A general quantum method to solve the graph K-colouring problem

Maurice Clerc

Graph colouring: a polynomial complexity quantum algorithm

Maurice Clerc[*]

1 Introduction

In [2] the authors define an heuristics for quantum graph colouring. Their experiments suggest a polynomial-time complexity. We propose here another algorithm (more precisely a quantum circuit) whose classical width$\times$depth complexity is proved to be polynomial.

For a graph with N nodes and for a given number K of colours, the number of qubits to define the binary colouring matrix is $N \times K$ so this algorithm is simply called *NK-method*. Note this is a general method that does not assume any particular structure of the graph.

2 The NK-method

The binary colouring matrix is defined as follows:

- N lines of K columns
- the (i, j) cell is set to 1 iff the colour of the node i is j.
- all the others cells are set to zero.

To build a quantum circuit the algebraic approach of the sub-section 3.1 of the Appendix was a guideline but we do not need to exactly follow it. Informally the steps of the algorithm are, for a given number of colours K:

- Initialize, that is, generate a superposition of all possible colorings.
- Compare to graph. If for all edges the ends are of different colors, mark the coloring as valid, using an ancillary qubit set to $|1>$.
- Amplify valid coloring probabilities
- Measure and propose solutions.

Each step can be performed thanks to a quantum sub-circuit. A Qiskit code ([1]) for simulation is given in the Appendix 3.2. And, of course, for a complete method the above algorithm has to be run for several K values (at most $N-2$, or less by dichotomy).

2.1 Initialization

This step consists of generating a superposition of all possible colorings. The q_i qubits defining these colorings can be presented as a $N \times K$ matrix

$$\mathbb{Q} = \begin{pmatrix} q_0 & q_1 & \cdots & q_{K-1} \\ q_K & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ q_{(N-1)K} & \cdots & \cdots & q_{NK-1} \end{pmatrix}$$

[*]Maurice.Clerc@WriteMe.com

Each of the N lines of K elements must have one 1 and only one, therefore K possible ways. Thus, the total number of coloring matrices is K^N. It is in fact to realize W quantum states (named after one of its inventors Wolfgang Dür).

The algorithm first builds the initialization circuit that generates these W states, one for each node.

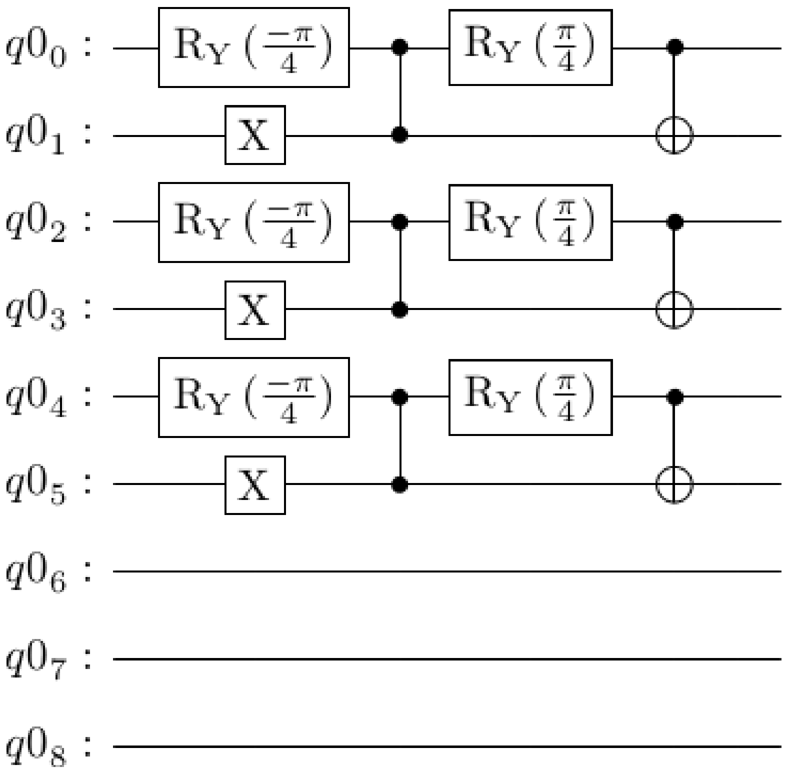


Figure 1: Generation of coloring matrices (W states).

At its output the state vector is then

$\frac{\sqrt{2}}{4}|000010101\rangle + \frac{\sqrt{2}}{4}|000010110\rangle + \frac{\sqrt{2}}{4}|000011001\rangle + \frac{\sqrt{2}}{4}|000011010\rangle$
$+\frac{\sqrt{2}}{4}|000100101\rangle + \frac{\sqrt{2}}{4}|000100110\rangle + \frac{\sqrt{2}}{4}|000101001\rangle + \frac{\sqrt{2}}{4}|000101010\rangle$

Each binary sequence is read from right to left. The first NK bits represent a coloring matrix. For example for the second state we have 010110 which gives

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$$

meaning

- color 2 for the node 1
- color 1 for the node 2
- color 1 for the node 3

which is actually an invalid coloring.

Be careful, these states are only visible in a simulation. On a quantum machine you can only obtain one of them after measurement. As they all have the same probability $\frac{1}{8}$ this would be useless at this stage. Hence, later, the need for an amplification of those corresponding to a valid coloring.

2.2 Mark valid colorings

An ancillary qubit is used for each arc of the graph. Using C2X and X gates each of them is positioned at $|0>$ if both nodes have the same color or $|1>$ otherwise (valid arc).

