# Analyzing, Fixing and Optimizing a Space-Efficient Quantum Circuit for the Graph K-Coloring Problem

Oscar Belletti
*Department of Electronics, Information and Bioengineering*
*Politecnico di Milano*
Milano, Italy
oscar1.belletti@mail.polimi.it

Simone Reale
*Department of Electronics, Information and Bioengineering*
*Politecnico di Milano*
Milano, Italy
simone1.reale@polimi.it

Elisabetta Di Nitto
*Department of Electronics, Information and Bioengineering*
*Politecnico di Milano*
Milano, Italy
elisabetta.dinitto@polimi.it

***Abstract*—K-Coloring is a widely studied problem with both profound theoretical implications and practical applications. Its implementation with quantum means is significant because it can serve as a forerunner for a broad category of related subproblems, further enlarging the already available portfolio of quantum algorithms. In order to go beyond the state of the art we propose three different variants of a state of the art algorithm to account for the trade-off between the width and the depth of the circuit, taken as cost metrics. In particular, the third solution significantly improves the literature's best asymptotic complexity, calculated as the product of the circuit's width and depth, moving from $O(N^3 logN)$ to $O(N^2 logN)$, where N is the number of nodes in the considered graph.**

***Index Terms*—K-Coloring, Graph Coloring, Quantum circuit**

## I. Introduction

QUANTUM computing leverages the physical properties of quantum mechanics to perform computation, which, for some problems, allows for asymptotically better performance compared to classical computing. NP-complete problems have exponential complexity on classical hardware, making them particularly alluring for speedup, but so far, no quantum algorithm is known that is capable of solving them in non-exponential time. This means humbler improvements in efficiency are an interesting avenue to explore.

The K-Coloring problem is one of the 21 NP-Complete problems presented by Richard Karp in his seminal work [1]. It is a widely studied problem in graph theory, offering many practical applications as well as critical theoretical insights. Its characteristics make it an interesting playground for the definition of a suitable quantum solution.

Our work started with a detailed analysis of the state-of-the-art quantum approaches to the K-Coloring problem, particularly the gate model implementations. Our interest was captivated by the paper by Saha et al. [2] that presents a space-efficient implementation of the problem leveraging the Grover algorithm [3]. We set out to replicate the results but encountered some missing information and bugs in its definition, so we shifted our focus towards fixing the approach. We then noticed there was room for further improvements. So, we identified the intrinsic trade-off between *width*, i.e., the total number of qubits in the circuit, and *depth*, i.e., the maximum number of gates encountered from inputs to outputs, as our primary focus; then, we developed two solutions, each minimizing either the width or the depth, and a third solution that tries to balance the two aspects. We evaluated the asymptotic complexity of each solution under the width and depth metrics and analyzed their performance in a noiseless setting with the help of IBM's quantum platform.

The paper is structured as follows. Section II presents the state of the art about quantum implementations of the K-Coloring problem. Section III introduces the overall structure of the quantum gate model algorithm for solving K-Coloring using binary encoding for the colorings representation, Section IV provides an analysis of the algorithm complexity in terms of two different metrics, Section V presents the three variants we have identified and the corresponding analysis of their complexity, Section VI validates the variants and Section VII refines the complexity analysis of all presented approaches considering a possible realization for the MCX gates. Finally, Section VIII concludes the paper.

## II. State of the art

Given a graph G composed of a set of vertices V and a set of edges E, finding a K-Coloring of the graph means assigning a color from a set of k different colors to each vertex, with the constraint that for every edge $(v_i, v_j) \in E$, the colors assigned to each of the vertices connected by the edge are different.

In the quantum computing context, the K-Coloring problem has been considered from multiple perspectives both in the quantum annealing and gate programming models.

Problem formulations adopting the gate model can be found in Saha et al. [2] and Clerc [4]. The difference between the two approaches lies in their different color encoding; while in the work by Clerc, each vertex is assigned a number of qubits equal to the number of colors in a one-hot encoding fashion, Saha et al. use a binary encoding to obtain a representation of the possible colorings using a logarithmic number of qubits for each vertex. In particular, the work by Saha et al. [2] is interesting as it aims to build an automated end-to-end framework that takes any non-directed, unweighted graph along with a specified number of colors (k) and automatically generates the quantum circuit necessary to implement the K-Coloring problem on NISQ devices in a space-efficient way.

In the context of hybrid quantum-classical computation, [5] uses the Grover algorithm to speed up the exhaustive search

of the solution space quadratically. In this case, the problem under analysis is the List Coloring problem, which finds a possible proper coloring for a graph given a list of available colors for each vertex. This problem can be reduced to the standard K-Coloring if, for each vertex, the list contains k colors.

In [6], a possible formulation of graph K-Coloring is presented as an Ising problem; this formulation can be used to develop implementations based on Quantum Annealing (QA) [7], a meta-heuristics derived from an alternative quantum paradigm called Adiabatic Quantum Computation [8]. [9] presents a space-efficient QA implementation that utilizes only $n\log(k)$ qubits instead of the usual $nk$ where $n$ is the number of vertices and $k$ is the number of colors of the specific graph K-Coloring instance. An alternative process to leverage QA to solve our problem of interest is contained in [10] where the graph K-Coloring instance is written down as a set of propositional logic constraints and, through the SATyrus [11] approach, transformed into an energy minimization problem, the final step can then be either execution on real quantum hardware or quantum annealing simulators.

