Quantum Algorithms for Matchings and Vertex-Colorings in Graphs

Author: Nabiha Asghar

1 Introduction

1.1 Motivation

One of the main aims of theoretical computer science is to reduce the computational complexity of various problems by exploiting their structural properties. Graphs are found at the heart of a wide range of natural and synthetic structures, and many, if not most, problems in the complexity classes P and NP can be formulated graphically. In addition, there is a plethora of practical applications of graph theory, such as navigation systems, large-scale network design, optimization of processes and path planning in robotics. It is imperative, therefore, to study the impact of the superior information processing power of quantum resources on graph-theoretic algorithms.

1.2 Goals & Layout

This report is a self-contained[1] discourse on some recently published quantum algorithms for two standard graph-theoretic problems, matchings and vertex-colorings. The main goal is to study these algorithms in detail, and to compare them to their classical counterparts and analyse the differences. While presenting the work of other authors in this report, we place a high premium on improved presentation of their arguments, and endeavour to provide proofs and necessary details that are omitted in the original papers intentionally or otherwise.

The layout of this report is as follows. Section 2 defines vertex-colorings and gives a standard classical algorithm for obtaining a 2-coloring of a given graph. It goes on to describe two quantum algorithms for obtaining 2-colorings, and a partial (that is, incomplete) quantum algorithm to obtain a 3-coloring of a graph. This is followed by some comparison and analysis. Section 3 focuses on matchings and states a classical algorithm to obtain bipartite matchings in a graph. This is followed by a quantum algorithm for the same problem, and some comparison between their computational complexity. Section 4 gives some concluding remarks.

[1]We assume that the reader is familiar with the basic concepts of quantum infomation processing.

2 Vertex-Colorings in Graphs

This section deals with algorithms for vertex-colorings in graphs. Section 2.1 defines the vertex-coloring problem, and also gives some terminology. Section 2.2 gives a classical algorithm for 2-colorings, and Section 2.3 describes two quantum algorithms for 2-colorings given by D'Hondt [3]. Section 2.4 analyses the differences between the classical and quantum algorithms for 2-colorings. Section 2.5 gives a partial (incomplete) algorithm for 3-colorings given by D'Hondt [3], and describes why this problem may be hard to solve, even with quantum resources.

2.1 Definitions & Terminology

Let $G = (V, E)$ be an arbitrary undirected graph, where $|V| = n$ and $|E| = m$.

A connected graph with no cycles is called a **tree**. A k-vertex-coloring of a graph (referred to as **k-coloring** throughout this report) is an assignment of k colors to the vertices of G such that the end-vertices of any edge receive different colors. More formally, it is a function $c : V \to \{1, ..., k\}$ such that $c(u) \neq c(v)$ for all $(u, v) \in E$. It is worthwhile to note straight away that if G has a loop, then it does not have any coloring. Also, having multiple copies of some edge does not affect the way this graph may be k-colored. Hence, we may assume the graph is simple[2].

The **'k-Coloring Problem'** in graph theory is defined as follows. Given a graph G and a positive integer k, obtain a k-coloring of G, and if no k-coloring exists, output '*No k-coloring exists*'. For $k = 2$, this problem is in P, but for higher values of k, it is in NP. In particular, for $k = 3$, this is an NP-complete problem.

A quantum system is denoted by $|\psi\rangle$, using the standard Dirac's bra-ket notation. A system of n **qubits** is described by a superposition of 2^n terms and a system of n **qutrits**[3] corresponds to 3^n superposition terms.

For qubits, the unitary operators we use are the Pauli flip matrix X and the Hadamard matrix H, defined as

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

For qutrits, the Pauli flip matrix X is

$$X = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

[2]A simple graph is a graph that does not contain any loops or multi-edges.

[3]A qutrit is a unit of quantum information that can exist in 3 possible states, as opposed to a qubit that can exist in 2 possible states.

It is easy to check that X maps $|0\rangle$ to $|1\rangle$, $|1\rangle$ to $|2\rangle$ and $|2\rangle$ to $|0\rangle$. In the case of qutrits, we only need to know the Hadamard matrix's behaviour on $|0\rangle$, which is given by

$$H|0\rangle = \frac{1}{\sqrt{3}}(|0\rangle + |1\rangle + |2\rangle).$$

One can verify that in both n-qubit and n-qutrit systems, the operator $H^{\otimes n}$ creates an equally weighted superposition of all possible terms. As a general convention throughout this report, for both the systems, any of these two operators X and H with a subscript j implies that the operator in question acts upon the j-th qubit/qutrit. For example, in a 4-qubit system, $X_3|0110\rangle = |0100\rangle$.

A **quantum measurement**, denoted by M, is defined by a set of orthogonal projectors that sum to the identity matrix of the Hilbert space under consideration. The effect of a measurement on the quantum state is a projection onto one of the subspaces determined by the projectors, and is a probabilistic process. The measurement outcome is just taken to be a label distinguishing what projection occured. The measurements we encounter in this section typically split up the Hilbert space into two, where one projection corresponds to an intended action and the other does not, situations labelled with outcomes 0 and 1 respectively.

2.2 A Classical Algorithm for 2-Coloring

Given an arbitrary graph G, the goal is to determine, using classical resources, whether G is 2-colorable, and to obtain a 2-coloring if there exists one. Observe that if each component of G is 2-colorable, then G is 2-colorable too. So, a 2-coloring of a disconnected G can be obtained by 2-coloring each component first and then taking the union of these colorngs. We may, therefore, assume throughout this section that G is connected.