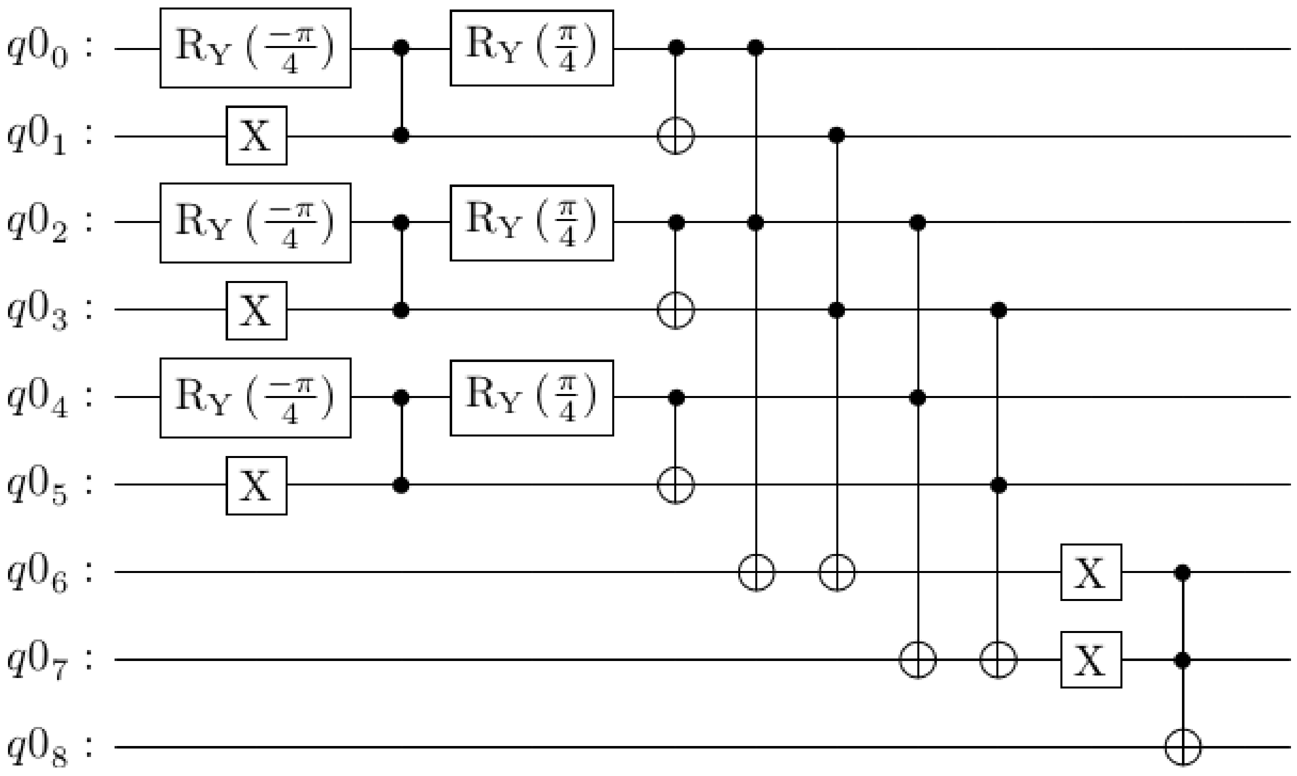


Figure 2: After initialization, marking of valid colorings,

After this step the state vector becomes
$\frac{\sqrt{2}}{4}|000010101\rangle + \frac{\sqrt{2}}{4}|000101010\rangle + \frac{\sqrt{2}}{4}|001010110\rangle + \frac{\sqrt{2}}{4}|001101001\rangle$
$+\frac{\sqrt{2}}{4}|010011010\rangle + \frac{\sqrt{2}}{4}|010100101\rangle + \frac{\sqrt{2}}{4}|111011001\rangle + \frac{\sqrt{2}}{4}|111100110\rangle$
Note that only the last two states correspond to valid colorings (the leftmost bit is set to 1).