The starting idea of this work was that of analyzing pre-existing examples of quantum K-Coloring implementations to find gaps and chances of improvement, and in order to do that, we found in Saha et al. [2] a good candidate for starting our research since its approach left a lot of unexplored possibilities to be exploited for better results.

## III. Structure of the algorithm

This section presents the overall structure and methodology to solve the K-Coloring problem in a quantum context, leveraging the Grover Algorithm [3] and going beyond the trivial one-hot encoding for the possible colorings. This structure originates from our replication of the Saha et al. approach [2], which resulted in the identification and correction of two main errors present in the original paper, as discussed in the following.

To exemplify the algorithm, we will consider a specific completely connected graph with $N = 3$ vertices and $k = 3$ colors, for which the circuit in Figure 1 is obtained. For the sake of clarity, we have divided the figure into a set of numbered boxes, indicating the different portions of the circuit.

### A. Coloring Representation

An important part of the algorithm is concerning the representation of vertices colors. As discussed in Section II, these can be represented in multiple ways, entailing different compromises in terms of the total number of qubits required versus the total number of gates. In the approach under analysis, we have that, given $k$ the number of colors, each vertex is assigned $q = \lceil \log_2 k \rceil$ data qubits, which hold the binary representation of an integer indicating the color assigned to the vertex. Given $N$ the number of vertices in the input graph we have then that this representation requires only $N * \lceil \log_2 k \rceil$ qubits, but introduces an excess of color encodings – called *invalid colors* – whose number can be trivially calculated as $2^q - k$.

### B. Initialization

The initialization step deals with the following qubits:

- $m = N * \lceil \log_2 k \rceil$ data qubits, which are collectively referred to as $|\psi\rangle$ and set to ground state $|\psi\rangle = |0\rangle^{\otimes m}$, used to encode the possible colorings;
- $r = N$ ancilla qubits referred collectively as $|\theta\rangle$ and set to excited state $|\theta\rangle = |1\rangle^{\otimes r}$, used to store generic intermediate results;
- $s = 2^q - k$ additional ancilla qubits referred as $|\zeta\rangle$ and set to the ground state $|\zeta\rangle = |0\rangle^{\otimes s}$, used to store intermediate results of the invalid color detection step;
- $o = 1$ output qubit referred as $|\phi\rangle$ and set to the state $|\phi\rangle = |-\rangle$ which triggers for the states with a valid coloring inverting their phase;

Thus, after initialization, the state of our system is:

$$|\psi\rangle\,|\theta\rangle\,|\zeta\rangle\,|\phi\rangle = |0\rangle^{\otimes m}\,|1\rangle^{\otimes r}\,|0\rangle^{\otimes(2^q-k)}\,|-\rangle$$

In our example of a complete graph with 3 vertices and 3 colors, we have $m = 6$ data qubits, $r = 3$ plus the two ancilla qubits $|\zeta\rangle$ and $|\phi\rangle$. Figure 1 shows these qubits and, in box 1, the gates belonging to the initialization step. A Hadamard transformation is performed on $|\psi\rangle$ to obtain an equally weighted superposition of the data qubits, and another is applied to $|\phi\rangle$ to obtain $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ on the output qubit. All the other groups of qubits remain unchanged.

### C. Oracle Application

The oracle is implemented using a gate set composed simply by NOT (X), CNOT (CX), Toffoli (CCX), and multi-controlled Toffoli gates (MCX), and includes two main types of sub-components, one for invalid colors detection and the other for checking if two adjacent vertices have the same color, followed by a closing part.

*1) Invalid color detection:* As already mentioned, some data qubits may represent invalid colors. We deal with this problem by marking all states that use them as invalid on an additional qubit line. This is done by applying, for each invalid color, an appropriate qubit activation followed by invalid color detection. In general, the qubit activation is a mask composed of X gates, defined for a specific color, that turns the data qubits of a vertex to $|1\rangle$ for that color. In practice, it assigns an X gate to every qubit line that would be $|0\rangle$ for that color. In Figure 2a and Figure 2b this is exemplified by the gates in the red boxes, referring to two different masks respectively.

After the qubit activation, the invalid color detector checks every vertex for the configuration of all $|1\rangle$ using a MCX gate (see the central part of Figures 2a and 2b). If no such vertices are found, the state is marked as valid by turning the $|\zeta\rangle$ ancilla qubit to $|1\rangle$. Then, the qubit activation is repeated in order to return the data qubits to the initial state, which allows for additional invalid colors to be checked or to proceed to the phase that deals with checking coloring conditions on the edges.

