CHICAGO JOURNAL OF THEORETICAL COMPUTER SCIENCE
http://cjtcs.cs.uchicago.edu/

Learning graph based quantum query algorithms for finding constant-size subgraphs[*]

Troy Lee[†] Frédéric Magniez[‡] Miklos Santha[§]

December 17, 2012

Abstract: Let H be a fixed k-vertex graph with m edges and minimum degree $d > 0$. We use the learning graph framework of Belovs to show that the bounded-error quantum query complexity of determining if an n-vertex graph contains H as a subgraph is $O(n^{2-2/k-t})$, where

$$t = \max\left\{\frac{k^2 - 2(m+1)}{k(k+1)(m+1)}, \frac{2k-d-3}{k(d+1)(m-d+2)}\right\} > 0 \ .$$

The previous best algorithm of Magniez et al. had complexity $\widetilde{O}(n^{2-2/k})$.

1 Introduction

Quantum query complexity. Quantum query complexity has been a very successful model for studying the power of quantum computation. Important quantum algorithms, in particular the search algorithm of Grover [10] and the period finding subroutine of Shor's factoring algorithm [18], can be formulated in

[*]Partially supported by the French ANR Defis project ANR-08-EMER-012 (QRAC) and the European Commission IST STREP project 25596 (QCS). Research at the Centre for Quantum Technologies is funded by the Singapore Ministry of Education and the National Research Foundation.

[†]`troyjlee@gmail.com`

[‡]`frederic.magniez@univ-paris-diderot.fr`

[§]`miklos.santha@liafa.univ-paris-diderot.fr`

Key words and phrases: quantum algorithms, query complexity, finding subgraphs, learning graph

this model, yet it is still simple enough that one can often prove tight lower bounds. This model is the quantum analog of deterministic and randomized decision tree complexities; the resource measured is the number of queries to the input and all other operations are for free.

For promise problems the quantum query complexity can be exponentially smaller than the classical complexity, the Hidden Subgroup Problem [19, 9] being the most striking example. The situation is dramatically different for total functions, as Beals et al. [2] showed that in this case the deterministic and the quantum query complexities are polynomially related.

One rich source of concrete problems are functions related to properties of graphs. Graph problems were first studied in the quantum query model by Buhrman et al. [5] and later by Buhrman et al. [6], who looked at Triangle Finding together with Element Distinctness. This was followed by the exhaustive work of Dürr et al. [8] who investigated many standard graph problems including Connectivity, Strong Connectivity, Minimum Spanning Tree, and Single Source Shortest Paths. All these approaches were based on clever uses of Grover's search algorithm. The groundbreaking work of Ambainis [1] using quantum walks for Element Distinctness initiated the study of quantum walk based search algorithms. Magniez et al. [15] used this technique to design quantum query algorithms for finding constant size subgraphs, and recently Childs and Kothari found a novel application of this framework to decide minor-closed graph properties [7]. The results of [15] imply that a k-vertex subgraph can be found with $\widetilde{O}(n^{2-2/k})$ queries, and moreover Triangle Finding is solvable with $\widetilde{O}(n^{1.3})$ queries. Later, quantum phase estimation techniques [14] were also applied to these problems, and in particular the quantum query complexity of Triangle Finding was improved to $O(n^{1.3})$. The best lower bound known for finding any constant sized subgraph is the trivial $\Omega(n)$.

The general adversary bound and learning graphs. Recently, there have been exciting developments leading to a characterization of quantum query complexity in terms of a (relatively) simple semidefinite program, the general adversary bound [17, 13]. Now to design quantum algorithms it suffices to exhibit a solution to this semidefinite program. This plan turns out to be quite difficult as the minimization form of the general adversary bound (the easiest form to upper bound) has exponentially many constraints. Even for simple functions it is difficult to directly come up with a feasible solution, much less worry about finding a solution with good objective value.

Belovs [4] recently introduced the model of learning graphs, which can be viewed as the minimization form of the general adversary bound with additional structure imposed on the form of the solution. This additional structure makes learning graphs much easier to reason about. In particular, it ensures that the feasibility constraints are *automatically* satisfied, allowing one to focus on coming up with a solution having a good objective value. Learning graphs are a very promising model and have already been used to improve the complexity of Triangle Finding to $O(n^{35/27})$ [4] and to give an $o(n^{3/4})$ algorithm for k-Element Distinctness [3], improving the previous bound of $O(n^{k/(k+1)})$ [1].

Our contribution. We give two learning graph based algorithms for the problem of determining if a graph G contains a fixed k-vertex subgraph H. Throughout the paper we will assume that $k > 2$, as the problem of determining if G contains an edge is equivalent to search. We denote by m the number of edges in H. The first algorithm we give has complexity $O(n^{2-2/k-t})$ where $t = (k^2 - 2(m+1))/(k(k+1)(m+1)) > 0$. The second algorithm depends on the minimum degree of a vertex in H. Say that the smallest degree of a vertex in H is $d > 0$. This is without loss of generality as isolated vertices of H can be removed and the theorem applied to the resulting graph H'. The second algorithm has complexity

$O(n^{2-2/k-t})$ where $t = (2k-d-3)/(k(d+1)(m+2)) > 0$. Both algorithms thus improve on the previous best general subgraph finding algorithm of [15], which has complexity $\widetilde{O}(n^{2-2/k})$. The first algorithm performs better, for example, on dense regular graphs H, while the second algorithm performs better on the important case where H is a triangle, having complexity $O(n^{35/27})$, equal to that of the algorithm of Belovs [4].

To explain these algorithms, we first give a high level description of the learning graph algorithm in [4] for Triangle Finding, and its relation to the quantum walk algorithm given in [15]. The learning graph algorithm in [4] for Triangle Finding is roughly a translation of the quantum walk algorithm on the Johnson graph of [15] into the learning graph framework, with one additional twist. This is to maintain a database not of all edges present in G amongst a subset of r-vertices but rather a random sample of these edges. We will refer to this as sparsifying the database. While in the quantum walk world this idea does not help, in the context of learning graphs it leads to a better algorithm.

The quantum walk of [15] works by looking for a subgraph $H' = H \setminus \{v\}$, where v is a vertex of minimal degree in H, and then (using the algorithm for element distinctness) finding the vertex v and the edges linking it to H' to form H. Our second learning graph algorithm translates this procedure into the learning graph framework, and again applies the trick of sparsifying the database. Our first algorithm is simpler and translates the quantum walk searching for H directly to the learning graph framework, again maintaining a sparsified database.

The way we apply sparsification differs from how it is used in [4]. There every edge slot is taken independently with some fixed probability, while in our case the sparse random graphs are chosen uniformly from a set of structured multipartite graphs whose edge pattern reflects that of the given subgraph. The probability space evolves during the algorithm, but at every stage the multipartite graphs have a very regular degree structure. This uniformity of the probability space renders the structure of the learning graph very transparent.

Related contribution. Independently of our work, Zhu [20] also obtained Theorem 4.3. His algorithm is also based on learning graphs, but differs from ours in working with randomly sparsified cliques as in the algorithm of Belovs [4] for Triangle Finding, rather than graphs with specified degrees as we do.

2 Preliminaries

We denote by $[N]$ the set $\{1,2,\ldots,N\}$. The *quantum query complexity* of a function f, denoted $Q(f)$, is the number of input queries needed to evaluate f with error at most $1/3$. We refer the reader to the survey [11] for precise definitions and background.

For a boolean function $f : \mathcal{D} \to \{0,1\}$ with $\mathcal{D} \subseteq \{0,1\}^N$, the general adversary bound [12], denoted $\mathrm{ADV}^{\pm}(f)$, can be defined as follows (this formulation was first given in [16]):

$$\begin{aligned} \mathrm{ADV}^{\pm}(f) = \underset{u_{x,i}}{\text{minimize}} \quad & \max_{x\in\mathcal{D}} \sum_{i\in[N]} \|u_{x,i}\|^2 \\ \text{subject to} \quad & \sum_{\substack{i\in[N] \\ x_i\neq y_i}} \langle u_{x,i}|u_{y,i}\rangle = 1 \text{ for all } f(x)\neq f(y) \quad . \end{aligned} \tag{2.1}$$

As the general adversary bound characterizes quantum query complexity [17], quantum algorithms can be developed (simply!) by devising solutions to this semidefinite program. This turns out not to be so simple, however, as even coming up with *feasible* solutions to Equation (2.1) is not easy because of the large number of strict constraints.

Learning graphs are a model of computation introduced by Belovs [4] that give rise to solutions of Equation (2.1) and therefore quantum query algorithms. The model of learning graphs is very useful as it ensures that the constraints are satisfied automatically, allowing one to focus on coming up with a solution having a good objective value.

