Quantum Algorithm for k-distinctness with Prior Knowledge on the Input

Aleksandrs Belovs* Troy Lee†

Abstract

It is known that the dual of the general adversary bound can be used to build quantum query algorithms with optimal complexity. Despite this result, not many quantum algorithms have been designed this way. This paper shows another example of such algorithm.

We use the learning graph technique from [Bel11b] to give a quantum algorithm for k-distinctness problem that runs in $o(n^{3/4})$ queries, for a fixed k, given some prior knowledge on the structure of the input. The best known quantum algorithm for the unconditional problem uses $O(n^{k/(k+1)})$ queries.

1 Introduction

This paper is a sequel of [Bel11b] on applications of span programs, or, more generally, dual of the Adversary Bound, for constructing quantum query algorithms for functions with 1-certificate complexity bounded by a constant. Also, we use the computational model of a learning graph. In the aforementioned paper, a reduction of a learning graph to a quantum query algorithm was done using the notion of a span program, another computational model, proven to be equivalent to quantum query algorithms in the papers of Reichardt *et al.* [Rei11, LMR+11].

Two questions remained open from the last paper. Firstly, the logarithmic increase in the complexity for functions with non-Boolean input; and whether a learning graph that uses values of the variables to weight its arcs has more power than the one that doesn't. We fully resolve the first concern by switching from span programs to a more general notion of the dual of the adversary bound that possesses the same properties, and, thus, getting a query algorithm with the same complexity as the learning graph, up to a constant factor.

For the analysis of the second problem, we have chosen the k-distinctness problem for $k > 2$. This is the most symmetric problem for which the knowledge of the values of variables can be important in construction of the learning graph. Let us define this problem here.

The *element distinctness* problem consists in computing the function $f\colon [m]^n \to \{0,1\}$ that evaluates to 1 iff there is a pair of equal elements (known as *collision*) in the input, i.e., $f(x_1,\dots,x_n) = 1$ iff $\exists i \neq j : x_i = x_j$. The quantum query complexity of the element distinctness problem is well understood. It is known to be $\Theta(n^{2/3})$, the algorithm given by Ambainis [Amb07] and the lower bound shown by Aaronson and Shi [AS04] for the case of large alphabet size $\Omega(n^2)$ and by Ambainis [Amb05] in the general case. Even more, the lower bound $\Omega(n^{2/3})$ holds if one assumes there is either no, or exactly one collision in the input.

The *k-distinctness problem* is a direct generalization of the element distinctness problem. Given the same input, function evaluates to 1 iff there is a set of k input elements that are all equal. The situation with the quantum query complexity of the k-distinctness problem is not so clear.

*Faculty of Computing, University of Latvia, Raina bulv. 19, Riga, LV-1586, Latvia, stiboh@gmail.com.
†Centre for Quantum Technologies

As element distinctness reduces to the k-distinctness problem by repeating each element $k-1$ times, the lower bound of $\Omega(n^{2/3})$ carries over to the k-distinctness problem (this argument is attributed to Aaronson in [Amb07]). However, the best known algorithm requires $O(n^{k/(k+1)})$ quantum queries [Amb07].

As the above reduction shows, the k-distinctness problem can only become more difficult as k increases. There is another difficulty that arises when $k > 2$—this the huge diversity in the inputs. For element distinctness, all inputs that are distinct are essentially the same— they are all related by an automorphism of the function. Similarly, without loss of generality, one may assume that an input which is not distinct has a unique collision, and again all such inputs are related by an automorphism. When $k > 2$ this is no longer the case. For example, for $k = 3$ inputs can differ in the number of unique elements.

Main theorem In this paper, we show how one can fight the first difficulty, but we ignore the second one. Before explaining how we do so, let us give some additional definitions.

Let $(x_i)_{i\in[n]}$ be the input variables for the k-distinctness problem. Assume some subset $J \subseteq [n]$ is fixed. A subset $I \subseteq J$ is called *t-subtuple* with respect to J if

$$\forall i,j \in I : x_i = x_j, \qquad \forall i \in I\ \forall j \in J \setminus I : x_i \neq x_j \qquad \text{and} \qquad |I| = t, \tag{1}$$

i.e., if it is a maximal subset of equal elements and it has size t. For the important special case $J = [n]$, we call them *t-tuples*. If I is such that only the first condition of (1) is satisfied, we call it *subset of equal elements*.

We give a quantum algorithm for the k-distinctness problem that runs in $o(n^{3/4})$ queries for a fixed k, but with the prior knowledge on the number of t-tuples in the input. Using the same reduction as in [Amb07], it is easy to show the complexity of this problem is $\Omega(n^{2/3})$ as well.

Theorem 1. *Assume we know the number of t-tuples in the input for the k-distinctness problem for all $t = 1,\dots,k-1$ with precision $O(\sqrt[4]{n})$. Then, the problem can be solved in $O(n^{1-2^{k-2}/(2^k-1)})$ quantum queries. The constant behind the O depends on k, but not on n.*

The precision in the formulation of the theorem can be loosened, $O(\sqrt[4]{n})$ is the most obvious value that works for all k's. See Section 4.3 for more details. Concerning the complexity of the algorithm, it is the exact one, and we do not know whether it can be improved.

Organization of the Paper The paper is organized as follows. In Section 2, we define basic notions from quantum query complexity and probability theory. In Section 3, we define learning graphs and give a quantum algorithm for computing them. In Section 4, we develop some tools and get ready for Section 5, where we prove Theorem 1.

2 Preliminaries

Let $[m]$ denote the set $\{1,2,\dots,m\}$ and consider a function $f\colon [m]^n \supseteq \mathcal{D} \to \{0,1\}$. We identify the set of input indices of f with $[n]$. An *assignment* is a function $\alpha\colon [n] \supset S \to [m]$. One should think of this function as fixing values for input variables in S. We say input $x = (x_i)_{i\in[n]}$ *agrees* with assignment α if $\alpha(i) = x_i$ for all $i \in S$. If $S \subseteq [n]$, by x_S, we denote the only assignment on S that agrees with x.

An assignment α is called a *b-certificate* for f if any input from $\mathcal{D}$, consistent with α, is mapped to b by f. The *certificate complexity* $C_x(f)$ of function f on input x is defined as the minimal size of a certificate for f that agrees with x. The b-certificate complexity $C^{(b)}(f)$ is defined as $\max_{x\in f^{-1}(b)} C_x(f)$.

We use $[a,b]$ and $]a,b[$ to denote closed and open, respectively, intervals of $\mathbb{R}$; $\mathbb{R}^+$ to denote the set of non-negative reals. For the real vector space $\mathbb{R}^m$, we use the ℓ_∞-norm, $\|x\|_\infty = \max_i |x_i|$. In particular, we denote the ℓ_∞-ball of radius d around x by $\mathcal{B}(x,d)$. We use $\mathcal{B}(d)$ to denote the ball with center zero and radius d.

For the complex vector spaces $\mathbb{C}^m$, however, we use a more common ℓ_2-norm, $\|x\| = \sqrt{\sum_i |x_i|^2}$.

2.1 Adversary bound

In this paper, we work with query complexity of quantum algorithms, i.e., we measure the complexity of a problem by the number of queries to the input the best algorithm should make. Query complexity provides a lower bound on time complexity. For many algorithms, query complexity can be analyzed easier than time complexity. For the definition of query complexity and its basic properties, a good reference is [BdW02].

The adversary bound, originally introduced by Ambainis [Amb02], is one of the most important lower bound techniques for quantum query complexity. In fact, a strengthening of the adversary bound, known as the general adversary bound [HLŠ07], has recently been shown to characterize quantum query complexity, up to constant factors [Rei11, LMR$^+$11].

What we actually use in the paper, is the dual of the general adversary bound. It provides upper bounds on the quantum query complexity, i.e., quantum query algorithms. Due to the same results, it also is tight. Despite this tight equivalence, the actual applications of this upper bound (in the form of span programs) have been limited, mostly, to formulae evaluation [RŠ], and, recently, linear algebra problems [Bel11a]. In [Bel11b], it was used to give a variant of an optimal algorithm for the element distinctness problem, and an algorithm for the triangle problem having better complexity than the one known before. In this paper, we provide yet another application.

The (dual of the) general adversary bound is defined as follows.

Definition 2. Let $f\colon [m]^n \to \{0,1\}$ be a function.

$$\begin{aligned}\mathrm{Adv}^{\pm}(f) = \underset{\substack{k\in\mathbb{N}\\ u_{x,j}\in\mathbb{C}^k}}{\text{minimize}} \ & \max_x \sum_{j\in[n]} \|u_{x,j}\|^2 \\ \text{subject to } & \sum_{\substack{j\\ x_j\neq y_j}} \langle u_{x,j}|u_{y,j}\rangle = 1 \text{ whenever } f(x)\neq f(y).\end{aligned} \tag{2}$$

For our application, it will be more convenient to use a different formulation of the objective value.

Claim 3.

$$\begin{aligned}\mathrm{Adv}^{\pm}(f) = \underset{\substack{k\in\mathbb{N}\\ u_{x,j}\in\mathbb{C}^k}}{\text{minimize}} \ & \sqrt{\left(\max_{x\in f^{-1}(1)} \sum_{j\in[n]} \|u_{x,j}\|^2\right)\left(\max_{y\in f^{-1}(0)} \sum_{j\in[n]} \|u_{y,j}\|^2\right)}. \\ \text{subject to } & \sum_{\substack{j\\ x_j\neq y_j}} \langle u_{x,j}|u_{y,j}\rangle = 1 \text{ whenever } f(x)\neq f(y).\end{aligned} \tag{3}$$

Proof. The objective value in Eq. (3) is less than that of Eq. (2) by the inequality of arithmetic and geometric means. For the other direction, note that the constraint is invariant under multiplying all vectors $u_{x,j}$ where $f(x)=1$ by c and all vectors $u_{y,j}$ where $f(y)=0$ by c^{-1}. In this way we can ensure that the maximum in Eq. (2) is the same over $f^{-1}(0)$ and $f^{-1}(1)$ and so equal to the geometric mean. □

The general adversary bound characterizes bounded-error quantum query complexity.