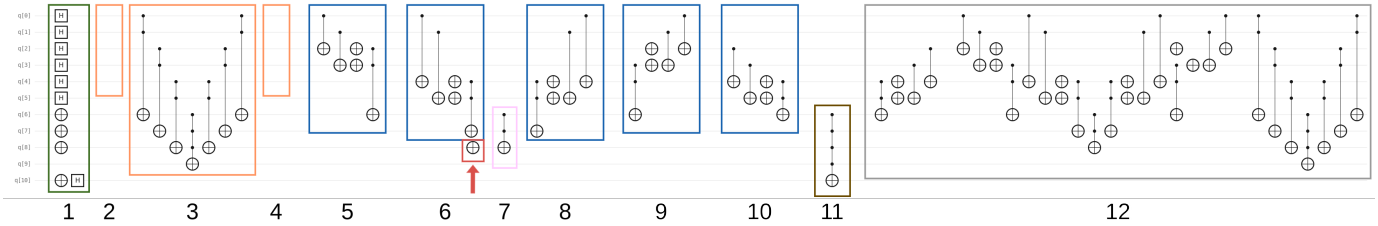
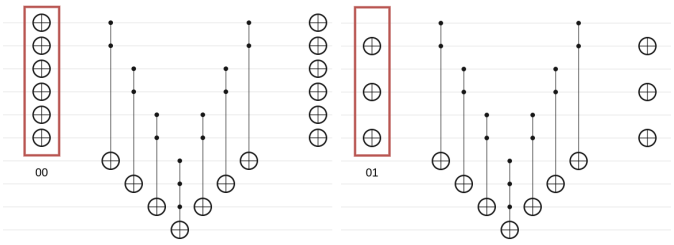



Fig. 1: Representation of the Oracle proposed in [2] to solve a 3-coloring problem with a graph of 3 vertices and three connected edges and corrected in this paper.



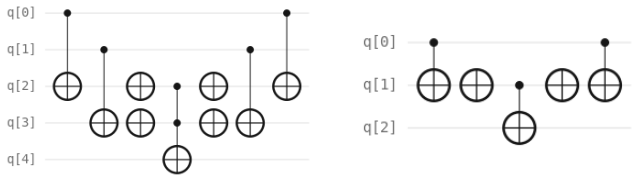

(a) Activation with color 00 mask. (b) Activation with color 01 mask.

Fig. 2: Examples of qubit activation and invalid color detection steps.

In Figure 1, we have a single invalid color, which by convention we pick to be the highest value. As a result, the qubit activation in box 2 is empty since it is targeting the color 11 which does not have zeroes. The invalid color detector is depicted in box 3, followed by the empty qubit activation in box 4. The first error of the Saha et al. approach lies in how the invalid colors are handled, and it can be seen as more problematic due to its conceptual nature. Saha et al. in their paper, suggest using only $N + 1$ ancillae, where the last one is used as a target for all the invalid color detectors.

An ancilla qubit, however, cannot be reused without first erasing its value because it is not possible to distinguish the case where its value is inverted two times from the case where it is never inverted; so it would not be possible to distinguish the case where there are two invalid colors present in a coloring from the case where there is none. Therefore, an additional ancilla qubit is required for each invalid color. Fortunately, while this does almost double the required number of ancillae in the worst case, the asymptotic limit stays $O(N)$. In general, the circuit uses $N+k_{inv}$ ancillae, where $k_{inv}$ is the number of invalid colors ($k_{inv} = 2^{\lceil \log_2(k) \rceil} - k$). The first $N$ ancillae are initialized to ones, the rest to zeros. Each invalid color detector writes to its own dedicated ancillae.

*2) Coloring comparison:* A single comparator component checks one arc to see whether its two adjacent vertices have the same color. Remembering that the ancilla qubits are initialized to $|1\rangle$, the comparator sets an ancilla qubit to the ground state $|0\rangle$ in the case the two corresponding vertices are of the same color and to the excited state $|1\rangle$ in the opposite case. In Figures 3a and 3b comparators between two vertices are defined for colors of two bits and one bit per vertex, respectively. Both comparators include the part setting the ancilla qubit value as explained above and another part resetting the state of all data qubits. In order to check all




(a) (b)

Fig. 3: Examples of binary comparators.

of the edges of the graph, the vertices are considered one at a time in order. For each vertex, the edges connecting it to all of the vertices with a bigger index are checked with a comparator writing to ancillae. In the case where there are two or more such edges, an MCX gate is applied with the corresponding ancillae as input to perform an AND operation, with the outcome written to a different ancilla. The value of $|1\rangle$ for that ancilla should imply that there is no color conflict between the vertex and the adjacent vertices of a higher index. Then, the comparators of this iteration are reapplied to clear the values of their target ancillae. This strategy guarantees that no more than one ancilla is consumed for iteration. In the first iteration, we are considering at most $N-1$ edges (exiting the first vertex), then at most $N-2$ for the second, and so on. The number of required ancillae declines by one per iteration, and one ancilla is written per iteration. Because of that, $N$ ancillae are sufficient to check every arc over the course of $N-1$ iterations.

In Figure 1, we have three different comparators. The first one is split between boxes 5 and 9 and the second between boxes 6 and 8, while the third comparator does not require the resetting part and is, therefore, included only in box 10. Box 7 is the MCX gate needed to combine the results of the first two comparators. The application of this gate is preceded in Figure 1 by the X gate in the red box between boxes 6 and 7. This was not present in the original circuit by Saha et al., causing the production of entirely incorrect results in most cases.

