Theory Comput Syst (2009) 45: 613–628
DOI 10.1007/s00224-008-9118-x

Quantum Algorithms for Matching Problems

Sebastian Dörn

Published online: 30 May 2008

Abstract We present quantum algorithms for the following matching problems in unweighted and weighted graphs with n vertices and m edges:

- Finding a maximal matching in general graphs in time $O(\sqrt{nm}\log^2 n)$.
- Finding a maximum matching in general graphs in time $O(n\sqrt{m}\log^2 n)$.
- Finding a maximum weight matching in bipartite graphs in time $O(n\sqrt{m}N\log^2 n)$, where N is the largest edge weight.

Our quantum algorithms are faster than the best known classical deterministic algorithms for the corresponding problems. In particular, the second result solves an open question stated in a paper by Ambainis and Špalek (Proceedings of STACS'06, pp. 172–183, 2006).

Keywords Quantum algorithms · Graph theory · Matching problems

1 Introduction

A matching in a graph is a set of edges such that for every vertex at most one edge of the matching is incident on the vertex. The task is to find a matching of maximum cardinality. The matching problem has many important applications in graph theory and computer science.

In this paper we study the complexity of algorithms for matching problems on quantum computers and compare these to the best known classical algorithms. Quantum algorithms have been presented for several problems from computer science [2, 8, 9], graph theory [3, 13–15, 25] and (linear) algebra [11, 16, 23].

S. Dörn (✉)
Institut für Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany
e-mail: sebastian.doern@uni-ulm.de

We will consider different versions of the matching problems, depending on whether the graph is bipartite or not and whether the graph is unweighted or weighted. For unweighted graphs, the best classical algorithm for finding a matching of maximum cardinality is based on augmenting paths and has running time $O(\sqrt{n}m)$ (see Micali and Vazirani [26]). Mucha and Sankowski [24] present an algorithm based on matrix multiplication that finds a maximum matching in general graphs in time $O(n^\omega)$, where $2 \leq \omega \leq 2.38$ is the exponent of the best matrix multiplication algorithm.

Ambainis and Špalek [3] published an $O(n^2(\sqrt{m/n} + \log n)\log^2 n)$ quantum time algorithm for computing a maximum matching. But this quantum algorithm is in no case better than the best classical algorithm from [26] and [24]. The authors of [3] state as an open question, to improve with a quantum algorithm the fastest known classical algorithm by Micali and Vazirani [26]. We solve this question by presenting an $O(n\sqrt{m}\log^2 n)$ quantum time algorithm for finding a maximum matching in general graphs. This algorithm is faster than the best classical deterministic algorithms for graphs with $m > n\log^4 n$.

For bipartite graphs the best classical matching algorithm has running time $O(\sqrt{n}m)$ (see Hopcroft and Karp [21]). Ambainis and Špalek [3] improved this bound with an $O(n\sqrt{m}\log^2 n)$ quantum algorithm for computing a maximum matching in these graphs. The algorithm is polynomially faster than the best classical algorithm, but not necessarily optimal. The time complexity of our quantum algorithm for the maximum matching problem in general graphs matches the complexity given in the algorithm from [3] for the restricted case of bipartite graphs.

If we consider weighted graphs, the best classical algorithms for computing a maximum weight matching in bipartite graphs were developed by Gabow and Tarjan [19] with running time $O(\sqrt{n}m\log(nN))$ and Kao et al. [22] with time complexity $O(\sqrt{n}W/k(n, W/N))$, where $k(x, y) = \log x/\log(x^2/y)$, N is the largest edge weight and W is the total edge weight of G.

We construct a quantum algorithm using the decomposition theorem of [22], and get a running time of $O(n\sqrt{m}N\log^2 n)$ for computing a maximum weight matching in bipartite graphs. If the largest edge weight N is constant, then we get an $O(n\sqrt{m}\log^2 n)$ time algorithm. This is faster than the best classical algorithms for this problem.

Berzina et al. [7] and Zhang [28] showed that $\Omega(n^{3/2})$ quantum queries to the adjacency matrix are required for computing a maximum matching in a bipartite graph. This implies a quantum lower bound of $\Omega(n^{3/2})$ for the quantum time complexity. Since the bipartite maximum matching problem is a special case of the other maximum matching problems, it follows that the $\Omega(n^{3/2})$ lower bound holds for all the maximum matching problems considered here. We show that the same lower bound holds also for the maximal matching problem. A maximal matching in a graph is matching which is contained in no other matching. Then we obtain a matching upper bound for this problem, thus showing that the quantum query complexity of the problem is $O(n^{3/2}\log n)$. For the running time we prove an upper bound of $O(\sqrt{nm}\log^2 n)$.

Our results are proved using several techniques: Grover search [18], quantum depth first search, topological numbering and quantum running time analysis of clas-

sical algorithms. For the case of weighted matching algorithms we use the quantum graph algorithms of Dürr [13] and as well as our new maximum matching algorithm.

The paper is organized as follows: In Sect. 2 we give necessary definitions and facts about graph theory and quantum computing. In Sect. 3 we present quantum algorithms for unweighted matching problems. We give a quantum algorithm for computing a maximal matching in general graphs in time $O(\sqrt{nm}\log^2 n)$. Then we present a maximum matching quantum algorithm for general graphs in time $O(n\sqrt{m}\log^2 n)$. In Sect. 4 we give quantum algorithms for the weighted matching problems. We show that finding a maximum weight matching in bipartite graphs can be done in time $O(n\sqrt{m}N\log^2 n)$ and finding a minimum weight perfect matching in bipartite graphs can be done in time $O(n\sqrt{nm}\log^3 n)$. We end with a Section with conclusion and open problems.