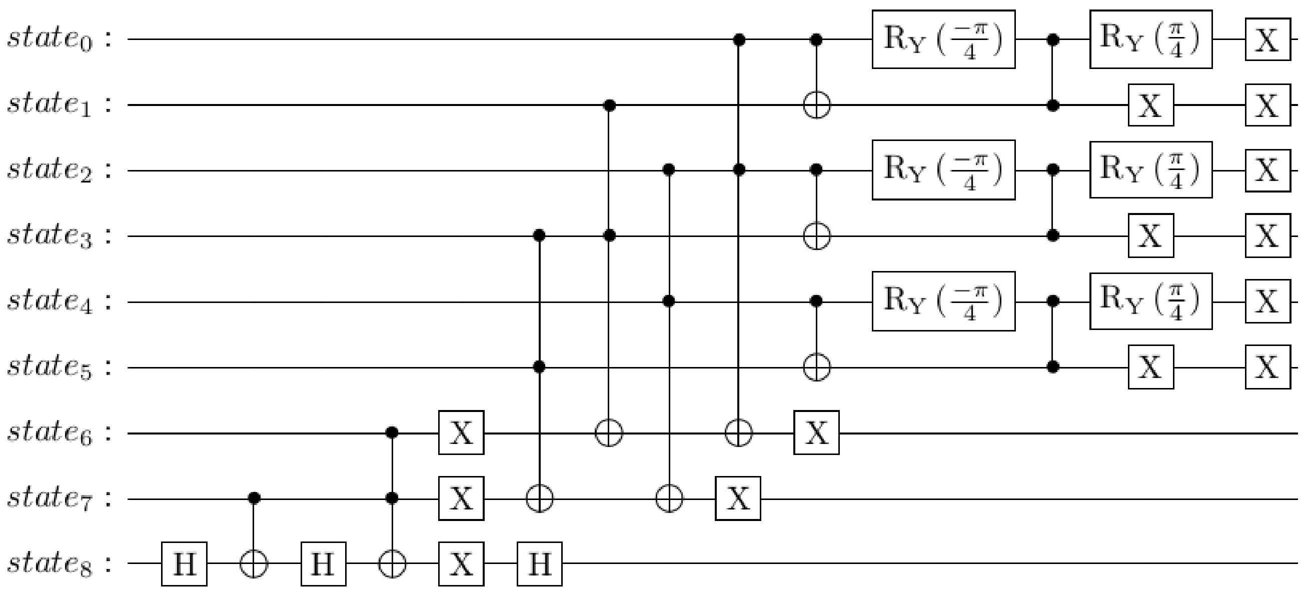
2.3 Amplifying probabilities

Naturally, increasing some probabilities necessarily leads to decreasing others, since their total must be 1. That is why a more complete denomination of this step is actually amplification-diffusion.

Here we use the classic Grover's method ([4]). The circuit is generated by the Qiskit code given in appendix and presented in detail in figure [3. We notice the quasi-symmetry due to the principle of the method, which applies a first circuit, then its inverse after a CnX gate controlling here the last qubit by all the others.

But in practice we can consider this circuit as a black box with all the qubits input, especially since this decomposition is not unique and depends on the tools used (machine and software).

Finally, the histogram of the figure 4 is obtained. For this very simple example, the probabilities of invalid colorings are completely null (compared to the accuracy of the computer) and therefore do not appear. This will no longer be the case for the other examples. The two solutions are (omitting auxiliary qubits) 011001 and 100110. To be read from right to left in two-bit sequences, from where (10,01,10) and (01,10,01) that is to say the colorings (2,1,2) and (1,2,1). Figure 5 represents the second.



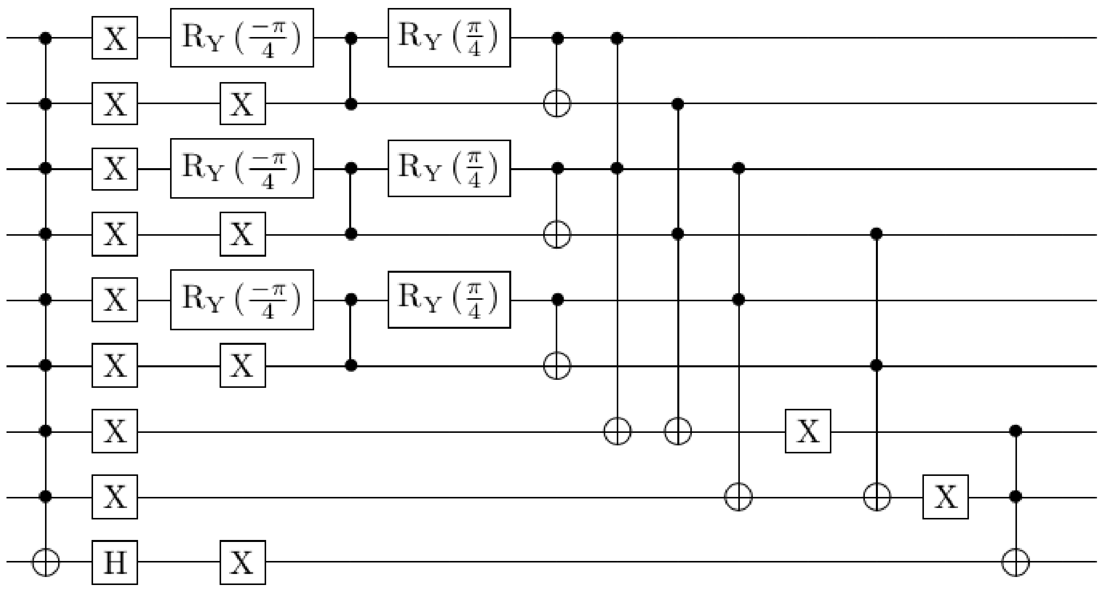

X
H
$R_Y\left(\frac{-\pi}{4}\right)$
$R_Y\left(\frac{\pi}{4}\right)$

Figure 3: Detail of the amplification circuit of the states whose last qubitis at$|1>$ (Grover's method).

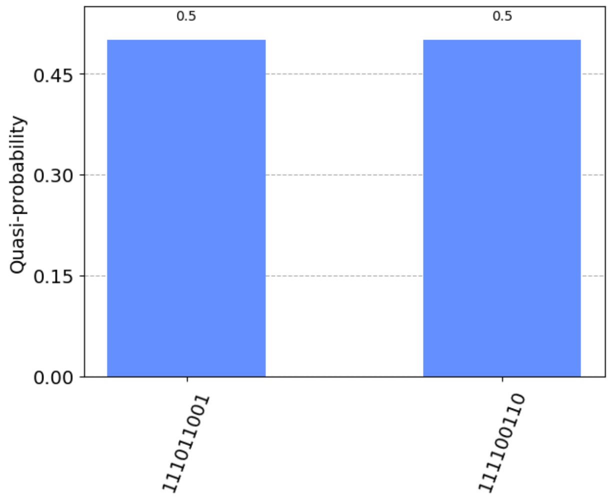


Figure 4: The two solutions for the 3-nodes linear graph.