Theorem 4 ([Rei11, LMR+11]). *Let f be as above. Then $Q_{1/4}(f) = \Theta(\mathrm{Adv}^{\pm}(f))$.*

2.2 Martingales and Azuma's Inequality

We assume the reader is familiar with basic notions of probability theory. In this section, we state some concentration results we will need in the proof of Theorem 1. The results are rather standard, can be found, e.g., in [AS08].

A *martingale* is a sequence $X_0, \ldots, X_m$ of random variables such that $\mathrm{E}[X_{i+1} \mid X_0, \ldots, X_i] = X_i$, for all i's.

Theorem 5 (Azuma's Inequality). *Let $0 = X_0, \ldots, X_m$ be a martingale such that $|X_{i+1} - X_i| \leq 1$ for all i's. Then*

$$\Pr[X_m > \lambda\sqrt{m}] < e^{-\lambda^2/2}.$$

for all $\lambda > 0$.

A standard way of defining martingales, known as *Doob martingale process*, is as follows. Assume $f(y_1, \ldots, y_m)$ if a real-valued function, and there is a probability distribution Y on the input sequences. The Doob martingale $D_0, \ldots, D_m$ is defined as

$$D_i = \mathop{\mathrm{E}}_{y' \in Y}[f(y') \mid \forall j \leq i : y'_j = y_j]$$

that is a random variable dependent on $y \in Y$. In particular, $D_0 = \mathrm{E}[f]$ and $D_m = f(y)$. This is a martingale, and Azuma's inequality states $f(y)$ isn't far away from its expectation with high probability, if revealing one input variable has little effect on the expectation of the random variable.

3 Learning graphs

3.1 Definitions

By Theorem 4, to upper bound the quantum query complexity of a function, it suffices to construct a feasible solution to Eq. (2). Trying to come up with vectors which satisfy all pairwise equality constraints, however, can be quite challenging even for simple functions.

A learning graph, introduced in [Bel11b], is a computational model that aids in the construction of such vectors for a function $f\colon [m]^n \supseteq \mathcal{D} \to \{0,1\}$ with boolean output. By design, a learning graph ensures that the constraint (3) is satisfied, allowing one to focus on minimizing the objective value.

Definition 6. A *learning graph* $\mathcal{G}$ is a directed acyclic connected graph with vertices labeled by subsets of $[n]$, the input indices. It has arcs connecting vertices S and $S \cup \{j\}$ only, where $S \subseteq [n]$ and $j \in [n] \setminus S$. Each arc e is assigned a *weight function* $w_e\colon [m]^S \to \mathbb{R}^+$, where S is the origin of e.

A learning graph can be thought of as modeling the development of one's knowledge about the input during a query algorithm. Initially, nothing is known, and this is represented by the root labeled by $\emptyset$. When at a vertex labeled by $S \subseteq [n]$, the values of the variables in S have been learned. Following an arc e connecting S to $S \cup \{j\}$ can be interpreted as querying the value of variable x_j. We say the arc *loads* element j. When talking about vertex labeled by S, we call S the set of *loaded elements*.

In order for a learning graph to compute function f correctly, for any $x \in f^{-1}(1)$, there should exist a vertex of the learning graph containing a 1-certificate for x. We call vertices containing a 1-certificate *accepting*.

Let e be a weighted arc of the learning graph from S to $S \cup \{j\}$. In the examples of learning graphs given in [Bel11b], it sufficed to assign e a weight w_e that depended only on the set S and element j, but not the values learned. Here, we follow Remark 4 of [Bel11b] and use a more general model where w_e can depend both on S and j, as well as on the values of the variables in S. We denote $w_e(x) = w_e(x_S)$. Although, this notation is convenient, it is important to keep in mind that values of the variables outside S *do not* affect the value of w_e. The weight 0 of an arc should be thought of as the arc is missing for this particular input.

By $\mathcal{G}(x)$, we denote the instance of $\mathcal{G}$ for input $x \in \mathcal{D}$, i.e., $\mathcal{G}(x)$ has the same vertices and arcs as $\mathcal{G}$ does, only the weight of arc e is a real number $w_e = w_e(x)$. Another way to think of a leaning graph, is like a collection of graphs $\mathcal{G}(x)$ such that arcs from S to $S \cup \{j\}$ in $\mathcal{G}(x^{(1)})$ and $\mathcal{G}(x^{(2)})$ have equal weight if $x^{(1)}_S = x^{(2)}_S$. The arcs with the latter property are called *identical*.

Arcs is the main constituent of the learning graph, and we use notation $e \in \mathcal{G}$ to denote that e is an arc of $\mathcal{G}$. Similarly, we write $e \in \mathcal{G}(x)$.

The *complexity* of a learning graph computing f is defined as the geometrical mean of its *positive* and *negative complexities*. The negative complexity $\mathcal{N}(\mathcal{G}(y))$ for $y \in f^{-1}(0)$ is defined as $\sum_{e \in \mathcal{G}(y)} w_e$. The negative complexity of the learning graph $\mathcal{N}(\mathcal{G})$ is defined as $\max_{y \in f^{-1}(0)} \mathcal{N}(\mathcal{G}(y))$. In order to define positive complexity, we need one additional notion.

Definition 7. The *flow* on $\mathcal{G}(x)$ for $x \in f^{-1}(1)$ is a real-valued function p_e where $e \in \mathcal{G}(x)$. It has to satisfy the following properties:

- vertex $\emptyset$ is the only source of the flow, and it has intensity 1. In other words, the sum of p_e over all e's leaving $\emptyset$ is 1;
- vertex S is a sink iff it is accepting. That is, if $S \neq \emptyset$ and S does not contain a 1-certificate of x for f then, for vertex S, the sum of p_e over all in-coming arcs equals the sum of p_e over all out-going arcs.

The complexity of the flow is defined as $\sum_{e \in \mathcal{G}(x)} p_e^2 / w_e$, with convention $0/0 = 0$. The positive complexity $\mathcal{P}(\mathcal{G}(x))$ is defined as the smallest complexity of a flow on $\mathcal{G}(x)$. The positive complexity of the learning graph $\mathcal{P}(\mathcal{G})$ is defined as $\max_{x \in f^{-1}(1)} \mathcal{P}(\mathcal{G}(x))$.

We often consider a collection of flows p for all $x \in f^{-1}(1)$. In this case, $p_e(x)$ denotes the flow p_e in $\mathcal{G}(x)$.

Let us briefly introduce some additional concepts connected with learning graphs. The i-th *step* of a learning graph is the set of all arcs ending in a vertex of cardinality i. If $E \subseteq \mathcal{G}$ is a set of arcs, we use notation $p_E = \sum_{e \in E} p_e$. Usually, E is a subset of a step. A special case is p_S with S being a vertex; it is used to denote the flow through vertex S, i.e., the sum of p_e over all arcs ending at S.

The following technical result is extracted from [Bel11b]

Lemma 8 (Conditioning)**.** *Suppose V is a subset of vertices such that no vertex is a subset of another. Let p_e be a flow from $\emptyset$ and ending at V of intensity 1, W be a subset of V and $t = \sum_{S \in W} p_S$. Then there exists a flow p' with the same properties, such that $p'_S = p_S/t$ for $S \in W$ and $p'_S = 0$, otherwise. Moreover, the complexity of p' is at most $1/t^2$ times the complexity of p.*

This lemma is applied as follows. One uses some construction to get a flow that ends at V. After that, another construction is applied to obtain a flow that starts at W and ends at the proper sinks, i.e., accepting vertices. In the second flow, vertices in $V \setminus W$ are dead-ends, i.e., no flow

should leave them. Then it is possible to apply Lemma 8 to glue both parts of the flow together and get a valid flow.

3.2 Reduction to Quantum Query Algorithms

In this section, we prove that if there is a learning graph for f of complexity C then $Q_{1/4}(f) = O(C)$. In the case of f with non-boolean input alphabet this solves an open problem from [Bel11b] by removing a logarithmic factor present there. We do this by showing how a learning graph can be used to construct a solution to Eq. (3) of the same complexity, and then appealing to Theorem 4.

Theorem 9. *If there is a learning graph for $f\colon [m]^n \to \{0,1\}$ with complexity C then $\mathrm{Adv}^{\pm}(f) \leq C$.*

Proof. Let $\mathcal{G}$ be the learning graph, w_e be the weight function, and p be the optimal flow.

We show how to construct the vectors $u_{x,j}$ satisfying (3) from $\mathcal{G}$. Let E_j be the set of arcs $e_{S,S\cup\{j\}}$ between S and $S\cup\{j\}$ for some S. Notice that the set of $\{E_j\}_{j\in[n]}$ partition all the arcs in the graph. If $e = e_{S,S\cup\{j\}}$, let $\alpha(e) \in [m]^S$ be an assignment of values to the set labeling the origin of e.

The vectors $u_{x,j}$ will live in a Hilbert space $\bigoplus_{e\in E_j,\alpha(e)} H_{e,\alpha(e)}$ where $\alpha(e) \in [m]^S$ is an assignment of values to the positions in S. In our case each $H_{e,\alpha(e)} = \mathbb{C}$. Thus we think of $u_{x,j} = \bigoplus_{e\in E_j,\alpha(e)} u_{x,j,e,\alpha(e)}$, and now go about designing these vectors.

First of all, if $e = e_{S,S\cup\{j\}}$ then $u_{x,j,e,\alpha(e)} = 0$ if $x_S \neq \alpha(e)$. Otherwise, if $f(y) = 0$ then we set $u_{y,j,e,\alpha(e)} = \sqrt{w_e(y)}$ and if $f(x) = 1$, we set $u_{x,j,e,\alpha(e)} = p_e(x)/\sqrt{w_e(x)}$.

