

# Quantum Algorithm for Triangle Finding in Sparse Graphs

François Le Gall      Shogo Nakajima

Department of Computer Science  
Graduate School of Information Science and Technology  
The University of Tokyo, Japan

## Abstract

This paper presents a quantum algorithm for triangle finding over sparse graphs that improves over the previous best quantum algorithm for this task by Buhrman et al. [SIAM Journal on Computing, 2005]. Our algorithm is based on the recent  $\tilde{O}(n^{5/4})$ -query algorithm given by Le Gall [FOCS 2014] for triangle finding over dense graphs (here  $n$  denotes the number of vertices in the graph). We show in particular that triangle finding can be solved with  $O(n^{5/4-\epsilon})$  queries for some constant  $\epsilon > 0$  whenever the graph has at most  $O(n^{2-c})$  edges for some constant  $c > 0$ .

## 1 Introduction

**Background.** Triangle finding asks to decide if a given undirected graph  $G = (V, E)$  contains a cycle of length three, i.e., whether there exist three vertices  $u_1, u_2, u_3 \in V$  such that  $\{u_1, u_2\} \in E$ ,  $\{u_1, u_3\} \in E$  and  $\{u_2, u_3\} \in E$ . This problem has received recently a lot of attention, for the following reasons.

First, several new applications of triangle finding have been discovered recently. In particular, Vassilevska Williams and Williams have shown a surprising reduction from Boolean matrix multiplication to triangle finding [17], which indicates that efficient algorithms for triangle finding may be used to design efficient algorithms for matrix multiplication, and thus also for a vast class of problems related to matrix multiplication. Relations between variants of the standard triangle finding problem (such as triangle finding over weighted graphs) and well-studied algorithmic problems (such as 3SUM) have also been shown in the past few years (see for instance [16, 18]).

Second, triangle finding is one of the most elementary graph theoretical problems whose complexity is unsettled. In the time complexity setting, the best classical algorithm uses a reduction to matrix multiplication [10] and solves triangle finding in time  $O(n^{2.38})$ , where  $n$  denotes the number of vertices in  $G$ . In the time complexity setting again, Grover search [9] immediately gives, when applied to triangle finding as a search over the set of triples of vertices of the graph, a quantum algorithm with time complexity  $\tilde{O}(n^{3/2})$ , which is still the best known upper bound for the quantum time complexity of this problem.<sup>1</sup> In the query complexity setting, where an oracle to the adjacency matrix of the graph is given and only the number of calls to this oracle is counted, a surge of activity has led to quantum algorithms with better complexity. Magniez, Santha and Szegedy [15] first presented a quantum algorithm that solves triangle finding with  $\tilde{O}(n^{1.3})$  queries. This complexity was later improved to  $O(n^{1.296\dots})$  by Belovs [4], then to  $O(n^{1.285\dots})$  by Lee, Magniez and Santha [13] and Jeffery, Kothari and Magniez [12], and further improved recently to  $\tilde{O}(n^{5/4})$  by Le Gall [8]. The main open problem now is to understand whether this  $\tilde{O}(n^{5/4})$ -query upper bound is tight or not. The best known lower bound on the quantum query complexity of triangle finding is the straightforward  $\Omega(n)$  lower bound.

Another reason why triangle finding has received much attention from the quantum computing community is that work on the quantum complexity of triangle finding has been central to the development of algorithmic techniques. Indeed, all the improvement mentioned in the previous paragraph have been obtained by introducing either new quantum techniques or new paradigms for the design of quantum

<sup>1</sup>In this paper the notation  $\tilde{O}(\cdot)$  removes polylog $n$  factors.

algorithms: applications of quantum walks to graph-theoretic problems [15], introduction of the concept of learning graphs [4] and improvements to this technique [13], introduction of quantum walks with quantum data structures [12], association of combinatorial arguments with quantum walks [8].

**Triangle finding in sparse graphs.** The problem we will consider in this paper is triangle finding over sparse graphs (the graphs considered are, as usual, undirected and unweighted). If we denote  $m$  the number of edge of the graph (i.e.,  $m = |E|$ ), the goal is to design algorithms with complexity expressed as a function of  $m$  and  $n$ . Ideally, we would like to show that if  $m = n^{2-c}$  for any constant  $c > 0$  then triangle finding can be solved significantly faster than in the dense case (i.e.,  $m \approx n^2$ ). Besides its theoretical interest, this problem is of practical importance since in many applications the graphs considered are sparse.

Classically, Alon, Yuster and Zwick [1] constructed an algorithm exploiting the sparsity of the graph and working in time  $O(m^{1.41})$ , which gives better complexity than the  $O(n^{2.38})$ -time complexity mentioned above when  $m \leq n^{1.68}$ . Understanding whether an improvement over the dense case is also possible for larger values  $m$  is a longstanding open problem. Note in the classical query complexity setting it is easy to show that the complexity of triangle finding is  $\Theta(n^2)$ , independently of the value of  $m$ .

In the quantum setting, using amplitude amplification, Buhrman et al. [6] showed how to construct a quantum algorithm for triangle finding with time and query complexity  $O(n + \sqrt{nm})$ . This upper bound is tight when  $m \leq n$  since the  $\Omega(n)$ -query lower bound for the quantum query complexity of triangle finding already mentioned also holds when  $m$  is a constant. Childs and Kothari [7] more recently developed an algorithm, based on quantum walks, that detects the existence of subgraphs in a given graph. Their algorithm works for any constant-size subgraph. For detecting the existence of a triangle, however, the upper bound they obtain is  $\tilde{O}(n^{2/3}\sqrt{m})$  queries for  $m \geq n$ , which is worse than the bound obtained in [6]. Buhrman et al.'s result in particular gives an improvement over the  $\tilde{O}(n^{5/4})$ -query quantum algorithm algorithm whenever  $m \leq n^{3/2}$ . A natural question is whether a similar improvement can be obtained for larger values of  $m$ . For instance, can we obtain query complexity  $\tilde{O}(n^{5/4-\epsilon})$  for some constant  $\epsilon > 0$  when  $m \approx n^{1.99}$ ? A positive answer would show that even a little amount of sparsity can be exploited in the quantum query setting, which is not known to be true in the classical setting as mentioned in the previous paragraph.