2 Preliminaries

2.1 Graph Theory

Let $G = (V, E)$ be an undirected graph, with $V = V(G)$ and $E = E(G)$ we denote the set of vertices and edges of G. Let $n = |V|$ be the number of vertices and $m = |E|$ the number of edges of G. We denote by $N_G(v)$ the set of all vertices adjacent to $v \in V$ and $d_G(v) := |N_G(v)|$. The graph G_{-S} is obtained from G by deleting the vertices $S \subset V$ and the incident edges. We denote by $[n]$ the set $\{1, \ldots, n\}$.

Definition 2.1 Let $G = (V, E)$ be an undirected graph, a *matching* is a subset $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is incident on v. We say that a vertex $v \in V$ is *matched* by M if some edge in M is incident on v, otherwise v is called *unmatched* or *free*. A matching is called *perfect*, if all vertices of G are matched. The set M is called *maximal*, if there is no matching $M' \subseteq E$ with $M \subset M'$. A *maximum matching* is a matching of maximum cardinality.

A *path* between two vertices s and t in G is a sequence $(v_1, \ldots, v_k)$ where $k \geq 1$ and $v_1, \ldots, v_k$ are distinct vertices of G such that $s = v_1$, $t = v_k$ and $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, k-1$. For a matching M, a path P in G is called *alternating* in M if edges in P are alternately in M and not in M. An alternating path P is called *augmenting path* for the matching M if the two end vertices of P are unmatched by M.

We consider the following models for accessing information in undirected graphs:

Adjacency matrix model: Given is the adjacency matrix $A \in \{0, 1\}^{n \times n}$ of G with $A_{i,j} = 1$ iff $\{i, j\} \in E$. Weighted graphs are encoded by a weight matrix A, where $A_{i,j}$ is the weight of edge $\{i, j\}$, and for convenience we set $A_{i,j} = \infty$ if $\{i, j\} \notin E$.

Adjacency list model: Given are the degrees $d_G(1), \ldots, d_G(n)$ of the vertices and for every $i \in V$ an array with its neighbours $f_i : [d_G(i)] \to [n]$. The value $f_i(j)$ is the j-th neighbour of i. Weighted graphs are encoded by a sequence of functions $f_i : [d_G(i)] \to [n] \times \mathbb{N}$, such that if $f_i(j) = (i', w)$ then there is an edge $\{i, i'\}$ with weight w and i' is the j-th neighbour of i.

In the following, we denote by **M** and **L** the input model of the graph as adjacency matrix (**M**) and as adjacency list (**L**).

2.2 Quantum Computing

For the basic notation on quantum computing, we refer the reader to the textbook by Nielsen and Chuang [27].

Quantum Query Model

In the query model, the input $x_1, \ldots, x_N$ is contained in a black box or oracle and can be accessed by queries to the black box. As a query we give i as input to the black box and the black box outputs x_i. The goal is to compute a Boolean function $f : \{0,1\}^N \to \{0,1\}$ on the input bits $x = (x_1, \ldots, x_N)$ minimizing the number of queries. The classical version of this model is known as decision tree.

The quantum query model was explicitly introduced by Beals et al. [4]. In this model we pay for accessing the oracle, but unlike the classical case, we use the power of quantum parallelism to make queries in superposition. The state of the computation is represented by $|i, b, z\rangle$, where i is the query register, b is the answer register, and z is the working register. A quantum computation with T queries is a sequence of unitary transformations

$$U_0 \to O_x \to U_1 \to O_x \to \cdots \to U_{T-1} \to O_x \to U_T,$$

where each U_j is a unitary transformation that does not depend on the input x, and O_x are query (oracle) transformations. The oracle transformation O_x can be defined as $O_x : |i, b, z\rangle \to |i, b \oplus x_i, z\rangle$. The computation consists of the following three steps:

1. Go into the initial state $|0\rangle$.
2. Apply the transformation $U_T O_x \cdots O_x U_0$.
3. Measure the final state.

The result of the computation is the rightmost bit of the state obtained by the measurement.

The quantum computation determines f with bounded error, if for every x, the probability that the result of the computation equals $f(x_1, \ldots, x_N)$ is at least $1 - \epsilon$, for some fixed $\epsilon < 1/2$. In the query model of computation each query adds one to the query complexity of an algorithm, but all other computations are free. The time complexity of the algorithm is usually measured in terms of the total circuit size for the unitary operations U_i.

Quantum Complexity

All results in our paper hold in the bounded-error model. For the quantum algorithms we use the following two complexity measures:

- The *quantum query complexity* of a graph algorithm $\mathcal{A}$ is the number of queries to the adjacency matrix or to the adjacency list of the input graph made by $\mathcal{A}$.

- The *quantum time complexity* of a graph algorithm $\mathcal{A}$ is the number of basic quantum operations made by $\mathcal{A}$.

Remark 2.2

1. Since each quantum query takes one unit step, the quantum time complexity is at least as large as the quantum query complexity.
2. For most polynomial time quantum algorithms, the time complexity is equal the query complexity with a polylog-factor.
3. In many graph problems where the graph has a large number of edges ($m = \Theta(n^2)$), the number of queries to the adjacency matrix is the same as the number of queries to the adjacency list. An exception for example is the test whether a graph is connected (see [13]).