Let us check the objective value. If $f(y) = 0$ then we have

$$\sum_j \|u_{y,j}\|^2 = \sum_j \sum_{e\in E_j} w_e(y) = \sum_{e\in\mathcal{G}} w_e(y) = \mathcal{N}(\mathcal{G}(y)).$$

If $f(x) = 1$ then

$$\sum_j \|u_{x,j}\|^2 = \sum_j \sum_{e\in E_j} \frac{p_e(x)^2}{w_e(x)} = \sum_{e\in\mathcal{G}} \frac{p_e(x)^2}{w_e(x)} = \mathcal{P}(\mathcal{G}(x)).$$

Thus the geometric mean of these quantities it is at most C.

Let us now see that the constraint is satisfied.

$$\begin{aligned}\sum_{j:x_j\neq y_j} \langle u_{x,j}|u_{y,j}\rangle &= \sum_{j:x_j\neq y_j} \sum_{\substack{e_{S,S\cup\{j\}}\\ x_S=y_S}} \langle u_{x,j,e,x_S}|u_{y,j,e,x_S}\rangle \\ &= \sum_j \sum_{\substack{e_{S,S\cup\{j\}}\\ x_S=y_S, x_j\neq y_j}} \frac{p_e(x)}{\sqrt{w_e(x)}}\sqrt{w_e(y)} = \sum_j \sum_{\substack{e_{S,S\cup\{j\}}\\ x_S=y_S, x_j\neq y_j}} p_e(x) = 1\ .\end{aligned}$$

The second equality from the end holds because $w_e(x) = w_e(x_S) = w_e(y_S) = w_e(y)$ due to the construction of the weight function. To see why the last equality holds, note that the set of arcs $e_{S,S\cup j}$ where $x_S = y_S$ and $x_j \neq y_j$ is the cut induced by the vertex sets $\{S \mid x_S = y_S\}$ and $\{S \mid x_S \neq y_S\}$. Since the source is in the first set, and all the sinks are in the second set, the value of the cut is equal to the total flow which is one. □

4 Getting Ready

This section is devoted to the analysis of the applicability of learning graphs for the k-distinctness problem, without constructing the actual learning graph. In Section 4.1, we review the tools

of [Bel11b] to the case when the arcs of the learning graph depend on the values of the variables. In Section 4.2, we make the tools of in Section 4.1 easier to apply. In Section 4.3, we describe the conventions on the input variables we assume for the rest of the paper. In Section 4.4, we develop an important notion of almost symmetric flow that is a generalization of symmetric flow used in [Bel11b]. Finally, in Section 4.5, we describe a learning graph that is equivalent to the previous quantum algorithm for the k-distinctness problem.

4.1 Symmetries

In [Bel11b], the symmetries under consideration were those of the indices of the input variables. This was sufficient because values of the variables did not affect the learning graph. In this paper, we consider a wider group of symmetries, namely $\mathcal{S}_n \times \mathcal{S}_m$, where $\mathcal{S}$ is the full symmetric group, that in the first multiplier permutes the indices, and the second one the values of the variables, i.e., an input $x = (x_i)_{i\in[n]}$ gets mapped by $\sigma = \sigma_{\mathrm{i}} \times \sigma_{\mathrm{o}}$ to $\sigma x = (\sigma_{\mathrm{o}}(x_{\sigma_{\mathrm{i}} i}))_{i\in[n]}$.

Let $\Sigma \subseteq \mathcal{S}_n \times \mathcal{S}_m$ be the symmetry group of the problem, i.e., such that $f(\sigma x) = f(x)$ for all $x \in \mathcal{D}$ and $\sigma \in \Sigma$. For the k-distinctness problem, Σ equals the whole group $\mathcal{S}_n \times \mathcal{S}_m$.

We extend the mapping $x \mapsto \sigma x$ to assignments, as well as vertices and arc of learning graphs in an obvious way. For example, an arc $e \in \mathcal{G}(x)$ from S to $S \cup \{v\}$ is mapped to the arc $\sigma e \in \mathcal{G}(\sigma x)$ from $\sigma_{\mathrm{i}} S$ to $\sigma_{\mathrm{i}}(S \cup \{v\})$. Actually, graph $\mathcal{G}(\sigma x)$ may also not contain the latter arc. To avoid such inconvenience, we assume $\mathcal{G}$ is embedded into the complete graph having all possible arcs of the form $e_{S,S\cup\{v\}}$, with the unused arcs having weights and flow equal to 0. Then, it is easy to see any $\sigma \in \Sigma$ maps a valid flow on $\mathcal{G}(x)$ to a valid flow on $\mathcal{G}(\sigma x)$ in the sense of Definition 7. Of course, the complexity of the latter can be huge, even $+\infty$, because it may have a non-zero flow through an arc having weight 0. Consider two arcs:

$$e_i \in \mathcal{G}(x^{(i)}) \text{ originating in } S_i \text{ and loading } v_i, \text{ for } i = 1, 2. \tag{4}$$

In this section, as well as in Section 4.2, we are going to define various equivalence relations between them, mostly, to avoid the increase in the complexity of the flow under transformations from Σ. Note, in contrary to [Bel11b], we define equivalences between arcs, not transitions, i.e., chains of arcs.

Equivalency Arcs e_1 and e_2 are called *equivalent* iff there exists $\sigma \in \Sigma$ such that $\sigma_{\mathrm{i}}(v_1) = v_2$ and $\sigma(x^{(1)}_{S_1}) = x^{(2)}_{S_2}$. It is natural to assume equivalent arcs have equal weight. We give a formal argument in Proposition 10.

Denote by $\mathcal{E}$ the set of all equivalency classes of $\mathcal{G}$ under this relation. Also, we use notation $\mathcal{E}_i$ for all equivalency classes of step i (an equivalency class is fully contained in one step, hence, this is a valid notion). If $E \in \mathcal{E}$, we use notation $E(x)$ to denote the subset of arcs of $\mathcal{G}(x)$ that belongs to E.

For the k-distinctness, the equivalence is characterized by the structure of the subtuples of S. We capture this by the *specification* $\beta(S)$ of the vertex, i.e., by a list of non-negative integers $(b_1, b_2, \ldots, b_{k-1})$ such that S contains exactly b_t t-subtuples. (If S contains a k- or a larger subtuple it is an accepting vertex and no arcs are leaving it). In particular, $|S| = \sum_t t b_t$. Thus, two arcs are equivalent iff the specifications of their origins are equal.

Strong equivalency The first part of this section describes the equivalence relation for arcs e_1 and e_2 with respect to their weight. We would like to get a stronger equivalence that captures the flow through an arc. This kind of equivalency has already been used in [Bel11b] without explicit definition.

Arcs e_1 and e_2 from (4) are called *strongly equivalent* iff there exists an element $\sigma \in \Sigma$ such that

$$\sigma(x^{(1)}) = x^{(2)}, \qquad \sigma_{\mathrm{i}}(S_1) = S_2 \qquad \text{and} \qquad \sigma_{\mathrm{i}}(v_1) = v_2. \tag{5}$$

Again, due to symmetry, it is natural to assume the flow through strongly equivalent arcs is equal. If, for some positive inputs $x^{(1)}$ and $x^{(2)}$, there is $\sigma \in \Sigma$ such that the first condition of (5) holds, the task of finding a flow for $x^{(2)}$ is reduced to finding a flow for $x^{(1)}$, that is again a corollary of Proposition 10. See also Proposition 11.

Formal argument We give a proof that, without loss in complexity, we may assume weight and flow is constant on equivalent and strongly equivalent arcs, respectively.

Proposition 10. *For any learning graph $\mathcal{G}$, it is possible to construct a learning graph $\mathcal{G}'$ and a flow p' on it with the same or smaller complexity, so that equivalent arcs have the same weight and strongly equivalent arcs have the same flow through them.*

Proof. The proof is a standard application of symmetry. Let p be an optimal flow for $\mathcal{G}$. We define the weights of arcs in $\mathcal{G}'$ and the flow through it as follows:

$$w'_e(\alpha) = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} w_{\sigma e}(\sigma\alpha), \qquad \text{and} \qquad p'_e(x) = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} p_{\sigma e}(\sigma x).$$

If arcs of (4) are equivalent, there exists σ' such that $\sigma' e_1 = e_2$ and $\sigma'(x^{(1)}_{S_1}) = x^{(2)}_{S_2}$. Hence,

$$w'_{e_2}(x^{(2)}) = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} w_{\sigma e_2}(\sigma(x^{(2)}_{S_2})) = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} w_{\sigma\sigma' e_1}(\sigma\sigma'(x^{(1)}_{S_1})) = w'_{e_1}(x^{(1)}),$$

since Σ is a group. The equality of flows is proven in a same way. Let us check the complexity. For a negative input y, we have:

$$\mathcal{N}(\mathcal{G}'(y)) = \sum_{e \in \mathcal{G}} \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} w_{\sigma e}(\sigma y) = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} \mathcal{N}(\mathcal{G}(\sigma y)).$$

Hence, for at least one σ, $\mathcal{N}(\mathcal{G}'(y)) \le \mathcal{N}(\mathcal{G}(\sigma y))$. Thus, $\mathcal{N}(\mathcal{G}') \le \mathcal{N}(\mathcal{G})$.

For the positive case, at first note that p' is a valid flow, as a convex combination of valid flows. For any $x \in f^{-1}(1)$, we have

$$\begin{aligned}\mathcal{P}(\mathcal{G}'(x)) &\le \sum_{e \in \mathcal{G}} \left(\frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} p_{\sigma e}(\sigma x) \right)^2 \left(\frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} w_{\sigma e}(\sigma x) \right)^{-1} \\ &\le \sum_{e \in \mathcal{G}} \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} \frac{p_{\sigma e}(\sigma x)^2}{w_{\sigma e}(\sigma x)} = \frac{1}{|\Sigma|} \sum_{\sigma \in \Sigma} \mathcal{P}(\mathcal{G}(\sigma x)).\end{aligned}$$

The second inequality follows from the Jensen's inequality for the square function $\left(\sum_{\sigma \in \Sigma} \gamma_\sigma z_\sigma\right)^2 \le \sum_{\sigma \in \Sigma} \gamma_\sigma z_\sigma^2$, with $\gamma_\sigma = w_{\sigma e}(\sigma x) / \left(\sum_{\sigma \in \Sigma} w_{\sigma e}(\sigma x)\right)$ and $z_\sigma = p_{\sigma e}(\sigma x)/\gamma_\sigma$. Due to the same argument, $\mathcal{P}(\mathcal{G}') \le \mathcal{P}(\mathcal{G})$. □

4.2 Loosening equivalencies

Although equivalencies defined in the previous section are optimal, they are not always convenient to work with. They turn out to be too strong, that results in a vast number of equivalency classes that should be treated separately. In this section, we describe a number of ways to loosen these equivalences, thus reducing the number of classes and making them easier to work with.