**Our results.** In this paper we answer positively to the above question. Our main result is as follows.

**Theorem 1.** *There exists a quantum algorithm that solves, with high probability, the triangle finding problem over graphs of  $n$  vertices and  $m$  edges with query complexity*

$$\begin{cases} O(n + \sqrt{nm}) & \text{if } 0 \leq m \leq n^{7/6}, \\ \tilde{O}(nm^{1/14}) & \text{if } n^{7/6} \leq m \leq n^{7/5}, \\ \tilde{O}(n^{1/6}m^{2/3}) & \text{if } n^{7/5} \leq m \leq n^{3/2}, \\ \tilde{O}(n^{23/30}m^{4/15}) & \text{if } n^{3/2} \leq m \leq n^{13/8}, \\ \tilde{O}(n^{59/60}m^{2/15}) & \text{if } n^{13/8} \leq m \leq n^2. \end{cases}$$

The complexity bounds of Theorem 1 are depicted in Figure 1. For the dense case (i.e.,  $m \approx n^2$ ) we recover the same complexity  $\tilde{O}(n^{5/4})$  as in [8] — in this case it turns out that our algorithm applies exactly the same procedure as in [8]. Whenever  $m = n^{2-c}$  for some constant  $c > 0$  (in particular, for  $m \approx n^{1.99}$ ), we indeed obtain query complexity  $\tilde{O}(n^{5/4-\epsilon})$  for some constant  $\epsilon > 0$  depending on  $c$ . The query complexity of our algorithm is better than the query complexity of Buhrman et al.'s algorithm [6] whenever  $m \gtrsim n^{7/6}$ . When  $m \lesssim n^{7/6}$  we obtain the same complexity  $O(n + \sqrt{nm})$  as in [6] — in this case it turns out that our algorithm applies exactly the same procedure as in [6].

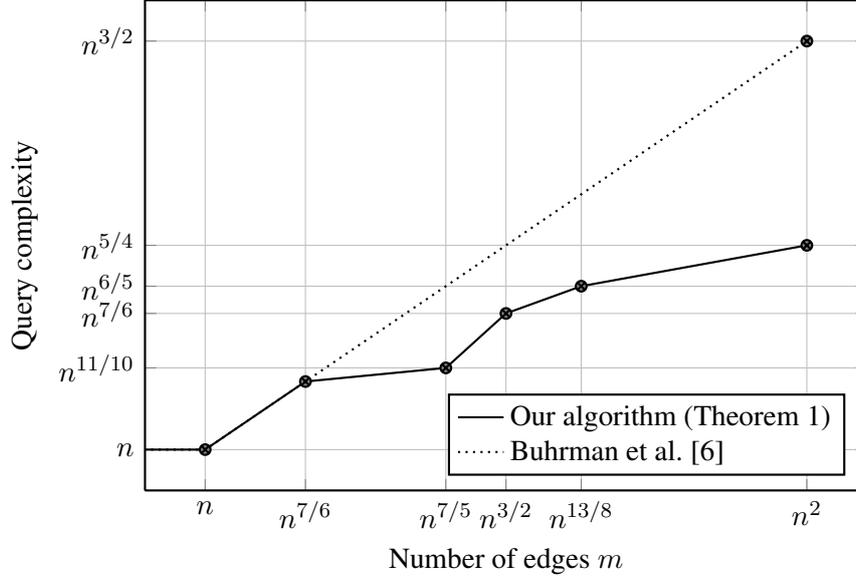


Figure 1: Quantum query complexity of triangle finding on a graph with  $n$  vertices and  $m$  edges.

**Overview of our techniques.** The main idea is to adapt the  $\tilde{O}(n^{5/4})$ -query quantum algorithm for triangle finding [8] to handle sparse graphs. This algorithm works in two steps: a first step based on Grover search that detects the existence of a triangle in a well-chosen small part of the graph  $G$ , and a second step based on recursive quantum walks that detects the existence of a triangle in the remaining (large) part of the graph. A first simple observation is that the first step can be implemented faster in the case of sparse graphs by applying the quantum algorithm by Buhrman et al. [6] on the small part of  $G$  instead of using Grover search. This observation alone however does not give any interesting speed-up unless sparsity is exploited in the second step as well. The hard part is actually to adapt the recursive quantum walk approach to the case of sparse graphs, and we outline below our main ideas to achieve this goal.

The main issue is that the implementation of the recursive quantum walks described [8] is really tailored for their application on dense graphs; when trying to use the same implementation for sparse graphs prohibitive intermediate costs (of order  $\tilde{O}(n^{5/4})$ , which is fine for the dense case, but not for the sparse case) appear. To overcome this difficulty, we need to modify partially the original approach in several ways, such as modifying how the inner quantum walk checks if it has found a solution and adjusting how the sets of marked states of the walk are defined, to fully exploit the sparsity of the graph.

Several more technical issues have also to be dealt with. The complexity of a quantum walk basically depends on the complexity of three operations performed by the walk: the set up cost (creating the data structure corresponding to the initial state of the walk), the update cost (updating the database after updating the current state of the walk) and the checking cost (checking if the current state of the walk is marked or not). The sparsity of the graph  $G$  can be immediately exploited to reduce the cost of these three operations if the graph is “perfectly balanced”, i.e., if each vertex of the graph has degree  $\Theta(m/n)$ . However, while the average degree will indeed be  $\Theta(m/n)$ , in general the graph can have many vertices with degree exceeding this estimate (in particular this can happen for vertices of the triangle we are looking for). This is a significant complication since to analyze a quantum walk one need an upper bound on the worst case (i.e., for the worse state of the walk) complexity of the three operations. Indeed, there is no general technique to analyze quantum walks when only an upper bound on the average update cost or checking cost is available. To overcome this difficulty, our approach is to partition the vertices of  $V$  into two sets: the set of vertices with degree larger than  $n^d$  (which we call below high-degree vertices) and the set of vertices with degree smaller than  $n^d$  (low-degree vertices), where  $d$  is a parameter. Obtaining the classification can be done by combining quantum search and quantum counting, but is costly when  $d$

is small, which means that we need to be careful when choosing  $d$ . Once this classification has been obtained, we only need to search separately for four types of triangles: triangles with three low-degree vertices, triangles with two low-degree vertices and one high-degree vertex, triangles with one low-degree vertex and two high-degree vertices, and triangles with three high-degree triangles. Since we know that each low-degree vertex has degree at most  $n^d$ , we can derive a worst-case upper bound for the corresponding update costs and checking costs. For high-degree vertices we do not have any upper bound on their degree, but we know that the number of high-degree vertices is at most  $m/n^d$ , which will be significantly smaller than  $n$  if  $d$  is well chosen and lead to some improvement for the corresponding complexity of the walks, since the graph has at most  $m$  edges. Combined with the ideas described in the previous paragraph, this strategy enables us to obtain the upper bounds given in Theorem 1.

## 2 Preliminaries

### 2.1 Query complexity for graph-theoretic problems

In this paper we adopt the standard model of quantum query complexity for graph-theoretic problems. The presentation given below will follow the description of these notions given in [8].

For any finite set  $T$  and any  $r \in \{1, \dots, |T|\}$  we denote  $\mathcal{S}(T, r)$  the set of all subsets of  $r$  elements of  $T$ . We use the notation  $\mathcal{E}(T)$  to represent  $\mathcal{S}(T, 2)$ , i.e., the set of unordered pairs of elements in  $T$ .