Quantum Query Lower Bounds

In this paper, we use the following special case of an adversary method by Ambainis [1] to prove lower bounds for the quantum query complexity.

Theorem 2.3 [1] *Let $A \subseteq \{0,1\}^n$, $B \subseteq \{0,1\}^n$ and $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = 1$ for all $x \in A$, and $f(y) = 0$ for all $y \in B$. Let m and m' be numbers such that*

1. *for every $(x_1, \ldots, x_n) \in A$ there are at least m values $i \in \{1, \ldots, n\}$ such that $(x_1, \ldots, x_{i-1}, 1 - x_i, x_{i+1}, \ldots, x_n) \in B$,*
2. *for every $(x_1, \ldots, x_n) \in B$ there are at least m' values $i \in \{1, \ldots, n\}$ such that $(x_1, \ldots, x_{i-1}, 1 - x_i, x_{i+1}, \ldots, x_n) \in A$.*

Then every bounded-error quantum algorithm that computes f has quantum query complexity $\Omega(\sqrt{m \cdot m'})$.

Remark 2.4

1. A lower bound for the quantum query complexity implies also the same lower bound for the quantum time complexity.
2. The trivial upper bound for the quantum query complexity for graph problems is $O(n^2)$.

Quantum Search

A search problem is a subset $P \subseteq [N]$ of the search space $[N]$. With P we associate its characteristic function $f_P : [N] \to \{0,1\}$ with

$$f_P(x) = \begin{cases} 1, & \text{if } x \in P, \\ 0, & \text{otherwise.} \end{cases}$$

Any $x \in P$ is called a solution to the search problem. Let $k = |P|$ be the number of solutions of P.

Theorem 2.5 [5, 6, 18] *Let $P \subseteq [N]$ be a search problem and k the number of solutions of P.*

1. *Finding one solution of P can be done in $O(\sqrt{N/k})$ expected quantum queries to f_P with probability of at least a constant. The search algorithm does not require prior knowledge of k.*
2. *Finding one solution of P can be done in $O(\sqrt{N})$ quantum queries to f_P with probability of at least a constant, provided there is one.*
3. *Whether $k > 0$ can be decided in $O(\sqrt{N})$ quantum queries to f_P with probability of at least a constant.*
4. *Finding all solutions of P can be done in $O(\sqrt{kN})$ quantum queries to f_P with probability of at least a constant.*

Remark 2.6

1. The Grover search outputs a correct answer with probability of at least $1/2$. If we want to reduce the error probability to less than $1/n$, we have to repeat the quantum search $O(\log n)$ times. This increases the query complexity by a logarithmic factor. Suppose our quantum algorithm calls $l < n$ different Grover subroutines, then it outputs a correct answer with success probability at least $(1 - 1/n)^l \geq 1/e$.
2. The time complexity of our quantum algorithms is bigger than its query complexity by another logarithmic factor, since the running time complexity of Grover search is logarithmic bigger than its query complexity.

3 Unweighted Matchings

We regard the following matching problems in unweighted graphs:

Maximal Matching Given a graph G, compute a maximal matching in G.

Maximum Matching Given a graph G, compute a maximum matching in G.

3.1 Maximal Matching

In this section we construct a quantum query algorithm for the maximal matching problem. Then we prove that this algorithm is nearly optimal in the adjacency matrix model.

Theorem 3.1 *The quantum query complexity of the maximal matching problem is $O(n^{1.5} \log n)$ in the adjacency matrix model and $O(\sqrt{nm} \log n)$ in the adjacency list model.*

Proof Let $G = (V, E)$ be a graph with vertex set $V = \{v_1, \ldots, v_n\}$ and $k = 1$ an integer. We compute a set M of matching edges which is maximal in G. At the beginning $M = \emptyset$ and we mark every vertex of G as enabled. While there are some enabled vertices of G, we search with the Grover algorithm an adjacent edge of the

vertex v_k in G. If there is no such edge, we mark the vertex v_k as disabled. Otherwise we use the edge $\{v_k, v\}$ for the matching M, and we mark the two vertices v_k and v as disabled. Then we increase the value of k by one.

In the matrix model, one search can be done in $O(\sqrt{n})$ quantum queries. In total we use $O(n^{1.5})$ quantum queries to the adjacency matrix for computing a maximal matching. In the list model $\sqrt{d_G(v_k)}$ queries are required for every vertex v_k, and in total we use $\sum_{k=1}^{n} \sqrt{d_G(v_k)} = O(\sqrt{nm})$ quantum queries. In order to get a constant success probability, we need to amplify the success probability of each subroutine by repeating it $O(\log n)$ times, see Remark 2.6. Then we obtain a maximal matching M in the indicated quantum query complexity. □

Corollary 3.2 *The quantum time complexity of the maximal matching problem is* $O(n^{1.5} \log^2 n)$ *in the adjacency model and* $O(\sqrt{nm} \log^2 n)$ *in the adjacency list model.*

Now we show a nearly optimal quantum query lower bound in the adjacency matrix model, by using the adversary method of Ambainis [1] and analog to [7].

Theorem 3.3 *The maximal matching problem requires* $\Omega(n^{1.5})$ *quantum queries to the adjacency matrix.*

Proof We construct the sets A and B for the usage of Theorem 2.3. Let f be the Boolean function which is one, iff there is a maximal matching set of size n. The set A consists of all graphs $G = (V, E)$ with $|V| = 3n + 1$ satisfying the following requirements: (1) There are n mutually not connected red vertices. (2) There are $2n$ green vertices not connected with the red ones. Green vertices are grouped in pairs and each pair is connected by edge. (3) There is a black vertex which is connected to all red and green vertices. Let M be the set of edges between every pair of green vertices of G. Then M is a maximal matching in G of size n. The value of the function f for all graphs $G \in A$ is 1.