The main ingredient we require for our classical algorithm is an ordering of the edges of the graph by what is known as the *depth first traversal* (DFT) of the given graph. Intuitively, a DFT is a means of restructuring a graph as a tree with certain edges added, the latter indicating the non-tree-like structure. To construct a DFT, one first selects an arbitrary vertex to be the root of the tree, then exploring as far as possible along each branch before backtracking. We now give a precise definition of the procedure.

DFT Algorithm [7]
Given a connected, undirected graph $G = (V, E)$ (typically in the form of an adjacency list):

```
1. Select a fixed but arbitrary root $r \in V$ and mark it as *visited*.
2. For each edge $e$ in the adjacency list of $r$, execute the following steps:
3.     If $w$, the other end-vertex of $e$, has not been *visited* yet,
4.         Label $e$ as a *tree edge*.
5.         Recursively call the DFT algorithm with $w$ as the root.
```

```
6.     Else,
7.         Label $e$ as a back edge.
```

It is not hard to see that the tree edges form a spanning tree of G, called a DFT tree. As the name suggests, each back edge always connects two nodes, one of which is the ancestor of the other in the DFT tree. Therefore, each back edge creates *exactly* one cycle in the DFT tree. Lastly, each vertex of G is processed at most once, and each edge of the graph G is traversed at most once, which implies that the DFT algorithm takes $O(n+m)$ time and hence it is in P.

We are now ready to give the classical algorithm for 2-coloring. The main idea is to use DFT to determine whether G has any cycles of odd length or not. Note that if G has a cycle, then the only way to color the vertices of this cycle with 2 colors is to assign them the colors red and blue alternately. But if this cycle has an odd number of vertices, we always end up getting two consecutive reds or blue. In fact, a standard graph-theoretic result says that a graph is 2-colorable if and only if it has no odd cycles. So the 2-coloring algorithm works as follows. We make the DFT label the vertices as '1' or '2' as they are visited for the first time via tree-edges. Every time we reach a vertex v that has a back-edge leading to an ancestor w, we check to see whether the labels of v and w are the same. If they are the same, it means that the cycle created by this back edge is odd. Thus, the graph cannot be 2-colored and the algorithm stops. On the other hand, if the labels of v and w are different, we continue. The label-keeping and back-edge-checking can be implemented by keeping an additional array of length n, in which a value '0' indicates that the vertex has not been visited, a value '1' indicates that the vertex has been visited and labelled '1', and the value '2' indicates that the vertex has been visited and labelled '2'. The algorithm is given below. Essentially, it is a modified version of DFT.

Classical Algorithm for 2-Coloring

Given a connected, undirected graph $G=(V,E)$ (typically in the form of an adjacency list):

```
1. Keep a global variable called $L$. Initialize it with the value '1'.
2. Select a fixed but arbitrary root $r \in V$ and assign it the label $L$.
3. For each edge $e$ in the adjacency list of $r$, execute the following steps:
4.     If $w$, the other end-vertex of $e$, is unlabelled,
5.         Label $w$ as $L+1 \pmod 2$.
6.         Set $L$ to be $L+1 \pmod 2$, and recursively call the DFT algorithm with $w$ as root.
7.     Else,
8.         Check $w$'s label. If it is the same as $r$'s label, stop and return 'No 2-coloring exists'.
```

If all the vertices of G are labelled at the end of one run of this algorithm, we obtain a 2-coloring of G, otherwise G has an odd cycle and hence no 2-coloring exists. The running time analysis is similar to that of DFT. Each vertex is processed at most once and each edge is traversed at most once, hence the algorithm takes $O(n+m)$ time.

2.3 Quantum Algorithms for 2-Coloring

We now give two quantum algorithms, given by D'Hondt [3], that output possible 2-colorings of an arbitrary undirected, connected graph, or the answer that there do not exist any. These algorithms rely heavily on the DFT structure of a graph and the distinction between tree edges and back edges; this is primarily why the running time of these algorithms is no better than the classical running time.

The main strategy is to first assume that no edges exist in the graph, and generate the set of all possible 2-colorings. Then each edge is considered one by one as a constraint on this set, and the elements of the set that violate this constraint are removed. This process is repeated until either all edge constraints have been successfully considered, or an edge constraint is discovered that is violated by all the elements of the set, in which case no solution exists.

Given a connected, undirected graph $G = (V, E)$, we represent each of the n vertices of the given graph G with a qubit, and the two available colors are encoded into the possible basis states of the qubits, $|0\rangle$ (red) and $|1\rangle$ (blue). We start with the state $|0\rangle^{\otimes n}$, which corresponds to assigning the color red to all the vertices of the graph. Next, we apply the Hadamard operation to each qubit to obtain a superposition of all possible colorings of this graph when edges are ignored:

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle. \tag{1}$$

For example, let J be a graph where $V(J) = \{i, j, k\}$ and $E(J) = \{e_{ij}, e_{jk}, e_{ik}\}$. Let the first, second and third qubits represent i, j and k respectively. Then for J, we obtain the state $|\psi\rangle = \frac{1}{2\sqrt{2}}(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |001\rangle + |110\rangle + |111\rangle)$.