Let  $G = (V, E)$  be an undirected and unweighted graph, where  $V$  represents the set of vertices and  $E \subseteq \mathcal{E}(V)$  represents the set of edges. We write  $n = |V|$ . In the query complexity setting, we assume that  $V$  is known, and that  $E$  can be accessed through a quantum unitary operation  $\mathcal{O}_G$  defined as follows. For any pair  $\{u, v\} \in \mathcal{E}(V)$ , any bit  $b \in \{0, 1\}$ , and any binary string  $z \in \{0, 1\}^*$ , the operation  $\mathcal{O}_G$  maps the basis state  $|\{u, v\}\rangle|b\rangle|z\rangle$  to the state

$$\mathcal{O}_G|\{u, v\}\rangle|b\rangle|z\rangle = \begin{cases} |\{u, v\}\rangle|b \oplus 1\rangle|z\rangle & \text{if } \{u, v\} \in E, \\ |\{u, v\}\rangle|b\rangle|z\rangle & \text{if } \{u, v\} \notin E, \end{cases}$$

where  $\oplus$  denotes the bit parity (i.e., the logical XOR). We say that a quantum algorithm computing some property of  $G$  uses  $k$  queries if the operation  $\mathcal{O}_G$ , given as an oracle, is called  $k$  times by the algorithm. We also assume that we know the number of edges of the input graph (i.e., we know  $m = |E|$ ). All the results in this paper can be easily generalized to the case where  $m$  is unknown.

**Quantum enumeration.** Let  $f_G: \{1, \dots, N\} \rightarrow \{0, 1\}$  be a Boolean function depending on the input graph  $G$ , and let us write  $M = f^{-1}(1)$ . Assume that for any  $x \in \{1, \dots, N\}$  the value  $f_G(x)$  can be computed using at most  $t$  queries to  $\mathcal{O}_G$ . Grover search enables us to find an element  $x$  such that  $f_G(x) = 1$ , if such an element exists, using  $\tilde{O}(\sqrt{N/M} \times t)$  queries to  $\mathcal{O}_G$ . A folklore observation is that we can then repeat this procedure to find all the elements  $x \in \{1, \dots, N\}$  such that  $f_G(x) = 1$  with  $\tilde{O}\left(\left(\sqrt{\frac{N}{M}} + \sqrt{\frac{N}{M-1}} + \dots + \sqrt{\frac{N}{1}}\right) \times t\right) = \tilde{O}(\sqrt{N \times M} \times t)$  queries. We call this procedure *quantum enumeration*.

**Quantum walk over Johnson graphs.** Let  $T$  be a finite set and  $r$  be a positive integer such that  $r \leq |T|$ . Let  $f_G: \mathcal{S}(T, r) \rightarrow \{0, 1\}$  be a Boolean function depending on the input graph  $G$ . We say that a set  $A \in \mathcal{S}(T, r)$  is marked if  $f_G(A) = 1$ . Let us consider the following problem. The goal is to find a marked set, if such a set exists, or otherwise report that there is no marked set. We are interested in the number of calls to  $\mathcal{O}_G$  to solve this problem. The quantum walk search approach developed by Ambainis [2] solves this problem using a quantum walk over a Johnson graph.

The Johnson graph  $J(T, r)$  is the undirected graph with vertex set  $\mathcal{S}(T, r)$  where two vertices  $R_1, R_2 \in \mathcal{S}(T, r)$  are connected if and only if  $|R_1 \cap R_2| = r - 1$ . In a quantum walk over a Johnson graph  $J(T, r)$ , the state of the walk corresponds to a node of the Johnson (i.e., to an element  $A \in \mathcal{S}(T, r)$ ).

A data structure  $D(A)$ , which in general depends on  $G$ , is associated to each state  $A$ . There are three costs to consider: the set up cost  $S$  representing the number of queries to  $\mathcal{O}_G$  needed to construct the data structure of the initial state of the walk, the update cost  $U$  representing the number of queries to  $\mathcal{O}_G$  needed to update the data structure when one step of the quantum walk is performed (i.e., updating  $D(A)$  to  $D(A')$  for some  $A' \in \mathcal{S}(T, r)$  such that  $|A \cap A'| = r - 1$ ), and the checking cost  $C$  representing the number of queries to  $\mathcal{O}_G$  needed to check if the current state  $A$  is marked (i.e., checking whether  $f_G(A) = 1$ ). Let  $\varepsilon > 0$  be such that, for all input graphs  $G$  for which at least one marked set exists, the fraction of marked states is at least  $\varepsilon$ . Ambainis [2] (see also [14]) has shown that the quantum walk search approach outlined above finds with high probability a marked set if such set exists (or otherwise report that there is no marked set) and has query complexity  $\tilde{O}\left(S + \frac{1}{\sqrt{\varepsilon}}(\sqrt{r} \times U + C)\right)$ .

## 2.2 Quantum algorithm for dense triangle finding

In this subsection we outline the  $\tilde{O}(n^{5/4})$ -query quantum algorithm for triangle finding over a dense graph by Le Gall [8]. We actually present a version of this algorithm that solves the following slightly more general version of triangle finding, since this will be more convenient when describing our algorithms for sparse graphs in the next section: given two (non necessarily disjoint) sets  $V_1, V_2 \subseteq V$ , find a triangle  $\{v_1, v_2, v_3\}$  of  $G$  such that  $v_1 \in V_1$  and  $v_2, v_3 \in V_2$ , if such a triangle exists. Note that the original triangle finding problem is the special case  $V_1 = V_2 = V$ .

**Definitions and lemmas.** Let  $V_1$  be any subset of  $V$ . For any sets  $X \subseteq V_1$  and  $Y \subseteq V$ , we define the set  $\Delta_G(X, Y) \subseteq \mathcal{E}(Y)$  as follows:

$$\Delta_G(X, Y) = \mathcal{E}(Y) \setminus \bigcup_{u \in X} \mathcal{E}(N_G(u)),$$

where  $N_G(u)$  denotes the set of neighbors of  $u$ . For any vertex  $w \in V$ , we define the set  $\Delta_G(X, Y, w) \subseteq \Delta_G(X, Y)$  as follows:

$$\Delta_G(X, Y, w) = \left\{ \{u, v\} \in \Delta_G(X, Y) \mid \{u, w\} \in E \text{ and } \{v, w\} \in E \right\}.$$

An important concept used in [8] is the notion of *k-good sets*.

**Definition 2.1.** Let  $k$  be any constant such that  $0 \leq k \leq 1$ , and  $V_1$  be any subset of  $V$ . A set  $X \subseteq V_1$  is *k-good* for  $(G, V_1)$  if the inequality  $\sum_{w \in V_1} |\Delta_G(X, Y, w)| \leq |Y|^2 |V_1|^{1-k}$  holds for all  $Y \subseteq V$ .

Note that [8] considered only Definition 2.1 for the case  $V_1 = V$ . In our paper we will need the slightly generalized version described here. The point is that *k-good sets* can be constructed very easily.

**Lemma 2.1** ([8]). Let  $k$  be any constant such that  $0 \leq k \leq 1$ . Suppose that  $X$  is a set obtained by taking uniformly at random, with replacement,  $\lceil 3|V_1|^k \log n \rceil$  elements from  $V_1$ . Then  $X$  is *k-good* for  $(G, V_1)$  with probability at least  $1 - 1/n$ .

Lemma 2.1 was proved in [8] only for the case  $V_1 = V$ , but the generalization is straightforward.