The set B consists of all graphs $G' = (V, E)$ with $|V| = 3n + 1$ satisfying the following requirements: (1) There are $n - 2$ mutually not connected red vertices. (2) There are $2n + 2$ green vertices not connected with red ones, green vertices are grouped in pairs and each pair is connected by edge. (3) There is a black vertex which is connected to all red and green vertices. The value of the function f for all graphs $G' \in B$ is 0, since there no maximal matching of size n in G'.

From each graph $G \in A$, we can obtain $G' \in B$ by adding one edge between two red vertices (then the two vertices become green), then $l = \Omega(n^2)$. From each graph $G' \in B$, we can obtain $G \in A$ by deleting an edge between two green vertices, then $l' = \Omega(n)$. By Theorem 2.3, the quantum query complexity is $\Omega(\sqrt{l \cdot l'}) = \Omega(n^{1.5})$. □

3.2 Maximum Matching

The best classical algorithms for finding a matching of maximum cardinality based on augmenting paths with running time $O(\sqrt{nm})$ (Micali and Vazirani [26]) and on

matrix multiplication with running time $O(n^{\omega})$ (Mucha and Sankowski [24]), where $2 \leq \omega \leq 2.38$ is the exponent of the best matrix multiplication algorithm. Ambainis and Špalek [3] constructed a $O(n^2(\sqrt{m/n} + \log n)\log^2 n)$ quantum algorithm for computing a maximum matching. Unfortunately this quantum algorithm is in no case better than the best classical algorithm from [26] and [24].

In this subsection we present an $O(n\sqrt{m}\log^2 n)$ quantum time algorithm for finding a maximum matching in a general graphs. This algorithm is faster than the best classical one for computing a maximum matching in a graph with $m > n\log^4 n$.

We speed up the $O(\sqrt{n}m)$ algorithm by Micali and Vizirani [26] by different Grover subroutines. This algorithm works in phases. In each phase a maximal set of disjoint minimum length augmenting paths is found, and the existing matching is increased along this paths. We implement such a phase in time $O(\sqrt{mn}\log^2 n)$ with quantum search. Only $O(\sqrt{n})$ such phases are needed for finding a maximum matching, see [21, 26].

First we determine the quantum query complexity of two basic graph problems: depth first search and topological numbering. We use this procedures in the quantum maximum matching algorithm.

Depth First Search (DFS) Given a graph $G = (V, E)$ and a vertex $s \in V$. Compute a depth-first tree T from s. The tree T is rooted at s and contains all the vertices of G that are reachable from s.

Lemma 3.4 *The quantum query complexity of DFS is* $O(n^{1.5}\log n)$ *in the adjacency matrix model and* $O(\sqrt{nm}\log n)$ *in the adjacency list model.*

Proof In DFS, we construct a tree T that contains all the vertices of G that are reachable from the root s. We can construct this tree by the following simple procedure: For each edge $(s, w) \in E$ in which w has not been discovered by T, make w the next child of s in T, and recall the procedure with $s = w$.

We use the Grover search to construct the tree T. Every vertex is discovered by T at most once. In the adjacency matrix model, every vertex is found in $O(\sqrt{n})$ quantum queries. In total we are using $O(n^{1.5})$ quantum queries in the adjacency matrix model. In the list model, finding an adjacent vertex of s which has not been discovered uses $O(\sqrt{d_G(s)})$ quantum queries. In total we have $\sum_s \sqrt{d_G(s)} = O(\sqrt{mn})$ quantum queries in the adjacency list model. By using Remark 2.6, the error probability is constant. □

Topological Numbering (TN) Given a directed acyclic graph $G = (V, E)$. Compute a numbering $I : V \to \{1, \ldots, n\}$ for each vertex in G, such that each edge is directed from lower number to higher number, i.e. if $(u, v) \in E$ then $I(u) < I(v)$.

Lemma 3.5 *The quantum query complexity of topological numbering is* $O(n^{1.5}\log n)$ *in the adjacency matrix model and* $O(\sqrt{nm}\log n)$ *in the adjacency list model.*

Proof We grow a DFS path P until a sink t (vertex with outdegree 0) is reached. Then we set $I(t) = n$, decrease n by 1 and delete t from the path P and the graph G. We continue with the DFS procedure until G has no vertices.

In each iteration we grow the DFS path P by starting with the previous P and extending it, if possible. Since we use DFS, we apply Lemma 3.4 to obtain the quantum query complexity of topological numbering. □

Before we explain the maximum matching algorithm, we give some important definitions.

Definition 3.6 Let $G = (V, E)$ be a graph and M be a matching in G.

- The *evenlevel* (*oddlevel*) of a vertex v is the length of a minimum even (odd) length alternating path from v to a free vertex, if there is one, and infinite otherwise.
- The *level* is the length of a minimum alternating path from v to a free vertex.
- A vertex is *outer* (*inner*) iff its level is even (odd). If v is outer (inner) then its oddlevel (evenlevel) will be refered to as the *other level* of v.
- An edge (u, v) in a graph G with matching M is called a *bridge* if either both evenlevel(u) and evenlevel(v) are finite, or both oddlevel(u) and oddlevel(v) are finite.
- A *blossom* B in a matched graph G is a cycle of odd length k, in which the edges are maximally matched. In every blossom there is one free vertex, called *basis* of B. The two vertices at distance $\lfloor k/2 \rfloor$ from the basis are called *peaks* of B.
- Let u be a vertex of G which is not matched. If u is inner and oddlevel(u) $= 2i + 1$ then v is called a *predecessor* of u iff evenlevel(v) $= 2i$ and $(u, v) \in E$. If u is outer then v is a predecessor of u iff (u, v) is a matched edge.
- The *ancestor* is the transitive closure of the relation predecessor.