Definition 2.1 (Learning graph)**.** A learning graph $\mathcal{G}$ is a 5-tuple $(\mathcal{V},\mathcal{E},w,\ell,\{p_y : y \in Y\})$ where $(\mathcal{V},\mathcal{E})$ is a rooted, weighted and directed acyclic graph, the weight function $w : \mathcal{E} \to \mathbb{R}$ maps learning graph edges to positive real numbers, the length function $\ell : \mathcal{E} \to \mathbb{N}$ assigns each edge a natural number, and $p_y : \mathcal{E} \to \mathbb{R}$ is a unit flow whose source is the root, for every $y \in Y$.

Definition 2.2 (Learning graph for a function)**.** Let $f : \{0,1\}^N \to \{0,1\}$ be a function. A learning graph $\mathcal{G}$ for f is a 5-tuple $(\mathcal{V},\mathcal{E},S,w,\{p_y : y \in f^{-1}(1)\})$, where $S : \mathcal{V} \to 2^N$ maps $v \in \mathcal{V}$ to a set $S(v) \subseteq [N]$ of variable indices, and $(\mathcal{V},\mathcal{E},w,\ell,\{p_y : y \in f^{-1}(1)\})$ is a learning graph for the length function ℓ defined as $\ell((u,v) = |S(v) \setminus S(u)|$ for each edge (u,v). For the root $r \in \mathcal{V}$ we have $S(r) = \emptyset$, and every learning graph edge $e = (u,v)$ satisfies $S(u) \subseteq S(v)$. For each input $y \in f^{-1}(1)$, the set $S(v)$ contains a 1-certificate for y on f, for every sink $v \in \mathcal{V}$ of p_y.

Note that it can be the case for an edge (u,v) that $S(u) = S(v)$ and the length of the edge is zero. In Belovs [4] what we define here is called a reduced learning graph, and a learning graph is restricted to have all edges of length one.

Definition 2.3 (Flow preserving edge sets)**.** A set of edges $E \subseteq \mathcal{E}$ is *flow preserving*, if in the subgraph $G = (V,E)$ induced by E, for every vertex $v \in V$ which is not a source or a sink in G, $\sum_{u\in V} p_y((u,v)) = \sum_{w\in V} p_y((v,w))$, for every y. For a flow preserving set of edges E we let $p_y(E)$ denote *the value of the flow p_y over E*, that is $p_y(E) = \sum_{s:\text{source in } G} \sum_{v\in V} p_y((s,v))$.

Observe that $p_y/p_y(E)$ is a unit flow over E whenever $p_y(E) \neq 0$, and that $p_y(\mathcal{E}) = 1$ for every y. The complexity of a learning graph is defined as follows.

Definition 2.4 (Learning graph complexity)**.** Let $\mathcal{G}$ be a learning graph, and let $E \subseteq \mathcal{E}$ a set of flow preserving learning graph edges. The negative complexity of E is $C_0(E) = \sum_{e\in E} \ell(e)w(e)$. The positive complexity of E under the flow p_y is

$$C_{1,y}(E) = \sum_{e\in E} \frac{\ell(e)}{w(e)} \left(\frac{p_y(e)}{p_y(E)}\right)^2, \text{ if } p_y(E) > 0, \quad \text{and 0 otherwise.}$$

The positive complexity of E is $C_1(E) = \max_{y\in Y} C_{1,y}(E)$. The complexity of E is $C(E) = \sqrt{C_0(E)C_1(E)}$, and the learning graph complexity of $\mathcal{G}$ is $C(\mathcal{G}) = C(\mathcal{E})$. The learning graph complexity of a function f, denoted $\mathcal{LG}(f)$, is the minimum learning graph complexity of a learning graph for f.

The usefulness of learning graphs for quantum query complexity is given by the following theorem.

Theorem 2.5 (Belovs)**.** $Q(f) = O(\mathcal{LG}(f))$.

We study functions $f : \{0,1\}^{\binom{n}{2}} \to \{0,1\}$ whose input is an undirected n-vertex graph. We will refer to the vertices and edges of the learning graph as L-vertices and L-edges so as not to cause confusion with the vertices/edges of the input graph. Furthermore, we will only consider learning graphs where every L-vertex is labeled by a k-partite undirected graph on $[n]$, where k is some fixed positive integer. Different L-vertices will have different labels, and we will identify an L-vertex with its label.

3 Analysis of learning graphs

We first review some tools developed by Belovs to analyze the complexity of learning graphs and then develop some new ones useful for the learning graphs we construct. We fix for this section a learning graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w, \ell, \{p_y\})$. By *level d* of $\mathcal{G}$ we refer to the set of vertices at distance d from the root. A *stage* is the set of edges of $\mathcal{G}$ between level i and level j, for some $i < j$. For a subset $V \subseteq \mathcal{V}$ of the L-vertices let $V^+ = \{(v,w) \in \mathcal{E} : v \in V\}$ and similarly let $V^- = \{(u,v) \in \mathcal{E} : v \in V\}$. For a vertex v we will write v^+ instead of $\{v\}^+$, and similarly for v^- instead of $\{v\}^-$. Let E be a stage of $\mathcal{G}$ and let V be some subset of the L-vertices at the beginning of the stage. We set $E^{\rightarrow}_V = \{(v,w) \in E :$ v is u or a descendent of u for some $u \in V\}$. For a vertex v we will write $E^{\rightarrow}_v$ instead of $E^{\rightarrow}_{\{v\}}$.

Given a learning graph $\mathcal{G}$, the easiest way to obtain another learning graph is to modify the weight function of $\mathcal{G}$. We will often use this reweighting scheme to obtain learning graphs with better complexity or complexity that is more convenient to analyze. When $\mathcal{G}$ is understood from the context, and when w' is the new weight function, for any subset $E \subseteq \mathcal{E}$ of the L-edges, we denote the complexity of E with respect to w' by $C^{w'}(E)$.

An illustration of the reweighting method is the following lemma of Belovs which states that we can upper bound the complexity of a learning graph by partitioning it into a constant number of stages and summing the complexities of the stages.

Lemma 3.1 (Belovs)**.** *If $\mathcal{E}$ can be partitioned into a constant number k of stages $E_1, \dots, E_k$, then there exists a weight function w' such that $C^{w'}(\mathcal{G}) = O(C(E_1) + \ldots + C(E_k))$.*

Now we will focus on evaluating the complexity of a stage. Belovs has given a general theorem to simplify the calculation of the complexity of a stage for flows with a high degree of symmetry (Theorem 6 in [4]). Our flows will possess this symmetry but rather than apply Belovs' theorem, we develop one from scratch that takes further advantage of the regular structure of our learning graphs.

Definition 3.2 (Consistent flows)**.** Let E be a stage of $\mathcal{G}$ and let $V_1, \dots, V_s$ be a partition of the L-vertices at the beginning of the stage. We say that $\{p_y\}$ is consistent with $E^{\rightarrow}_{V_1}, \dots, E^{\rightarrow}_{V_s}$ if $p_y(E^{\rightarrow}_{V_i})$ is independent of y for each i.

Lemma 3.3. *Let E be a stage of $\mathcal{G}$ and let $V_1, \dots, V_s$ be a partition of the L-vertices at the beginning of the stage. Set $E_i = E^{\rightarrow}_{V_i}$, and suppose that $\{p_y\}$ is consistent with $E_1, \dots, E_s$. Then there is a new weight function w' for $\mathcal{G}$ such that*

$$C^{w'}(E) \leq \max_i C(E_i).$$

Proof. Since by hypothesis $p_y(E_i)$ is independent from y, denote it by α_i. We assume that $\alpha_i > 0$ for each i; if $\alpha_i = 0$ then $p_y((u,v)) = 0$ for every y and $(u,v) \in E_i$, and these edges can be deleted from the graph without affecting anything. For $e \in E_i$, we define the new weight $w'(e) = \alpha_i C_1(E_i) w(e)$. Let us analyze the complexity of E under this weighting.

To evaluate the positive complexity observe that $p_y(E) = 1$ for every y, since E is a stage, and thus $\sum_i \alpha_i = 1$. Therefore

$$C_1^{w'}(E) = \max_y \sum_i \sum_{e\in E_i} \frac{\ell(e)p_y(e)^2}{w'(e)} \le \sum_i \frac{\alpha_i}{C_1(E_i)} \max_y \sum_{e\in E_i} \frac{\ell(e)p_y(e)^2}{w(e)\alpha_i^2} = \sum_i \alpha_i = 1.$$

The negative complexity can be bounded by

$$C_0(E) = \sum_i \sum_{e\in E_i} \ell(e)w'(e) = \sum_i \alpha_i C_1(E_i) \sum_{e\in E_i} \ell(e)w(e) = \sum_i \alpha_i C_1(E_i) C_0(E_i) \le \max_i C(E_i)^2.$$

□

At a high level, we will analyze the complexity of a stage E as follows. First, we partition the set of vertices V into equivalence classes $[u] = \{\sigma(u) : \sigma \in S_n\}$ for some appropriate action of S_n that we will define later, and use symmetry to argue that the flow is consistent with $\{E^{\rightarrow}_{[u]}\}$. Thus by Lemma 3.3, it is enough to focus on the maximum complexity of $E^{\rightarrow}_{[u]}$. Within $E^{\rightarrow}_{[u]}$, our flows will be of a particularly simple form. In particular, incoming flow will be uniformly distributed over a subset of $[u]$ of fixed size independent of y. The next two lemmas evaluate the complexity of $E^{\rightarrow}_{[u]}$ in this situation.