**Quantum algorithm for dense triangle finding.** Let  $a, b$  and  $k$  be three constants such that  $0 < b < a < 1$  and  $0 < k < 1$ . The values of these constants will be set later. The quantum algorithm in [8] works as follows.

The algorithm first takes a set  $X \subseteq V_1$  obtained by choosing uniformly at random  $\lceil 3|V_1|^k \log n \rceil$  elements from  $V_1$ , and checks if there exists a triangle of  $G$  with a vertex in  $X$  and two vertices in  $V_2$ . This can be done using Grover search with

$$O\left(\sqrt{|X| \times |\mathcal{E}(V_2)|}\right) = \tilde{O}\left(|V_1|^{k/2} |V_2|\right) \quad (1)$$

queries. If no triangle has been reported, we know that any triangle of  $G$  with one vertex in  $V_1$  and two vertices in  $V_2$  must have an edge in  $\Delta_G(X, V_2)$ .

Now, in order to find a triangle with an edge in  $\Delta_G(X, V_2)$ , if such a triangle exists, the idea is to search for a set  $A \in \mathcal{S}(V_2, \lceil |V_2|^a \rceil)$  such that  $\Delta_G(X, A)$  contains an edge of a triangle. To find such a set  $A$ , the algorithm performs a quantum walk over the Johnson graph  $J(V_2, \lceil |V_2|^a \rceil)$ . The states of this walk correspond to the elements in  $\mathcal{S}(V_2, \lceil |V_2|^a \rceil)$ . The state corresponding to a set  $A \in \mathcal{S}(V_2, \lceil |V_2|^a \rceil)$  is marked if  $\Delta_G(X, A)$  contains an edge of a triangle of  $G$ . In case the set of marked states is not empty, the fraction of marked states is

$$\varepsilon = \Omega\left(|V_2|^{2(a-1)}\right).$$

The data structure of the walk stores the set  $\Delta_G(X, A)$ . Concretely, this is done by storing the couple  $(v, N_G(v) \cap X)$  for each  $v \in A$ , since this information is enough to construct  $\Delta_G(X, A)$  without using any additional query. The setup cost is  $S = |A| \times |X| = \tilde{O}(|V_2|^a |V_1|^k)$  queries. The update cost is  $U = 2|X| = \tilde{O}(|V_1|^k)$  queries. The query complexity of the quantum walk is

$$\tilde{O}\left(S + \sqrt{1/\varepsilon} \left(|V_2|^{a/2} \times S + C\right)\right), \quad (2)$$

where  $C$  is the cost of checking if a state is marked.

The checking procedure is done as follows: check if there exists a vertex  $w \in V_1$  such that  $\Delta_G(X, A)$  contains a pair  $\{v_1, v_2\}$  for which  $\{v_1, v_2, w\}$  is a triangle of  $G$ . For any  $w \in V_1$ , let  $Q(w)$  denote the query complexity of checking if there exists a pair  $\{v_1, v_2\} \in \Delta_G(X, A)$  such that  $\{v_1, v_2, w\}$  is a triangle of  $G$ . Using Ambainis' variable cost search [3] this checking procedure can be implemented using

$$C = \sqrt{\sum_{w \in V_1} Q(w)^2}$$

queries. It thus remains to give an upper bound on  $Q(w)$ . Let us fix  $w \in V_1$ . First, a tight estimator of the size of  $\Delta_G(X, A, w)$  is computed: the algorithm computes an integer  $\delta(X, A, w)$  such that  $|\delta(X, A, w) - |\Delta_G(X, A, w)|| \leq \frac{1}{10} \times |\Delta_G(X, A, w)|$ , which can be done in  $\tilde{O}(|V_1|^k)$  queries using (classical) sampling. The algorithm then performs a quantum walk over the Johnson graph  $J(A, \lceil |V_2|^b \rceil)$ . The states of this walk correspond to the elements in  $\mathcal{S}(A, \lceil |V_2|^b \rceil)$ . We now define the set of marked states of the walk. The state corresponding to a set  $B \in \mathcal{S}(A, \lceil |V_2|^b \rceil)$  is marked if  $B$  satisfies the following two conditions:

- (i) there exists a pair  $\{v_1, v_2\} \in \Delta_G(X, B, w)$  such that  $\{v_1, v_2\} \in E$  (i.e., such that  $\{v_1, v_2, w\}$  is a triangle of  $G$ );
- (ii)  $|\Delta_G(X, B, w)| \leq 10 \times |V_2|^{2(b-a)} \times \delta(X, A, w)$ .

The fraction of marked states is

$$\varepsilon' = \Omega\left(|V_2|^{2(b-a)}\right).$$

The data structure of the walk will store  $\Delta_G(X, B, w)$ . Concretely, this is done by storing the couple  $(v, e_v)$  for each  $v \in B$ , where  $e_v = 1$  if  $\{v, w\} \in E$  and  $e_v = 0$  if  $\{v, w\} \notin E$ . The setup cost is  $S' = \lceil |V_2|^b \rceil$  queries since it is sufficient to check if  $\{v, w\}$  is an edge for all  $v \in B$ . The update cost is  $U' = 2$  queries. The checking cost is

$$C'_w = O\left(\sqrt{|\Delta_G(X, B, w)|}\right) = O\left(\frac{|V_2|^b}{|V_2|^a} \sqrt{\delta(X, A, w)}\right) = O\left(\frac{|V_2|^b}{|V_2|^a} \sqrt{|\Delta(X, A, w)|}\right).$$

We thus obtain the bound

$$Q(w) = \tilde{O}\left(|V_1|^k + S' + \sqrt{1/\varepsilon'} \left(|V_2|^{b/2} \times U' + C'_w\right)\right),$$

and conclude that

$$C = \tilde{O} \left( \sqrt{|V_1|} \left( |V_1|^k + S' + \frac{|V_2|^{b/2} \times U'}{\sqrt{\varepsilon'}} \right) + \frac{|V_2|^{b-a}}{\sqrt{\varepsilon'}} \times \sqrt{\sum_{w \in V_1} |\Delta(X, A, w)|} \right).$$

The final key observation is that, since the set  $X$  is  $k$ -good for  $(G, V_1)$  with high probability, as guaranteed by Lemma 2.1, the term  $\sum_{w \in V} |\Delta(X, A, w)|$  in the above expression can be replaced by  $O(|V_2|^{2a} |V_1|^{1-k})$ , which enables us to express  $C$  as a function of  $a$ ,  $b$  and  $k$ , and then the complexity of the second part of the algorithm (Expression (2)) as a function of  $a$ ,  $b$  and  $k$ . The complexity of the whole algorithm (the maximum of Expression (1) and Expression (2)) can thus be written as a function of  $a$ ,  $b$  and  $k$  as well.

For the original triangle finding problem (i.e., for the case  $V_1 = V_2 = V$ ), taking  $a = \frac{3}{4}$  and  $b = k = \frac{1}{2}$  gives query complexity  $\tilde{O}(n^{5/4})$ .

### 3 Quantum Algorithm for Sparse Triangle Finding

In this section we describe our quantum algorithm for triangle finding in sparse graphs and prove Theorem 1.