The classical algorithm by [26] for computing a maximal matching in a general graph consists of a main routine SEARCH, and three subroutines: BLOSS-AUG, FINDPATH and TOPOLOGICAL ERASE. We describe shortly the four parts of the algorithm, for details see [26]:

SEARCH:
Given a graph $G = (V, E)$ and a matching M, SEARCH constructs simultaneously for every free vertex v of G a Breadth First Search (BFS) tree which is rooted at v, to find the oddlevel and evenlevel of each vertex in G. At the start of the subroutine the two levels of each vertex of G are set to infinity. Then SEARCH grows the BFS trees by incrementing the search level by one each time.
When SEARCH detects that a certain edge (u, v) is a bridge, it calls the subroutine BLOSS-AUG with the parameter u and v.

BLOSS-AUG:
This subroutine is called with vertices u and v such that the edge (u, v) is a bridge. The result is either the formation of a new blossom, or a minimum augmenting path. A new blossom is formed if and only if the following condition holds:

1. There exist a vertex z such that z is an ancestor of vertex u and v.
2. The vertices u and v do not have any ancestors, other than z, whose level is equal to the level of z.

If the condition holds for bridge (u, v) and b is the vertex whose level is maximum, then the new blossom is the set of vertices w such that:

1. Vertex w does not belong to any other blossom when B is formed.
2. Either $w = u$ or $w = v$ or w is an ancestor of u or w is an ancestor of v.
3. The vertex b is an ancestor of w.

From this follows that b is the base of B and u and v are the peaks of B.
BLOSS-AUG performs a Double Depth First Search (DDFS), consisting in growing two DFS trees T_1 and T_r simultaneously, i.e. if at a certain stage, the centers of activities of T_1 and T_r are at v_1 and v_r, then the DDFS grows T_1 if level$(v_1) \geq$ level(v_r), and its grows T_r otherwise. T_1 and T_r are rooted at u and v.
For the special details of the DDFS see [26, p. 21]. During the DDFS, the two trees can find two different free vertices, then an augmentation of the matching is possible.

FINDPATH:
When BLOSS-AUG finds the presence of a minimum augmenting path, we use FINDPATH to search for such a path P. FINDPATH is called with two vertices v_h and v_l and a blossom B as parameters. It holds level$(v_h) \geq$level(v_l) and they both belong to a common minimum augmenting path.
The procedure returns a path between v_l and v_r with a DFS starting at v_h to find v_l. The present matching is increased along the minimum augmenting path P.

TOPOLOGICAL ERASE:
After FINDPATH has found the minimum augmenting path P and the matching has been increased along this path, this subroutine deletes from the graph the path P and all those edges which cannot be part of a minimum augmenting path disjoint from P.
This subroutine uses a topological sort. Each vertex has a counter which at any stage indicates the number of its unerased predecessor edges. A vertex is deleted with all incident edges, either when its counter is zero or when it enters a minimum augmenting path detected by FINDPATH. The counter of the free vertices is one at the start and during the phase, since they have no predecessor.

Theorem 3.7 *There is a quantum algorithm for the maximum matching problem with running time of* $O(n^2 \log^2 n)$ *in the adjacency matrix model and* $O(n\sqrt{m} \log^2 n)$ *in the adjacency list model.*

Proof We show that a phase consisting of the four subroutines SEARCH, BLOSS-AUG, FINDPATH and TOPOLOGICAL ERASE can be implemented with quantum search in time $O(n^{1.5} \log^2 n)$ in the adjacency matrix model and in time $O(\sqrt{nm} \log^2 n)$ in the adjacency list model. Only $O(\sqrt{n})$ such phases are needed for finding a maximum matching, see [21, 26]. We regard the above four subroutines.

In the SEARCH procedure we perform a Breadth First Search to find the evenlevel and oddlevel of each vertex in G as follows: All free vertices of G get the evenlevel 0 and the other levels are infinite. SEARCH constructs simultaneously for every free vertex v of G a BFS tree to find the two level numbers of each vertex in G. In the adjacency matrix model a vertex is found in $O(\sqrt{n})$ quantum queries. Every vertex is processed at most twice, since we have two level numbers for each vertex. In the list model, processing a vertex v costs $O(\sqrt{d_G(v) \cdot n_v})$ quantum queries, where n_v is

the number of vertices adjacent to v with a infinite level number. Since $\sum_v n_v \leq 2n$, then the total quantum query complexity is upper-bounded by the Cauchy-Schwarz inequality:

$$\sum_v \sqrt{d_G(v)n_v} \leq \sqrt{\sum_v d_G(v)} \sqrt{\sum_v n_v} = O(\sqrt{mn}).$$

The procedure BLOSS-AUG performs a Double Depth First Search, consisting in growing of two DFS trees. With quantum search we perform these two DFS in $O(n^{1.5})$ quantum queries to the adjacency matrix model and in $O(\sqrt{nm})$ quantum queries to the adjacency list model, see Lemma 3.4.