Equivalency Assume the weight function is decomposed as $w_e(\alpha) = w_e(\theta(\alpha))$, where θ is some "filter" that captures the properties of α we are interested in. It is good to assume symmetry preserves θ, i.e., $\theta(\alpha_1) = \theta(\alpha_2)$ implies $\theta(\sigma\alpha_1) = \theta(\sigma\alpha_2)$ for any $\sigma \in \Sigma$. Transitions e_1 and e_2 are called θ-*equivalent* iff there exists $\sigma \in \Sigma$ such that $\sigma_{\mathrm{i}}(v_1) = v_2$ and $\sigma(\theta(x^{(1)}_{S_1})) = \theta(x^{(2)}_{S_2})$. It is again natural to assume θ-equivalent arcs have the same weight. Two main examples are:

θ_1, the identity. This results in the relation from the previous section. This is the main equivalency used in Section 5;

θ_2, mapping α to its domain $\mathcal{D}(\alpha)$. For the k-distinctness, arcs e_1 and e_2 from (4) are θ_2-equivalent iff $|S_1| = |S_2|$. This is the equivalency used in [Bel11b].

Strong equivalency Unlike equivalency, strong equivalency turns out to be too strong for all our applications. We can weaken it by considering $\mathcal{G}$ as a learning graph for function $\tilde{f}$ that gets as input $\tilde{x}$, the original input x with some information removed.

More precisely, extend the output alphabet $[m]$ with a set of *special characters* Q. Let $\vartheta\colon f^{-1}(1) \to ([m] \cup Q)^n$ be the function that maps x to $\tilde{x}$. We extend elements of Σ to $([m] \cup Q)^n$ by assuming $\sigma_{\mathrm{o}}(c) = c$, if $\sigma_{\mathrm{i}} \times \sigma_{\mathrm{o}} \in \Sigma$ and $c \in Q$.

The function $\tilde{f}\colon ([m] \cup Q)^n \to \{0,1\}$ is defined as $\tilde{f}(\vartheta(x)) = 1$ for all $x \in f^{-1}(1)$. Let $\tilde{\mathcal{G}}$ be the same learning graph as $\mathcal{G}$ but calculating $\tilde{f}$. Arcs e_1 and e_2 from (4) are called ϑ-*strongly equivalent* iff the corresponding arcs $\tilde{e}_1 \in \tilde{\mathcal{G}}(\vartheta(x^{(1)}))$ and $\tilde{e}_2 \in \tilde{\mathcal{G}}(\vartheta(x^{(2)}))$ are strongly equivalent.

For this construction to work, we require a stronger definition of a 1-certificate for $\tilde{f}$. We say an assignment $\alpha\colon [n] \supseteq M \to [m] \cup Q$ is a *1-certificate* if, for all $x \in f^{-1}(1)$ such that $\vartheta(x)_M = \alpha$, x_M is a 1-certificate of x in f. With this definition, any valid flow in $\tilde{\mathcal{G}}(\vartheta(x))$ is simultaneously a valid flow in $\mathcal{G}(x)$.

We give three examples of ϑ's.

ϑ_1, the identity. This results in the relation from the previous section. We have no example of using this equivalency;

ϑ_2, the equivalency used in [Bel11b]. Let $Q = \{\cdot, \star\}$. For a positive input x, fix some 1-certificate α. Let M be the domain of α. The elements of M are called *marked*. Define $\tilde{x} = \vartheta_2(x)$ as

$$\tilde{x}_i = \begin{cases} \star, & i \in M; \\ \cdot, & \text{otherwise.} \end{cases}$$

Clearly, $\vartheta_2(x)_M$ is a 1-certificate. Refer to Section 4.5 for an example of usage of this equivalency.

ϑ_3, defined in Section 4.3. The main equivalency used in Section 5.

Again, we assume the flow through ϑ-strongly equivalent arcs is equal. The equivalencies we use in the paper possess two additional symmetric properties. Firstly, if $x^{(1)}, x^{(2)} \in f^{-1}(1)$ and $\sigma \in \Sigma$ are such that $\sigma(\vartheta(x^{(1)})) = \vartheta(x^{(2)})$ then, for any 1-certificate α of $\vartheta(x^{(1)})$, $\sigma\alpha$ is a 1-certificate of $\vartheta(x^{(2)})$. Secondly, any ϑ-strong equivalency class, having non-zero flow through it, is completely contained in some θ-equivalency class.

Proposition 11. *If θ, $\vartheta, x^{(1)}, x^{(2)}$ and a flow p_e on $\tilde{\mathcal{G}}(\vartheta(x^{(1)}))$ satisfy the above conditions and θ-equivalent arcs in $\mathcal{G}$ have the same weight, then σp_e is a valid flow on $\tilde{\mathcal{G}}(\vartheta(x^{(2)}))$ with the same complexity (that, also, is a valid flow on $\mathcal{G}(x^{(2)})$).*

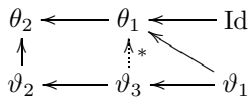

Figure 1: Relations between equivalencies. Arrows are from a more strong relation to a weaker one. Id is the identity relation from Section 3.1, θ's are equivalencies and ϑ's are strong equivalencies. Implication from ϑ_3 to θ_1 only holds for arcs with non-zero flow.

4.3 Conventions for k-distinctness

Strong equivalency, as defined in Section 4.1, turns out to be too strong for the k-distinctness problem, because for most of the pairs $x^{(1)}, x^{(2)} \in f^{-1}(1)$ there is no σ such that $\sigma(x^{(1)}) = x^{(2)}$, and, hence, no arcs from $\mathcal{G}(x^{(1)})$ and $\mathcal{G}(x^{(2)})$ can be strongly equivalent, whatever $\mathcal{G}$ is. We use the loosening tool of Section 4.2 to define ϑ_3 so that there always exists σ that maps $\vartheta_3(x^{(1)})$ to $\vartheta_3(x^{(2)})$. Then, by Proposition 11, defining a flow for any positive input x is enough to get a flow for all positive inputs.

Let the set of special characters be $Q = \{\cdot\}$. Fix an arbitrary positive input x. First of all, we identify a subset M of k equal elements (the marked elements in terminology of ϑ_2). Next, due to the condition in Theorem 1, we may assume there are non-negative integers $\ell_1, \ldots, \ell_{k-1}$ such that in any valid input (either positive, or negative) there are at least ℓ_t t-tuples and

$$n - \sum_{t=1}^{k-1} t\ell_t = O(\sqrt[4]{n}).$$

We arbitrary select ℓ_t t-tuples. Denote by A_t the union of the selected t-tuples. We also use notation $A_{\geq t}$ to denote $\bigcup_{j=t}^{k-1} A_j$. We define $\tilde{x} = \vartheta_3(x)$ as

$$\tilde{x}_i = \begin{cases} x_i, & i \in A_{\geq 1} \cup M; \\ \cdot, & \text{otherwise.} \end{cases}$$

Clearly, assignment $\tilde{x}_M$ is a 1-certificate. In the learning graph for $\tilde{f}$, defined using ϑ_3, we will have $p_e(x) = 0$ if the origin of e has at least one $\cdot$ in $\vartheta_3(x)$. This convention assures that θ_1 and ϑ_3 satisfy the conditions of Proposition 11. Further, we are going to ignore vertices having $\cdot$'s in them. Figure 1 describes which of the defined equivalence relations imply which.

Let us use this spot to mention one more convention on the input. Namely,

$$\forall t \leq k-1 : \ell_t = \Omega(n). \tag{6}$$

Any other case can be reduced to this one by extending the input by n t-tuples with elements outside the range of the original problem, for each $t \leq k-1$.

The strong equivalency class (with respect to ϑ_3) of an arc depends solely on the *types* of its initial and target vertices. The type $\tilde{\beta}(S)$ of vertex S is an $(k-1) \times k$-matrix $(b_{t,s})$, where $b_{t,s}$ is the number of t-subtuples of S contained in A_s (or M, if $s = k$). Most of the time, we will implicitly assume $b_{t,k} = 0$ for all t's, hence, describe the type of a vertex by an $(k-1) \times (k-1)$-matrix, assuming the removed row contains only zeroes. Note also that the specification (b_t) can be expressed using the type: $b_t = \sum_{s=1}^{k} b_{t,s}$.

We will have to measure distance between types. When doing so, we treat them as vectors. I.e., the distance between types $\tilde{\beta}(S) = (b_{s,t})$ and $\tilde{\beta}(S') = (b'_{s,t})$ is defined as

$$\|\tilde{\beta}(S) - \tilde{\beta}(S')\|_\infty = \max_{s,t} |b_{s,t} - b'_{s,t}|.$$

Negative complexity Here we estimate how the restriction from the actual number of t-tuples in the input to ℓ_t ones in $A_{\geq 1}$ affects the negative complexity.