*3) Closing part:* Boxes 11 and 12 in Figure 1 include the last two parts of the circuit. More specifically, box 11 includes

an MCX gate on the output ancilla, which is triggered by the states with a valid coloring inverting their phase. Finally, box 12 repeats all gates in the boxes from 2 to 10 in the reverse order to perform an un-computation step, restoring the ancillae to their initial state and making them available for further Grover iterations if necessary.

### D. Diffusion

The final step of Grover's search algorithm is diffusion, which amplifies the probabilities of the marked states. This is a pretty standard operator which remains the same regardless of the oracle and is not of particular interest in the context of our discussion.

## IV. Algorithm Complexity Analysis

To assess the complexity of the algorithms already presented in Section III and the alternatives that will be described in the following, we introduce two primary metrics: depth and width of the circuit. Since our scope is mainly theoretical, we focus on asymptotic descriptions of these metrics.

### A. Width

In the context of this work, width is defined as the total number of qubits employed in the circuit. From a high-level perspective, all the models presented utilize binary encoding for the colorings and have a width that depends mainly on data qubits and ancillae.

### B. Depth

In [12] the depth of a circuit is defined as the maximum number of gates encountered on any path from an input bit or qubit to an output bit or qubit in the circuit. According to [13] if we visualize the circuit as being divided into a sequence of discrete time-slices, where the application of a single gate requires a single time-slice, the depth of a circuit is its total number of time- slices. This metric is closely related to the number of gates, but how gates are organized to form paths also plays an important role. In the circuit we are analyzing, the comparators asymptotically dominate the depth. The number of comparators used in a circuit depends on the number of arcs to check, which is intrinsic to the specific K-Coloring problem that is being considered. The other factor affecting the asymptotic depth concerns the fact that gates operating on different qubits (and belonging exclusively to different gate-to-gate paths) can be thought of as executing in parallel, thus impacting the depth as just one gate. In Figure 4, we see the ordering approach presented in Saha et al, applied to a complete graph of four nodes where arcs are grouped by nodes. The dots indicate the qubits used by the comparators (in this example, only one data qubit is assigned for each node). Unfortunately, this ordering, combined with how ancillae are used, is close to the worst possible strategy for parallelism. In fact, comparators 1 to 3 cannot be parallelized as they all operate on $q[0]$; the same happens to 4 and 5 and to 5 and 6. The only comparators that could be executed in parallel are comp 3 and 4, unless they share any ancilla qubit. Figure 5 shows a possible ordering of the components that is more parallelizable: the pairs of comparators 1 and 6, 2 and 5, 3 and 4 can all execute with the depth of just one comparator, as long as the ancillae allow for it. This is important because, with an appropriate application of the components, it is possible to cut by half the overall depth, thus asymptotically reducing it by a factor of $O(N)$.

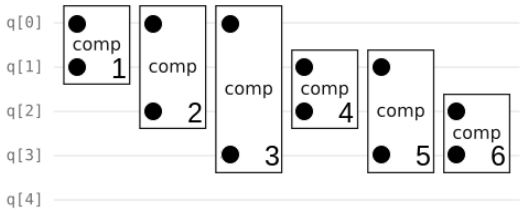



Fig. 4: The ordering of the components in Saha et al.

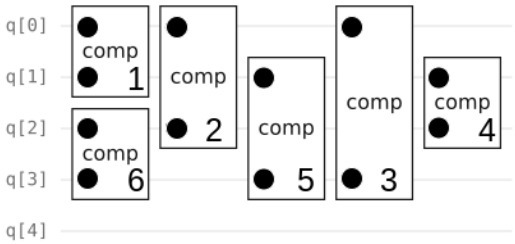



Fig. 5: An alternative ordering of components.

As such, these are the two fundamental quantities that determine the asymptotic depth:

- E = number of comparators used in the circuit
- P = circuit parallelism factor, which indicates the average number of comparators executed in parallel, meaning, in $O(1)$ depth.

Then, the asymptotic depth can be defined as:

$$D = O(E/P)$$

Generally speaking, $E$ is always at least equal to the number of conditions that must be checked, which is in the order of $O(N^2)$, but it can also be higher in specific cases that will be discussed later on. Anyhow, the value of $E$ can be compensated by our ability to parallelize the circuit, given by $P$.

### C. Overall Complexity

Since we want to explicitly consider the trade-off between the depth and width of the circuits analyzed, we define the overall complexity C as follows:

$$C = W \times D$$

Notice that the data qubits set a minimum asymptotic width of $O(Nlog_2N)$, then different oracles will entail different compromises in terms of the number of ancillae.

### D. Complexity of Saha et al. approach

Referring to the width, as $N$ grows larger and larger, the number of data qubits $O(N * \lceil log_2(k) \rceil) = O(NlogN)$ dominates over the ancillae $O(N)$.

As for depth, in our K-Coloring circuit, it is asymptotically dominated by the comparators. A single comparator has a depth of five, regardless of the number of colors (and resulting qubits assigned per vertex). This is because the maximum number of elementary gates within a comparator found on each of the paths associated with qubits is at most five (see Figure 3). We consider it to be $O(1)$.