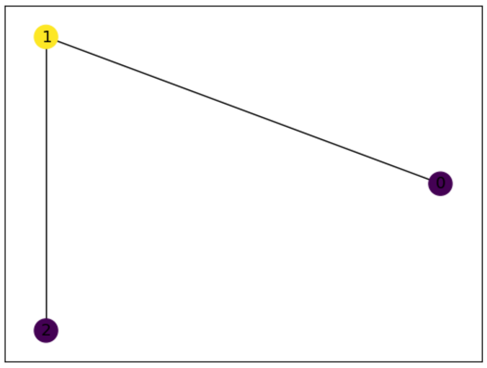


Figure 5: One of the two valid colorings of the 3-nodes lineaar graph.

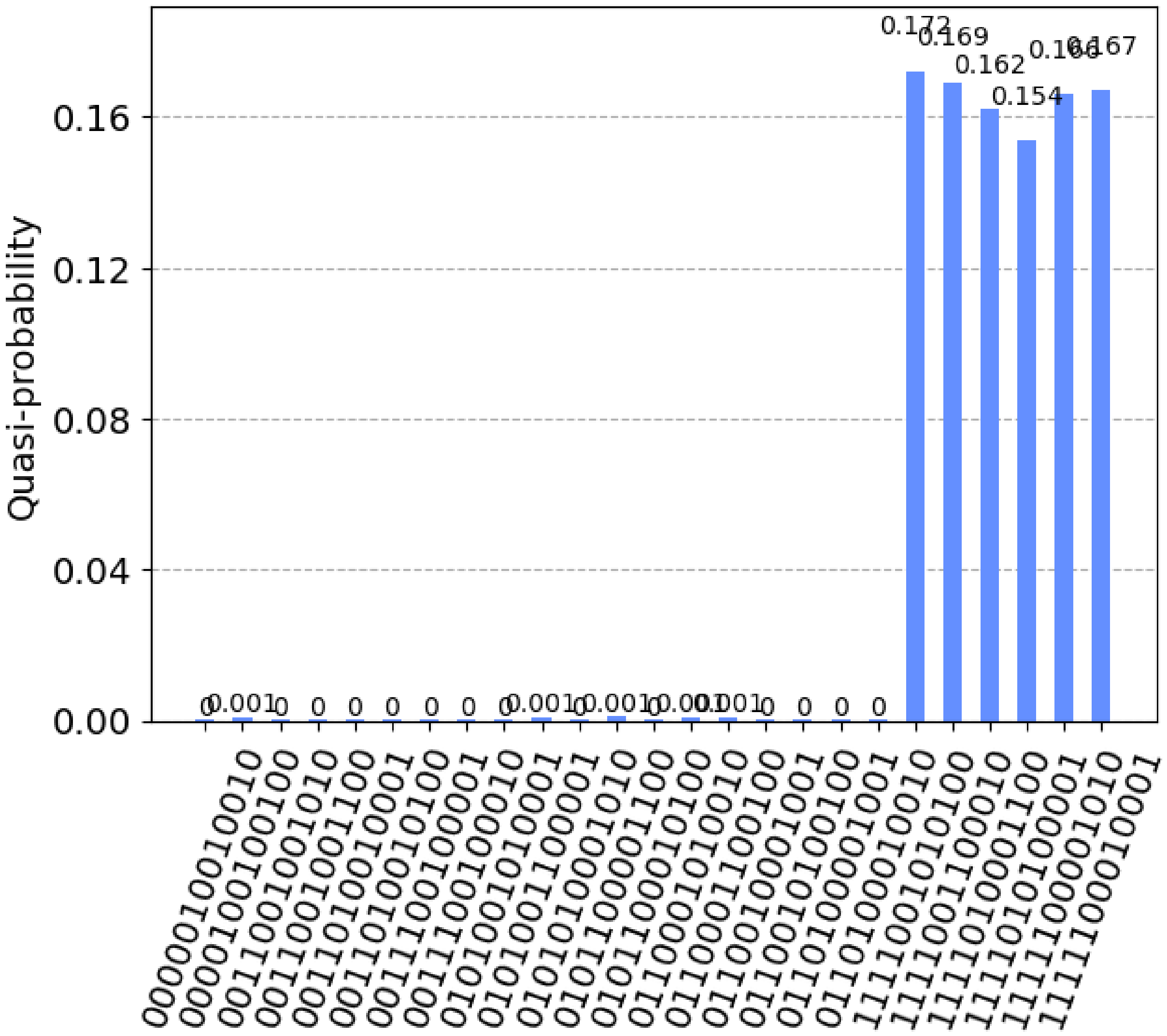


Figure 6: The six solutions for the triangular graph.

2.4 Examples

Some examples where it becomes difficult to display the circuit and state vectors. They are nevertheless very simple, because, as we will see, the number of qubits required increases quadratically with the size of the graph.

2.4.1 The triangular graph

If we are looking for a bicoloriage the answer of the algorithm is «No solution». With three colors, it finds the six solutions, as shown in figure 6. Their probabilities are well amplified, but the others are no longer strictly null. So by being (very) unlucky, the measure might not give a valid solution.

2.4.2 Five nodes, five arcs, three colors

Note G5_3_3 the graph in question. A valid coloring found by the algorithm is given in the figure [fig:G5_3_3 colored]. In fact the 24 possible are found simultaneously[1],essential difference of with conventional algorithms, but there are two disadvantages:

[1]Assuming the algorithm runs a real quantum machine, of course.

- The algorithm already requires 18 qubits.
- Even when amplified, the maximum probability becomes smaller and smaller with the number of possible colorings. So, sometimes it corresponds to an invalid coloring. In this case I had to execute the algorithm twice in order for this maximum probability (of the order of 0.03) to be that of a valid coloring.