Lemma 12. *Consider a set $A'_{\geq 1}$ that is defined similarly to $A_{\geq 1}$, only it has ℓ'_t t-tuples. Assume $|\ell_t - \ell'_t| \leq d = o(n)$ for all t's. Let (b_t) be any specification such that $b = \sum_t b_t = o(n)$. Then the ratio of the number of subsets satisfying (b_t) in $A_{\geq 1}$ and $A'_{\geq 1}$ is at most $e^{O(db/n)}$.*

Proof. It is straight-forward to calculate the number of subsets of $A_{\geq 1}$ satisfying specification (b_t). Indeed, it equals

$$\sum_{(b_{t,s})} \prod_{s=1}^{k-1} \left[\binom{\ell_s}{b_{1,s}} \binom{s}{1}^{b_{1,s}} \binom{\ell_s - b_{1,s}}{b_{2,s}} \binom{s}{2}^{b_{2,s}} \cdots \binom{\ell_s - b_{1,s} - \cdots - b_{s-1,s}}{b_{s,s}} \binom{s}{s}^{b_{s,s}} \right] \tag{7}$$

where the summation is over all types $(b_{t,s})$ that agree with specification (b_t). Eq. (7), with ℓ_t replaced by ℓ'_t, gives the corresponding number of subsets in $A'_{\geq 1}$. It is enough to show that each multiplier featuring ℓ_s in (7) changes by at most a factor of $e^{O(db/n)}$. But we have:

$$\binom{\ell'_s - b_{1,s} - \cdots - b_{t-1,s}}{b_{t,s}} \Big/ \binom{\ell_s - b_{1,s} - \cdots - b_{t-1,s}}{b_{t,s}} = \left(1 + O\left(\frac{d}{n}\right)\right)^{O(b)} = e^{O(db/n)},$$

where we used that $\ell_s = \Theta(n)$, because of (6). □

Since the complexity mentioned in Theorem 1 is $o(n^{3/4})$, it is natural to assume no vertex of the learning graph has more elements. It's actually the case, as described in Section 5. The precision $O(\sqrt[4]{n})$ in the formulation of Theorem 1 has been chosen so that restriction of the flow to $A_{\geq 1} \cup M$ does not hurt the negative complexity, as it can be seen from the next

Corollary 13. *Fix any possible negative input y, and any valid specification (b_t) with all entries $o(n^{3/4})$. Then, the number of subsets of $[n]$ satisfying (b_t) is bounded by a constant times the number of such subsets included in $A_{\geq 1}$.*

Because of this, we may act as if the set of input variables is $A_{\geq 1} \cup M$, not $[n]$.

4.4 Almost symmetric flows

Assume the following scenario. We have chosen which equivalency classes will be present in the learning graph. Also, for each positive input, we have constructed a flow. The task is to weight the arcs of the learning graph to minimize its complexity. In this section, we define a way of performing this task, if the flow satisfies some requirements.

For the k-distinctness problem, all arcs leaving a vertex are equivalent, hence, to specify which equivalency classes are present, it is enough to define which vertices have arcs leaving them. For each step, we define a set of *valid specifications*. If a vertex before the step satisfies one of them, we draw all possible arcs out of it. Otherwise, we declare it a *dead-end* and draw no arcs out of it.

The flow is called *symmetric* in [Bel11b] if, for each equivalency class, the flow through an arc of it is either 0, or p, where p does not depend on the input, but may depend on the equivalency class; also it is required that the number of arcs having flow p does not depend on the input as well. This notion was sufficient for the applications in that paper, because ϑ_2-strong equivalence was used, and that is easy to handle. In this paper, we use ϑ_3-strong equivalence, and it is not enough with symmetric flows. Thus, we have to generalize this notion.

Definition 14. The flow is called *almost symmetric* if, for each equivalency class E, there exist constants $\pi(E)$ and $\tau(E)$ such that, for each positive input x, there exists a subset $G(E,x) \subseteq E(x)$ such that

$$\tau(E)|G(E,x)| = \Theta\left(\max_{y\in f^{-1}(0)} |E(y)|\right), \qquad \sum_{e\in G(E,x)} p_e(x)^2 = \Omega\left(\sum\nolimits_{e\in E(x)} p_e(x)^2\right) \tag{8}$$
$$\text{and} \quad \forall e \in G(E,x) : p_e(x) = \Theta(\pi(E)).$$

The elements inside $G(E,x)$ are called *typical* arcs. Number $\tau(E)$ is called the *speciality* of the equivalency class (as well, as of any arc in the class). We also define the *typical flow* through E as $\mu(E) = \pi(E)\max_{x\in f^{-1}(1)} |G(E,x)|$. It is straight-forward to check that

$$\forall x \in f^{-1}(1) : \mu(E) = O(p_E(x)). \tag{9}$$

Theorem 15. *If the flow is almost symmetric, the learning graph can be weighted so that its complexity becomes* $O\left(\sum_{E\in\mathcal{E}} \mu(E)\sqrt{\tau(E)}\right)$.

Proof. For each arc e in an equivalency class E, we assign weight $w_e = \pi(E)/\sqrt{\tau(E)}$. Let us calculate the complexity. For each $y \in f^{-1}(0)$, we have the following negative complexity

$$\sum_{E\in\mathcal{E}} w_E|E(y)| = \sum_{E\in\mathcal{E}} \frac{\pi(E)}{\sqrt{\tau(E)}}|E(y)| = O\left(\sum_{E\in\mathcal{E}} \pi(E)\sqrt{\tau(E)} \max_{x\in f^{-1}(1)} |G(E,x)|\right).$$

For a positive input $x \in f^{-1}(1)$, we have

$$\sum_{E\in\mathcal{E}} \frac{1}{w_E} \sum_{e\in E(x)} p_e(x)^2 = O\left(\sum_{E\in\mathcal{E}} \frac{\sqrt{\tau(E)}}{\pi(E)}|G(E,x)|\pi(E)^2\right) = O\left(\sum_{E\in\mathcal{E}} \mu(E)\sqrt{\tau(E)}\right).$$

By combining both estimates, we get the statement of the theorem. □

For each step i, define $T_i = \max_{E\in\mathcal{E}_i} \tau(E)$. Then Theorem 15 together with (9) and the observation that the total flow through all arcs on any step is at most 1, implies the following

Corollary 16. *If the flow is almost symmetric, the learning graph can be weighted so that its complexity becomes* $O\left(\sum_i \sqrt{T_i}\right)$ *where the sum is over all steps.*

4.5 Previous Algorithm for k-distinctness

As an example of application of Corollary 16, we briefly describe a variant of a learning graph for the k-distinctness problem. It is a direct analog of an algorithm from [Amb07] using learning graphs and a straightforward generalization of the learning graph for element distinctness from [Bel11b].

To define equivalencies between arcs, we use θ_2 and ϑ_2 from Section 4.2. The learning graph consists of loading $r + k$ elements without any restrictions (as imposed by θ_2), where r is some parameter to be specified later. We refer to the first r steps as to the *first stage*, and to the last k steps as to the *second stage*.

Clearly, all arcs of the same step are equivalent. Consider strong equivalency. Let x be a positive input and let M be a subset of k equal elements in it. We use M as the set of marked elements to define ϑ_2. Then, the strong equivalence class of an arc is determined by the number

of elements in its origin, the number of marked elements among them, and whether the element being loaded is marked.

The flow is organized as follows. On the first stage, only arcs without marked elements are used. On the second stage, only arcs loading marked elements are used. Thus, on each step only one strong equivalency class is used, and the flow among all arcs in it is equal.

It is easy to check this is a valid flow for k-distinctness and it is symmetric. Let us calculate the specialities. The first r steps have speciality $O(1)$. The speciality of the i-th step of the second stage is $O(n^i/r^{i-1})$. This is because the fraction of $(r+i-1)$-subsets of $[n]$ containing $i-1$ marked elements is $\Theta(r^{i-1}/n^{i-1})$; and $k-i+1$ arc only, out of $\Theta(n)$ originating in such vertex, is used by the flow. Hence, by Corollary 16, the complexity of the learning graph is $O\left(r+\sqrt{n^k/r^{k-1}}\right)$ that is optimized when $r=n^{k/(k+1)}$ and the optimal value is $O\left(n^{k/(k+1)}\right)$.

5 Algorithm for k-distinctness

The purpose of this section is to prove Theorem 1. In Section 5.1, we give some intuition behind the learning graph. In Section 5.2, we describe the learning graph, or, more precisely, define valid specifications for each step, as described in Section 4.3. In Section 5.3, we define the flow, and give preliminary estimates of the complexity. Finally, in Section 5.4, we prove the flow defined in Section 5.3 is almost symmetric and prove the estimates therein are correct.

5.1 Intuition behind the algorithm

There is another way to analyze the complexity of the learning graph in Section 4.5.

Lemma 17. *Assume convention (6) on the input. The expected number of t-subtuples in an r-subset of $A_{\geq 1}$, chosen uniformly at random, is $\Theta(r^t/n^{t-1})$.*

Proof. Let S be the random subset. Denote $n'=|A_{\geq 1}|$. The probability a fixed subset of t equal elements from A_s forms a t-subtuple in S is $\binom{n'-s}{r-t}/\binom{n'}{r}=\Theta(r^t/n^t)$. The number of such subsets is $\sum_s \ell_s\binom{s}{t}=\Theta(n)$. Hence, by linearity of expectation, the expected number is $\Theta(r^t/n^{t-1})$. □

Consider the following informal argument. Let M be the set of marked elements as in Section 4.5. Before the last step, the flow only goes through vertices S having $|S\cap M|=k-1$. Fix a vertex S and let $M'=M\cap S$. One may say, M' as a $(k-1)$-subtuple, is hidden among other $(k-1)$-subtuples of S. The expected number of such is $\Theta(r^{k-1}/n^{k-2})$, total number of $(k-1)$-tuples is $\Theta(n)$, hence, the fraction of the vertices used by the flow on this step is $\Theta(r^{k-1}/n^{k-1})$. Thus, the speciality of the arc loading the missing marked element is $\Theta(n^k/r^{k-1})$ that equals the estimate in Section 4.5.

As such, this is just a more difficult and less strict analysis of the learning graph. But one can see that the speciality of the last steps depends on the number of t-subtuples in the vertices. We cannot get a large quantity of them by loading elements blindly without restrictions, but it is quite possible, we can deliberately enrich vertices of the learning graph in large subtuples by gradually filtering out vertices containing a small number of them.