Lemma 3.4. *Let E be a stage of $\mathcal{G}$ and let V be some subset of the L-vertices at the beginning of the stage. For each y let $W_y \subseteq V$ be the set of vertices in V which receive positive flow under p_y. Suppose that for every y the following is true:*

1. $E^{\rightarrow}_u \cap E^{\rightarrow}_v = \emptyset$ *for* $u \ne v \in V$,
2. $|W_y|$ *is independent of* y,
3. *for all* $v \in W_y$ *we have* $p_y(E^{\rightarrow}_v) = p_y(E^{\rightarrow}_V)/|W_y|$.

Then

$$C(E^{\rightarrow}_V) \le \sqrt{\max_{v\in V} C_0(E^{\rightarrow}_v) \max_{v\in V} C_1(E^{\rightarrow}_v) \frac{|V|}{|W_y|}}\ .$$

Proof. The negative complexity can easily be upper bounded by

$$C_0(E^{\rightarrow}_V) = \sum_{v\in V} C_0(E^{\rightarrow}_v) \le |V| \max_{v\in V} C_0(E^{\rightarrow}_v).$$

For the positive complexity we have

$$\begin{aligned} C_1(E_V^{\rightarrow}) &= \max_y \sum_{v \in W_y} \sum_{e \in E_v^{\rightarrow}} \frac{\ell(e) p_y(e)^2}{w(e) p_y(E_V^{\rightarrow})^2} \\ &\leq \frac{1}{|W_y|^2} \sum_{v \in W_y} \max_y \sum_{e \in E_v^{\rightarrow}} \frac{\ell(e) p_y(e)^2 \omega^2}{w(e) p_y(E_V^{\rightarrow})^2} \\ &\leq \frac{\max_{v \in V} C_1(E_v^{\rightarrow})}{|W_y|}. \end{aligned}$$

□

Observe that when E is a stage between two consecutive levels, that is between level i and $i+1$ for some i, and V is a subset of the vertices at the beginning of the stage, then $E_V^{\rightarrow} = V^+$. We will use Lemma 3.3 in conjunction with Lemma 3.4 first in this context.

Lemma 3.5. *Let E be a stage of $\mathcal{G}$ between two consecutive levels. Let V be the set of L-vertices at the beginning of the stage and suppose that each $v \in V$ has outdegree d and all L-edges e of the stage satisfy $w(e) = 1$ and $\ell(e) \leq \ell$. Let $V_1, \ldots, V_s$ be a partition of V, and for all y and i, let $W_{y,i} \subseteq V_i$ be the set of vertices in V_i which receive positive flow under p_y. Suppose that*

1. *the flows $\{p_y\}$ are consistent with $\{V_i{}^+\}$,*
2. *$|W_{y,i}|$ is independent from y for every i, and for all $v \in W_{y,i}$ we have $p_y(v^+) = p_y(V_i{}^+)/|W_{y,i}|$,*
3. *there is a g such that for each vertex $v \in W_{y,i}$ the flow is directed uniformly to g of the d many neighbors.*

Then there is a new weight function w' such that

$$C^{w'}(E) \leq \max_i \ell \sqrt{\frac{d}{g} \frac{|V_i|}{|W_{y,i}|}} \quad . \tag{3.1}$$

Proof. By hypothesis (1) we are in the realm of Lemma 3.3 and therefore $C^{w'}(E) \leq \max_i C(V_i{}^+)$. To evaluate $C(V_i{}^+)$, we can apply Lemma 3.4 according to hypothesis (2). The statement of the lemma then follows, since for every $v \in V$ we have $C_0(v^+) = \ell d$, and $C_1(v^+) = \ell/g$ by hypothesis (3). □

This lemma will be the main tool we use to analyze the complexity of stages. Note that the complexity in Equation (3.1) can be decomposed into three parts: the length ℓ, the degree ratio d/g, and the *maximum vertex ratio* $\max_i |V_i|/|W_{y,i}|$. This terminology will be very helpful to evaluate the complexity of stages.

We will use symmetry to decompose our flows as a convex combinations of uniform flows over disjoint sets of edges. Recall that each L-vertex u is labeled by a k-partite graph on $[n]$, say with color classes $A_1, \ldots, A_k$, and that we identify an L-vertex with its label. For $\sigma \in S_n$ we define the action of σ on u as $\sigma(u) = v$, where v is a k-partite graph with color classes $\sigma(A_1), \ldots, \sigma(A_k)$ and edges $\{\sigma(i), \sigma(j)\}$ for every edge $\{i, j\}$ in u.

Define an equivalence class $[u]$ of L-vertices by $[u] = \{\sigma(u) : \sigma \in S_n\}$. We say that S_n *acts transitively* on flows $\{p_y\}$ if for every y, y' there is a $\tau \in S_n$ such that $p_y((u,v)) = p_{y'}((\tau(u), \tau(v))$ for all L-edges (u,v).

As shown in the next lemma, if S_n acts transitively on a set of flows $\{p_y\}$ then they are consistent with $[v]^+$, where v is a vertex at the beginning of a stage between consecutive levels. This will set us up to satisfy hypothesis (1) of Lemma 3.5.

Lemma 3.6. *Consider a learning graph $\mathcal{G}$ and a set of flows $\{p_y\}$ such that S_n acts transitively on $\{p_y\}$. Let V be the set of L-vertices of $\mathcal{G}$ at some given level. Then $\{p_y\}$ is consistent with $\{[u]^+ : u \in V\}$, and, similarly, $\{p_y\}$ is consistent with $\{[u]^- : u \in V\}$.*

Proof. Let $p_y, p_{y'}$ be two flows and $\tau \in S_n$ such that $p_y((u,v)) = p_{y'}((\tau(u), \tau(v))$ for all L-edges (u,v). Then

$$\begin{aligned} p_y([u]^+) &= \sum_{v \in [u]} \sum_{w:(v,w) \in \mathcal{E}} p_y((v,w)) \\ &= \sum_{v \in [u]} \sum_{w:(v,w) \in \mathcal{E}} p_{y'}((\tau(v), \tau(w))) \\ &= \sum_{\tau^{-1}(v) \in [u]} \sum_{\tau^{-1}(w):(\tau^{-1}(v), \tau^{-1}(w)) \in \mathcal{E}} p_{y'}((v,w)) \\ &= \sum_{v \in [u]} \sum_{w:(v,w) \in \mathcal{E}} p_{y'}((v,w)) = p_{y'}([u]^+). \end{aligned}$$

The statement $p_y([u]^-) = p_{y'}([u]^-)$ follows exactly in the same way. □

The next lemma gives a sufficient condition for hypothesis (2) of Lemma 3.5 to be satisfied. The partition of vertices in Lemma 3.5 will be taken according to the equivalence classes $[u]$. Note that unlike the previous lemmas in this section that only consider a stage of a learning graph, this lemma speaks about the learning graph in its entirety.

Lemma 3.7. *Consider a learning graph and a set of flows $\{p_y\}$ such that S_n acts transitively on $\{p_y\}$. Suppose that for every L-vertex u and flow p_y such that $p_y(u^-) > 0$,*

1. *the flow from u is uniformly directed to $g^+([u])$ many neighbors,*

2. *for every L-vertex w, the number of incoming edges with from $[w]$ to u is $g^-([w],[u])$.*

Then for every L-vertex u the flow entering $[u]$ is uniformly distributed over $W_{y,[u]} \subseteq [u]$ where $|W_{y,[u]}|$ is independent of y.

Proof. We first use hypotheses (1),(2) of Lemma 3.7 to show that for every flow p_y and for every L-vertex u, the incoming flow $p_y(u^-)$ to u is either 0 or $\alpha_y([u]) > 0$, that is it depends only on the equivalence class of u. We then use transitivity and hypothesis (2) of Lemma 3.7 to reach the conclusion of the lemma.