Incidentally, given the number of states, the histogram would be unreadable.

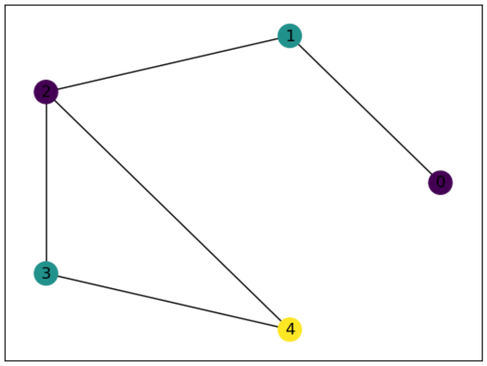


Figure 7: Graph G5_3_3, a valid 3-coloring.

2.4.3 Five nodes, eight arcs, four colors

Note this graph G5_8_4. Here you need 29 qubits. After two iterations of amplification the algorithm finds the 48 solutions with a probability of approximately 0.01 for each, for example 11111111110000001010000100001 which, by omitting the 6 auxiliary bits, is decoded from right to left in $(1000, 0100, 0010, 1000, 0001)$, the colors $(1, 2, 3, 1, 4)$ pour les nœuds $(0, 1, 2, 3, 4)$.

2.5 Complexities

For $K \geq 3$ determining whether any graph is K-colorizable is a NP-Complete problem. In practice this means that on a conventional computer (Türing machine) our simulations made with Qiskit require resources (memory, calculation time) growing exponentially with the size of the graph. So my 6 Gb laptop cannot even solve locally the 3-coloring of a graph with 4 nodes and 5 edges (error message: *Insufficient memory*). For larger graphs the code given in appendix then uses a remote IBM machine.

We can estimate theoretical complexities, according to different measures. They are all polynomial of low degree. Other approaches are exponentially complex, even for the decision problem of colorability ([3]). Still others suggest a polynomial complexity ([2]), but only experimentally.

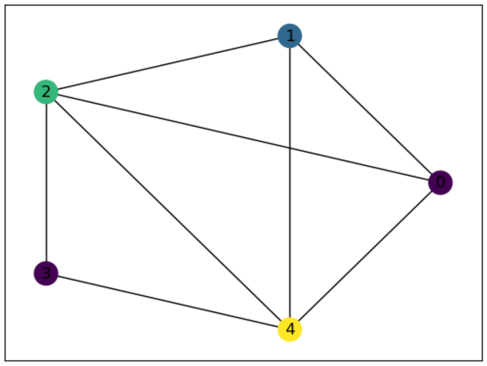


Figure 8: Graphe G5_8_4, un 4-coloriage valide

2.5.1 Number of qubits

Let A be the number of arcs of the graph. We use

- NK qubits for coloring matrix superposition.
- As many auxiliary qubits as arcs. Maximum $\frac{N(N-1)}{2}$ for a complete graph.
- 1 ancillary qubit to mark valid colorings.

In all

$$Q = NK + A + 1 \tag{1}$$

As $K \leq N$ the complexity in number of qubits is then

$$C_q = \frac{3N^2}{2} - \frac{N}{2} + 1 \tag{2}$$

i.e. $O\left(N^2\right)$ with, moreover, a small coefficient (3/2)

2.5.2 Number of gates

The table 1 shows the distribution of gates by type.

	Initialization	Marking	Amplification
H			4
X	N	A	$2\left(2A+1+N+NK\right)$
CX	$N(K-1)$		$2N\left(K-1\right)+1$
C_2X		AK	$2AK$
C_AX			2
CZ	$N(K-1)$		$2N\left(K-1\right)$
R_y	$2N\left(K-1\right)$		$4N\left(K-1\right)$
C_QX			1
Maximum ($K=N$ and $A=N(N-1)/2$)	$4N^2-3N$	$\frac{1}{2}\left(N^3-N^2\right)$	N^3+11N^2-8N+9

Table 1: Number of gates.

For amplification the number of gates used must be multiplied by the number of iterations (see [subsec:Number of iterations]).

2.5.3 Size of the circuit

A classic measure of the complexity of a quantum circuit is the product width$\times$depth. The width is actually just the number of qubits used. The depth is the longest path in this circuit, between an input and an output. Its precise value depends on how the real circuit is realized. Some systems count for example one unit for a Z gate and others three (because $Z = HXH$). Nevertheless it is always a linear transformation on the number of gates and, therefore, this depth is of the order of $O\left(N^3\right)$.

It can also be requested in the Qiskit code. The table 2 and the attached figure, where complete graphs are considered, confirm a cubic complexity.

So, even with the worst strategy possible, consisting in successively trying 2 colors, then 3, ... then $N-1$, the complexity of searching for an optimal coloring is in $O(N^4)$.

More cleverly, we can proceed by dichotomy on the number of colors K:

- try $K_1 = 2$
- if no solution, $K_2 = N-1$.
- if no solution, N colors are required, one different for each node.