The number of comparators used in the circuit depends on the number of edges to check, which is intrinsic to the specific K-Coloring problem that is being considered. Since we are taking into consideration fully-connected graphs the number of comparator applications $E$ is $O(N^2)$. When it comes to circuit parallelism, Saha et al apply color comparators in the worst possible order: they are grouped by vertices, which means that they are never in parallel. Therefore, the circuit parallelism factor $P$ is equal to 1, and the depth is $O(N^2)$. This brings the overall complexity to $O(N^3 \log N)$.

## V. New Oracles

The complexity metrics defined in the previous section suggest that possible improvements of the original approach include finding ways to limit the number of used ancillae to reduce the width and increasing the parallelism of gates to reduce the depth. In this section, we discuss the possible strategies that can be applied to achieve these objectives, and then we present three different variants of the K-Coloring approach that use these strategies to improve, respectively, width, depth, and the overall circuit complexity.

### A. Width improvement

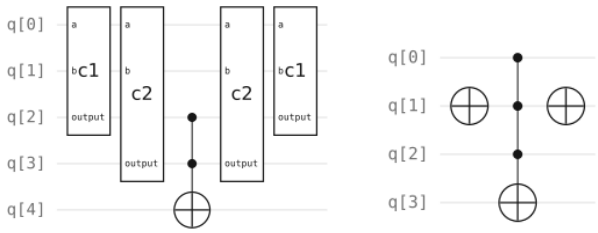
Given that the number of data qubits depends on the problem size (number of graph vertices and colors), qubit reduction and, thus, width improvement can be accomplished by finding a way to reduce the amount of used ancilla qubits. This goal can be obtained by systematically applying what we define as the *AND circuit pattern*. This is a circuit fragment that, as shown in Figure 6a, consists of an MCX gate over a series of input qubits, writing the AND of their values to an output qubit. Subsequently, the input qubits can be erased (by repeating whatever gates set their value) and reused. The writing and erasing of the inputs entails that their values need to be computed twice, increasing the depth; in exchange, all of the input qubits are free to be used again. In the original circuit, this is applied to each iteration of the comparator application with two or more edges.

### B. Depth improvement

As discussed in Section IV, the depth can be reduced by increasing the parallelism of different circuit components. By the term *component*, we mean modular elements that may occur in a circuit multiple times to perform operations on different qubits. Referring to the original approach, we have already discussed about the comparator components (see Figure 3), which focus on comparing pairs of graph vertices and appear in the circuit a number of times that depends on the number of pairs to be compared (see the elements in all blue boxes in Figure 1) and we have shown in Figure 5 how they can be parallelized.

Another portion of the original circuit that could be organized in parallelizable components is the invalid color detection part (see the orange boxes in Figure 1). To this purpose, we define a new component, the Single vertex Color Detector (SNCD), which checks whether a given vertex is of a given color. It could be considered a sub-component of the invalid color detector.

A SNCD is defined as described in Figure 6b. First, a mask made of X gates for the specific color is applied, which turns all of the qubits into ones for that color. Then, an MCX gate detects that configuration. Finally, the X mask is applied again to restore the original state. It is important to note that the MCX activates when the condition is **not** respected. In practice, the invalid color detection circuit fragment in the approach by Saha et al consists of $N$ SNCDs and an application of the AND circuit pattern. We show in the following that SNCDs can be parallelized with other components, thus reducing the depth metrics.




(a) AND circuit pattern on two ancillae. (b) SNCD for color 101.

Fig. 6: Circuit elements.

The performance improvement achieved by the new approaches presented in the following sections stems from the parallelization-friendly application of both comparators and SNCDs that leads to a depth of $O(N)$ instead of $O(N^2)$, even if there are $O(N^2)$ components. The following algorithm returns a group of components that can be executed in a $O(1)$ step. It is sufficient to execute this algorithm repeatedly to subdivide all of the components into such groups, which will be $O(N)$ in number, even for $O(N^2)$ components [1].

1: $available\ vertices \leftarrow all\ vertices$
2: $group \leftarrow empty\ list$
3: $component \leftarrow next\ component(available\ vertices)$
4: **while** $compoent$ **do**

[1] An argument in favor of this claim can be found in https://github.com/Oscar-Belletti/Analyzing-Fixing-and-Optimizing-a-Space-Efficient-Quantum-Circuit-for-the-Graph-K-Coloring-Problem/blob/master/parall.pdf

5: $add\ component\ to\ group$
6: $remove\ component\ vertices\ from\ available\ vertices$
7: $component \leftarrow next\ component(available\ vertices)$
8: **end while**

Each component operates on one or two graph vertices. This means that for each vertex, there is a list of components that operate on it. $next\ component$ returns a random component from list of the vertex with the longest list. This helps keep the lists balanced for bigger groups in later iterations. Another detail to note is that, while in the approach by Saha et al. invalid color detection is accomplished entirely before the color comparison, in our variants, we can mix SNCDs and comparators freely, targeting the maximum possible parallelization.