The subroutine FINDPATH returns a path with a DFS starting at v_h to find v_l. Clearly with Lemma 3.4, the quantum query complexity is $O(n^{1.5})$ in the adjacency matrix model and $O(\sqrt{nm})$ in the adjacency list model.

The quantum query complexity of the TOPOLOGICAL ERASE procedure is the same as the above three subroutines, by using Lemma 3.5.

In order to get a constant success probability, we need to amplify the success probability of each subroutine by repeating it $O(\log n)$ times. □

4 Weighted Matching

In this section, we look at weighted matchings in bipartite graphs. Let $G = (V, E)$ be a graph, $w(u, v)$ is the weight of an edge $\{u, v\} \in E$, if u is not adjacent to v, let $w(u, v) = 0$. We denote by N the largest edge weight and with W the total edge weight of G.

We regard the following two matching problems in weighted graphs:

Maximum Weight Bipartite Matching Given a bipartite graph G with positive integer weights on the edges and without isolated vertices, compute a matching M in G such that the sum of the weights of the edges in M is maximum over all possible matchings.

Minimum Weight Perfect Bipartite Matching Given a weighted bipartite graph G, compute a perfect matching M in G such that the sum of the weights of the edges in M is minimum over all possible perfect matchings.

4.1 Maximum Weight Bipartite Matching

The best classical algorithms for computing a maximum weight matching in bipartite graphs with positive integer weights have been developed by Gabow and Tarjan [19] with running time $O(\sqrt{n}m \log(nN))$ and by Kao et al. [22] with time complexity $O(\sqrt{n}W/k(n, W/N))$, where $k(x, y) = \log x / \log(x^2/y)$.

We give a quantum algorithm with running time $O(n\sqrt{m}N \log^2 n)$ for computing a maximum weight matching in bipartite graphs. If the largest edge weight N is constant, then we get an $O(n\sqrt{m} \log^2 n)$ time algorithm. For the construction of our quantum algorithm we use the decomposition theorem of [22]. First we need some definitions and facts about a minimum weight cover in a bipartite graph.

Definition 4.1 Let $G = (X \cup Y, E)$ be a bipartite graph. A *cover* of G is a function $C : X \cup Y \to \mathbb{N}$ such that $C(x) + C(y) \geq w(x, y)$ for all $x \in X$ and $y \in Y$. Let $w(C) := \sum_{z \in X \cup Y} C(z)$ be the weight of C. The cover C is of *minimum weight*, if $w(C)$ is the smallest possible value.

The algorithms for computing a maximum weight matching in bipartite graphs use the following problem:

Minimum Weight Cover Given a bipartite graph G with positive integer weights, compute a minimum weight cover of G.

Remark 4.2 A minimum weight cover is dual to the maximum weight matching in a bipartite graph G, i.e. from a maximum matching in G we can find a minimum weight cover of G in time $O(m)$, see [10].

Definition 4.3 Let $G = (X \cup Y, E)$ be a bipartite graph and $h \in \{1, \ldots, N\}$ be an integer. Define G_h as the graph which is formed by the edges $\{u, v\}$ of G with $w(u, v) \in [N - h + 1, N]$ and the edge weight $\{u, v\}$ of G_h is $w(u, v) - (N - h)$.

Let C_h be a minimum weight cover of G_h and G_h^* is formed by the edges $\{u, v\}$ of G with $w(u, v) - C_h(u) - C_h(v) > 0$ and the edge weight $\{u, v\}$ of G_h^* is $w(u, v) - C_h(u) - C_h(v)$.

Now we present the Decomposition theorem of [22].

Theorem 4.4 [22] *Let h, G, G_h, C_h, G_h^* as in Definition 4.3 and let C_h^* be any minimum weight cover of G_h^*. If $D : X \cup Y \to \mathbb{N}$ is a function such that for every $u \in V(G)$,*

$$D(u) = C_h(u) + C_h^*(u),$$

then D is a minimum weight cover of G.

Using this theorem, a minimum weight cover of G can be computed with the following recursive procedure, see [22].

The correctness of the Algorithm 1 follows from Theorem 4.4. We use our maximum matching quantum algorithm for computing a minimum weight cover of the graph G.

Theorem 4.5 *The quantum time complexity of the* Minimum Weight Cover *algorithm is $O(n\sqrt{m}N \log^2 n)$ in the adjacency list model and $O(n^2 N \log^2 n)$ in the adjacency matrix model.*

Proof We analyse the running time in the adjacency list model. We initialize a maximum heap in $O(m)$ time to store the edges of G according to their weights.

Let $T(n, W, N)$ be the running time of the Minimum Weight Cover quantum algorithm. Let L be the set of all edges in G_1, i.e. the heaviest edges in G. Then Step 1 takes $O(|L| \log m)$ time. In Step 2, we can compute a maximum matching of G_1 in

Algorithm 1 MINIMUM WEIGHT COVER

Input: Bipartite graph $G = (X \cup Y, E)$ with positive integer weights.
Output: Minimum weight cover $D : X \cup Y \to \mathbb{N}$.
Complexity: **M:** $O(n^2 N \log^2 n)$, **L:** $O(n\sqrt{m}N \log^2 n)$ quantum steps.