Nodes	Width	Depth	Complexity
2	6	30	180
3	13	45	585
4	23	60	1380
5	36	75	2700
6	52	90	4680
7	71	105	7455
8	93	120	11160
9	118	135	15930
10	146	150	21900

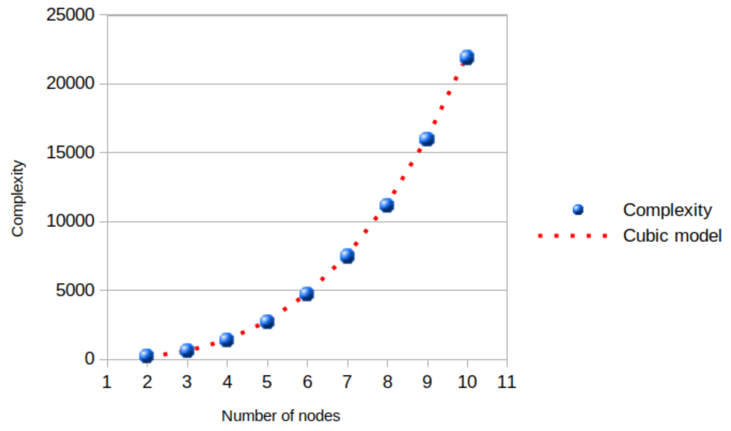


Table 2: Complexity of the circuit for the NK method. It grows polynomially (cubic) with the size of the graph.

- if a solution, we know that the optimum is in $]K_1, K_2]$and we try the integer K_3 closest to its middle
- etc.

At most we must try $k$values with 2^{k-1}\leq N-3\leq2^{k}. The complexity then becomes

$$O\left(\left(\ln\left(N-3\right)+1\right)N^{3}\right) \tag{3}$$

2.5.4 Number of iterations

The Grover method requires a number of iterations for maximum amplification. It doesn't take too much or too little. The theoretical formula is

$$i_{optim}=\frac{\pi}{4}\sqrt{\frac{number\ of\ \ states}{k}} \tag{4}$$

The parameter k is the number of solutions, which is *a priori* unknown. In many applications the number of states is 2^{Q} where, remember, Q is the number of qubits. Then i_{optim} increases very quickly with the size of the problem.

Here, thanks to initialization (2.1) the number of states is reduced to K^{N}. Moreover k is generally in the order of $O\left(N!\right)$. Thus, for the complete graph we have

$$i_{optim}=\frac{\pi}{4}\sqrt{\frac{N^{N}}{N!}}$$

So, by applying the Moivre-Stirling formula to $N!$, we see that i_{optim} still increases as $\left(\frac{e^{N}}{N}\right)$, which remains fast. For example, for $N=10$ we have to do 42 iterations, but for $N=20$ we already get a value of the order of 5000.

However, this is not so considerable if we note that the probability of finding a solution by random search is only $2,32 \times 10^{-8}$. On the other hand it becomes prohibitive if the execution of the algorithm is only simulated on a conventional computer, because, in this case, the state vector must be kept in memory and it contains about 10^{26} elements...

3 Appendix

The mathematical analysis of the first section below has inspired the code of the second one. However it can be ignored for the code is self-sufficient to compute the complexity.

3.1 A bit of algebra

We can define the validity conditions of a coloring by a purely matrix approach. The C coloring is coded by its binary matrix and the adjacency matrix G of the graph.

We define.

$$\Sigma = C \oplus C \tag{5}$$

$$G_K = \sqcup_K G \tag{6}$$

$$\Gamma = G_K \odot \Sigma \tag{7}$$

where $\oplus$ is the outer sum, $\sqcup_K$ the 'horizontal' concatenation operator repeated K times and $\odot$ the element wise product.

Then the indicator is

$$v_{C,G} = \max(\Gamma) \tag{8}$$

so that the validity is given the condition

$$v_{C,G} < 2 \tag{9}$$

that is equivalent to

$$v_{C,G} = 0 \vee v_{C,G} = 1 \tag{10}$$

easier to verify by manipulations of qubits. You can also use a modified outer sum so that Γ is binary. The condition is then simply

$$\Gamma = \mathbf{0} \tag{11}$$

where $\mathbf{0}$ is a null matrix.

Octave/Matlab©algorithm

```
    Sigma=outerSum(C);
    % Variant:
    % Sigma=max(0,Sigma-1);
    GK=repmat(G, K,1);
    Gamma=GK.*Sigma;
    valid=max(Gamma(:))<2 ;
    % Variants:
    % valid=max(Gamma(:))<1; % valid=Gamma=zeros(size(C));
    ...
    function Sigma=outerSum(C)
     % This a simplified version
     % The complete version allows two input matrices
    [~,K]=size(C);

        Sigma=[];
            for k=1:K
               Ck=meshgrid(C(:,k));
               Ck=Ck+Ck';
               Sigma=[Sigma Ck];
            end
    end
```

Proof

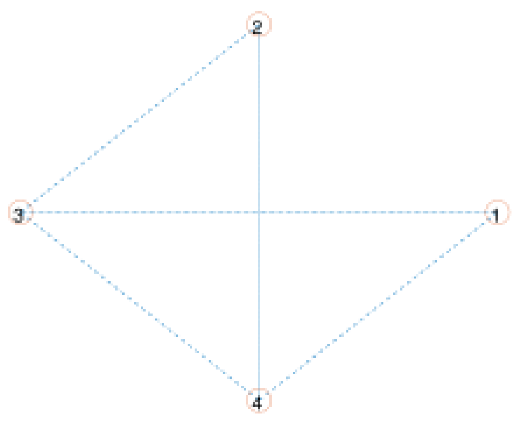
1. The only possible values in Σ are 0, 1 and 2.
2. The only possible values in Γ are 0, 1 and 2.
3. Let's consider $\Sigma(n,m)$. We have $m=(k-1)N+j$, $j\in[1,2,\cdots,N]$.
   (a) If $\Sigma(n,m)=0$ neither the node n nor the node j have the color k.
   (b) If $\Sigma(n,m)=1$ the node n has the color k and the node j another one or vice versa.
   (c) If $\Sigma(n,m)=2$ the nodes n and j have both the color k (and this is of course also true when $n=j$).
4. The value 2 is 'eliminated' in Γ if and only if $\Gamma(n,m)=0$, i.e. $G(n,j)=0$, which means 'no edge between n and j'.
5. Therefore if there is no 2 value in Γ then the coloring C is valid.

