Span Programs for Functions with Constant-Sized 1-certificates

Aleksandrs Belovs*

The abstract is based on [5] but also includes the subsequent developments from [6].

Quantum query complexity measures the complexity of a quantum algorithm by the number of queries to the input it uses. A number of strong lower bound methods for quantum query complexity have been developed. One of them is the adversary bound, first time formulated by Ambainis [1] and later generalized by Høyer *et al.* in [11]. The latter, (general) adversary bound was shown to be tight by Reichardt *et al.* [16, 12].

The adversary bound admits a formulation as a semi-definite program (SDP), hence, any feasible solution for its dual is an upper bound on quantum query complexity, and this bound is also tight! For functions with Boolean input, the dual SDP can be represented as a span program. In principle, this means that span programs give an optimal quantum algorithm for any Boolean function. However, their actual applications have been mostly limited to formula evaluation [17]. We show how to use them for a wider class of functions.

Quantum algorithms for functions with bounded 1-certificate complexity form the second large class of quantum algorithms, after the hidden subgroup problem. It includes OR function, solvable by the Grover algorithm [10], the element distinctness, the triangle and other problems.

One can distinguish two main design paradigms for quantum algorithms for such functions. The first one includes application of the Grover search and its close relative — quantum amplitude amplification [8]. This paradigm resulted in the algorithm for the collision problem with complexity $O(n^{1/3})$ by Brassard *et al.* [7], the $O(n^{3/4})$-algorithm for the element distinctness problem by Buhrman *et al.* [9], the $O(n^{10/7})$-algorithm for the triangle problem by Magniez *et al.* [14], and others.

The second paradigm is based on quantum walks on the Johnson graph. It was pioneered by Ambainis with his $O(n^{2/3})$-algorithm for the element distinctness problem, and a more general $O(n^{k/(k+1)})$-algorithm for the k-distinctness problem [2]. The triangle-finding algorithm with complexity $O(n^{13/10})$ by Magniez *et al.* [14] also belongs to this class.

We propose an alternative approach using a computational model of a *learning graph*. It is a directed 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, and $[m]$ is the set of possible values of variables.

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.

Using this computational model, in [5], we obtain an algorithm for the triangle problem with complexity $O(n^{35/27})$. In [6], we build an algorithm for k-distinctness with complexity $o(n^{3/4})$ that

*Faculty of Computing, University of Latvia, Latvia, stiboh@gmail.com. This work has been supported by the European Social Fund within the project "Support for Doctoral Studies at University of Latvia".

requires prior information on how many maximal subsets of equal elements of sizes $1, 2, \ldots, k-1$ are there in the input, with precision $O(\sqrt[4]{n})$. Both these algorithms are the first improvements for those problems in 8 years (since papers [14] and [2]).

Let us describe the definition of a learning graph in more detail, and the main ideas of the algorithms. Let $\mathcal{G}$ be a learning graph. By $\mathcal{G}(x)$, we denote the graph with the same vertices and arcs as $\mathcal{G}$, only the weight of arc e is a real number $w_e = w_e(x)$ that depends on the input x. We use notation $e \in \mathcal{G}(x)$ to denote that e is an arc of $\mathcal{G}(x)$.

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 $\mathcal{G}$ is defined as $\max_{y \in f^{-1}(0)} \mathcal{N}(\mathcal{G}(y))$.

A *flow* p_e on $\mathcal{G}(x)$ for $x \in f^{-1}(1)$ is a flow of value 1, with $\emptyset$ being the only source, and S being a sink iff S contains a 1-certificate for x. The complexity of p_e 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 is $\max_{x \in f^{-1}(1)} \mathcal{P}(\mathcal{G}(x))$.

Papers [5] and [6] feature two alternative ways of reducing a learning graph to a quantum query algorithm: using span programs, and the dual of the adversary bound, respectively. We sketch the idea of the second reduction. Recall the dual SDP for the adversary bound [15]:

$$\underset{k \in \mathbb{N},\, u_{x,j} \in \mathbb{C}^k}{\text{minimize}} \ \max_x \sum_{j \in [n]} \|u_{x,j}\|^2 \quad \text{subject to} \quad \sum_{j:\, x_j \neq y_j} \langle u_{x,j} | u_{y,j} \rangle = 1 \text{ whenever } f(x) \neq f(y).$$

The constraint is satisfied in the following way. Vectors $u_{x,j}$ are constructed so that the sum in the constraint equals the value of the cut between the vertex sets $\{S \mid \forall i \in S : x_i = y_i\}$ and $\{S \mid \exists i \in S : x_i \neq y_i\}$ for the flow on $\mathcal{G}(x)$. The source $\emptyset$ is in the first set and all the sinks are in the second one. This implies the value of the cut equals the value of the flow, i.e., 1. The objective of the SDP equals the complexity of the learning graph.

We divide the learning graph into stages. Each stage consists of parallel chains of arcs (transitions), so that any path from the source to a sink uses exactly one transition of every stage. Each stage has a task to load some specific elements. Also, we divide all transitions into equivalence classes, according to some kind of symmetry. We call the flow symmetric if, in each class, the flow takes only two values, one of them being 0. The flows in this abstract are going to be symmetric.

The speciality of a class is the ratio of the number of transitions in it to the number of ones used by the flow. One can prove, that if the flow is symmetric, the complexity of the learning graph is $O\left(\sum_i L_i \sqrt{T_i}\right)$, where the sum is over stages, L_i is the length of the stage (number of arcs in a transition) and T_i is the maximal speciality.