Now we see how to add edges one by one and change $|\psi\rangle$ accordingly. Let an edge between two vertices a and b be denoted by e_{ab}. Observe that for any $e_{ab} \in E$, $|\psi\rangle$ should not have any superposition terms containing $|00\rangle_{ab}$ or $|11\rangle_{ab}$. That is, we need to eliminate all the superposition terms where the a'th qubit and the b'th qubit have the same value. This must be done such that all other terms remain intact. To do this, we use the following unitary operation,

$$E_{ab} = \begin{bmatrix} 1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

called the *tree edge operation* for any $e_{ab} \in E$. It can be verified that $E^{\dagger}_{ab}E_{ab} = I$, so it is indeed a unitary matrix. Its action on basis states is

$|00\rangle_{ab} \to \frac{1}{\sqrt{2}}(|01\rangle_{ab} + |00\rangle_{ab})$
$|01\rangle_{ab} \to \frac{1}{\sqrt{2}}(|01\rangle_{ab} - |00\rangle_{ab})$
$|10\rangle_{ab} \to \frac{1}{\sqrt{2}}(|10\rangle_{ab} + |11\rangle_{ab})$

$|11\rangle_{ab} \to \frac{1}{\sqrt{2}}(|10\rangle_{ab} - |11\rangle_{ab})$.
Let us do this for our example graph J. For edge e_{ij}, we get $E_{ij}|\psi\rangle = \frac{1}{2}(|010\rangle + |011\rangle + |100\rangle + |101\rangle)$. So we have obtained a state in which the first two qubits are different in each superposition term; we did this by eliminating precisely those superposition terms that conflicted with the edge under consideration. Thus, we have successfully processed one edge; so far so good.

Note that this is true for arbitrary edges, *provided that all color combinations for qubits a and b are present in the superposition, and that too with equal amplitudes.* The latter is ensured by the form of the unitary, which eliminates *exactly* half of the superposition terms and changes the amplitudes such that every term in the new state has the same amplitude. The former is ensured if one end-vertex of the edge-operation is still unprocessed, i.e., if the edge under consideration is a tree edge. Therefore, in order to ensure that this operation works for as many edges as possible, we need to process the edges in the DFT order. Returning to our example, we note that e_{jk} is also a tree edge, so we compute $E_{jk}(E_{ij}|\psi\rangle) = E_{jk}(\frac{1}{2}(|010\rangle + |011\rangle + |100\rangle + |101\rangle)) = \frac{1}{\sqrt{2}}|010\rangle + |101\rangle$. The remaining edge e_{ik} is not a tree edge. This means that in our newly obtained state, not all color combinations for the i'th and k'th qubit are present, hence we cannot apply the tree edge operation.

It follows that back edges would have to be handled in a different way. This is because each time we encounter a back edge, its endpoints have already been assigned a fixed color via tree edge operations. Thus, as expected, back edges only determine whether 2-colorings exists for the graph at all. We handle back edges as follows. For each back edge e_{ab}, consider the projective measurement M_{ab} with the projection operators O_{ab} and $I - O_{ab}$, where

$$O_{ab} = |01\rangle_{ab}\langle 01| + |10\rangle_{ab}\langle 10|,$$

$$I - O_{ab} = |00\rangle_{ab}\langle 00| + |11\rangle_{ab}\langle 11|.$$

We assume that a measurement outcome '0' corresponds to a projection on the subspace determined by O_{ab} and a measurement outcome '1' corresponds to a projection on the subspace determined by $I - O_{ab}$. Clearly, everytime we do the projective measurement M_{ab} for a back edge e_{ab}, we want the outcome to be '0', which implies that the qubits a and b are different in every term of the superposition. Note that this measurement is deterministic if the edges are processed in the DFT order; there is exactly one possible outcome for each back edge measurement operation. In our example, applying the measurement operation M_{ik} on $\frac{1}{\sqrt{2}}|010\rangle + |101\rangle$ yields '1' obviously, hence J has no 2-coloring.

We are ready to concisely state the algorithm now.

Quantum Algorithm # 1 for 2-colorings
Given an undirected, connected graph $G = (V, E)$:

1. Create the state in (1).
2. For each edge $e_{ab} \in E$ in DFT order, execute the following steps:

```
3.      If $e_{ab}$ is a tree edge,
4.          Apply $E_{ab}$.
5.      Else,
6.          Apply $M_{ab}$.
7.          If measurement outcome is ‘1’, stop and return *‘No 2-coloring exists’*.
8. Measure the final superposition to obtain an arbitrary 2-coloring of the graph, or use the
   superposition of all 2-colorings of $G$ for further processing.
```

The running time analysis is similar to the classical algorithm. Since we consider edges in the DFT order, each vertex is processed at most once and each edge is traversed at most once, hence the algorithm takes $O(n + m)$ time.

D'Hondt [3] also gives a completely measurement-based algorithm for 2-colorings. The idea is that back edge measurements can be executed for tree edges as well, except that now the measurement outcome would not be deterministic. This is because there is more than one possible outcome for each tree edge. To make it deterministic, a correction operation is inserted immediately after the measurement if a ‘1’ is obtained. The key observation is that when the operation M_{ab} yields outcome ‘1’ for some edge tree $e_{ab} \in E$, we can flip the value of the qubit b (by applying the Pauli X matrix) to obtain the right coloring for vertices a and b; this makes the measurement deterministic and results in the same state that would be obtained if E_{ab} was used. Unfortunately, this does not allow us to process the edges arbitrarily. We still need to follow the DFT order, otherwise the qubit corrections alter the correlations fixed earlier. Using this approach, the algorithm is given below. Here, s_{ab} is the outcome of the measurement and $X^{s_{ab}}$ denotes the execution of the Pauli X matrix conditioned on s_{ab}. Clearly, the running time is still $O(n + m)$.