Let  $d$  be a real number such that  $0 \leq d \leq 1$ . The value of this parameter will be set later. Define the following two subsets of  $V$ :

$$\mathcal{V}_h^d = \{v \in V \mid \deg(v) \geq \frac{9}{10} \times n^d\},$$

$$\mathcal{V}_l^d = \{v \in V \mid \deg(v) \leq \frac{11}{10} \times n^d\}.$$

A crucial observation is that  $|\mathcal{V}_h^d| = O(m/n^d)$ , since the graph  $G$  has  $m$  edges. The following proposition shows how to efficiently classify all the vertices of  $V$  into vertices in  $\mathcal{V}_h^d$  and vertices in  $\mathcal{V}_l^d$ .

**Proposition 3.1.** *There exists a quantum algorithm using  $Q_1 = \tilde{O}(n^{1-d} \sqrt{m})$  queries that partitions the set  $V$  into two sets  $V_h^d$  and  $V_l^d$  such that, with high probability,  $V_h^d \subseteq \mathcal{V}_h^d$  and  $V_l^d \subseteq \mathcal{V}_l^d$ .*

*Proof.* Let  $v$  be any vertex in  $V$ . Using quantum counting [5] we can compute, using  $\tilde{O}(\sqrt{\frac{n}{n^d}})$  queries, a value  $a(v)$  such that  $|a(v) - \deg(v)| \leq n^d/100$  with probability at least  $1 - 1/\text{poly}(n)$ . We use  $a(v)$  to classify  $v$  as follows: we decide “ $v$  is in  $\mathcal{V}_h^d$ ” if  $a(v) \geq n^d$ , and decide “ $v$  is in  $\mathcal{V}_l^d$ ” if  $a(v) < n^d$ . This decision is correct with probability at least  $1 - 1/\text{poly}(n)$ .

We can thus apply quantum enumeration as described in Section 2.1 to obtain a set  $V_h^d \subseteq V$  of vertices such that, with high probability, all the vertices in  $V_h^d$  are in  $\mathcal{V}_h^d$  and all the vertices in  $V \setminus V_h^d$  are in  $\mathcal{V}_l^d$ . We then take  $V_l^d = V \setminus V_h^d$ . The overall complexity of this approach is  $\tilde{O}(\sqrt{n \times \frac{m}{n^d}} \times \sqrt{\frac{n}{n^d}}) = \tilde{O}(n^{1-d} \sqrt{m})$  queries, since  $|\mathcal{V}_h^d| = O(m/n^d)$ .  $\square$

In the remaining of the section we assume that the algorithm of Proposition 3.1 outputs a correct classification (i.e.,  $V_h^d \subseteq \mathcal{V}_h^d$  and  $V_l^d \subseteq \mathcal{V}_l^d$ ), which happens with high probability. In particular we assume that  $|\mathcal{V}_h^d| = O(m/n^d)$ . We will say that a vertex  $v \in V$  is  $d$ -high if  $v \in V_h^d$ , and say it is  $d$ -low if  $v \in V_l^d$ . Once the vertices have been classified, checking if  $G$  has a triangle can be divided into four subproblems: checking if  $G$  has a triangle with three  $d$ -low vertices, checking if  $G$  has a triangle with two  $d$ -low vertices and one  $d$ -high vertex, checking if  $G$  has a triangle with one  $d$ -low-degree vertex and two  $d$ -high vertices, and checking if  $G$  has a triangle with three high-degree triangles. We now present six procedures to handle these cases (for some cases we present more than one procedure to allow us to choose which procedure to use according to the value of  $m$ ).

**Proposition 3.2.** *Let  $a_1, k_1$  and  $b_1$  be any constants such that  $0 < a_1, k_1 < 1$  and  $0 < b_1 < a_1$ . There exists a quantum algorithm that finds a triangle of  $G$  consisting of three  $d$ -low vertices, if such a triangle exists, with high probability using  $Q_2 = \tilde{O}(n + n^{k_1/2}m^{1/2} + n^{a_1+d/2+k_1-1/2} + n^{1/2+d/2+k_1-a_1/2} + n^{3/2+k_1/2-a_1} + n^{1+b_1+d/2-a_1} + n^{3/2-b_1/2} + n^{3/2-k_1/2})$  queries.*

The proof of Proposition 3.2 will use the following key lemma.

**Lemma 3.1.** *Let  $k$  be any constant such that  $0 < k < 1$ . Suppose that  $X$  is a set of size  $|X| = \lceil 3n^k \log n \rceil$  obtained by taking uniformly at random vertices from  $V_l^d$ . Then, with probability at least*

$$1 - \frac{1}{n \exp(\frac{231}{10}n^{d+k-1} \log n)},$$

the inequality

$$|N_G(v) \cap X| < \frac{33}{10}n^{d+k-1} \log n + 2 \log n$$

holds for all vertices  $v \in V_l^d$ .

*Proof.* Without loss of generality assume that  $|N_G(v) \cap V_l^d| = \frac{11}{10}n^d$  for any  $v \in V_l^d$  (remember that we assume that  $V_l^d \subseteq \mathcal{V}_l^d$ ). The quantity  $|N_G(v) \cap X|$  is a random variable distributed according to the hypergeometric distribution. The expected value of  $|N_G(v) \cap X|$  is  $\mu = |N_G(v) \cap V_l^d| \times \frac{|X|}{|V_l^d|} = \frac{33}{10} \times n^{d+k-1} \log n$ . By Corollary 2.4 and Theorem 2.10 of [11],  $\Pr[|(N_G(v) \cap X)| \geq 7\mu + 2 \log n] \leq \exp(-7\mu) \frac{1}{n^2}$ . Thus the inequality

$$|N_G(v) \cap X| < \frac{33}{10}n^{d+k-1} \log n + 2 \log n$$

holds for all  $v \in V_l^d$  with probability at least

$$1 - |V_l^d| \times \frac{1}{n^2 \exp(\frac{231}{10}n^{d+k-1} \log n)} = 1 - \frac{1}{n \exp(\frac{231}{10}n^{d+k-1} \log n)},$$

as claimed.  $\square$

*Proof of Proposition 3.2.* We adapt the algorithm for the dense case presented in Section 2.2. We take  $V_1 = V_2 = V_l^d$ , and  $X \subseteq V_1$  of size  $|X| = \lceil 3|V_1|^{k_1} \log n \rceil$ .

