi. 2024. Quantum Circuit Design for the Lee-Brickell Based Information Set Decoding. In *Applied Cryptography and Network Security Workshops - ACNS 2024 Satellite Workshops, AAC, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14587)*, Martin Andreoni (Ed.). Springer, 8–28. doi:10.1007/978-3-031-61489-7_2

[32] Miklos Santha. 2008. Quantum Walk Based Search Algorithms. In *Theory and Applications of Models of Computation, 5th International Conference, TAMC 2008, Xi'an, China, April 25-29, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 4978)*, Manindra Agrawal, Ding-Zhu Du, Zhenhua Duan, and Angsheng Li (Eds.). Springer, 31–46. doi:10.1007/978-3-540-79228-4_3

[33] Neil Shenvi, Julia Kempe, and K. Birgitta Whaley. 2003. Quantum Random-Walk Search Algorithm. *Physical Review A: Atomic, Molecular, and Optical Physics* 67, 5 (May 2003), 052307. doi:10.1103/PhysRevA.67.052307

[34] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. 2010. Quantum Addition Circuits and Unbounded Fan-Out. *Quantum Information & Computation* 10, 9&10 (2010), 872–890. doi:10.26421/QIC10.9-10-12

[35] Virginia Vassilevska. 2009. Efficient Algorithms for Clique Problems. *Inform. Process. Lett.* 109, 4 (2009), 254–257. doi:10.1016/J.IPL.2008.10.014