5.2 Description of the Learning graph

We would like to apply Corollary 16, hence, it is enough to give valid specifications for each step. We do this using a pseudo-code notation in Algorithm 1.

Here $r_1,\ldots,r_{k-1}$ are some parameters with $r_{i+1}=o(r_i)$, $r_1=o(n)$ and $r_{k-1}=\omega(1)$ to be specified later. Also, it will be convenient to denote $r_0=n$. The commands of the algorithm

Algorithm 1 Learning graph for the k-distinctness problem

1: **for** $j \leftarrow 1$ **to** r_1 **do**
2: Load an element
3: **end for**
4: Declare as dead-ends vertices having more than $c_t r_1^t / n^{t-1}$ t-subtuples for any $t = 2, \ldots, k-1$
5: **for** $i \leftarrow 2$ **to** $k-1$ **do**
6: **for** $j \leftarrow 1$ **to** r_i **do**
7: **for** $l \leftarrow 1$ **to** i **do**
8: Load an element of level l
9: **end for**
10: **end for**
11: **end for**
12: Load an element // The last element is no subject to any constraints

define the specifications as follows. The loop in lines 1—3 says there is no constraint on the first r_1 steps. Line 4 introduces the *original specifications*. Here, $c_t > 0$ are some constants we specify later.

The loop in Lines 5—11 describes how the specifications change with each step. Assume a step, described on Line 8, loads an element of *level* l. Then, a valid specification (b_t) before the step is transformed into a valid specification (b'_t) after the step as follows

$$b'_t = \begin{cases} b_t + 1, & t = l; \\ b_t - 1, & t = l-1; \\ b_t, & \text{otherwise.} \end{cases}$$

In other words, if there is an arc between vertices of specifications (b_t) and (b'_t) and it load v then there exists an $(l-1)$-subtuple Q of S such that $Q \cup \{v\}$ is an l-subtuple of $S \cup \{v\}$. In fact, only such arcs will be used by the flow, as it is described in more detail in Section 5.3.

Hence, for each specification in Lines 5—12, it is possible to trace it back to its original specification. For example, if (b_t) is a specification of the vertex after step in Line 8 with the values of the loop counters i, j and l, the original specification is given by $(\tilde{b}_t) - (\delta_t^l)$, where

$$\tilde{b}_t = \begin{cases} b_t - r_t, & 2 \le t < i; \\ b_t - j + 1, & t = i; \\ b_t, & \text{otherwise;} \end{cases} \quad \text{and} \quad \delta_t^l = \begin{cases} 1, & t = l; \\ 0, & \text{otherwise.} \end{cases}$$

Moreover, the use of the arcs in the flow, as described in the previous paragraph, implies the flow through all vertices having some fixed original specification is the same for all steps.

Finally, the step on Line 12 loads the last element, and there is no need for the dead-end conditions, because after the last step all vertices have no arcs leaving them.

Naming convention We use the following convention to name the steps of the learning graph. The step on Line 2 is referred as the j-th step of the first stage. The step on Line 8 is referred using triple (i, j, l), except for the case $i = k-1$ and $j = r_{k-1}$. The latter together with the step on line Line 12 is referred as the steps $1, 2, \ldots, k$ of the *last stage*. The steps of the form $(i, \cdot, \cdot)$ are called the *i-th stage*. Altogether, all steps of the form $(\cdot, \cdot, \cdot)$ are called the *preparatory phase*.

5.3 Flow

We two possible ways to define a flow. The first one is to set the flow through the arcs on each step so that the flow through all vertices on each step is the same. We believe this can be done, but we lack techniques to deal with this kind of arguments.

Instead of that, we select the second way. For each vertex, we divide the flow evenly among all possible arcs. Because of this, the ratio of the maximal and the minimal flow accumulates with each step, and at the end it is quite large. We avoid this complication by applying the concentration results stating that for large n's almost all flow will be concentrated on some typical subset of arcs and will be distributed almost evenly on it.

First Stage For the first stage, we use θ_2- and ϑ_2-based equivalencies, akin to the first stage of the flow in Section 4.5. Consider the uniform flow, i.e., such that distributes all the in-coming flow among all out-going arcs, leading to an element of $A_{\geq 1}$, equally. Clearly, it is symmetric, and the flow through any vertex $S \subseteq A_{\geq 1}$ after the first stage is $\binom{|A_{\geq 1}|}{r_1}^{-1}$. The speciality of each step in this flow is $O(1)$ because of Corollary 13.

But this flow is non-zero for vertices declared as dead-ends in Line 4. We fix this by applying Lemma 8. We have to choose $c_t > 0$ so that, with probability, say, $1/2$, an uniformly picked subset of size r_1 satisfies a valid specification. And it is possible to do so due to Lemma 17 and Markov's inequality.

After performing the conditioning, the complexity of the flow in the first stage increases by at most a constant factor (that can be ignored), and all non-dead-end vertices have the same flow through them, we denote p_o.

Preliminary Estimates For the remaining stages, we use θ_1 and ϑ_3 to define equivalences between arcs. Here we informally analyze the flow for Lines 5—12 of Algorithm 1, assuming there is flow p_o through all non-dead-end vertices after Line 4. The formal analysis is done in Section 5.4.

Roughly speaking, the flow is organized as follows. On step (i, j, l), an element, not in M, belonging to level l is loaded. On any step of the last stage, an element of M is loaded. Let us estimate the complexity of the learning graph. Assume for the moment the flow is almost symmetric.

Approximately n arcs are leaving a vertex on each step. Let (i, j, l) be a step of the preparatory phase and assume $l > 1$. An element of level l is loaded, and there are $\Omega(r_{l-1})$ $(l-1)$-subtuples in the vertex that can be extended. Hence, $\Omega(r_{l-1})$ arcs leaving the vertex can be used by the flow. This makes the speciality of the step equal to $O(n/r_{l-1})$. This is true for $l = 1$ as well, because of the convention $r_0 = n$.

Now turn to the last stage. Let us calculate the speciality of a vertex used by the flow on step $j > 1$ of the last stage. Let V_0 be the vertices contained in $A_{\geq 1}$ having a valid specification, and V_M be the vertices of $A_{\geq 1} \cup M$ that can be used by the flow. Define relation φ, where $S_0 \in V_0$ is in relation with $S_M \in V_M$ if S_M can be obtained from S_0 by removing one of its $(j-1)$-subtuples and adding $j-1$ elements from M instead. Each S_0 has $\Omega(r_{j-1})$ images and each S_M has $O(n)$ preimages. Hence, $|V_0|/|V_M| = O(n/r_{j-1})$. Because only $O(1)$, out of $\Theta(n)$ arcs leaving a vertex from V_M, can be used by the flow, we have the speciality of step j of the last stage equal to $O(n^2/r_{j-1})$. This also is true for $j = 1$. All this is summarized in Table 1.

If we could apply Corollary 16, we would get the complexity

$$O\left(r_1 + r_2\sqrt{n/r_1} + r_3\sqrt{n/r_2} + \cdots + r_{k-1}\sqrt{n/r_{k-2}} + n/\sqrt{r_{k-1}}\right).$$

Step	First stage	Preparatory, $(\cdot,\cdot,l)$	Last stage, j-th
Speciality	1	n/r_{l-1}	n^2/r_{j-1}
Number	r_1	r_l	1

Table 1: Parameters (up to a constant factor) of the stages of the learning graph for the k-distinctness problem.

Denote $\rho_i = \log_n r_i$ and assume all the addends are equal. Then

$$\frac{1}{2} + \rho_i - \frac{\rho_{i-1}}{2} = \frac{1}{2} + \rho_{i+1} - \frac{\rho_i}{2}, \qquad i = 1, \ldots, k-1$$

where we assume $\rho_0 = 1$ and $\rho_k = 1/2$. It is equivalent to $\rho_i - \rho_{i+1} = (\rho_{i-1} - \rho_i)/2$. Hence,

$$1/2 = \rho_0 - \rho_k = (2^k - 1)(\rho_{k-1} - \rho_k).$$

Thus, the optimal choice of ρ_1 is $1 - 2^{k-2}/(2^k - 1)$.

We use these calculations to make our choice of $r_i = n^{\rho_i}$. It remains to strictly define the flow, prove it is almost symmetric and the estimates in Table 1 are correct. Before doing so, we combine some estimates on the values of r_i's in the following

Proposition 18. *We have $\sqrt{r_1} r_2 = o(n)$ and $\sqrt{r_1} = o(r_i)$ for any i. Also, any valid specification on stage i, has $\Theta(r_j)$ j-subtuples for $j < i$.*

Proof. The first equation follows from $\rho_1 < 3/4$ and $\rho_2 < 5/8$. The second inequality follows from $\rho_i \geq 1/2$ for all i's.

Due to Line 4 of the algorithm, after the first stage, any valid specification has $O(r_1^i / n^{i-1})$ i-subtuples. For $i > 1$, it is $o(\sqrt{n}) = o(r_j)$ for any j. Hence, after the first stage there are $\Theta(r_1)$ 1-subtuples, and this number does not substantially change after that. Similarly, if one doesn't take into account the ± 1-fluctuations, the number of j-subtuples is changed only on stage j, when r_j j-subtuples are added. □

Values of the flow Let us describe how the flow is defined. Fix some stage i. A vertex before a step of the form $(i, \cdot, 1)$ is called a *key vertex*. Consider a key vertex S with type $(b_{t,s})$. The flow from S is distributed evenly among all *succeeding* key vertices, where S' is a succeeding key vertex for S iff $S' \setminus S$ is a subset of equal elements having a value different from any element of S. The number of succeding key vertices for S is

$$N(S) = D_i\left(\sum\nolimits_{t=1}^{k-1} b_{t,1}, \ldots, \sum\nolimits_{t=1}^{k-1} b_{t,k-1}\right)$$

where

$$D_i(z_1, \ldots, z_{k-1}) = \sum_{s=i}^{k-1} (\ell_s - z_s) \binom{s}{i}$$

is the number of possible i-subtuples to extend the vertex with, when z_s s-tuples have already been used.