Let V_t be the set of vertices at level t and fix a flow p_y. The proof is then by induction on the level t on a stronger statement for every $\sigma, \sigma' \in S_n$ and L-vertices $u \in V_t$ and $v, v' \in V_{t+1}$:

$$[p_y((u,v)) > 0 \text{ and } p_y((\sigma(u), v')) > 0] \implies p_y((u,v)) = p_y((\sigma(u), v')), \tag{3.2}$$

$$[p_y(\sigma(u)^-) > 0 \text{ and } p_y(\sigma'(u)^-) > 0] \implies p_y(\sigma(u)^-) = p_y(\sigma'(u)^-). \tag{3.3}$$

At level $t = 0$, the statement is correct since the root is unique, has incoming flow 1, and outgoing edges with flow 0 or $1/g^+(\text{root})$.

Assume the statements hold up to and including level t. Hypothesis 1 implies that when $p_y((u,v)) > 0$ for $u \in V_t$, it satisfies $p_y((u,v)) = p_y(u^-)/g^+([u])$, and similarly $p_y((\tau(u),v')) = p_y(\tau(u)^-)/g^+([u])$. Therefore, Equation 3.3 at level t implies Equation 3.2 at level $t+1$.

We now turn to Equation 3.3 at level $t+1$. Fix $v \in V_{t+1}$ and $\sigma, \sigma' \in S$ such that $\sigma(v)$ and $\sigma'(v)$ have positive incoming flows. Then

$$p_y(\sigma(v)^-) = \sum_{u \in V_t} p_y((u,\sigma(v))) \quad \text{and} \quad p_y(\sigma'(v)^-) = \sum_{u \in V_t} p_y((u,\sigma'(v))).$$

We will show that $p_y(\sigma(v)^-) = p_y(\sigma'(v)^-)$ by proving the following equality for every u

$$\sum_{\tau \in S_n} p_y((\tau(u),\sigma(v))) = \sum_{\tau \in S_n} p_y((\tau(u),\sigma'(v))).$$

By Equation 3.2 at level t, all nonzero terms in the respective sum are identical. By Hypothesis 2, the number of nonzero terms is $g^-([u],[v])$ in both sums. Therefore the two sums are identical.

We now have concluded that the incoming flow to an L-vertex u is either 0 or $\alpha_y([u]) > 0$. This implies that the flow entering u is uniformly distributed over some set $W_{y,[u]} \subseteq [u]$. We now show that the size of this set is independent of y.

If the flow is transitive then $\alpha_y([u])$ is independent of y and furthermore by the second statement of Lemma 3.6 applied to the level of u, $\sum_{v \in [u]} p_y(v^-) = \sum_{v \in [u]} p_{y'}(v^-)$. Thus the number of terms in each sum must be the same and $|W_{y,[u]}|$ is independent of y. □

4 Algorithms

We first discuss some basic assumptions about the subgraph H. Say that H has k vertices and minimum degree d. First, we assume that $d \geq 1$, that is H has no disconnected vertices. Recall that we are testing if G contains H as a subgraph, not as an induced subgraph. Thus if H' is H with disconnected vertices removed, then G will contain H if and only if G contains H' and $n \geq k$. Furthermore, the algorithms we give in this section behave monotonically with k, and so will have smaller complexity on the graph H'. Additionally, we assume that $k \geq 3$ as if $k = 2$ and $d = 1$ then H is simply an edge and in the case the complexity is known to be $\Theta(n)$ as it is equivalent to search on $\Theta(n^2)$ items.

Thus let H be a graph on vertex set $\{1,2,\ldots,k\}$, with $k \geq 3$ vertices. We present two algorithms in this section for determining if a graph G contains H. Following Belovs, we say that a stage *loads* an edge $\{a,b\}$ if for all L-edges (u,v) with flow in the stage, we have $\{a,b\} \in S(v) \setminus S(u)$. Both algorithms will use a subroutine, given in Section 4.1, to load an induced subgraph of H. For some integer $1 \leq u \leq k$, let $H_{[1,u]}$ be the subgraph of H induced by vertices $1,2,\ldots,u$. The first algorithm, given in Section 4.2, will take $u = k$ and load H directly; the second algorithm, given in Section 4.3, will first load $H_{[1,k-1]}$, and then search for the missing vertex that completes H.

4.1 Loading a subgraph of H

Fix $1 \leq u \leq k$ and and let $e_1, \dots, e_m$ be the edges of $H_{[1,u]}$, enumerated in some fixed order. We assume that $m \geq 1$. For any positive input graph G, that is a graph G which contains a copy of H, we fix k vertices $a_1, a_2, \dots, a_k$ such that $\{a_i, a_j\}$ is an edge of G whenever $\{i, j\}$ is an edge of H.

We define a bit of terminology that will be useful. For two sets $Y_1, Y_2 \subseteq [n]$, we say that a bipartite graph between Y_1 and Y_2 is of type $(\{(n_1, d_1), \dots, (n_j, d_j)\}, \{(m_1, g_1), \dots, (m_\ell, g_\ell)\})$ if Y_1 has n_i vertices of degree d_i for $i = 1, \dots, j$, and Y_2 has m_i vertices of degree g_i for $i = 1, \dots, \ell$, and this is a complete listing of vertices in the graph, i.e. $|Y_1| = \sum_{i=1}^{j} n_i$ and $|Y_2| = \sum_{i=1}^{\ell} m_i$.

Vertices of our learning graph will be labeled by a u-partite graph Q on disjoint sets $X_1, \dots, X_u \subseteq [n]$. The global structure of Q will mimic the edge pattern of $H_{[1,u]}$. Namely, for each edge $e_t = \{i, j\}$ of $H_{[1,u]}$, there will be a bipartite graph Q_t between X_i and X_j with a specified degree sequence. There are no edges between X_i and X_j if $\{i, j\}$ is not an edge of $H_{[1,u]}$. The mapping $S : V \to 2^{\binom{n}{2}}$ from learning graph vertices to query indices returns the union of the edges of Q_t for $t = 1, \dots, m$.

We now describe the stages of our first learning graph. Let V_t denote the L-vertices at the beginning of stage t (and so the end of stage $t-1$ for $t > 0$). The L-edges between V_t and V_{t+1} are defined in the obvious way—there is an L-edge between $v_t \in V_t$ and $v_{t+1} \in V_{t+1}$ if the graph labeling v_t is a subgraph of the graph labeling v_{t+1}. We initially set the weight of all L-edges to be one, though some edges will be reweighted in the complexity analysis using Lemma 3.5. The root of the learning graph is labeled by the empty graph.

The algorithm depends on two parameters r, s which will be optimized later. The parameter $r \in [n]$ will control the number of vertices, and $s \in [0, 1]$ the edge density, of graphs labeling the L-vertices.

Learning graph $\mathcal{G}_1$:

Stage 0: Setup (Figure 1). V_1 consists of all L-vertices labeled by a u-partite graph Q with color classes $A_1, \dots, A_u \subseteq [n]$, each of size $r-1$. The edges will be the union of the edges in bipartite graphs $Q_1, \dots, Q_m$, where if $e_\ell = \{i, j\}$ is an edge of $H_{[1,u]}$, then Q_ℓ is a bipartite graph of type $(\{(r-1-rs, rs), (rs, rs-1)\}, \{(r-1-rs, rs), (rs, rs-1)\})$ between A_i and A_j. The number of edges added in this stage is $O(sr^2)$. Flow is uniform from the root of the learning graph, whose label is the empty graph, to all L-vertices such that $a_1, \dots, a_k \notin A_i$ for $i = 1, \dots, u$.

Stage t for $t = 1, \dots, u$: Load a_t (Figures 2 and 3). V_{t+1} consists of all L-vertices labeled by a u-partite graph Q with color classes $B_1, \dots, B_t, A_{t+1}, \dots A_u$, where $|B_i| = r$, and $|A_i| = r-1$. The edges of Q are the union of edges of bipartite graphs $Q_1, \dots, Q_m$, where if $e_\ell = \{i, j\}$ then the type of Q_ℓ is given by the following cases:

- If $t < i < j$, then Q_ℓ is of type $(\{(r-1-rs, rs), (rs, rs-1)\}, \{(r-1-rs, rs), (rs, rs-1)\})$ between A_i and A_j.
- If $i \leq t < j$, then Q_ℓ is of type $(\{(r-rs, rs), (rs, rs-1)\}, \{(r-1, rs)\})$ between B_i and A_j.
- If $i < j \leq t$, then Q_ℓ is of type $(\{(r, rs)\}, \{(r, rs)\})$ between B_i and B_j.

The number of edges added at stage t is $O(rs)$. The flow is directed uniformly on those L-edges where the element added to A_t is a_t and none of the edges $\{a_i, a_j\}$ are present.

Stage $u+1$: Hiding (Figure 4). Now we are ready to start loading edges $\{a_i, a_j\}$. If we simply loaded the edge $\{a_i, a_j\}$ now, however, it would be uniquely identified by the degrees of a_i, a_j since only these vertices would have degree $rs+1$. This means that for example at the last stage of the learning graph the vertex ratio would be $\Omega(n^{k-1})$, no matter what r is. Thus in this stage we first do a "hiding" step, adding edges so that half of the vertices in every set have degree $rs+1$.