We replace the first step of the algorithm, which checks if there exists a triangle of  $G$  with a vertex in  $X$  and two vertices in  $V_2$ , by the following procedure based on [6]. We take a random edge  $\{u, v\} \in \mathcal{E}(V_2) \cap E$  and then try to find a vertex  $w$  from  $X$  such that  $\{u, v, w\}$  is a triangle of  $G$ . Note that this can be implemented using two Grover searches in  $\tilde{O}(\sqrt{|\mathcal{E}(V_2)|/|\mathcal{E}(V_2) \cap E|} + \sqrt{|X|})$  queries, and that in the worst case (i.e., when there is only one triangle) the success probability of this approach is  $\Theta(1/|\mathcal{E}(V_2) \cap E|)$ . Using amplitude amplification we can then check with high probability the existence of such a triangle with total query complexity

$$\tilde{O}\left(\sqrt{|\mathcal{E}(V_2) \cap E|} \times (\sqrt{|\mathcal{E}(V_2)|/|\mathcal{E}(V_2) \cap E|} + \sqrt{|X|})\right) = \tilde{O}(n + \sqrt{n^{k_1}m}). \quad (3)$$

We now show how to adapt the second step of the algorithm presented in Section 2.2 to exploit the sparsity of the graph. First, as observed in [8], the cost of estimating the size of  $\Delta_G(X, A, w)$  can be reduced to  $\tilde{O}(\sqrt{n^{k_1}})$  queries by using quantum counting instead of random sampling (quantum counting was not used in [8] since it did not result in any speed-up for the dense case, but for the sparse case this is necessary). We now describe our main ideas to exploit the sparsity of the graph, and show how to reduce the cost of two quantum walks.

First, we describe how to reduce the setup cost  $S$  and the update cost  $U$  as follows. By Lemma 3.1, we know that  $|N_G(v) \cap X| < t$  for all  $v \in V_2$ , where  $t = \frac{33}{10}n^{d_1+k_1-1} \log n + 2 \log n$ . Therefore we can use quantum enumeration to find all vertices in  $N_G(v) \cap X$  with

$$\tilde{O} \left( \sqrt{\frac{|X|}{t}} + \dots + \sqrt{\frac{|X|}{1}} \right) = \tilde{O}(\sqrt{|X|t})$$

queries. The setup cost  $S$  is thus

$$S = \tilde{O} \left( |A| \times \sqrt{|X|t} \right) = \tilde{O}(n^{a_1+k_1+d/2-1/2})$$

queries, and the update cost  $U = \tilde{O}(\sqrt{|X|t}) = \tilde{O}(n^{k_1+d/2-1/2})$  queries.

Next, we describe how to reduce the setup cost  $S'$ . This set up requires to obtain the couple  $(v, e_v)$  for each  $v \in B$ , where  $w$  is a fixed vertex in  $V_1$ ,  $e_v = 1$  if  $\{v, w\} \in E$  and  $e_v = 0$  if  $\{v, w\} \notin E$ . Let  $\mu(w) = n^d \times \frac{|B|}{|V|} = n^{d+b_1-1}$  be the average of  $|N_G(w) \cap B|$  over all  $B$ . We use quantum enumeration to find at most  $10 \times \mu(w)$  vertices in  $N_G(w) \cap B$  from  $B$ . Thus the cost of this procedure is

$$S' = \tilde{O} \left( \sqrt{\frac{|B|}{|\mu(w)|}} + \dots + \sqrt{\frac{|B|}{1}} \right) = \tilde{O}(\sqrt{|B| \times |\mu(w)|}) = \tilde{O}(n^{b_1+d/2-1/2})$$

queries. Note that this procedure will not correctly prepare the database for all  $B$ 's (since  $|N_G(w) \cap B|$  may exceeds  $10 \times \mu(w)$  for some  $B$ 's); it will prepare correctly the database only for a large fraction of the  $B$ 's. This is nevertheless not a problem since the initial state of the quantum walk is a uniform superposition of all the  $B$ 's: this procedure will thus prepare a state close enough to the ideal state, which will modify only in a negligible way the final success probability of the whole walk.

We also modify the definition of a marked state for the second walk (we add one condition). Namely, the state corresponding to a set  $B \in \mathcal{S}(A, [|V_2|^{b_1}])$  will be marked if  $B$  satisfies the following three conditions:

- (i) there exist two vertices  $v_1, v_2 \in B$  such that  $\{v_1, v_2\} \in E$  (i.e., such that  $\{v_1, v_2, w\}$  is a triangle of  $G$ );
- (ii)  $|\Delta_G(X, B, w)| \leq 10 \times |V_2|^{2(b_1-a_1)} \times \delta(X, A, w)$ ;
- (iii)  $|N_G(w) \cap B| \leq 10 \times \mu(w)$ .

It is easy to show that adding the third condition does not change significantly the fraction of marked states:

$$\begin{aligned} \varepsilon' &= \Pr[v_1 \in B \text{ and } v_2 \in B \text{ and } |N_G(w) \cap B| \leq 10\mu(w)] \\ &= \Pr[v_1 \in B] \Pr[v_2 \in B \mid v_1 \in B] \Pr[|N_G(w) \cap B| \leq 10\mu(w) \mid v_1 \in B \text{ and } v_2 \in B] \\ &= \Pr[v_1 \in B] \Pr[v_2 \in B \mid v_1 \in B] \Pr[|N_G(w) \cap B'| \leq 10\mu(w) - 2] \\ &\geq (1 - \Pr[|N_G(w) \cap B'| \geq 10\mu(w) - 2]) \times \Pr[v_1 \in B] \Pr[v_2 \in B \mid v_1 \in B] \\ &\geq (1 - \Pr[|N_G(w) \cap B'| \geq 10\mu(w)' - 2]) \times \Pr[v_1 \in B] \Pr[v_2 \in B \mid v_1 \in B] \\ &\geq (1 - \Pr[|N_G(w) \cap B'| \geq 2\mu(w)']) \times \Pr[v_1 \in B] \Pr[v_2 \in B \mid v_1 \in B] \\ &\geq \left(1 - \frac{1}{2}\right) \times \Pr[v_1 \in B] \Pr[v_2 \in B \mid v_1 \in B] = \Omega \left( n^{2(b_1-a_1)} \right), \end{aligned}$$

where  $B' \in \mathcal{S}(A \setminus \{v_1, v_2\}, [|V_2|^{b_1}] - 2)$  and  $\mu(w)' = n^d \times \frac{|B'|}{|V|}$  be the average of  $|N_G(w) \cap B'|$  over all  $B'$ .

The checking procedure of the second walk (and thus its cost  $C'_w$ ) is the same as in the dense case.

By evaluating the performance of the walks as done for the dense case in Section 2.2, but replacing Expression (1) by Expression (3) and replacing in the evaluation of Expression (2) the quantities  $S$ ,  $U$ ,  $S'$  and the cost of estimating  $|\Delta_G(X, A, w)|$  by the expressions we just derived, we obtain the claimed query complexity for the whole algorithm. For instance, the checking cost of the first walk is

$$\begin{aligned}
C &= \tilde{O} \left( \sqrt{|V_1|} \left( |V_1|^{k_1/2} + S' + \frac{|V_2|^{b_1/2} \times U'}{\sqrt{\varepsilon'}} \right) + \frac{|V_2|^{b_1-a_1}}{\sqrt{\varepsilon'}} \times \sqrt{\sum_{w \in V_1} |\Delta(X, A, w)|} \right) \\
&= \tilde{O} \left( n^{1/2+k_1/2} + n^{1/2} \times S' + n^{1/2+a_1-b_1/2} + \sqrt{\sum_{w \in V_1} |\Delta(X, A, w)|} \right) \\
&= \tilde{O} \left( n^{1/2+k_1/2} + n^{b_1+d/2} + n^{1/2+a_1-b_1/2} + n^{1/2+a_1-k_1/2} \right)
\end{aligned}$$

queries.  $\square$

**Proposition 3.3.** *Let  $a_2$ ,  $k_2$  and  $b_2$  be any constants such that  $0 < a_2 < 1$ ,  $1 < n^{k_2} < |V_h^d|$  and  $0 < b_2 < a_2$ . A triangle of  $G$  consisting of two  $d$ -low vertices and one  $d$ -high vertex can be detected with high probability using  $Q_3 = \tilde{O}(n + n^{k_2/2}m^{1/2} + n^{a_2+d+k_2}m^{-1/2} + n^{1+d+k_2-a_2/2}m^{-1/2} + n^{1+k_2/2-a_2-d/2}m^{1/2} + n^{1+b_2-a_2-d/2}m^{1/2} + n^{1-b_2/2-d/2}m^{1/2} + n^{1-d/2-k_2/2}m^{1/2})$  queries.*