3-coloring example (see the figure 9)

$$G=\begin{pmatrix}0&0&1&1\\0&0&1&1\\1&1&0&1\\1&1&1&0\end{pmatrix}$$

$$C=\begin{pmatrix}1&0&0\\1&0&0\\0&1&0\\0&0&1\end{pmatrix}$$

$$\Sigma=\begin{pmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\0&0&0&0\end{pmatrix}+\begin{pmatrix}1&1&0&0\\1&1&0&0\\1&1&0&0\\1&1&0&0\end{pmatrix}$$


(a) The graph

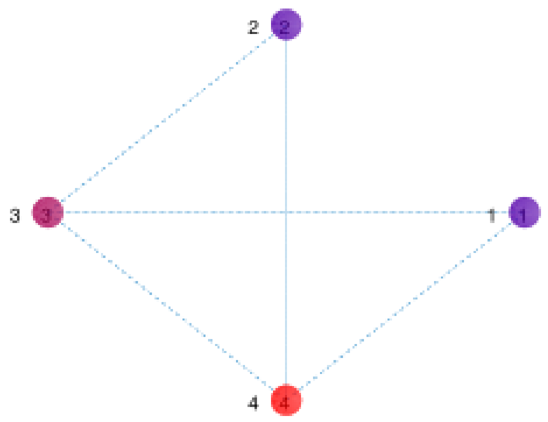

(b) A valid 3-coloring

Figure 9: Optimal coloring of a 4-nodes 5-edges graph.

$$\sqcup \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\sqcup \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 2 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 2 & 2 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 2 \end{pmatrix}$$

$$G_K = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Note that the 2 values in Σ are eliminated thanks to the element wise product with G_K. Let's look carefully how Σ is built in order to find where are the 2 values. For each color k

- duplicate 'horizontally' N times the corresponding column of the matrix C in order to build a square matrix Σ_k
- add it to its transpose

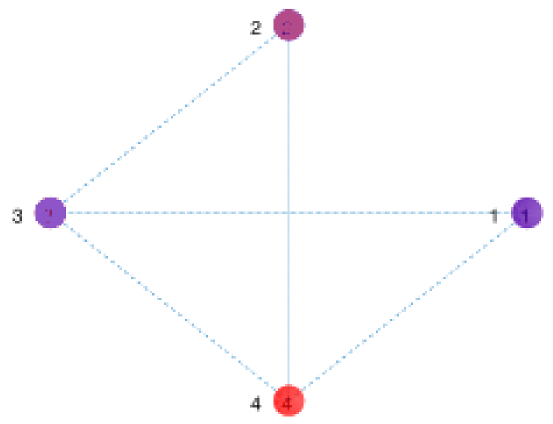


Figure 10: Un 3-coloriage invalide. Les nœud 1 et 3 ont la même couleur, alors qu'ils sont reliés par un arc.

- if $k > 1$,concatenate 'horizontally' the result to the previous one.

So a 2 value is generated iff we have

$$\Sigma_k(i,j) = \Sigma_k(j,i) = 1 \tag{12}$$

But $\Sigma_k(i,j) = \Sigma_k(i,1)$ and $\Sigma_k(j,i) = \Sigma_k(j,1)$, indicating that the same color k is assigned to the nodes i and j.So, after a long detour, the rule is finally very simple:

There is a 2 value in Σ iff two nodes have the same color

Here is an example of an invalid coloring of our 4-nodes 5 edges graph (see the figure 10).

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Gamma = \begin{pmatrix} 0 & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