### C. Minimum Width (MW) Oracle

The original oracle is quite space-efficient regarding ancillae, using only $O(N)$ of them. However, this metric can be improved further by exploiting the AND circuit pattern as described below. Given some number of ancillae:

1: *Apply X gates to the first two ancillae.*
2: *Apply two components to the first two ancillae.*
3: **for all** *subsequent ancillae* **do**
4: *Apply AND circuit pattern with all of the previous ancillae as sources and the current ancilla as target*
5: *After the previous ancillae are reset, compute new values for them again in the same way*
6: **end for**

In the end, an MCX reading from all of the ancillae writes to the qubit dedicated for the phase kickback.

*Complexity:* Unlike the one presented in the original paper, this oracle does not require a number of ancillae determined solely by the number of vertices. Instead, their quantity depends on the number of edges and invalid colors; this alone makes a big difference for large, sparse graphs, but in the worst case, however, it does not matter.

Two ancillae allow for two conditions. For every ancilla added beyond that, the number of conditions that can be checked doubles. This is because each new ancilla is assigned a value by the AND circuit pattern, which alone enforces the same number of values as all of the previous ancillae combined. This makes it evident that the number of ancillae required is $O(logN^2) = O(logN)$ where $O(N^2)$ is the number of conditions in the worst case. Despite this low requirement of ancillae, the width of the overall circuit is the same as in the original circuit since it is dominated by the $N * \lceil log(k) \rceil = O(NlogN)$ data qubits.

The depth, however, is worse than the approach by Saha et al: for two ancillae, the number of components applied to the circuit is two. For each next ancilla, the number of component applications is tripled. This happens because the new AND circuit pattern doubles the number of components' applications until this point, and then new values are calculated for all the previous ancillae; this process takes the same number of component applications as before. In this way, each new ancilla qubit doubles the number of enforced constraints and triples the number of component applications. keeping in mind that the number of requirements is $O(N^2)$, the number $E$ of overall component applications is therefore equal to:

$$E = 2 * 3^A = 2 * 3^{\log_2 O(N^2)}$$
$$= 2 * 3^{\frac{log_3 O(N^2)}{log_3 2}} = O(N^2)^{\frac{1}{log_3 2}} \approx O(N^{3.17})$$

where $A$ is the total number of ancillae. At best, the circuit parallelism factor $P$ is equal to 2 because the components always write to just two ancillae and so cannot be executed in parallel; this does not affect the asymptotic depth, which remains $O(N^{3.17})$. The overall width $\times$ depth complexity is therefore $O(N^{4.17}logN)$

### D. Minimum Depth (MD) Oracle

If we want to minimize depth rather than width, we simply can avoid using the AND circuit pattern so each required component is applied only once. This can be done by dedicating an ancilla qubit for every condition that needs to be enforced, applying all the components, and then applying an MCX gate on all of the ancillae for the phase kickback. Given $A$, the number of ancillae:

1: $i \leftarrow 1$
2: **while** $i < A$ **do**
3: apply component to ancilla $i$
4: $i \leftarrow i + 1$
5: **end while**

*Complexity:* The number of ancillae is, in the worst case, $O(N^2)$. This dominates the data qubits and defines the width. However, it is essential to note that, unlike in the original paper, the number of ancillae is not fixed and depends on the number of conditions that need to be enforced for the specific graph in question. This makes a difference for sparse graphs.

When it comes to depth, the number of component applications $E$ is equal to the number of conditions, therefore $O(N^2)$ in the worst case. The parallelism factor $P$, on the other hand, is $O(N)$ if the components are sorted right. Remember that two or more gates operating on different qubits have a depth of one. This means that two or more components, whether SNCDs or comparators have a depth of $O(1)$ if they operate on different vertices and write to different ancillae. There are only $N$ vertices, so no more than $N$ vertex color detectors, $N/2$ comparators, or a mix of the two can have a depth of $O(1)$. It is possible to execute, on average, $O(N)$ components in $O(1)$ steps with appropriate sorting of the components. The final depth, therefore, is:

$$D = O(E/P) = O(N^2/N) = O(N)$$

The overall width $\times$ depth complexity is: $O(N^3)$.

### E. Balanced (B) Oracle

The balanced oracle aims to minimize the $depth \times width$ complexity. The ancillae are filled iteratively, starting from the last one and moving upwards. At each iteration, all free

| Metrics | Original | MW | MD | B |
|---|---|---|---|---|
| W | $NlogN$ | $NlogN$ | $N^2$ | $NlogN$ |
| D | $N^2$ | $N^{3.17}$ | $N$ | $N$ |
| C | $N^3logN$ | $N^{4.17}logN$ | $N^3$ | $N^2logN$ |

TABLE I: Oracles asymptotic complexities (for brevity O is omitted).

ancillae but the last are written to by single components, then the *last* of the free ancillae is written to by an AND circuit pattern operating on all of the previous ancillae. In the following pseudocode, $A$ represents the number of ancillae, while $last$ represents the last of the clean ancillae.