Quantum Algorithm # 2 for 2-colorings

Given an undirected, connected graph $G = (V, E)$:

```
1. Create the state in (1).
2. For each edge $e_{ab} \in E$ in DFT order, execute the following steps:
3.      If $e_{ab}$ is a tree edge,
4.          Apply $X^{s_{ab}} M_{ab}$.
5.      Else,
6.          Apply $M_{ab}$.
7.          If measurement outcome is ‘1’, stop and return *‘No 2-coloring exists’*.
8. Measure the final superposition to obtain an arbitrary 2-coloring of the graph, or use the
   superposition of all 2-colorings of $G$ for further processing.
```

2.4 Classical vs. Quantum 2-Coloring: A Comparison

The most glaring similarity between the classical and the quantum algorithms we have seen is the use of depth first traversal on the edges of the graph. In both the algorithms, the difference between tree edges and back edges is exploited. Moreover, the role of back edges is the same; they determine whether any 2-colorings exist for the graph or not. This is why the quantum algorithm's complexity is no better than its classical counterpart, even though in general we expect a quantum algorithm to provide at least polynomial speedup. D'Hondt [3] states, "We stress that we have not started out by looking at the classical solutions; rather the fact that we unearthed the same results suggests that it is a typical property of the problem rather than the solution".

On the other hand, an argument can be made that the outputs of the classical and quantum algorithms are not *quite* the same, hence it is hard to compare the time complexities of these algorithms. Indeed, the classical algorithm outputs one possible 2-coloring, whereas the quantum algorithm outputs a superposition of all possible 2-colorings, thus demonstrating its power of parallel processing. The final entangled state could potentially be used to derive some properities of all 2-colorings via some other quantum computation, though it is not yet clear how that should be done.

D'Hondt's approach is novel in the sense that it aims to discover and establish a typical set of useful quantum operations and procedures, that could be used as primitives in higher level quantum computations. Thus, the goal is to start a systematic, bottom-up development of quantum computations. The bottom-up approach exploits the structure of the problem, rather than trying to utilize the parallelisability to look for a solution by brute force. Thus, it proceeds through structure-dependent operations, potentially paving the way for more fundamental speed-ups.

2.5 A Partial Quantum Algorithm for 3-Coloring

We describe a partial algorithm for 3-coloring, given by D'Hondt [3]. The algorithm is incomplete mainly because finding a 3-coloring of a graph and/or determining whether one exists at all is an NP-complete problem. However, it is interesting to see how and why the k-coloring problem for $k = 2$ suddenly becomes so much more difficult to solve for $k = 3$, even for a quantum computer.

Given an undirected, connected graph G, we represent each of the n vertices as a qutrit now, because each vertex can be colored with three possible colors, 0, 1 and 2. Again, we start with the state $|0\rangle^{\otimes n}$ and employ the Hadamard operation for qutrits to generate all possible 3-colorings, ignoring the edges:

$$|\psi\rangle = H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{3^n}} \sum_{x\in\{0,1,2\}^n} |x\rangle. \tag{2}$$

As one would expect, we start looking at each edge one by one, in DFT order, as a constraint in a measurement-based way. The measurement M_{ab} for each tree edge $e_{ab} \in E$ is a generalization of what we saw earlier; we split the Hilbert space into two subspaces, given below, associated with the allowed and disallowed colorings of vertices a and b respectively, and project onto these subspaces.

$$O_{ab} = |01\rangle_{ab}\langle 01| + |02\rangle_{ab}\langle 02| + |10\rangle_{ab}\langle 10| + |12\rangle_{ab}\langle 12| + |20\rangle_{ab}\langle 20| + |21\rangle_{ab}\langle 21|,$$
$$I - O_{ab} = |00\rangle_{ab}\langle 00| + |11\rangle_{ab}\langle 11| + |22\rangle_{ab}\langle 22|.$$

Again, we assume that a measurement outcome '0' corresponds to a projection on the subspace determined by O_{ab} and a measurement outcome '1' corresponds to a projection on the subspace determined by $I - O_{ab}$. However, one main difference that we have now, from the 2-coloring case, is that, for any two qutrits a and b, 6 of the 9 terms correspond to outcome '0' and the remaining correspond to '1'. That is, the probability for the state $|00\rangle + |01\rangle + |02\rangle + |10\rangle + |11\rangle + |12\rangle + |20\rangle + |21\rangle + |22\rangle$ to collapse to the subspace cedetermined by O_{ab} is $\frac{2}{3}$, and the probability for it to collapse to the subspace determined by $I - O_{ab}$ is $\frac{1}{3}$. Previously, these probabilities were $\frac{1}{2}$ and $\frac{1}{2}$. Thus, the state obtained after an outcome of '1' is

$$\frac{1}{\sqrt{3^{n-1}}}(|00\rangle + |11\rangle + |22\rangle) \sum_{x\in\{0,1,2\}^{n-2}} |x\rangle. \tag{3}$$

Hen, we cannot simply insert a correction operation (of flipping the second qutrit via Pauli X matrix for qutrits) whenever we get the outcome '1'. The reason is that flipping the second qutrit of this state would give us

$$\frac{1}{\sqrt{3^{n-1}}}(|01\rangle + |12\rangle + |20\rangle) \sum_{x\in\{0,1,2\}^{n-2}} |x\rangle,$$

but what we actually want is

$$\frac{1}{\sqrt{2(3^{n-1})}}(|01\rangle + |12\rangle + |20\rangle + |02\rangle + |10\rangle + |21\rangle) \sum_{x\in\{0,1,2\}^{n-2}} |x\rangle. \tag{4}$$