More precisely, let e be an arc of step (i, j, l) originating in a non-dead-end vertex S' and loading an element v. Then the flow through this arc is defined using the values of the flow through key vertices before step $(i, j, 1)$ as follows:

$$p_e = \begin{cases} \binom{s}{i}\binom{s}{l}^{-1} \frac{p_{S' \setminus Q}}{lN(S' \setminus Q)}, & |Q| = l \text{ and } s \geq i, \text{ where } Q = \{\iota \in S' \cup \{v\} \mid x_\iota = x_v\} \\ & \text{and } s \text{ is such that } Q \text{ is contained in } A_s; \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

If the first case in (10) holds, vertex $S' \setminus Q$ is called the key vertex *preceeding* arc e. Note that it is uniquely defined.

5.4 Analysis of the flow

Typical vertices The point of this section is to prove the flow defined in Section 5.3 is almost symmetric. For this, we should identify the set of typical arcs. Before doing so, we define *typical vertices*.

Let $\beta = (b_t)$ be a valid specification of the preparatory phase. Select any $t \in [k-1]$ and let X_t be the collection of all subsets of $A_{\geq 1}$ consisting of b_t t-subtuples. In other words, elements of X_t satisfy specification $(0, \dots, 0, b_t, 0 \dots, 0)$. Denote $e_{t,s} = \mathrm{E}_{S \in X_t}[b_{t,s}(S)]$, where $b_{t,s}(S) = |S \cap A_s|/t$ is the element of $\tilde{\beta}(S)$. Denote $\varepsilon_\beta = (e_{t,s})$.

A type $(b_{t,s})$, consistent with β, is called *typical* if it is inside $\mathcal{B}(\varepsilon_\beta, C\sqrt{r_1})$, where C is a constant to be specified later, i.e., if for all t and s holds $|b_{t,s} - e_{t,s}| \leq C\sqrt{r_1}$. A typical vertex is one of a typical type. Let us state some properties of the typical vertices.

Lemma 19. *Let $(b_{t,s})$ be the type of any typical vertex on the i-th stage. Then, for all $t < i$ and $s \geq t$, we have $b_{t,s} = \Omega(r_t)$.*

Proof. Since $\sqrt{r_1} = o(r_t)$, it is enough to show that $e_{t,s} = \Omega(r_t)$. Let S be an element of X_t. Arbitrarily order its subtuples: $S = \{s_1, \dots, s_{b_t}\}$. Clearly, the expectation is the same for ordered and unordered lists of subtuples, so let us consider the former.

By linearity of expectation, $e_{t,s} = b_t \Pr[s_1 \subseteq A_s]$. The number of sequences having s_1 in A_s is $\ell_s \binom{s}{t}$ times the number of ways to pick the remaining $b_t - 1$ t-subtuples out of $A_{\geq 1}$ where one s-tuple cannot be used. By (6) and Lemma 12, these numbers are equal for different $s \geq t$, up to a constant factor. Hence, the probability is $\Omega(1)$, and since $b_t = \Omega(r_t)$, we have $e_{t,s} = \Omega(r_t)$. □

Lemma 20. *For any valid specification β of the preparatory phase and for any $\lambda > C\sqrt{r_1}$,*

$$\Pr[\|\tilde{\beta}(S) - \varepsilon_\beta\|_\infty > \lambda] < e^{-\Omega(\lambda^2/r_1)}, \tag{11}$$

where the probability is uniform over all subsets S of $A_{\geq 1}$ satisfying β.

We derive the lemma from the following two pure technical results

Proposition 21. *Let H be the disjoint union of $(H_t)_{t \in [k]}$ where each H_t is a rectangular array of dots, having ℓ_t columns and m_t rows. Let X be the set of all r-element subsets of H where no subset has more than 1 dot from any column of any H_t. Occupy X with the uniform probability distribution and let $h_t \colon X \ni S \mapsto |S \cap H_t|$. Assume $k = O(1)$, $\ell_t = \Theta(n)$ and $r = o(n)$. Then:*

$$\Pr\left[|h_t - \mathrm{E}[h_t]| > \lambda\right] < e^{-\Omega(\lambda^2/r)}, \tag{12}$$

for any $\lambda > 0$.

Proof. This is a standard application of Azuma's inequality (Theorem 5). Suppose, we sort the elements of each $S \in X$ in any order: $S = \{s_1, \dots, s_r\}$. Clearly, the probability equals for unsorted and for sorted lists. We use both interchangeably in the proof.

Let D_i be the Doob martingale with respect to this sequence. We have to prove that $|D_i - D_{i-1}| = O(1)$, i.e., the expectation of h_t does not change much when a new element of the sequence is revealed. To simplify notations, we prove only $|D_1 - D_0| = O(1)$, the remaining inequalities being similar.

For the proof, we define two other classes of probability distributions, all being uniform:

- Y_i: r-subsets of $H \setminus Q$, where Q is a fixed column of H_i;
- Z_i: $(r-1)$-subsets of $H \setminus Q$.

For the martingale, it is enough to prove that, for all i:

$$|\mathrm{E}[h_t \mid X] - (\mathrm{E}[h_t \mid Z_i] + \delta_{i,t})| = O(1),$$

where $\delta_{i,t}$ is the Kronecker delta. We have

$$\mathrm{E}[h_t \mid X] = \Pr[Q \cap S \neq \emptyset]\,(\mathrm{E}[h_t \mid Z_i] + \delta_{i,t}) + \Pr[Q \cap S = \emptyset]\,\mathrm{E}[h_t \mid Y_i]. \tag{13}$$

Hence, it is enough to prove that

$$|\,\mathrm{E}[h_t \mid Z_i] - \mathrm{E}[h_t \mid Y_i]| = O(1). \tag{14}$$

Denote $\ell'_j = \ell_j - \delta_{i,j}$; and let K_h and K'_h be the number of elements S of Z_i and Y_i, respectively, having $h_t(S) = h$. Note that $K'_h = \gamma_h K_{h-1}$, where $h \in [r]$ and $\gamma_h = ((\ell'_t - h + 1)m_t)/h$. Thus

$$\mathrm{E}[h_t \mid Y_i] \leq \frac{\sum_{h=1}^{r} hK'_h}{\sum_{h=1}^{r} K'_h} = \frac{\sum_{h=1}^{r} h\gamma_h K_{h-1}}{\sum_{h=1}^{r} \gamma_h K_{h-1}} \leq \frac{\sum_{h=1}^{r} hK_{h-1}}{\sum_{h=1}^{r} K_{h-1}} = 1 + \mathrm{E}[h_t \mid Z_i],$$

where the second inequality holds because γ_h monotonely decreases. Thus, by linearity of expectation, $\mathrm{E}[h_t \mid Y_i] \geq \mathrm{E}[h_t \mid Z_i] - k + 1$, for all i, thus proving (14). An application of Azuma's inequality finishes the proof of the proposition. □

Lemma 22. *Assume $i, j \in [k-1]$, at least one of them is not 1, and $m = O(1)$ is an integer. Let μ be a probability distribution on $\mathbb{R}^m$ such that $\mu(\mathbb{R}^m \setminus \mathcal{B}(\lambda)) \leq e^{-C_1\lambda^2/r_i}$ for any $\lambda \geq C_2\sqrt{r_i}$. Assume w is a positive real function, defined on the support of μ, such that $w(x)/w(y) \leq e^{C_3 r_j \|x-y\|_\infty/n}$ for any x, y. Here C_1, C_2, C_3 are some positive constants. Then there exists a constant $C > 0$ such that*

$$\int_{\mathbb{R}^m \setminus \mathcal{B}(\lambda)} w(x)\,d\mu(x) = e^{-\Omega(\lambda^2/r_1)} \int_{\mathbb{R}^m} w(x)\,d\mu(x)$$

for any $\lambda \geq C\sqrt{r_1}$.

Proof. In the proof, C with a subindex denotes a positive constant that may depend on other C's.

Let ν be a measure on $]C_2\sqrt{r_i}, +\infty[$ such that $\nu(]\lambda, +\infty[) = \mu(\mathbb{R}^m \setminus \mathcal{B}(\lambda))$. The worst case, when the mass of μ is as far from the origin as possible, is when $\nu(]\lambda, +\infty[) = e^{-C_1\lambda^2/r_i}$. In this case, $\nu(t) = g(t)\,dt$ with $g(t) = \frac{2C_1 t}{r_i} e^{-C_1 t^2/r_i}$.