1: Construct G_1 from G.
2: Find a minimum weight cover C_1 of G_1.
3: Construct G_1^* from G and C_1.
4: **if** $G_1^* = \emptyset$ **then**
5: **return**$[C_1]$
6: **else**
7: $C_1^* :=$ MINIMUM WEIGHT COVER$[G_1^*]$
8: $D(u) := C_1(u) + C_1^*(u)$ for all u in G
9: **return**$[D]$
10: **end if**

$O(n\sqrt{|L|}\log^2 n)$ time steps with bounded error, by using the maximum matching quantum algorithm. From this matching, C_1 can be found in $O(|L|)$ deterministic time, see [10]. Let L_1 be the set of the edges of G adjacent to some vertex u with $C_1(u) > 0$. Step 3 updates every edge of L_1 in $O(l_1 \log m)$ time, where $l_1 = |L_1|$.

The total running time of steps 1 to 3 is $O(n\sqrt{l_1}\log^2 n)$, since $L \subseteq L_1$. The total weight of G_1^* is at most $W - l_1$. Step 7 uses then at most $T(n, W - l_1, N')$ time, where $N' < N$ is the maximum edge weight of G_1^* and it follows

$$T(n, W, N) = O(n\sqrt{l_1}\log^2 n) + T(n, W - l_1, N'),$$

where $T(n, 0, N') = 0$. We apply the procedure recursively for some positive integers $l_1, l_2, \ldots, l_p$ with $p \le N$ and $\sum_{1 \le i \le p} l_i = W$, it follows

$$T(n, W, N) = O\left(n \log^2 n \sum_{i=1}^{p} \sqrt{l_i}\right).$$

Since $\sum_{i=1}^{p} \sqrt{l_i} \le \sqrt{p \sum_{i=1}^{p} l_i}$, then

$$T(n, W, N) = O\left(n \log^2 n \sqrt{p \sum_{i=1}^{p} l_i}\right) = O\left(n \log^2 n \sqrt{NW}\right).$$

Since $W \le Nm$ it follows $T(n, W, N) = O(n\sqrt{m}N \log^2 n)$ in the list model. The running time for the matrix model follows setting $m = n^2$. □

Now we use the Algorithm 2 by [22] to recover a maximum weight matching of a bipartite graph G from a minimum weight cover of G.

Theorem 4.6 *There is a quantum algorithm for computing a maximum weight matching in a bipartite graph with running time of $O(n\sqrt{m}N \log^2 n)$ in the adjacency list model and $O(n^2 N \log^2 n)$ in the adjacency matrix model.*

Algorithm 2 MAXIMUM WEIGHT MATCHING

Input: Bipartite graph $G = (X \cup Y, E)$ with positive integer weights.
Output: Maximum weight matching M.
Complexity: **M:** $O(n^2 N \log^2 n)$, **L:** $O(n\sqrt{m}N \log^2 n)$ quantum steps.

1: $D :=$MINIMUM WEIGHT COVER$[G]$
2: $A := \{\{u, v\} \in E \mid w(u, v) = D(u) + D(v)\}$
3: $H := (V, A)$
4: Make two copies of H, call them H^a and H^b. For each vertex u of H, let u^a and u^b denote the corresponding vertex in H^a and H^b.
5: Union H^a and H^b to form H^{ab}, and add to H^{ab} the set of edges $\{(u^a, u^b) \mid u \in V(H),\ D(u) = 0\}$.
6: Find a maximum matching K of H^{ab}.
7: $M = \{(u, v) \mid (u^a, v^a) \in K\}$

Proof We use the above algorithm. The graph H^{ab} has at most $2n$ nodes and at most $3m$ edges. The maximum matching K can be constructed by a quantum algorithm in time $O(n\sqrt{m}\log^2 n)$ in the adjacency list model and in time $O(n^2 \log^2 n)$ in the adjacency matrix model. □

Corollary 4.7 *There is a quantum algorithm for computing a maximum weight matching in a bipartite graph with constant edge weight with running time of* $O(n\sqrt{m}\log^2 n)$ *in the adjacency list model and* $O(n^2 \log^2 n)$ *in the adjacency matrix model.*

4.2 Minimum Weight Perfect Bipartite Matching

We give the quantum time complexity for computing a minimum weight perfect matching in a bipartite graph. For this task we use shortest paths.

Shortest Paths Given a weighted graph $G = (V, E)$ and a source vertex v_s of the graph, compute a tree T, such that the shortest paths from v_s to all the other vertices is in T.

The best deterministic algorithm for computing a minimum weight perfect matching in a bipartite graph computes shortest paths in a graph, see Cook et al. [12]. The time complexity for such an algorithm is given by the following Theorem.

Theorem 4.8 (see [12]) *There is an algorithm for the minimum weight perfect matching problem for bipartite graphs with running time* $O(nS(n, m))$, *where* $S(n, m)$ *is the time needed to solve the shortest path problem on a digraph with* n *vertices and* m *arcs.*

For the shortest path problem we use the quantum algorithm by Dürr et al. [13].

Theorem 4.9 [13] *The shortest path problem can be solved by a quantum algorithm in time* $O(n^{1.5}\log^3 n)$ *in the adjacency matrix model and in time* $O(\sqrt{nm}\log^3 n)$ *in the adjacency list model.*

From this, we get the running time to compute the minimum weight perfect matching in a bipartite graph.