Thus, we need to use some unitary operation other than X. Note that applying the operation $C = \frac{1}{\sqrt{2}}(X + X^2)$ to (3) gives us (4), which is exactly what we want. However, it can be verified that C is not a unitary operation. So we use the next best thing, which is

$$C = \frac{2}{3}X + \frac{2}{3}X^2 - \frac{1}{3}I. \tag{5}$$

Proving that this unitary operation achieves the best possible result is an optimization problem; we omit the proof here. Applying (5) to (3) gives us

$$\left(\frac{1}{\sqrt{3^{n-1}}}\frac{2}{3}(|01\rangle + |12\rangle + |20\rangle + |02\rangle + |10\rangle + |21\rangle) \sum_{x\in\{0,1,2\}^{n-2}} |x\rangle\right)$$

$$- \left(\frac{1}{\sqrt{3^{n-1}}}\frac{1}{3}(|00\rangle + |11\rangle + |22\rangle) \sum_{x\in\{0,1,2\}^{n-2}} |x\rangle\right). \tag{6}$$

We observe that the success and failure probabilities (that is, the probabilities of obtaining a projection onto O_{ab} and $I - O_{ab}$ respectively) for this new state are $\frac{8}{9}$ and $\frac{1}{9}$ respectively, as opposed to the older probabilities $\frac{2}{3}$ and $\frac{1}{3}$. Thus, by enough repetitions, the success probability can be made arbitrarily close to 1. Hence, we have found an arbitrarily 'good' way of dealing with tree edges.

Now that we have a state which contains only those superposition terms that are compliant with all tree edges, we focus our attention on treating back edges. In the case of 2-coloring, all the terms in the superposition were either compatible or incompatible with a given back edge; this is not the case here. Thus, we cannot use a measurement deterministically. Neither can we use M_{ab} measurements for back edges, because this may destroy the correlations established earlier. This is where the problem becomes significantly hard, compared to 2-colorings.

One possible way to deal with this tough situation is to try and use unitary transformations to eliminate the 'bad' superposition terms for each back edge. This may require a lot of case analysis. For instance, we can count the number, say p, of vertices between the endpoints of a back edge in the DFT tree (that is, if e_{ab} is the back edge and C is the cycle created by it, let p be the number of vertices in this cycle minus 2) and treat each possible value of this number as a separate case. The first case is $p = 1$; let us analyze this case. Consider the three vertices i, j, k and a back edge e_{ik} in some graph $G = (V, E)$, and let $|\psi\rangle$ be the quantum state after G's tree edges have been processed. Then, half the superposition terms in $|\psi\rangle$ are compatible with the back edge and half are not. We can exploit this fact by defining the unitary back edge operation E_{ijk} as mapping all basis states onto themselves, except the following:

$$|c, c\oplus 1, c\rangle_{ijk} \to |c, c\oplus 1, c\oplus 2\rangle_{ijk} - |c, c\oplus 1, c\rangle_{ijk}$$

$$|c, c\oplus 1, c\oplus 2\rangle_{ijk} \to |c, c\oplus 1, c\oplus 2\rangle_{ijk} + |c, c\oplus 1, c\rangle_{ijk}$$

$$|c, c\oplus 2, c\rangle_{ijk} \to |c, c\oplus 2, c\oplus 1\rangle_{ijk} - |c, c\oplus 2, c\rangle_{ijk}$$

$$|c, c\oplus 2, c\oplus 1\rangle_{ijk} \to |c, c\oplus 2, c\oplus 1\rangle_{ijk} + |c, c\oplus 2, c\rangle_{ijk}$$

where $c \in \{0, 1, 2\}$ is some color and $\oplus$ denotes addition modulo 3. We can see from these four mappings that all terms of the type $|c, c\oplus 1, c\rangle_{ijk}$ and $|c, c\oplus 2, c\rangle_{ijk}$ are eliminated, thus leaving us with only those terms that are compatible with the edge e_{ik}. So, we have the right operator for the case $p = 1$. However, note that this operator is operating on three vertices rather than two. This points to the fact that the vertices in the path ijk are already entangled through three edge operations and thus we can no longer process the endpoints of the back edge separately from the intermediary ones. D'Hondt [3] conjectures that, for back edges connecting longer chains, all intermediary vertices will play a role in the associated back edge operations. Whether this strategy works for other values of p is still an open question.

Even though this algorithm is partial and the chances of its completion seem slim, there are a few interesting things to be noted and some lessons to be learned. The problems we face in this algorithm give us a flavour of *why* this problem is so hard. Also, it links the hardness of the problem to the (poorly understood) concept of entanglement in a very fundamental way. It also suggests why there can be no brute force quantum algorithms for problems in NP and why it is important to exploit the structure of the problem, if we hope to create efficient quantum algorithms for hard problems.

3 Matchings in Graphs

In this section, we present classical and quantum algorithms for bipartite matchings in graphs. Section 3.1 gives some definitions and preliminaries. Sections 3.2 describes a well known classical polynomial time algorithm that computes a bipartite matching of a given graph; Section 3.3 gives a quantum algorithm for the same problem that achieves polynomially faster time by making use of Grover's Search Algorithm. Section 4.4 gives some concluding remarks.