1: $last \leftarrow A$
2: **while** $last > 2$ **do**
3: apply components to ancillae $[1, last - 1]$
4: apply AND pattern with sources $[1, last - 1]$ and target $last$
5: $last \leftarrow last - 1$
6: **end while**
7: apply two components to the first two ancillae

*Complexity:* The first two ancillae hold one condition each. The third two. The fourth three, and so on. This means that $A$ ancillae can hold up to $1+\sum_i^{A-1} i = O(A^2)$ conditions, which means that $O(N^2)$ conditions require $O(N)$ ancillae. It is important to note however that, unlike in the original paper, the number of ancillae is not fixed and depends on the number of conditions that need to be enforced for the specific graph in question. This makes a difference for sparse graphs. Anyway, the width is dominated by the data qubits and is $O(NlogN)$

Since the AND circuit pattern is not applied recursively, every component is applied at most twice, so the number of component applications $E$ is $O(N^2)$. The parallelization factor $P$ is $O(N)$ because the sorting algorithm from the previous section is applied, and there are generally enough ancillae available to be used as targets. Therefore, the depth is $O(N^2/N) = O(N)$. Note that while this is a quadratic improvement compared to the original paper, it only refers to the depth, not to the number of gates, which we expect to be similar.

The overall width $\times$ depth complexity is $O(N^2logN)$, the best of all the oracles examined in this paper (see Table I and Figure 7 for a comparison among all approaches).

## VI. Validation

| M | Original | MW | MD | B |
|---|---|---|---|---|
| W | $N \log N$ | $N \log N$ | $N^2$ | $N \log N$ |
| D | $N^2$ $\log(\log N)$ | $N^{3.17}$ $\log(\log N)$ | $N$ $\log(\log N)$ | $N$ $\log(\log N)$ |
| C | $N^3(\log N)$ $\log(\log N)$ | $N^{4.17}(\log N)$ $\log(\log N)$ | $N^3$ $\log(\log N)$ | $N^2(\log N)$ $\log(\log N)$ |

TABLE III: Oracles refined asymptotic complexities (for brevity O is omitted, M is for Metrics).

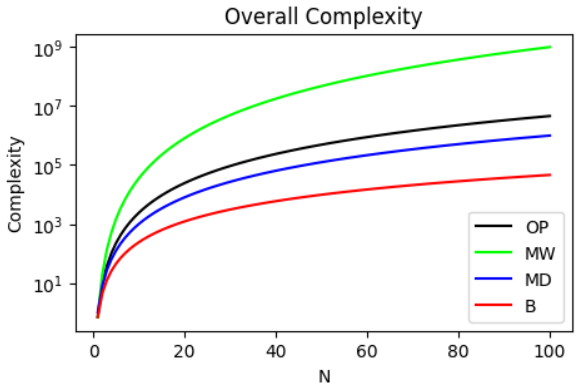



Fig. 7: Semi-logarithmic plot of the asymptotic complexities of the Original Paper (OP), Balanced Oracle approach (B), Minimum Width approach (MW), and Minimum Depth approach (MD).

Since the Grover algorithm is not practical on NISQ devices, we performed tests using noiseless simulations with `qiskit-aer` to verify the theoretical correctness of the algorithms presented. The results show an accuracy above 95% for all tested configurations (graphs with 3–8 vertices and 2–5 colors), with the 5% error being due to the limited floating-point arithmetic precision.

We also provide a complete replication package [2] along with a set of simulations that take into account various kind of noise models derived from actual IBM quantum computers. We have also synthesized circuits for complete graphs of sizes 5 to 50 vertices using various oracles, measuring their widths, depths, and depth × width complexities, the empirical outcomes can be observed in Table II and are aligned to our asymptotic results.

## VII. Complexity metrics refinement

The assumption that an MCX gate has a depth of $O(1)$ and uses no ancillae regardless of the number of inputs is not realistic, but we have relied on it so far to maintain a transpilation-agnostic view, similar to the one in the work by Saha et al. In this section, we re-examine the complexity metrics of the oracles discussed with a more accurate model of the MCX gate. We rely on the realization proposed by [14], where an MCX gate with I control bits uses $O(I)$ ancillae and has a depth of $O(\log I)$.

Comparators and SNCDs are no longer constant time but have a depth of $O(\log I) = O(\log(\lceil \log_2(k) \rceil)) = O(\log(\log N))$ ($I = \lceil \log_2(k) \rceil$, the number of qubits assigned to one vertex). The depth of the comparators and color-detecting circuitry for all oracles is $O(N^2 \log(\log N))$ for sequential execution and $O(N \log(\log N))$ for parallel. In order to execute $O(N)$ of them in parallel, the Balanced and Minimum Depth approaches now require an additional $O(N \log N)$ ancillae, this fact changes the asymptotic number

[2] https://github.com/Oscar-Belletti/Analyzing-Fixing-and-Optimizing-a-Space-Efficient-Quantum-Circuit-for-the-Graph-K-Coloring-Problem

| Metrics | Original paper | | | Minimum width | | | Minimum depth | | | Balanced | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nodes | 5 | 20 | 50 | 5 | 20 | 50 | 5 | 20 | 50 | 5 | 20 | 50 |
| Width | 24 | 133 | 365 | 22 | 111 | 313 | 41 | 531 | 2226 | 24 | 131 | 364 |
| Depth | 225 | 3,929 | 24,653 | 619 | 32,359 | 330,063 | 73 | 301 | 603 | 143 | 749 | 1,643 |
| Complexity | 5,400 | 522,557 | 8,998,345 | 13,618 | 3,591,849 | 103,309,719 | 2,993 | 159,831 | 1,342,278 | 3,432 | 98,119 | 598,052 |

TABLE II: Complexity metrics for one grover iteration for complete graphs of 5, 20 and 50 nodes in the simulation environment.

of ancillae of the Balanced but not its overall width, which is dominated by the data qubits.