Formally, V_{u+2} consists of all L-vertices labeled by a u-partite graph Q with color classes $B_1, \ldots, B_u$, where $|B_i| = r$. The edges of Q are the union of edges of bipartite graphs $Q_1, \ldots, Q_m$, where if $e_\ell = \{i, j\}$ then Q_ℓ is of type $(\{(r/2, rs), (r/2, rs+1)\}, \{(r/2, rs), (r/2, rs+1)\})$ between B_i and B_j. The number of edges added in this stage is $O(r)$. The flow is directed uniformly to those L-vertices where for every $e_\ell = \{i, j\}$, both a_i and a_j have degree rs in Q_ℓ.

Stage $u+t+1$ for $t = 1, \ldots, m$: Load $\{a_i, a_j\}$ if $e_t = \{i, j\}$ (Figure 5). Take an L-vertex at the beginning of stage $u+t+1$ whose edges are the union of bipartite graphs $Q_1, \ldots, Q_m$. In stage $u+t+1$ only Q_t will be modified, by adding single edge $\{b_i, b_j\}$ where $b_i \in B_i$ and $b_j \in B_j$ have degree rs in Q_t. The flow is directed uniformly along those L-edges where $b_i = a_i$ and $b_j = a_j$.

Thus at the end of stage $u+m+1$, the L-vertices are labeled by the edges in the union of bipartite graphs $Q_1, \ldots, Q_m$ each of type $(\{(r/2-1, rs), (r/2+1, rs+1)\}, \{(r/2-1, rs), (r/2+1, rs+1)\})$. The incoming flow is uniform over those L-vertices where $a_i \in B_i$ for $i = 1, \ldots, u$, and if $e_\ell = \{i, j\}$ then the edge $\{a_i, a_j\}$ is present in Q_ℓ for $\ell = 1, \ldots, m$, and both a_i, a_j have degree $rs+1$ in Q_ℓ.

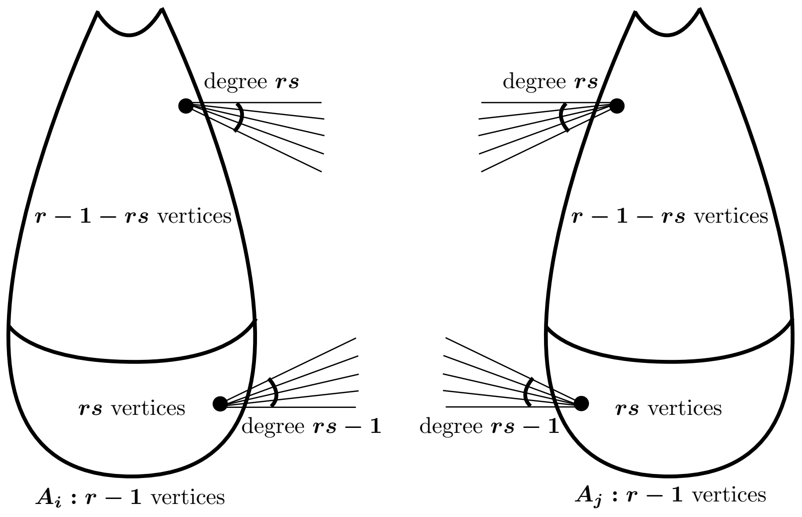

Figure 1: **Stage 0:** Edges added to G_ℓ when $e_\ell = \{i, j\}$ is an edge of K. The flow is uniform to instances with $a_1, \ldots, a_k \notin A_i$ for $i = 1, \ldots, k-1$.

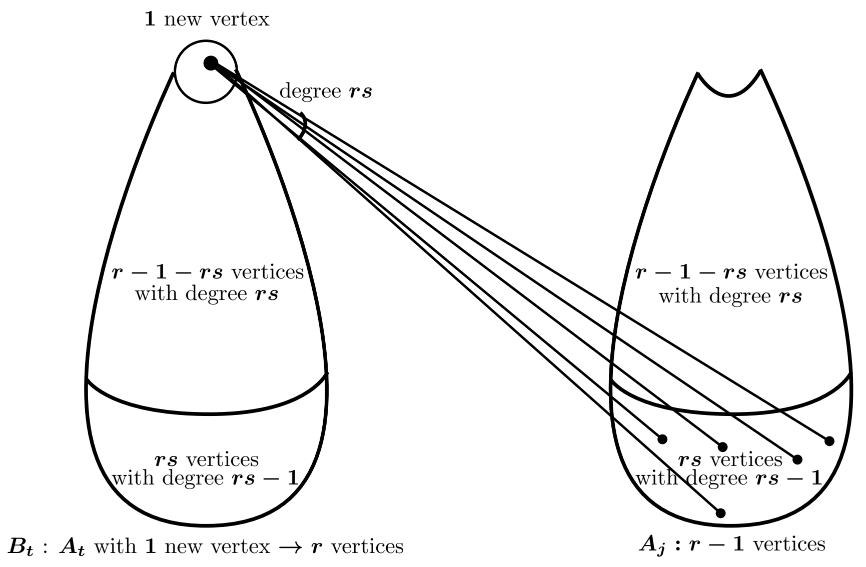
$\mathbf{1}$ new vertex
degree rs
$r-1-rs$ vertices with degree rs
rs vertices with degree $rs-1$
B_t : A_t with $\mathbf{1}$ new vertex $\rightarrow r$ vertices
$r-1-rs$ vertices with degree rs
rs vertices with degree $rs-1$
A_j : $r-1$ vertices

Figure 2: **Stage** t **for** $t = 1,\dots,u$**:** rs added edges in some G_ℓ at stage t, when $e_\ell = \{t, j\}$ with $t < j$. See Figure 3 for the case $e_\ell = \{i,t\}$ with $i < t$. (No edge is added to G_ℓ at stage t when $e_\ell = \{i, j\}$ with $t \neq i$ and $t \neq j$.) The added edges are between the new vertex of A_t and the rs vertices in A_j, respectively B_i, of degree $(rs-1)$. The flow is directed to instances where the new vertex of A_t is a_t.

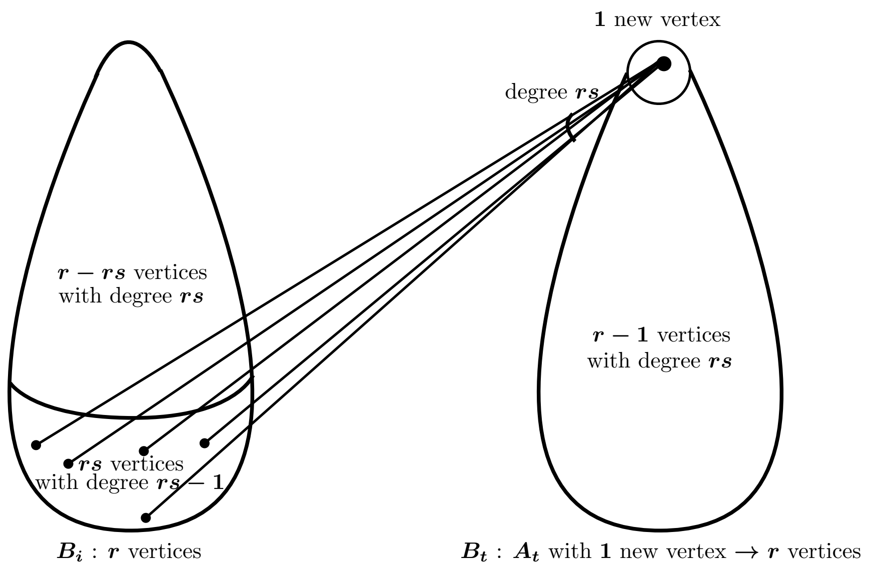
$\mathbf{1}$ new vertex
degree rs
$r-rs$ vertices with degree rs
rs vertices with degree $rs-1$
B_i : r vertices
$r-1$ vertices with degree rs
B_t : A_t with $\mathbf{1}$ new vertex $\rightarrow r$ vertices

Figure 3: **Stage** t **for** $t = 1,\dots,u$**:** rs added edges in some G_ℓ at stage t, when $e_\ell = \{i,t\}$ with $i < t$. See Figure 2 for the case $e_\ell = \{t, j\}$ with $t < j$. (No edge is added to G_ℓ at stage t when $e_\ell = \{i, j\}$ with $t \neq i$ and $t \neq j$.) The added edges are between the new vertex of A_t and the rs vertices in A_j, respectively B_i, of degree $(rs-1)$. The flow is directed to instances where the new vertex of A_t is a_t.