3.1 Definitions & Preliminaries

A graph in which the vertices can be partitioned into two sets A and B, so that all the edges join a vertex in A to a vertex in B, is called a **bipartite graph**. A **layered network** is a graph whose vertices are ordered into a number of layers, and whose edges only go from the i'th layer to the $(i + 1)$'th layer. A **matching** in a graph is a set M of edges such that no two edges in M share a common endpoint. A vertex v is called a **covered vertex** in the matching if there exists an edge in the matching that is incident to v, otherwise v is called a **free vertex**. An **augmenting path** in G is a path such that the edges in this path alternately lie in and outside the current matching; moreover, this path starts and ends with a free vertex. If there exists an augmenting path P in G, then the size of the current matching M can be increased by one, by taking the symmetric difference of M and the edges of P, denoted by $M \triangle E(P)$. This procedure is called **augmentation** of a matching.

The **Maximum Bipartite Matching Problem** asks for a matching of maximum size when the input is an undirected, bipartite graph (note that the fastest known algorithm for finding a maximum matching in a general graph has a running time of $O(\sqrt{n}m)$ [6], where n is the number of vertices in the graph and m is the number of edges). In this section, however, we will give a classical algorithm that runs in time $O(n^{2.5})$. Even though the input to a Maximum Matching Problem is an undirected graph, we will work with directed graphs at some stages of our algorithms. To access directed graphs, the following two black-box models are considered:

- *Adjacency Model:* the input graph is specified by an $n \times n$ Boolean matrix A, where $A[v, w] = 1$ if and only if there is a directed edge from v to w in the graph (that is, $(v, w) \in E$).
- *List Model:* the input is specified by a set S of n arrays, one for each vertex. The array for a vertex v is called N_v. Thus, $S = \{N_v : v \in V\}$, where d_v denotes the length of N_v and $1 \leq d_v \leq n$. For each vertex v, an entry of N_v is either a label of a neighbour of v or a hole. Thus, $N_v \setminus \{holes\} = \{w : (v, w) \in E\}$. The reasons for including holes in these adjacency lists will become clear later.

As we will see later, the quantum algorithm consists mostly of classical steps, with one exception; it uses the generalized Grover's search algorithm for database searching. The **Generalized Grover's Search Algorithm** [4] finds k items in a search space of size q in time $O(\sqrt{kq})$. An additional time $O(\sqrt{q})$ is needed to detect that there are no more solutions to be found; this term is important when $k = 0$. The Grover's search algorithm may output an incorrect answer with a constant probability k, and it may be used a polynomial number n^c of times in the quantum algorithm for bipartite matching, so we need to repeat it a few times to ensure that the success probability is close to 1. It turns out that $O(\log n)$ repetitions are needed to make the probability of wrong answer less than $\frac{1}{n^{c+1}}$. To see this, note that we can assume $k \leq 0.5$. Then

$$\text{Prob[Grover's alg. gives a wrong ans. after } (c{+}1)log(n) \text{ trials]} \leq k^{log(n^{c+1})} \leq \frac{1}{2^{log(n^{c+1})}} = \frac{1}{n^{c+1}}.$$

Therefore,

$$\text{Prob}[n^c \text{ runs of Grover's alg. gives a wrong answer}] \leq \frac{n^c}{n^{c+1}} = \frac{1}{n}.$$

This gives us

$$\text{Prob}[n^c \text{ runs of Grover's alg. gives the right answer}] \geq 1 - \frac{1}{n}.$$

Thus, as stated prevously, we need to repeat Grover's algorithm $O(\log n)$ times to make the success probability very close to 1. This increases the running time of the quantum bipartite matching algorithm by a $\log n$ factor.

3.2 Classical Algorithm for Bipartite Matching

We are given an undirected, bipartite graph $G = (V_1 \cup V_2, E)$ and the task is to find a matching of maximum size. We assume $m \geq n$, since zero-degree vertices can be eliminated in time $O(n)$. The following algorithm, known as the Hopcroft-Karp Algorithm [5], achieves this task in time $O(n^{2.5})$.

Hopcroft-Karp Algorithm for Bipartite Matching

1. Set M to an empty matching.
2. Create a new directed graph $H = (V', E')$ from G as follows. Add two dummy nodes a, b to G. Put a directed edge from a to every node in V_1. Put a directed edge from each node in V_2 to b. Finally for any undirected edge $(v, w) \in E \setminus M$, orient it from v to w, and for any undirected edge $(v, w) \in M$, orient it from w to v.
3. Construct a layered subgraph H' of H.
4. Find a maximal set of $\mathcal{P}$ vertex-disjoint paths from a to b in H' as follows.
   (a). Apply the DFT algorithm with a as the root, and stop when b is reached. Let P be the found path. If no path is found, go to (5).
   (b). Delete P from H'. Go back to step (a).
5. Augment the matching M by each path in $\mathcal{P}$ (that is, compute $M \triangle \mathcal{P}$).
6. If $\mathcal{P} \neq \varnothing$, go to (2), otherwise output the matching M.

The output of this algorithm is clearly a matching. The correctness of the algorithm follows from the follows two facts.