Theorem 4.10 *There is a quantum algorithm for the minimum weight perfect matching problem for bipartite graphs with running time of* $O(n^{2.5}\log^3 n)$ *in the adjacency matrix model and* $O(n\sqrt{nm}\log^3 n)$ *in the adjacency list model.*

5 Conclusion and Open Problems

We give a summary of the classical and quantum time upper bounds for the regarded matching problems in unweighted and weighted graphs:

Problem	Quantum time complexity	Theorem	Classical time complexity	Reference
Maximal matching	**M:** $O(n^{1.5}\log^2 n)$	3.2	$O(m)$	Trivial
	L: $O(\sqrt{nm}\log^2 n)$			
Maximum matching	**M:** $O(n^2\log^2 n)$	3.7	$O(\sqrt{n}m)$	[26]
	L: $O(n\sqrt{m}\log^2 n)$		$O(n^w)$, $w \le 2.376$	[24]
Maximum weight	**M:** $O(n^2 N\log^2 n)$	4.6	$O(\sqrt{n}W)$	[22]
Bipartite matching	**L:** $O(n\sqrt{m}N\log^2 n)$		$O(\sqrt{n}m\log(nN))$	[19]
Minimum weight	**M:** $O(n^{2.5}\log^3 n)$	4.10	$O(nS(n,m))$	[12]
Perf. Bip. matching	**L:** $O(n\sqrt{nm}\log^3 n)$			

We give some interesting open questions in the area of quantum matching algorithms:

1. Is the $O(n^2\log^2 n)$ quantum algorithm for computing a maximum matching optimal? The best quantum lower bound for this problem is $\Omega(n^{1.5})$. Is there a better upper bound for computing a maximum matching in bipartite graphs?
2. Gabow [17] showed that both the minimum weight and the maximum weight matching problems can be solved in time $O(n(m+n\log n))$. He presents an algorithm in which a maximum weight augmenting path in a graph G can be found in time $O(m+n\log n)$. Enlarging in G the matching with the path, after n steps the maximum weight matching is found.

   It is open to find a quantum algorithm for the minimum weight and the maximum weight matching for general graphs improving the running time of the classical algorithm of Gabow.

Acknowledgements For helpful comments I am grateful to Jacobo Torán.

References

1. Ambainis, A.: Quantum lower bounds by quantum arguments. J. Comput. Syst. Sci. **64**, 750–767 (2002)

2. Ambainis, A.: Quantum walk algorithm for element distinctness. In: Proceedings of FOCS'04, pp. 22–31 (2004)
3. Ambainis, A., Špalek, R.: Quantum algorithms for matching and network flows. In: Proceedings of STACS'06, pp. 172–183 (2006)
4. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. J. Assoc. Comput. Mach. **48**, 778–797 (2001)
5. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschr. Phys. **46**(4–5), 493–505 (1998)
6. Buhrman, H., Cleve, R., de Wolf, R., Zalka, C.: Bounds for small-error and zero-error quantum algorithms. In: Proceedings of FOCS'99, pp. 358–368 (1999)
7. Berzina, A., Dubrovsky, A., Freivalds, R., Lace, L., Scegulnaja, O.: Quantum query complexity for some graph problems. In: Proceedings of SOFSEM'04, pp. 140–150 (2004)
8. Buhrman, H., Dürr, C., Heiligman, M., Høyer, P., Magniez, F., Santha, M., de Wolf, R.: Quantum algorithms for element distinctness. In: Proceedings of CCC'01, pp. 131–137 (2001)
9. Brassard, G., Hóyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Proceedings of LATIN'98, pp. 163–169 (1998)
10. Bondy, J., Murty, U.: Graph Theory with Applications. North-Holland, Amsterdam (1976)
11. Buhrman, H., Špalek, R.: Quantum verification of matrix products. In: Proceedings of SODA'06, pp. 880–889 (2006)
12. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)
13. Dürr, C., Heiligman, M., Hoyer, P., Mhalla, M.: Quantum query complexity of some graph problems. In: Proceedings of ICALP'04, pp. 481–493 (2004)
14. Dörn, S.: Quantum complexity bounds of independent set problems. In: Proceedings of SOFSEM'07 (SRF), pp. 25–36 (2007)
15. Dörn, S.: Quantum algorithms for graph traversals and related problems. In: Proceedings of CIE'07, pp. 123–131 (2007)
16. Dörn, S., Thierauf, T.: The quantum query complexity of algebraic properties. In: Proceedings of FCT'07, pp. 250–260 (2007)
17. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 434–443 (1990)
18. Grover, L.: A fast mechanical algorithm for database search. In: Proceedings of STOC'96, pp. 212–219 (1996)
19. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for network problems. SIAM J. Comput. **18**, 1013–1036 (1989)
20. Gross, J., Yellen, J.: Graph Theory and its Applications. CRC Press, Boca Raton (1999)
21. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973)
22. Kao, M.-Y., Lam, T.-W., Sung, W.-K., Ting, H.-F.: A decomposition theorem for maximum weight bipartite matchings. SIAM J. Comput. **31**, 18–26 (2001)
23. Magniez, F., Nayak, A.: Quantum complexity of testing group commutativity. In: Proceedings of ICALP'05, pp. 1312–1324 (2005)
24. Mucha, M., Sankowski, P.: Maximum matchings via Gaussian elimination. In: Proceedings of FOCS'04, pp. 248–255 (2004)
25. Magniez, F., Santha, M., Szegedy, M.: Quantum algorithms for the triangle problem. In: Proceedings of SODA'05, pp. 1109–1117 (2005)
26. Micali, S., Vazirani, V.V.: An $O(\sqrt{nm})$ algorithm for finding maximum matching in general graphs. In: Proceedings of FOCS'80, pp. 17–27 (1980)
27. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2003)
28. Zhang, S.: On the power of Ambainis's lower bounds. In: Proceedings of ICALP'04. Lecture Notes in Computer Science, vol. 3142, pp. 1238–1250. Springer, Berlin (2004)