There exists a point y in the support of μ such that $\|y\|_\infty \leq C_2\sqrt{r_i}$. Without loss of generality, we may assume $w(y) = 1$. Consider

$$D = \int_{\mathcal{B}(C_2\sqrt{r_i})} w(x)\,d\mu(x) \geq \mu(\mathcal{B}(C_2\sqrt{r_i})) \inf_{x \in \mathcal{B}(C_2\sqrt{r_i})} w(x) \geq (1 - e^{-C_1 C_2^2}) e^{-2C_2 C_3 r_j \sqrt{r_i}/n}.$$

Then, for any $\lambda \geq C_2\sqrt{r_i}$,

$$\begin{aligned} \frac{1}{D} \int_{\mathbb{R}^m \setminus \mathcal{B}(\lambda)} w(x)\,d\mu(x) &\leq \frac{1}{D} \int_{\lambda}^{+\infty} e^{C_3 r_j (t + C_2\sqrt{r_i})/n} g(t)\,dt \\ &= \int_{\lambda}^{+\infty} \frac{C_4 t}{r_i} \exp\left(C_5 \frac{r_j\sqrt{r_i}}{n} + C_3 \frac{r_j t}{n} - C_1 \frac{t^2}{r_i} \right) dt. \end{aligned} \tag{15}$$

Denote $\tilde{t} = t/\sqrt{r_1}$. Then the expression in the last exponent can be rewritten as

$$C_5\frac{r_j\sqrt{r_i}}{n}+C_3\frac{r_jt}{n}-C_1\frac{t^2}{r_i}=C_5\frac{r_j\sqrt{r_i}}{n}+C_3\frac{r_j\sqrt{r_1}}{n}\tilde{t}-C_1\frac{r_1}{r_i}\tilde{t}^2=\frac{r_1}{r_i}\left(C_5\frac{r_i^{3/2}r_j}{nr_1}+C_3\frac{r_ir_j}{n\sqrt{r_1}}\tilde{t}-C_1\tilde{t}^2\right).$$

The coefficients of the last polynomial can be estimated as follows:

$$\frac{r_i^{3/2}r_j}{nr_1}\le\frac{\sqrt{r_1}r_2}{n}=O(1)\quad\text{and}\quad\frac{r_ir_j}{n\sqrt{r_1}}\le\frac{\sqrt{r_1}r_2}{n}=O(1),$$

by Proposition 18. This means there exist $C_6, C_7 > 0$ such that, for any $\lambda \ge C_6\sqrt{r_1}$, the right hand side of (15) is at most

$$\int_\lambda^{+\infty}\frac{C_4t}{r_i}e^{-C_7t^2/r_i}\,dt=\frac{C_4}{2C_7}e^{-C_7\lambda^2/r_i}=e^{-\Omega(\lambda^2/r_1)},$$

if $\lambda \ge C\sqrt{r_1}$ for C large enough. □

Proof of Lemma 20. Let S be the random subset. Denote the set of t-subtuples of S by S_t. We apply Proposition 21 to S_t with $k-1$ H_t's given by $m_s = \binom{s}{t}$ and $r = b_t = O(r_t)$. Thus, if S_t had uniform distribution, Eq. (12) would hold, that would imply (11), because there are $O(1)$ possible choices of s and t.

But in S, S_t does not have uniform distribution. Each S_t is assigned weight w_{S_t} that is proportional to the number of subsets of $A'_{\ge 1}$ having specification (b'_t), where $A'_{\ge 1}$ has $\ell_j - h_j(S_t)$ j-tuples in the notations of Proposition 21, $b'_t = 0$ and $b'_j = b_j$ for $j \ne t$.

Take two S_t and S'_t, and assume $\|\tilde{\beta}(S_t) - \tilde{\beta}(S'_t)\|_\infty \le d$. We apply Lemma 12. There are two cases. If $t > 1$, the lemma implies $w_{S_t}/w_{S'_t} = e^{O(dr_1/n)}$. If $t = 1$ then $w_{S_t}/w_{S'_t} = e^{O(dr_2/n)}$. Anyway, either r in (12), or r in the estimation of $w_{S_t}/w_{S'_t}$ is not r_1, and, hence, Lemma 22 applies, finishing the proof of the lemma. □

Divergence in the flow After we have defined typical vertices, we are going to show that almost all flow goes through them. But before we do so, we show get an estimate of the divergence of the flow in the distance of the types.

Lemma 23. *Suppose two key vertices S and S' of the same specification satisfy $\|\tilde{\beta}(S)-\tilde{\beta}(S')\|_\infty \le d$. Then $p_S/p_{S'} = e^{O(dr_2/n)}$.*

Proof. Denote $(b_t) = \beta(S) = \beta(S')$, and $b = \sum_t b_t$. Let the original specification of the vertices be (c_t), and $c = \sum_t c_t$.

Fix some order of subtuples in S and S' so that the sizes of the i-th subtuple in S and S' are equal for any i. Denote this common value by $\nu(i)$. Also, let $\delta_s(i)$ be 1 if the i-th subtuple of S is contained in A_s, and 0 otherwise. Define δ' for S' similarly.

Let Σ be the set of possible sequences of how the subtuples could have been loaded. I.e., for each element of Σ, the first c subtuples have specification (c_t), and the remaining $b-c$ subtuples are in a non-decreasing order with respect to their sizes. Moreover, the order of the first c subtuples is irrelevant, i.e., no two distinct elements of Σ have their tails of last $b-c$ subtuples equal. In these notations,

$$p_S = p_o\sum_{\sigma\in\Sigma}\prod_{j=c+1}^{b}D_{\nu(\sigma j)}\left(\sum\nolimits_{i=1}^{j-1}\delta_1(\sigma i),\dots,\sum\nolimits_{i=1}^{j-1}\delta_{k-1}(\sigma i)\right)^{-1},\tag{16}$$

where p_o and D are defined in Section 5.3. A similar expression works for S' as well, if one replaces δ by δ'.

Since the distance between the types of S and S' is d, one can define the order of the subtuples so that $\delta_s(i) = \delta'_s(i)$ for all s's and all, except at most $O(d)$, i's. In this case, for all σ, s and j:

$$\left|\sum_{i=1}^{j-1} \delta_s(\sigma i) - \sum_{i=1}^{j-1} \delta'_s(\sigma i)\right| = O(d).$$

Then the ratio of the D's in (16) is at most $1 + O(d/n)$. Since there are $O(r_2)$ multipliers, the ratio of the products in (16) for the same σ is at most

$$\left(1 + O\left(\frac{d}{n}\right)\right)^{O(r_2)} = e^{O(r_2 d/n)}.$$

And the same estimate holds for the ratio of sums. □

Finishing the proof Finally, we are about to prove that the statement of Corollary 16 applies for the flow. Call an arc on preparatory or last stage typical if the preceding key vertex is typical and the flow through the arc is non-zero. We show that conditions of (8) hold for a fixed value of $x \in f^{-1}(1)$. Then the existence of a strong equivalence between any two positive inputs, as in Section 4.3, implies that (8) holds for all positive inputs x with the values of $\pi(E)$ and $\tau(E)$ independent on x.

Note that the factor $\binom{s}{i} / \left(l\binom{s}{l} N(S' \setminus Q)\right)$ from (10) is equal for all arcs from a fixed equivalence class of the preparatory stage, up to a constant factor. Thus, the main concern is about $p_{S' \setminus Q}$, that is flow through a key vertex. The same is true for the last stage as well.

We start with the third condition of (8). It is enough to show the flow differs by at most a constant factor for any two typical key vertices of the same specification. The latter follows from the fact the types of typical vertices are at distance $O(\sqrt{r_1})$, and, hence, by Lemma 23, the ratio of the flow is $e^{O(r_2\sqrt{r_1}/n)} = O(1)$.

We continue with the second condition. Again, it is enough to show its analog for key vertices. The latter is a direct consequence of Lemma 22 applied to the estimates of Lemmas 20 and 23. The constant C in the definition of the typical vertex is that from the last application of Lemma 22.

Finally, let us calculate the speciality of each step. Because of Corollary 13, we may calculate the speciality as if the set of input variables is reduced to $A_{\geq 1} \cup M$. Consider a typical arc e of step (i, j, l). Let S be the origin of e. Note that S is typical (this is a consequence of (14)). If $l = 1$ then we can add any element from an untouched tuple of $A_{\geq i}$. Due to (6), there are $\Omega(n)$ such elements.

Now assume $l > 1$. By the construction of the flow, the non-zero flow is through the arcs that load the l-th element for a subtuple from $A_{\geq i}$. By Lemma 19, in S, there are $\Omega(r_{l-1})$ $(l-1)$-subtuples from $A_{\geq i}$. In both cases, there are $\Omega(r_{l-1})$ arcs leaving S that are used by the flow. By Lemma 20, an $\Omega(1)$ fraction of all vertices is typical, hence, the speciality of an equivalence class of step (i, j, l) is $O(n/r_{l-1})$.

For the last stage, the same argument as in Section 5.3 applies, concluded by a fact an $\Omega(1)$ fraction of all vertices before the last stage is typical.

Thus, the flow is almost symmetric and estimates from Table 1 are correct. This proves Theorem 1.

6 Summary

An algorithm for k-distinctness problem is constructed in the paper, given the prior knowledge of the structure of the input. Is it true, the problem can be solved in the same number of queries without any prior knowledge?

Also, the algorithm in Section 4.5 can be used for any function such that its 1-certificate complexity is bounded by k. For the algorithm in Section 5, it is not clear. So, another (stronger) open problem is as follows. Is it true, any function with 1-certificate complexity bounded by constant can be calculated in $o(n^{3/4})$ quantum queries? If so, this would be a far-reaching generalization of the quantum algorithm in [CK11].

Acknowledgements

AB would like to thank Andris Ambainis for useful discussions. AB has been supported by the European Social Fund within the project "Support for Doctoral Studies at University of Latvia".

References

[Amb02] A. Ambainis. Quantum lower bounds by quantum arguments. *J. Comput. Syst. Sci.*, 64:750–767, 2002. Earlier version in STOC'00.

[Amb05] A. Ambainis. Quantum lower bounds for collision and element distinctness with small range. *Theory of Computing*, 1:37–46, 2005.

[Amb07] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37:210–239, 2007.

[AS04] S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004.

[AS08] N. Alon and J.H. Spencer. *The probabilistic method*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008.

[BdW02] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theor. Comput. Sci.*, 288:21–43, October 2002.

[Bel11a] A. Belovs. Span-program-based quantum algorithm for the rank problem. Technical Report arXiv:1103.0842, arXiv, 2011.

[Bel11b] A. Belovs. Span programs for functions with constant-sized 1-certificates. Technical Report arXiv:1105.4024, arXiv, 2011.

[CK11] A. Childs and R. Kothari. Quantum query complexity of minor-closed graph properties. In *Proc. 28th STACS*, pages 661–672, 2011.

[HLŠ07] P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proc. 39th ACM STOC*, pages 526–535, 2007.

[LMR+11] T. Lee, R. Mittal, B. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of the state conversion problem. In *Proc. 52nd IEEE FOCS*, 2011.

[Rei11] B. Reichardt. Reflections for quantum query algorithms. In *Proc. 22nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 560–569, 2011.

[RŠ] B. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas. In *Proc. 40th ACM STOC*, pages 103—112.