Fact 1: A matching M is maximum if and only if there is no augmenting path in G.
Proof: Suppose M is maximum and for the sake of contradiction, let P be an augmenting path joining u and v. Then $M \triangle E(P)$ is a matching that covers all nodes covered by M, plus u and v. Thus, M is not maximum, a contradiction. Conversely, suppose there is no augmenting path in G and for the sake of contradiction, assume M is not maximum. Let M' be a matching such that $|M'| > |M|$. Let $J = M \triangle M'$. Then J is a subgraph of G in which every vertex has degree at most 2. Thus, J is a union of disjoint paths and even-length cycles. Since the cycles are even, there must exist a path Q in J with more edges of M' than M. Clearly Q is an augmenting path, a contradiction. $\square$

Fact 2: The minimal length of an augmenting path is increased by at least one after every iteration.
Proof: Let k be the length of the shortest augmenting path with respect to M in some iteration. Now consider the shortest path π with respect to $M \triangle \mathcal{P}$. If π does not intersect any path from $\mathcal{P}$, its length has to be larger than k, otherwise its existence contradicts the assumption that $\mathcal{P}$ is a maximal set and that it contains shortest disjoint augmenting paths. $\square$

It follows from Facts 1 and 2 that the algorithm looks for augmenting paths of all possible sizes, and stops if when it cannot find any more of them. Hence, the matching obtained is maximum.

Let us analyze the running time of this algorithm. Step 2 takes no more time than $O(m) \approx O(n^2)$. Constructing the layered subgraph takes time $O(n^2)$ (it uses Breadth First Search algorithm, which processes each edge at most once. We omit the details of the Breadth First Search algorithm here. Interested readers can look it up on the web or in any standard textbook for algorithms). The repeated application of DFT algorithm also takes total time $O(n^2)$, because each edge is processed at most once, after which it is deleted. Thus, one iteration takes time no more than $O(n^2)$. We now show that there are at most $O(\sqrt{n})$ iterations, from which the $O(n^{2.5})$ bound for the entire algorithm follows.

Fact 3: Let M^* be a maximum matching and let M be any matching in $G = (V, E)$. If the length of the shortest augmenting path with respect to M is k, then $|M^*| - |M| \leq \frac{n}{k}$.
Proof: Consider the graph $G' = (V, M \triangle M^*)$. It contains at most $|M^*| - |M|$ augmenting paths with respect to M. The length of each of these paths must be at least k. Thus, the total length of these paths is no more than n, so by Pigeonhole Principle, there are no more than $\frac{n}{k}$ paths. $\square$

Fact 4: The Hopcroft-Karp algorithm runs in at most $O(\sqrt{n})$.
Proof: By Fact 2, the length of the shortest augmenting path increases in every iteration. Thus, after $O(\sqrt{n})$ iterations, the shortest augmenting path will have length at least $O(\sqrt{n})$. From Fact 2, we know that there are at most $O(\sqrt{n})$ augmenting paths left in the graph. So, at most $O(\sqrt{n})$ more iterations will occur before the algorithm stops. Therefore, the total number of iterations is at most $2\sqrt{n}$. $\square$

3.3 Quantum Algorithm for Bipartite Matching

The quantum algorithm we give here is due to Ambainis and Špalek [1]. It is exactly the same as the Hopcroft-Karp algorithm, except that it achieves a polynomial speedup by constructing the intermediate layered subgraph using Grover's algorithm as a subroutine, and also finds all the augmenting paths in one iteration.

We first show how to construct the layered subgraph quantumly. Given a connected, directed black-box graph $G = (V, E)$ and a starting vertex $a \in V$, we want to assign layers $L : V \to \mathbb{N}$ to its vertices such that $L(a) = 0$ and $L(y) = 1 + \min_{x:(x,y) \in E} L(x)$ otherwise. The following algorithm is a quantum implementation of the Breadth First Search algorthm, that computes layer numbers for all vertices.

1. Set $L(a) = 0$ and $L(x) = \infty$ for $x \neq a$. Create a one-entry queue $W = \{a\}$.
2. While $W \neq \varnothing$,

a. take the first vertex x from W,
b. find by Grover's search all its neighbours y with $L(y) = \infty$, set $L(y) := L(x) + 1$, and append y to W, and
c. remove x from W.

Theorem 3.3.1: The algorithm assigns layers in time $O(n^{1.5} \log n)$ in the adjacency model and in time $O(\sqrt{nm} \log n)$ in the list model.
Proof: In the adjacency model, every vertex gets found at most once (from one of its ancestors) in step 2b. This takes time $O(\sqrt{n})$, because Grover's algorithm finds one element in a row of length n in the adjacency matrix of the graph. Moreover, discovering that a vertex has no descendants costs the same. Thus, the total time for finding all the vertices is $O(n\sqrt{n})$. Since we repeat Grover's algorithm $O(\log n)$ times, the claimed running time of $O(n^{1.5} \log n)$ follows.