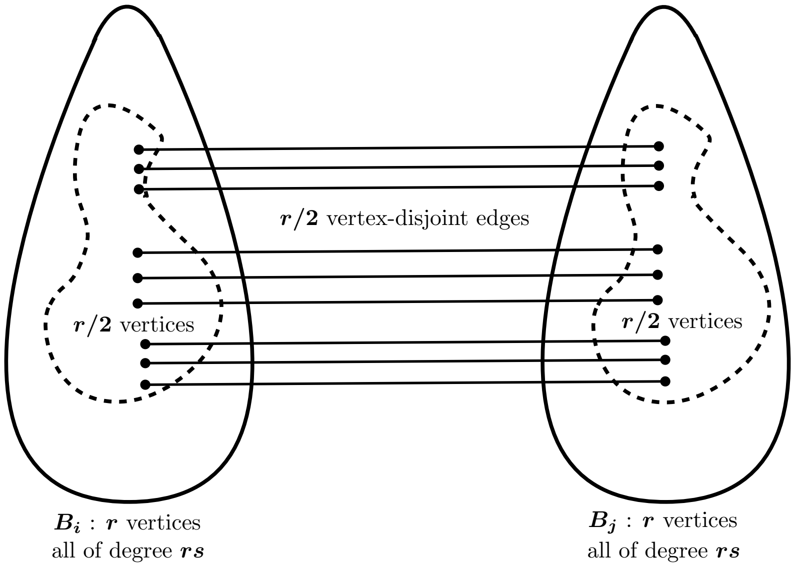

Figure 4: **Stage** $u+1$**:** We add $r/2$ vertex-disjoint edges to G_ℓ when $e_\ell = \{i, j\}$ is an edge of K. The flow is directed to instances where the degrees of a_i and a_j remain rs in G_ℓ.

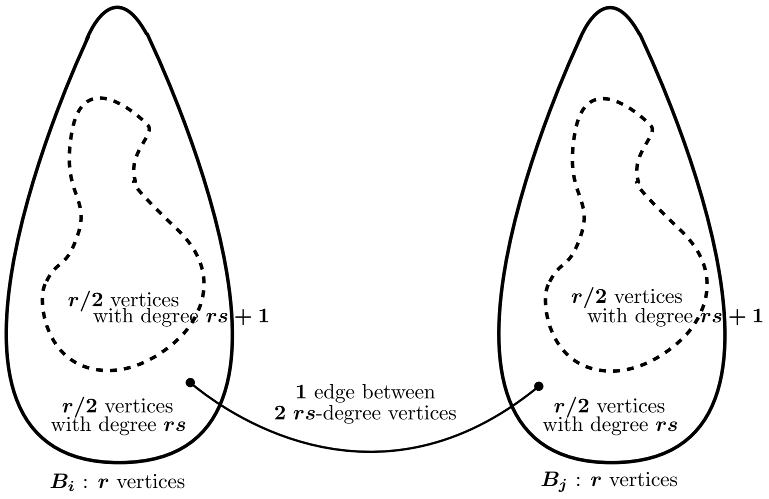

Figure 5: **Stage** $u+1+t$ **for** $t = 1, \ldots, m$**:** Let $e_t = \{i, j\}$. Then a single edge is added to Q_t between two vertices $b_i \in B_i$ and $b_j \in B_j$ of degree rs in Q_t. The flow is directed to instances where $b_i = a_i$ and $b_j = a_j$.

Complexity analysis of the stages Note that for an input graph y containing a copy of $H_{[1,u]}$ the definition of flow depends only on the vertices $a_1, \ldots, a_u$ that span H. As for any two graphs y, y' containing H there is a permutation τ mapping a copy of H in y to a copy of H in y' we see that S_n acts transitively on flows.

Furthermore, by construction of our learning graph, from a vertex v with $p_y(v^-) > 0$, flow is directed uniformly to g out of d many neighbors, where g, d depend only on the stage, not y or v. Additionally, by symmetry of the flow, hypothesis (2) of Lemma 3.7 is also satisfied. We will invoke Lemma 3.5 to evaluate the cost of each stage. Hypothesis (1) is satisfied by Lemma 3.6, hypothesis (2) by Lemma 3.7, and hypothesis (3) by construction of the learning graph.

- Stage 0: The set of L-vertices at the beginning of this stage is simply the root thus the vertex ratio (and maximum vertex ratio) is one. The degree ratio can be upper bounded by $((n-k)/(n-kr-k))^k = O(1)$, as we will choose $r = o(n)$ and k is constant. The length of this stage is $O(sr^2)$ and so its complexity is $O(sr^2)$.

- Stage t for $t = 1, \dots u$: An L-vertex in V_t will be used by the flow if and only if $a_i \in B_i$ for $i = 1, \dots, t-1$ and $a_i \notin B_1, \dots, B_t, A_{t+1}, \dots, A_u$ for $i = t, \dots, k$. For any vertex $v \in V_t$ the probability over $\sigma \in S_n$ that $\sigma(v)$ satisfies the second event is constant thus the vertex ratio is dominated by the first event which has probability $O((r/n)^{t-1})$. Thus the maximum vertex ratio is $O((n/r)^{t-1})$. The degree ratio is n. Since $O(sr)$ edges are added, the complexity is $O(sr\sqrt{n}(n/r)^{(t-1)/2})$.

- Stage $u+1$: As above, an L-vertex in V_{k+1} will be used by the flow if and only if $a_i \in B_i$ for $i = 1, \dots, u$. For any vertex $v \in V_{k+1}$ the probability over σ that this is satisfied by $\sigma(v)$ is $O((r/n)^u)$ therefore the maximum vertex ratio is $O((n/r)^u)$. For each $e_\ell = \{i, j\}$, half of the vertices in B_i and half of the vertices in B_j will have degree rs in Q_ℓ. Therefore, the degree ratio is $4^m = O(1)$. Since $O(r)$ edges are added, the complexity of this stage is therefore $O(r(n/r)^{u/2})$.

- Stage $u+t+1$ for $t = 1, \dots m$: In every stage, the degree ratio is $O(r^2)$. An L-vertex is in the flow at the beginning of stage $u+t+1$ if the following two conditions are satisfied:

$$a_i \in B_i \text{ for } i = 1, \dots u, \tag{4.1}$$

$$\text{if } e_\ell = \{i, j\} \text{ then } \{a_i, a_j\} \in Q_\ell \text{ with } a_i, a_j \text{ of degree } rs+1 \text{ in } Q_\ell, \text{ for } \ell = 1, \dots, t-1. \tag{4.2}$$

  The probability over σ that $\sigma(v)$ satisfies Equation (4.1) is $\Omega((r/n)^u)$. Among vertices in $[v]$ satisfying this condition, a further $\Omega(s^{t-1})$ fraction will satisfy Equation (4.2). This follows from Lemma 4.1 below, together with the independence of the bipartite graphs $Q_1, \dots, Q_m$. Thus the maximum vertex ratio is $O((n/r)^u s^{-(t-1)})$. As only one edge is added at this stage, we obtain a cost of $O(r(n/r)^{u/2} s^{-(t-1)/2})$.

Lemma 4.1. *Let Y_1, Y_2 be disjoint r-element subsets of $[n]$, and let $(y_1, y_2) \in Y_1 \times Y_2$. Let K be a bipartite graph between Y_1 and Y_2 of type $(\{(r/2-1, rs), (r/2+1, rs+1)\}, \{(r/2-1, rs), (r/2+1, rs+1)\})$. The probability over $\sigma \in S_n$ that the edge $\{y_1, y_2\}$ is in $\sigma(K)$ and both y_1 and y_2 are of degree $rs+1$, is at least $s/4$.*

Proof. The degree condition is satisfied with probability at least $1/4$. Given that the degree condition is satisfied, it is enough to show that for a bipartite graph K' of type $(\{(r, rs)\}, \{(r, rs)\})$ the probability over $\sigma \in S_n$ that $\sigma(K')$ contains the fixed edge (y_1, y_2) is at least s, since K is such a graph plus some additional edges.

Because of symmetry, this probability doesn't depend on the choice of the edge, let's denote it by p. Let $K_1,\dots,K_c$ be an enumeration of all bipartite graphs isomorphic to K'. We will count in two different ways the cardinality χ of the set $\{(e,h) : e \in K_h\}$. Every K_h contains sr^2 edges, therefore $\chi = csr^2$. On the other hand, every edge appears in pc graphs, therefore $\chi = r^2pc$, and thus $p = s$. □

4.2 Loading H

When $u = k$, the constructed learning graph determines if H is a subgraph of the input graph, since a copy of H is loaded on positive instances. Choosing the parameters s, r to optimize the total cost gives the following theorem.

Theorem 4.2. *Let H be a graph on $k \geq 3$ vertices and $m \geq 1$ edges. Then there is a quantum query algorithm for determining if H is a subgraph of an n-vertex graph making $O(n^{2-2/(k+1)-k/((k+1)(m+1))})$ many queries.*

Proof. By Theorem 2.5, it suffices to show that the learning graph $\mathcal{G}_1$ has the claimed complexity. We will use Lemma 3.1 and upper bound the learning graph complexity by the sum of the costs of the stages. As usual, we will ignore factors of k.