*Proof.* We again adapt the algorithm for the dense case presented in Section 2.2. We take  $V_1 = V_h^d$ ,  $V_2 = V_l^d$ ,  $a = a_2$ ,  $b = b_2$  and  $X \subseteq V_1$  of size  $|X| = \lceil 3n^{k_2} \log n \rceil$  (i.e., choose  $k$  such that  $|V_1|^k = n^{k_2}$ ). The algorithm is exactly the same as the algorithm of Proposition 3.2, except that this time we cannot use the sparsity of the graph (since the vertices in  $V_1$  are not  $d$ -low anymore) to reduce  $S'$ . Instead, we use the same set up procedure as for the dense case in the second walk.  $\square$

**Proposition 3.4.** *Let  $a_3$ ,  $k_3$  and  $b_3$  be constants such that  $1 < n^{a_3} < |V_h^d|$ ,  $0 < k_3 < 1$  and  $0 < b_3 < a_3$ . A triangle of  $G$  consisting of two  $d$ -high vertices and one  $d$ -low vertex can be detected with high probability using  $Q_4 = \tilde{O}(n + n^{k_3/2}m^{1/2} + n^{a_3+k_3} + n^{k_3-a_3/2-d}m + n^{1/2+k_3/2-a_3-d}m + n^{b_3-a_3-d/2}m + n^{1/2-b_3/2-d}m + n^{1/2-d-k_3/2}m)$  queries.*

*Proof.* We adapt the algorithm for the dense case to the case we are considering. This time we choose  $V_1 = V_l^d$ ,  $V_2 = V_h^d$ ,  $k = k_3$ , take  $a$  such that  $|V_2|^a = n^{a_3}$  and  $b$  such that  $|V_2|^b = n^{b_3}$ . The algorithm is again almost the same as the algorithm of Proposition 3.2, except that this time we cannot use the sparsity of the graph to reduce  $S$  and  $U$  since the vertices in  $V_2$  are not  $d$ -low anymore (but we can use the sparsity to reduce  $S'$  exactly as in the algorithm of Proposition 3.2). Instead, we use the same set up and checking procedures as for the dense case in the first walk.  $\square$

**Proposition 3.5.** *A triangle of  $G$  consisting of three  $d$ -high vertices can be detected with high probability using  $Q_5 = \tilde{O}((m/n^d)^{5/4})$  queries.*

*Proof.* We simply apply the original algorithm by Le Gall [8] for dense triangle finding over the subgraph of  $G$  induced by the vertices in  $V_h^d$ , and use the bound  $|V_h^d| = O(m/n^d)$ .  $\square$

**Proposition 3.6.** *Let  $b_4$  be any constant such that  $0 < b_4 < 1$ . A triangle of  $G$  consisting of three  $d$ -low vertices can be detected with high probability using  $Q_6 = \tilde{O}(n^{b_4+d/2} + n^{3/2-b_4/2} + n^{1/2+d})$  queries.*

*Proof.* We take  $V_1 = V_2 = V_l^d$ , and adapt the algorithm for the dense case as in Proposition 3.2, but we choose  $X = \emptyset$  and  $a = 1$ , i.e., we do not perform the first step of the algorithm and do not perform the first walk in the second step. That is, we only perform the second walk of the algorithm, with parameter  $b = b_4$ . The sparsity of the graph can again be used to reduce  $S'$ , exactly as in the algorithm of Proposition 3.2. The main difference is that we now use directly the sparsity of the graph for the

checking step  $C'$ : instead of performing a Grover search over  $\Delta_G(X, B, w)$ , as in the dense case, we simply do a Grover search over  $\mathcal{E}(N(w) \cap B)$ , at cost

$$\tilde{O}(\sqrt{|\mathcal{E}(N(w) \cap B)|}) = \tilde{O}(10 \times \mu(w)),$$

which gives a new upper bound  $C'_w$ . Replacing in the analysis of Section 2.2 the quantities  $S'$  and  $C'_w$  by these upper bounds, we obtain the claimed query complexity

$$\begin{aligned} C &= \tilde{O}\left(\sqrt{|V_1|}\left(S' + \sqrt{\frac{1}{\varepsilon'}}\left(|V_2|^{b_4/2} \times U' + C'_w\right)\right)\right) \\ &= \tilde{O}(n^{1/2} \times S' + n^{3/2-b_4/2} + n^{3/2-b_4} \times C'_w) \\ &= \tilde{O}(n^{b_4+d/2} + n^{3/2-b_4/2} + n^{3/2-b_4} \times \mu(w)) = \tilde{O}(n^{b_4+d/2} + n^{3/2-b_4/2} + n^{1/2+d}) \end{aligned}$$

□

**Proposition 3.7.** *A triangle consisting of at least one  $d$ -high vertex can be detected with high probability using  $Q_7 = O(n + n^{-d/2}m)$  queries.*

*Proof.* We use an algorithm similar to the procedure described in the first part of the proof of Proposition 3.2, based on [6]. We take a random edge  $\{u, v\} \in E$  and then try to find a vertex  $w$  from  $V_h^d$  such that  $\{u, v, w\}$  is a triangle of  $G$ . This can be implemented using two Grover searches in  $\tilde{O}(\sqrt{n^2/m} + \sqrt{|V_h^d|})$  queries, and that in the worst case (i.e., when there is only one triangle) the success probability of this approach is  $\Theta(1/m)$ . Using amplitude amplification we can then check with high probability the existence of such a triangle with total query complexity

$$\tilde{O}\left(\sqrt{m} \times \left(\sqrt{n^2/m} + \sqrt{|V_h^d|}\right)\right) = \tilde{O}\left(n + \frac{m}{\sqrt{n^d}}\right),$$

since  $|V_h^d| = O(m/n^d)$ . □

We are now ready to prove Theorem 1.

*Proof of Theorem 1.* From Propositions 1, 2, 3, 4, 5, 6 and 7 and the discussion before Proposition 2, the query complexity of our whole algorithm is

$$\min [(Q_1 + Q_6 + Q_7), (Q_1 + Q_3 + Q_4 + Q_5 + Q_6), (Q_1 + Q_2 + Q_3 + Q_4 + Q_5)].$$

We write  $m = n^\ell$ , for  $0 \leq \ell \leq 2$ , and optimize below the parameters.