In the list model, step 2b for a vertex v takes time $O(\sqrt{n_v d_v} + \sqrt{d_v})$ where n_v is the number of vertices inserted into W when processing v (the additional $\sqrt{d_v}$ is for the case when v has no neighbours). Thus, the total time for processing all the vertices is $\sum_{v \in V}(O(\sqrt{n_v d_v} + \sqrt{d_v}))$. Note that $\sum_{v \in V} n_v \leq n$ and $\sum_{v \in V} d_v \leq m$. By Cauchy-Schwarz inequality, we have

$$O\left(\sum_{v \in V} \sqrt{n_v d_v}\right) \leq O\left(\sqrt{\sum_{v \in V} n_v}\sqrt{\sum_{v \in V} d_v}\right) = O(\sqrt{nm})$$

and

$$O\left(\sum_{v \in V} \sqrt{d_v}\right) = O\left(\sum_{v \in V} \sqrt{d_v}\sqrt{1}\right) \leq O\left(\sqrt{\sum_{v \in V} d_v}\sqrt{\sum_{v \in V} 1}\right) = O(\sqrt{nm}).$$

This shows that processing all the vertices takes time $O(\sqrt{nm})$. Since we repeat Grover's algorithm $O(\log n)$ times, the claimed running time of $O(\sqrt{nm} \log n)$ follows. □

Theorem 3.3.2 [1]: Quantumly, a maximal bipartite matching can be found in time $O(n^2 \log n)$ in the adjacency model $O(n\sqrt{m} \log n)$ in the list model.
Proof: The quantum algorithm is the same as the Hopcroft-Karp algorithm, except that it finds all augmenting paths in one iteration in time $O(n^{1.5})$ (and $O(\sqrt{nm})$ respectively), times a log-factor for Grover's algorithm. Since the number of iterations is $O(\sqrt{n})$, the upper bound on the running time as stated in the theorem follows.

The intermediate directed graph H is generated on-line from the input black-box graph G and the current matching M, using no more than $O(n)$ queries in the following way. Note that the subgraph of H on $V_1 \times V_2$ is the same as G, except that some edges have been removed (in G, an undirected edge (v, w) is equivalent to a directed edge (v, w) and a directed edge (w, v). In H, one of these two directed edges have been removed). Thus, the fact that we allowed holes in the adjacency list model can be exploited to create H by removing some edges from

G. Moreover, two new vertices a and b are added, a list of neighbours of a with holes of total length n is added, and at most one neighbour b is added to the list of every vertex from V_2.

After creating H, the layered subgraph H' is created in time $O(n^{1.5} \log n)$ (and $O(\sqrt{nm} \log n)$ respectively), as stated in Theorem 3.3.1. All that remains to be shown is that the augmenting paths can be found in the same time. A maximal set of vertex-disjoint paths from a to b is found by depth first search, such that a descendant of a vertex is found by Grover's search over all unmarked vertices with layer numbers one bigger. All the vertices are unmarked initially. When a descendant is found, we mark it and continue backtracking. Either the vertex becomes a part of an augmenting path later on, or it does not belong to any and hence it never gets tried again. Thus, each vertex is visited at most once.

In the adjacency model, every vertex costs time $O(\sqrt{n})$ to be found and time $O(\sqrt{n})$ to discover that it does not have any descendant. In the list model, a vertex v costs time $O(\sqrt{n_v d_v} + \sqrt{d_v})$, where n_v is the number of unmarked vertices found from v. Note that $\sum_{v \in V} d_v$ increases by at most $2n$. Hence, the sum over all vertices is upper-bounded like in the proof of Theorem 3.3.1. □

3.4 Some Remarks

We see that the quantum algorithm achieved a running time that is faster by a factor of $O\left(\frac{\sqrt{n}}{\log n}\right)$, simply by using a quantum algorithm for brute force search. For $m = \Omega(n^{1+\epsilon})$ for some $\epsilon > 0$, this algorithm is actually faster than the best known classical algorithm (running in time $O(\sqrt{n}m)$) for bipartite matching, given by Micali and Vazirani [6]. Ambainis and Špalek [1] used this strategy, of using Grover's search as a subroutine, to find new quantum nonbipartite matching algorithms, but their algorithm turned out to be polynomially slower than the classical one. More recently, Dörn [2] has given quantum algorithms for bipartite and non-bipartite graphs (both weighted and unweighted) that are polynomially better than the best corresponding classical algorithms.

4 Conclusion

We have seen classical and quantum algorithms for two very standard problems in graph theory. Even though both the problems are classically solvable in polynomial time, it is interesting to see how quantum resources improve the complexity; this may potentially pave the way to understanding how problems in NP can be tackled quantumly in a better way. It is worth mentioning here that the two quantum algorithms are very different in flavour and were developed with two very different goals in mind. The 2-coloring and partial 3-coloring algorithms by D'Hondt [3] focused on exploiting the structure of the problem, rather than trying to conquer the hardness of the problems by brute force. In the process, some interesting primitive quantum operations were developed. Her algorithms were completely quantum, and did not use any classical subroutines. Also, a strong link between entanglement and the hardness of an NP-completeness were established, thus highlighting the importance of entanglement, which still remains a poorly understood concept. On the other hand, the algorithm by Ambainis and S̃palek showed us how quantum subroutines can be included in classical algorithms to improve their time complexity. This, by no means, is insignificant. Indeed, even if full-fledged quantum computers are never built, we could still stand to benefit from quantum resources by using them as sub-procedures in classical computers.

References

[1] A. Ambainis and R. Špalek. Quantum algorithms for matching and network flows. *Proceedings of STACS*, 2006.

[2] S. Dörn. Quantum algorithms for matching problems. *Theory of Computing Systems*, 45-3:613–628, 2009.

[3] E. D'Hondt. Quantum approaches to graph colouring. *Theoretical Computer Science*, 410:302–309, 2009.

[4] L.K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of 28th ACM STOC*, pages 212–219, 1996.

[5] J.E. Hopcroft and R.M. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[6] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matchings in general graphs. *Proceedings of 21st IEEE FOCS*, pages 17–27, 1980.

[7] J. van Leeuwen. Graph algorithms. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 525–632, 1990.