The complexity of stage 0 is:

$$S' = O\left(sr^2\right) \quad .$$

The complexity of each stage $1,\dots,k$, and also their sum, is dominated by the complexity of stage k:

$$U' = O\left(sr\sqrt{n}(n/r)^{(k-1)/2}\right) \quad .$$

The complexity of stage $k+1$ is:

$$U'' = O\left(r(n/r)^{k/2}\right) \quad .$$

Again, the complexity of each stage $k+2,\dots,k+m+1$, and also their sum, is dominated by the complexity of stage $k+m+1$:

$$U''' = O\left(r(n/r)^{k/2}s^{-(m-1)/2}\right) \quad .$$

Observe that $U'' = O(U''')$.

Therefore the overall cost can be bounded by $S' + U' + U'''$. Choosing $r = n^{1-1/(k+1)}$ makes $S' = U'$ for any value of s, as their dependence on s is the same. When $s = 1$ we have $U''' < S' = U'$ thus we can choose $s < 1$ to balance all three terms. Letting $s = n^{-t}$ we have $S' = U' = O(n^{2-2/(k+1)-t})$ and $U''' = O(n^{1+(k-2)/(2(k+1))+t(m-1)/2})$. Making these equal gives $t = k/((k+1)(m+1))$, and gives overall cost $O(n^{2-2/(k+1)-t})$. □

4.3 Loading the full graph but one vertex

Recall that H is a graph on vertex set $\{1,2,\dots,k\}$, with $k \geq 3$ vertices, $m \geq 1$ edges and minimum degree $d \geq 1$. By renaming the vertices, if necessary, we assume that vertex k has degree d.

Our second algorithm employs the learning graph $\mathcal{G}_1$ of Section 4.1 with $u = k-1$ to first load $H_{[1,k-1]}$. This is then combined with search to find the missing vertex and a collision subroutine to verify it links with $H_{[1,k-1]}$ to form H.

Again, let $H_{[1,k-1]}$ be the subgraph of H induced by vertices $1,2,\ldots,k-1$, and let $e_1,\ldots,e_{m'}$ be the edges of $H_{[1,k-1]}$, enumerated in some fixed order. Thus note that $m = m'+d$. For any positive input graph y, we fix k vertices $a_1,a_2,\ldots,a_k$ such that $\{a_i,a_j\}$ is an edge of y whenever $\{i,j\}$ is an edge of H. For notational convenience we assume that a_k is of degree d and connected to $a_1,\ldots,a_d$.

Learning graph $\mathcal{G}_2$:

Stages $0,1,\ldots,k+m'$**:** Learning graph $\mathcal{G}_1$ of Section 4.1.

Stage $k+m'+1$**:** We use search plus a d-wise collision subroutine to find a vertex v and d edges which link v to $H_{[1,k-1]}$ to form H. The learning graph for this subroutine is given in Section 4.4.

Complexity analysis of the stages All stages but the last one have been analyzed in Section 4.1, therefore only the last stage remains to study.

- Stage $k+m'+1$: Let $V_{k+m'+1}$ be the set of L-vertices at the beginning of stage $k+m'+1$. We will evaluate the complexity of this stage in a similar fashion as we have done previously. As S_n acts transitively on the flows, by Lemma 3.6 we can invoke Lemma 3.3 and it suffices to consider the maximum of $C(E^{\rightarrow}_{[u]})$ over equivalence classes $[u]$. Furthermore, as we have argued in Section 4.1, the learning graph also satisfies the conditions of Lemma 3.7, thus we can apply Lemma 3.4 to evaluate $C(E^{\rightarrow}_{[u]})$. The maximum vertex ratio over $[u]$ is $O(s^{-m'}(n/r)^{k-1})$. As shown in Section 4.4, the complexity of the subroutine learning graph attached to each $v \in V_{k+m'+1}$ is at most $O(\sqrt{n}r^{d/(d+1)})$. Thus by Lemma 3.4, the complexity of this stage is
$$O\left(s^{-m'/2}\left(\frac{n}{r}\right)^{(k-1)/2}\sqrt{n}r^{d/(d+1)}\right) \quad .$$

Choosing the parameters s,r to optimize the total cost gives the following theorem.

Theorem 4.3. *Let H be a graph on $k \geq 3$ vertices with minimal degree $d \geq 1$ and m edges. Then there is a quantum query algorithm for determining if H is a subgraph of an n-vertex graph making $O(n^{2-2/k-(2k-d-3)/(k(d+1)(m-d+2))})$ many queries.*

Proof. By Theorem 2.5, it suffices to show that the learning graph $\mathcal{G}_2$ has the claimed complexity. We will use Lemma 3.1 and upper bound the learning graph complexity by the sum of the costs of the stages. As usual, we will ignore factors of k.

The complexity of stage 0 is:
$$S' = O\left(sr^2\right) \quad .$$
The complexity of each stage $1,\ldots,k-1$, and also their sum, is dominated by the complexity of stage $k-1$:
$$U' = O\left(sr\sqrt{n}(n/r)^{(k-2)/2}\right) \quad .$$

The complexity of stage k is:

$$U'' = O\left(r(n/r)^{(k-1)/2}\right) .$$

Again, the complexity of each stage $k+1,\dots,k+m'$, and also their sum, is dominated by the complexity of stage $k+m'$:

$$U''' = O\left(r(n/r)^{(k-1)/2}s^{-(m'-1)/2}\right) .$$

Observe that $U'' = O(U''')$. Finally, denote the cost of stage $k+m'+1$ by

$$C' = O\left(s^{-m'/2}\left(\frac{n}{r}\right)^{(k-1)/2}\sqrt{n}r^{d/(d+1)}\right) .$$

Observe that $U''' = O(C')$, provided that $r^{1/(d+1)}s^{1/2} = O(n^{1/2})$. The later is always satisfied since $s \le 1$, $r \le n$ and $d \ge 1$. Therefore the overall cost can then be bounded by $S' + U' + C'$. Choosing $r = n^{1-1/k}$ makes $S' = U'$ for any value of s, as their s dependence is the same. When $s = 1$ we have $C' < S' = U'$ thus we can choose $s < 1$ to balance all three terms. Letting $s = n^{-t}$ we have $S' = U' = O(n^{2-2/k-t})$ and $C' = O(n^{2-2/k+1/(2k)-(k-1)/(k(d+1))+tm'/2})$. Making these equal gives $t = (2k-d-3)/(k(d+1)(m'+2))$. Since $k \ge 3$ we have $t > 0$ and thus $s < 1$. The overall cost of the algorithm is $O(n^{2-2/k-t})$. Noting that $m = m' + d$ gives the statement of the theorem. $\square$

Our main result is an immediate consequence of Theorem 4.2 and Theorem 4.3.

Theorem 4.4. *Let H be a graph on $k \ge 3$ vertices with minimal degree $d \ge 1$ and m edges. Then there is a quantum query algorithm for determining if H is a subgraph of an n-vertex graph making $O(n^{2-2/k-t})$ many queries, where*

$$t = \max\left\{\frac{k^2-2(m+1)}{k(k+1)(m+1)}, \frac{2k-d-3}{k(d+1)(m-d+2)}\right\} .$$

4.4 Graph collision subroutine

In this section we describe a learning graph for the graph collision subroutine that is used in the learning graph given in Section 4.3. For each vertex v at the end of stage $k+m'$ we will attach a learning graph $\mathcal{G}_v$. The root of $\mathcal{G}_v$ will be the label of v and we will show that it has complexity $\sqrt{n}r^{d/(d+1)}$. Furthermore for every flow p_y on $\mathcal{G}_v$, the sinks of flow will be L-vertices that have loaded a copy of H. We now describe $\mathcal{G}_v$ in further detail.

A vertex v at the end of stage $k+m'$ is labeled by a $(k-1)$-partite graph Q on color classes $B_1,\dots,B_{k-1}$ of size r. The edges of Q are the union of the edges in bipartite graphs $Q_1,\dots,Q_{m'}$ each of type $(\{(r/2-1,rs),(r/2+1,rs+1)\},\{(r/2-1,rs),(r/2+1,rs+1)\})$. This will be the label of the root of $\mathcal{G}_v$.