The Minimum Depth approach only has one MCX gate operating on the ancillae, the one that writes to the output reading from all $O(N^2)$ of the ancillae. This gate, for all of the oracles, requires at least doubling the number of ancillae, which has no asymptotic effect and a negligible depth of $O(\log A) = O(\log N)$.

The Original Paper oracle has $O(N)$ MCX gates operating on $2...O(N)$ inputs. Asymptotically, they add up to $O(N \log N)$ depth, which is less than the depth of the sequential application of comparators. The Balanced approach originally had a similar number and structure of MCX gates, but to reduce the depth, and considering that we already use $O(N \log N)$ ancillae, we can apply $O(logN)$ MCX gates in parallel on $O(logN)$ groups of $O(N)$ ancillae, which lowers their overall depth to $O(N)$.

The Minimum Width only has MCX gates with $O(\log N)$ inputs, which means that its ancillae are still $O(\log N)$. The comparator/SNCD applications are $O(N^{3.17})$, which leads to a depth of $O(N^{3.17} \log(\log N))$. The MCX gates over the ancillae are also in the order of $O(N^{3.17})$ with $O(\log N)$ inputs and so a depth of $O(N^{3.17} \log \log N)$ at worst. Table III summarises all refined complexities.

## VIII. Conclusion

In our study, we initially tried to analyze and replicate the results of a preexisting quantum gate approach to the K-Coloring presented by Saha et al. [2]; our replication efforts revealed substantial issues in the original implementation, leading to incorrect outcomes. To handle these issues, we proposed a series of revisions that solved the functional errors of the original implementation. At this point, we decided to extend our work beyond simple corrections and started exploring alternative designs to optimize the model. At the end of our work, we obtained a functioning version of the original implementation and three new models able to tackle the K-Coloring problem, all validated by a large set of replicable experiments. The different oracles offer distinct compromises, although asymptotically speaking the Balanced approach has the optimal performance in all metrics as well as the best overall complexity. For what concerns future works, an interesting path to explore is the one leading to the discussion about specific classes of sparse graphs and further optimizations based on their structure. Also, the substitution of the invalid color detection step by a less-than-k oracle able to mark natural numbers smaller than a given $k$ is possible, and its consequences in terms of complexity metrics can be investigated.

## Acknowledgments

This work has been partially supported by the projects CN-HPC-S10 and "QUASAR: QUAntum software engineering for Secure, Affordable, and Reliable systems", grant 2022T2E39C, under the PRIN 2022 MUR program funded by the EU - NGEU".

## References

[1] R. M. Karp, *Reducibility among Combinatorial Problems*. Boston, MA: Springer US, 1972, pp. 85–103.

[2] A. Saha, D. Saha, and A. Chakrabarti, "Circuit design for k-coloring problem and its implementation on near-term quantum devices," in *2020 IEEE International Symposium on Smart Electronic Systems (iSES) (Formerly iNiS)*, 2020, pp. 17–22.

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[4] M. Clerc, "A general quantum method to solve the graph K-colouring problem," 2023. [Online]. Available: https://hal.science/hal-02891847

[5] S. Mukherjee, "A grover search-based algorithm for the list coloring problem," *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–8, 2022.

[6] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, 2014.

[7] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse ising model," *Phys. Rev. E*, vol. 58, pp. 5355–5363, Nov 1998. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.58.5355

[8] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Rev. Mod. Phys.*, vol. 90, p. 015002, Jan 2018. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.90.015002

[9] Z. Tabi, K. H. El-Safty, Z. Kallus, P. Hága, T. Kozsik, A. Glos, and Z. Zimborás, "Quantum optimization for the graph coloring problem with space-efficient embedding," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 2020, pp. 56–62.

[10] C. Silva, A. Aguiar, P. M. V. Lima, and I. Dutra, "Mapping graph coloring to quantum annealing," *Quantum Machine Intelligence*, vol. 2, no. 2, p. 16, Nov 2020.

[11] P. Lima, M. Morveli-Espinoza, G. Pereira, and F. Franga, "Satyrus: a sat-based neuro-symbolic architecture for constraint processing," in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, 2005, pp. 6 pp.–.

[12] J. Watrous, *Quantum Computational Complexity*. New York, NY: Springer New York, 2009, pp. 7174–7201.

[13] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*. USA: Oxford University Press, Inc., 2007.

[14] Y. He, M.-X. Luo, E. Zhang, H.-K. Wang, and X.-F. Wang, "Decompositions of n-qubit toffoli gates with linear circuit complexity," *International Journal of Theoretical Physics*, vol. 56, no. 7, pp. 2350–2361, Jul 2017.