3.2 Qiskit code

```
NK-method
# k-coloring of a graph
# N = number of nodes
# K = number of colors
'''
In this method we use a coloring matrix:
1 line for each node
1 column for each color
one and only one 1 in each line, other values 0
It means we need at least NK qubits to describe such a matrix
On my laptop I can consider only very small graphs
```

```
but in principle it should work for any N and K.
Badly coded, I know.
Used here under Anaconda / Jupyter
Warnings:
1) The algorithm searches a solution for a given number of colors (K).
For a complete resolution it should loop on different K.
The worst approach is to check for K=2, then 3, ... N-1
but you can use dichotomy.
2) The algorithm looks for a valid coloring (for each edge the two ends have different colors).
If there is none it may propose a wrong one.
So you have to check the proposed non null solutions. Easy to do in polynomial time.
'''
import numpy as np
from qiskit import *
import qiskit.tools.jupyter
get_ipython().run_line_magic('qiskit_version_table', ")
get_ipython().run_line_magic('qiskit_copyright', ")
get_ipython().run_line_magic('matplotlib', 'inline')
#from qiskit import Aer
#from qiskit.providers.aer import AerError, QasmSimulator
#backend_sim = Aer.get_backend('statevector_simulator')
#backend_sim = Aer.get_backend('extended_stabilizer')
backend_sim = Aer.get_backend('qasm_simulator')
nNodes=3
nColors=3 # nColors <= nNodes. This is the number of colors to try.
nc=nColors+1 # For each node, nColors qubits that will be measured, + one ancillary
nn2=round((nNodes-1)*nNodes/2) # Number of pairs of different nodes and of possible edges
sc=round(nc*nNodes) # Number of qubits to skip before the ancillary qubits for pairs and edges
sg=round(nc*nNodes + nn2) # Beginning of the list of qubits that describe the graph
nqbits=sc + 2*nn2   # Total number of qubits
# Create a Quantum Circuit
q = QuantumRegister(nqbits)
c=ClassicalRegister(nColors*nNodes) # To measure to "extract" the coloring matrices
qc = QuantumCircuit(q,c)
# Add the graph (binary list of nNodes*(nNodes-1)/2 elements, because the graph is symmetric
# and there is no i=>i edges. Set to |1> the qubits corresponding to an edge.
'''
3 nodes, needs 3 colors
0 1 1
1 0 1
1 1 0
=> 1 1 1 for qc.x(sg), qc.x(sg+1), qc.x(sg+2)
3 nodes, needs 2 colors
0 1 0
1 0 1
0 1 0
=>  1 0 1
2 nodes
2 n
0 1
1 0
=> 1
4 nodes, needs 3 colors
0 0 1 1
0 0 1 1
1 1 0 1
```

```
1 1 1 0
=> 0 1 1 1 1 1
'''
'''
qc.x(sg) # Set to |1> iif there is an edge 1->2
qc.x(sg+1) #  if N>=3, edge 1->3
qc.x(sg+2)  # if N=3, edge 2->3. If N>3, edge 1-4
'''
# Complete graph, for test
for n in range(nNodes):
    qc.x(sg+n)
# ------------------------------------------Generate a coloring matrix
# Initialisation
# Hadamard gate for qubits that represent the coloring matrix
s=0
for n in range(nNodes):
    for k in range(nColors):
        qc.h(s+k)
    s=s+nc
#----------------------------
# Constraints
# A 1 and only one 1 in each of the coloring matrix
s=0
for n in range(nNodes):
    for k in range(nColors-1):
        for l in range(k+1,nColors):

           # Eliminate 11
            qc.ccx (s+k,s+l,s+nColors)
            qc.cx (s+nColors,s+k)
            qc.reset(s+nColors)

    # Eliminate 0* (no color assigned to the node n)
    for k in range(nColors): qc.x(s+k) #  if 0 => 1. Not needed if you can use
                                       # a negative multicontrolled gate

    cb=list(range(s,s+nColors) )
    qc.mcx (cb,s+nColors)

    for k in range(nColors): qc.x(s+k) # if 1 => 0

    qc.cx (s+nColors,s+nColors-1)
    qc.reset(s+nColors)
    s=s+nc

 # At this point, if we measure, we find coloring matrices (nNodes lines, nColors columns),
# one and only one 1 in each line (a node does have a color, and only one)
print('end of coloring matrices')
# Switch the ancillary qubits corresponding to pairs of nodes that have the same color
for k in range(nColors):
    s=nc*nNodes
    for n1 in range(nNodes-1):
        for n2 in range(n1+1,nNodes):
            n11=nc*n1+k # If q[n11]=|1> it means the node n1 has the color k
            n22=nc*n2+k # If q[n22]=|1> it means the node n2 has the color k
            qc.ccx(n11,n22,s) # If same color k, set s to |1>. Notice it can happens
                              # at most for one k
            s=s+1
            #print([n11,n22])
 # At this point, if we measure the (nNodes-1)*nNodes)/2 ancillary qubits,
 # we get binary strings,
# in which 1 means "same color" for n1 and n2
print('end of pairs of nodes')
# Compare to the graph.
for n in range(sc,sc+nn2): # For each pair of nodes
    # If same color and there is an edge "destroy" (set to |0*>) the coloring
    for node in range(nNodes):
        qnode=nc*node
        qnc=qnode+nColors
        for k in range(nColors):
            cb=[n,n+nn2,qnode+k]
            qc.mcx (cb,qnc)
            cb=[n,n+nn2,qnc]
            qc.mcx (cb,qnode+k)
            qc.reset(qnc)
print('end of compare to graph')
# Measure (only the qubits describing the coloring matrices)
cb=0
for n in range(nNodes):
    s=n*(nColors+1)
    for k in range(nColors):
        qb=s+k
        qc.measure(qb,cb)
        cb=cb+1
print('end of measures')
print(qc)
d=qc.depth()
print("Circuit depth: ",d)
print("Circuit width: ",nqbits)
print("Complexity: ",d*nqbits)
# Execute the circuit on a statevector simulator
#job = execute(qc, backend_state,shots=1000)
# Execute the circuit on the qasm simulator.
# Quick on small graphs, but memory error for 4 nodes, 3 colors
job = execute(qc, backend_sim,shots=1000)
#  This method should handle more qubits, but is awfully slow
#qobj = assemble(qc, backend=QasmSimulator(), shots=1000)
#job = QasmSimulator().run(qobj, backend_options={'method': 'extended_stabilizer'})
result=job.result()
#print(result)
print('end of execute')
# Grab the results from the job.
counts = result.get_counts() # result.get_counts(qc)
print(counts)
print('The solutions, if any,  are given by the strings that are not 0*')
if nColors>2:


    print('Please check: some of them may be a coloring with LESS colours')
print('You can rebuild the color matrix by reading the string from right to left')
print('(each row has ',nColors,' bits)')
```

References

[1] Qiskit, https://qiskit.org/.

[2] Alex Fabrikant and Tad Hogg. Graph Coloring with Quantum Heuristics. Edmonton, Alberta, Canada, 2002.

[3] Kazuya Shimizu and Ryuhei Mori. Exponential-Time Quantum Algorithms for Graph Coloring Problems. *Algorithmica*, June 2022.

[4] Wikipedia. Grover's algorithm, May 2023. Page Version ID: 1157281681.