On $\mathcal{G}_v$ we define a flow p'_y for every input y such that $p_y(v^-) > 0$ in the learning graph loading $H_{[1,k-1]}$. Say that y contains a copy of H and that vertices $a_1,\dots,a_k$ span H in y. For ease of notation, assume that vertex a_k (the degree d vertex removed from H) is connected to $a_1,\dots,a_d$. Recall that the L-vertex v will have flow if and only if $a_i \in B_i, a_k \notin B_i$ for $i = 1,\dots,k-1$ and if $e_\ell = \{i,j\}$ then the edge $\{a_i,a_j\}$ is present in Q_ℓ for $\ell = 1,\dots,m'$, and both a_i,a_j have degree $rs+1$ in Q_ℓ. Thus for each such y we will define a flow on $\mathcal{G}_v$. The flow will only depend on $a_1,\dots,a_k$. The complexity of $\mathcal{G}_v$ will depend on a parameter $1 \le \lambda \le r$, that we will optimize later.

Stage 0: Choose a vertex $u \notin B_i$ for $i = 1, \dots k-1$ and load λ edges between u and vertices of degree $rs+1$ in B_i, for each $i = 1, \dots d$. Flow is directed uniformly along those L-edges where $u = a_k$ and none of the edges loaded touch any of the $a_1, \dots, a_d$.

Stage t for $t = 1, \dots, d$: Load an additional edge between u and B_t. The flow is directed uniformly along those L-edges where the edge loaded is $\{a_k, a_t\}$.

Complexity analysis of the stages

- Stage 0: We use Lemma 3.4. As the vertices at the beginning of this stage consist only of the root, conditions (1) and (2) are trivially satisfied; Condition (3) is satisfied by construction. Flow is present in all L-edges of this stage where $u = a_k$, which is a $\Omega(1/n)$ fraction of the total number of L-edges. Thus the degree ratio $d/g = O(n)$. The length of the stage is λ, giving a total cost of $\lambda\sqrt{n}$.

- Stage t for $t = 1, \dots, d$: Let V_t be the set of vertices at the beginning of stage t. The definition of flow depends only on $a_1, \dots, a_k$, thus S_n acts transitively on the flows. Applying Lemma 3.6 gives that $\{p'(y)\}$ is consistent with $[u]^+$ for $u \in V_t$. Also by construction the hypothesis of Lemma 3.7 is satisfied, thus we are in position to use Lemma 3.5.

  The length of each stage is 1. The out-degree of an L-vertex in stage t is $O(r)$ while the flow uses just one outgoing edge, thus the degree ratio $d/g = O(r)$. Finally, we must estimate the fraction of vertices in $[u]$ with flow for $u \in V_t$. A vertex u in V_t has flow if and only if a_k was loaded in stage 0 and the edges $\{a_k, a_i\}$ are loaded for $i = 1, \dots, t-1$. The probability over $\sigma \in S_n$ that the first event holds in $\sigma(u)$ is $\Omega(1/n)$. Given that a_k has been loaded at vertex $u \in V_t$ the probability over σ that $\{a_k, a_i\} \in \sigma(u)$ is $\Omega(\lambda/r)$. Thus we obtain that the maximum vertex ratio at stage t is $n(r/\lambda)^{t-1}$. The complexity of stage t is maximized when $t = d$, giving an overall complexity $\sqrt{nr}(r/\lambda)^{(t-1)/2}$.

The sum of the costs $\lambda\sqrt{n}$ and $\sqrt{nr}(r/\lambda)^{(t-1)/2}$ is minimized for $\lambda = r^{d/(d+1)}$ giving a cost of $O(\sqrt{n}r^{d/(d+1)})$.

4.5 Comparison with the quantum walk approach

It is insightful to compare the cost of the learning graph algorithm for finding a subgraph with the the algorithm of [15] using a quantum walk on the Johnson graph. We saw in the analysis of the learning graph that there were three important terms in the cost, denoted S', U', C'. In the quantum walk formalism there are also three types of costs: setup, aggregated update, and aggregated checking, which we will denote by S, U, C. When the walk is done on the Johnson graph with vertices labeled by r-element subsets these costs are

$$\begin{aligned} S &= r^2 \\ U &= \left(\frac{n}{r}\right)^{(k-1)/2} r^{3/2} \\ C &= \left(\frac{n}{r}\right)^{(k-1)/2} \sqrt{n}r^{d/(d+1)}. \end{aligned}$$

Here d is the minimal degree of a vertex in H.

Here there is only one parameter, and in general r cannot be chosen to make all three terms equal. In the case of triangle finding ($k = 3, d = 2$), the choice $r = n^{3/5}$ is made. This makes $S = n^{1.2}$ and $U = C = n^{1.3}$. In the general case of finding H, the choice $r = n^{1-1/k}$ is made, giving the first and second terms equal to $n^{2-2/k}$ and the third term $C = n^{2-1/k(1+k/(d+1)+(d-1)/2(d+1))}$. Thus $C < S = U$ even for the largest possible value $d = k-1$. Because of this, the analysis gives $n^{2-2/k}$ queries for any graph on k vertices, independent of d.

Acknowledgments

We thank Aleksandrs Belovs for pointing out an error in the original version of the algorithm. We are very grateful to László Babai whose insightful remarks made us realize the importance of distinguishing the two algorithms presented here. We also thank the anonymous referees for their helpful comments for improving the presentation of the paper.

References

[1] A. AMBAINIS: Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. 2

[2] R. BEALS, H. BUHRMAN, R. CLEVE, M. MOSCA, AND R. DE WOLF: Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. 2

[3] A. BELOVS: Learning-graph-based quantum algorithm for k-distinctness. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, 2012. To appear. 2

[4] A. BELOVS: Span programs for functions with constant-sized 1-certificates. In *Proceedings of the ACM Symposium on the Theory of Computing*, pp. 77–84, 2012. 2, 3, 4, 5

[5] H. BUHRMAN, R. CLEVE, R. DE WOLF, AND C. ZALKA: Bounds for small-error and zero-error quantum algorithms. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pp. 358–368, 1999. 2

[6] H. BUHRMAN, C. DÜRR, M. HEILIGMAN, P. HØYER, F. MAGNIEZ, M. SANTHA, AND R. DE WOLF: Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005. 2

[7] A. CHILDS AND R. KOTHARI: Quantum query complexity of minor-closed graph properties. In LEIBNIZ INTERNATIONAL PROCEEDINGS IN INFORMATICS, editor, *Proceedings of Symposium on Theoretical Aspects of Computer Science*, volume 9, pp. 661–672, 2011. 2

[8] C. DÜRR, M. HEILIGMAN, P. HØYER, AND M. MHALLA: Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. 2

[9] M. ETTINGER, P. HØYER, AND E. KNILL: Hidden subgroup states are almost orthogonal. Technical Report quant-ph/9901034, arXiv, 1999. 2

[10] LOV K. GROVER: A fast quantum mechanical algorithm for database search. In *Proceedings of the ACM Symposium on the Theory of Computing*, pp. 212–219, 1996. 1

[11] P. HØYER AND R. ŠPALEK: Lower bounds on quantum query complexity. *Bulletin of the European Association for Theoretical Computer Science*, 87, 2005. Also arXiv report quant-ph/0509153v1. 3

[12] PETER HØYER, TROY LEE, AND ROBERT ŠPALEK: Negative weights make adversaries stronger. In *Proceedings of the ACM Symposium on the Theory of Computing*, pp. 526–535, 2007. [arXiv:quant-ph/0611054] 3

[13] T. LEE, R. MITTAL, B. REICHARDT, R. ŠPALEK, AND M. SZEGEDY: Quantum query complexity of state conversion. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pp. 344–353, 2011. 2

[14] F. MAGNIEZ, A. NAYAK, J. ROLAND, AND M. SANTHA: Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. 2

[15] F. MAGNIEZ, M. SANTHA, AND M. SZEGEDY: Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007. 2, 3, 18

[16] BEN W. REICHARDT: Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. In *Proceedings of IEEE Symposium on Foundations of Computer Science*, pp. 544–551, 2009. [arXiv:0904.2759 [quant-ph]] 3

[17] BEN W. REICHARDT: Reflections for quantum query algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pp. 560–569, 2011. 2, 4

[18] P. SHOR: Algorithms for quantum computation: Discrete logarithm and factoring. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. 1

[19] D. SIMON: On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. 2

[20] Y. ZHU: Quantum query complexity of subgraph containment with constant-sized certificates. Technical Report arXiv:1109.4165v1, arXiv, 2011. 3

AUTHORS

Troy Lee
senior research fellow
Centre for Quantum Technologies, National University of Singapore, Singapore 117543
troyjlee@gmail.com
http://www.research.rutgers.edu/~troyjlee

Frédéric Magniez
senior researcher
CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, F-75205 Paris, France
frederic.magniez@univ-paris-diderot.fr
http://www.liafa.univ-paris-diderot.fr/~magniez

Miklos Santha
senior researcher
CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris Cité, F-75205 Paris, France
and Centre for Quantum Technologies, National University of Singapore, Singapore 117543
miklos.santha@liafa.univ-paris-diderot.fr
http://www.liafa.univ-paris-diderot.fr/~santha