If  $\frac{7}{6} \leq \ell \leq \frac{7}{5}$ , the query complexity is upper bounded by

$$Q_1 + Q_6 + Q_7 = \tilde{O}(n^{1+\ell/2-d} + n^{3/2-b_4/2} + n^{b_4+d/2}),$$

which is optimized by taking  $b_4 = 1 - \frac{\ell}{7}$  and  $d = \frac{3\ell}{7}$ , giving the upper bound  $\tilde{O}(n^{1+\ell/14})$ .

If  $\frac{7}{5} \leq \ell \leq \frac{3}{2}$ , the query complexity is upper bounded by

$$Q_1 + Q_6 + Q_7 = \tilde{O}(n^{3/2-b_4/2} + n^{1/2+d} + n^{\ell-d/2}),$$

which is optimized by taking  $b_4 = \frac{8}{3} - \frac{4\ell}{3}$  and  $d = \frac{2\ell}{3} - \frac{1}{3}$ , giving the upper bound  $\tilde{O}(n^{1/6+2\ell/3})$ .

If  $\frac{3}{2} \leq \ell \leq \frac{13}{8}$ , the query complexity is upper bounded by

$$\begin{aligned} Q_1 + Q_3 + Q_4 + Q_5 + Q_6 &= \tilde{O}(n^{3/2-b_4/2} + n^{1/2+d} + n^{a_2+d+k_2-\ell/2} \\ &\quad + n^{1+b_2+\ell/2-a_2-d/2} + n^{1+\ell/2-b_2/2-d/2} + n^{1+\ell/2-d/2-k_2/2} \\ &\quad + n^{b_3+\ell-a_3-d/2} + n^{1/2+\ell-b_3/2-d} + n^{1/2+\ell-d-k_3/2}), \end{aligned}$$

which is optimized by taking  $a_2 = \frac{3}{10} + \frac{3\ell}{10}$ ,  $a_3 = \frac{23\ell}{15} - \frac{59}{30}$ ,  $b_2 = k_2 = \frac{1}{5} + \frac{\ell}{5}$ ,  $b_3 = k_3 = \frac{14\ell}{15} - \frac{16}{15}$ ,  $b_4 = \frac{22}{15} - \frac{8\ell}{15}$  and  $d = \frac{4}{15} + \frac{4\ell}{15}$ , giving the upper bound  $\tilde{O}(n^{23/30+4\ell/15})$ .

If  $\frac{13}{8} \leq \ell \leq 2$ , the query complexity is upper bounded by

$$\begin{aligned} Q_1 + Q_2 + Q_3 + Q_4 + Q_5 = & \tilde{O}(n^{a_1+d/2+k_1-1/2} + n^{1+b_1+d/2-a_1} \\ & + n^{3/2-b_1/2} + n^{3/2-k_1/2} + n^{a_2+d+k_2-\ell/2} + n^{1+b_2+\ell/2-a_2-d/2} + n^{1+\ell/2-b_2/2-d/2} \\ & + n^{1+\ell/2-d/2-k_2/2} + n^{b_3+\ell-a_3-d/2} + n^{1/2+\ell-b_3/2-d} + n^{1/2+\ell-d-k_3/2}), \end{aligned}$$

which is optimized by taking  $a_1 = \frac{3}{4}$ ,  $a_2 = \frac{19}{20} - \frac{\ell}{10}$ ,  $a_3 = \frac{3\ell}{5} - \frac{9}{20}$ ,  $b_1 = k_1 = \frac{31}{30} - \frac{4\ell}{15}$ ,  $b_2 = k_2 = \frac{19}{30} - \frac{\ell}{15}$ ,  $b_3 = k_3 = \frac{7}{30} + \frac{2\ell}{15}$  and  $d = \frac{4\ell}{5} - \frac{3}{5}$ , giving the upper bound  $\tilde{O}(n^{59/60+2\ell/15})$ .

For the case  $\ell \leq \frac{7}{6}$  we can obtain a better upper bound by using the  $O(n + \sqrt{nm})$ -query algorithm by Buhrman et al. [6]. This upper bound actually corresponds to a degenerate case appearing in our approach: the case  $d = 0$ . Indeed, observe that without loss of generality we can assume that  $\deg(v) \geq 1$  for all vertices  $v \in V$  (for instance by adding dummy vertices to the graph). In this case we have  $\mathcal{V}_h^d = V$  for  $d = 0$ , which means that we do not need to apply the algorithm of Proposition 3.1 in order to obtain a classification: we simply output  $V_h^0 = V$  and  $V_l^0 = \emptyset$  (i.e., all the vertices of the graph are 0-high). The only type of triangles we need to consider is triangles with three 0-high vertices, which can be found with complexity  $O(n + \sqrt{nm})$  by Proposition 3.7 (in this case the algorithm of Proposition 3.7 is exactly the same as the algorithm in [6]).  $\square$

## Acknowledgments

The authors are grateful to Mathieu Laurière, Keiji Matsumoto, Harumichi Nishimura and Seiichiro Tani for helpful comments. This work is supported by the Grant-in-Aid for Young Scientists (B) No. 24700005, the Grant-in-Aid for Scientific Research (A) No. 24240001 of the Japan Society for the Promotion of Science and the Grant-in-Aid for Scientific Research on Innovative Areas No. 24106009 of the Ministry of Education, Culture, Sports, Science and Technology in Japan.

## References

- [1] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:354–364, 1997.
- [2] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007.
- [3] Andris Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, 2010.
- [4] Aleksandrs Belovs. Span programs for functions with constant-sized 1-certificates: extended abstract. In *Proceedings of the 44th Symposium on Theory of Computing*, pages 77–84, 2012.
- [5] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum counting. In *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*, pages 820–831, 1998.
- [6] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005.
- [7] Andrew M. Childs and Robin Kothari. Quantum query complexity of minor-closed graph properties. *SIAM Journal on Computing*, 41(6):1426–1450, 2012.

- [8] François Le Gall. Improved quantum algorithm for triangle finding via combinatorial arguments. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, pages 216–225, 2014.
- [9] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Symposium on the Theory of Computing*, pages 212–219, 1996.
- [10] Alon Itai and Michael Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
- [11] Svante Janson, Tomasz Luczak, and Andrzej Rucinski. *Random graphs*, volume 45. John Wiley & Sons, 2011.
- [12] Stacey Jeffery, Robin Kothari, and Frédéric Magniez. Nested quantum walks with quantum data structures. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1474–1485, 2013.
- [13] Troy Lee, Frédéric Magniez, and Miklos Santha. Improved quantum query algorithms for triangle finding and associativity testing. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1486–1502, 2013.
- [14] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011.
- [15] Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.
- [16] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd Symposium on Theory of Computing*, pages 603–610, 2010.
- [17] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the 51th Symposium on Foundations of Computer Science*, pages 645–654, 2010.
- [18] Virginia Vassilevska Williams and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing*, 42(3):831–854, 2013.