For example, consider the element distinctness problem. Let a and b be the two equal elements in an input. The learning graph can be as follows: on stage I load a, and on stage II load b. Since we do not know a and b in advance, stage I has to contain all n arcs, and stage II — $n(n-1)$ arcs. All arcs of one stage are in the same equivalence class. This learning graph has complexity $\Theta(n)$, because the speciality of the second stage is $\Theta(n^2)$: only one arc is used by the flow.

The speciality of stage II is a product of two factors. The first one comes in because b is loaded, a specific element out of n possible. This factor is $\Theta(n)$. The second one comes in because the arc originates in a vertex containing a. In the learning graph from the last paragraph, this factor is $\Theta(n)$ as well.

It is not possible to reduce the first factor, but it is possible to reduce the second one. Assume, we prepend the learning graph by stage 0 that loads r elements not from $\{a, b\}$. After that, it proceeds as earlier. After a is loaded, it blends into other r elements of the vertex, and this way, the speciality of stage II is decreased r times.

Let us do some math. The speciality of stage 0 is 1 (as it loads common-or-garden elements) and its length is r. The lengths of stage I and II are both 1, and the specialities are $O(n)$ and

$O(n^2/r)$, respectively. Thus, the complexity of the learning graph is $O(r+n/\sqrt{r})$ that is optimized when $r = n^{2/3}$.

Consider the framework for quantum walk based on set-up, update and checking operations [13]. In this framework, the complexity estimate for the learning graph for element distinctness resembles one for the quantum-walk-based algorithm, where r is the set-up cost, and $n/\sqrt{r}$ is the term featuring the update cost. Similarly, it is possible to get algorithms that involve the checking cost. For instance, consider the triangle problem. Let a, b and c be the vertices of a triangle. The learning graph consists of loading a full subgraph on r vertices (stage I, corresponds to the set-up cost), adding vertices a and b to the subgraph (stage II, update cost), and a distinctness-type subroutine for loading edges ac and bc (stage III, checking cost). Working out the lengths and specialities, we obtain complexity $O(r^2 + \frac{n}{\sqrt{r}}r + \frac{n^{3/2}}{r}r^{2/3})$. It is optimized when $r = n^{3/5}$, the optimal value being $O(n^{13/10})$. This is in a complete correspondence to the algorithm from [14].

Until now we have demonstrated how learning graphs mimic the quantum-walk-based algorithms. Now we are about to show they can be superior. Note that only the second and the third terms in the complexity estimate for the triangle problem have complexity $O(n^{13/10})$, while the first term is just $O(n^{12/10})$. This suggests the resources of the set-up stage are not fully used. And this is because adding vertex b to the subgraph has too high cost due to its length — r edges must be loaded.

This concern can be fixed using the tool that is easy to apply for learning graphs and tremendously difficult for quantum walk. We mean amortization when different computational threads in a subroutine have different complexity. In text-book quantum algorithms, in order to continue with the algorithm, one has to wait until all threads are finished, hence, the complexity of the subroutine is *maximum* of all the threads. For a learning graph, we are interested in its complexity only, that is just a number, and it is easy to show the contribution of a subroutine is the *average* of all its threads. Attaining the average cost for a "variable-time" subroutine has been studied by Ambainis for Grover search [3] and amplitude amplification [4]. The algorithms are rather involved, and we are not aware of any solution for more complicated quantum walks.

Thus, assume stages I and II are performed as before with the exception that not a full subgraph is loaded, but its random subgraph, where each edge is taken, independently, with probability s. Stage III remains as it was. In this case, the average "set-up cost" is sr^2, "update cost" is $sn\sqrt{r}$, and "checking cost" is $n^{3/2}r^{-1/3}s^{-1/2}$. The first two costs decreased by a factor of s, because one has to load less edges, and the "checking cost" increased by a factor of $s^{-1/2}$, because the probability a subgraph on a superset of $\{a, b\}$ contains edge ab is exactly s. One also has to load edge ab (between stages II and III), but this has speciality $O(n^2)$ and length 1, hence, its contribution, $O(n)$, is negligible.

The optimal values of parameters are $r = n^{2/3}$ and $s = n^{-1/27}$. In this case all three "costs" are equal to the complexity of the learning graph $O(n^{35/27})$ that is better than $O(n^{13/10})$.

We briefly describe the algorithm of [6]. Denote by $a_1, \ldots, a_k$ the k equal elements in the input. Similarly to the element distinctness problem, a_i's is the last thing loaded in the vertex of the learning graph. Before that, the vertex is enriched in subsets of equal elements of smaller sizes, starting with 2, and ending with $k-1$. Thus, the speciality of loading $\{a_1, \ldots, a_k\}$ is reduced. For example, when a_k is loaded, $\{a_1, \ldots, a_{k-1}\}$ is hidden among all $(k-1)$-subsets of equal elements in the vertex. The knowledge of the number of ℓ-subsets of equal elements in the input, for $\ell = 1, \ldots, k-1$, is used to assure all negative inputs have approximately the same complexity. This type of learning graph is conceptually different from a quantum walk on the Johnson graph.

To conclude, the learning graph paradigm was proven to be superior to the existing paradigms for function with small 1-certificates, that was demonstrated by algorithms for the triangle and k-distinctness problems. We believe the mentioned algorithms is just a glimpse of what can be done using the dual of the adversary bound.

References

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

[2] Ambainis, A.: *Quantum walk algorithm for element distinctness.* In 45th IEEE Symposium on Foundations of Computer Science, 22–31 (2004)

[3] Ambainis, A.: *Quantum search with variable times.* In 25th International Symposium on Theoretical Aspects of Computer Science, 49–61 (2008)

[4] Ambainis, A.: *Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations.* arXiv:1010.4458v2 (2010)

[5] Belovs, A.: *Span programs for functions with constant-sized 1-certificates.* arXiv:1105.4024v1 (2011)

[6] Belovs, A., Troy, L.: *Quantum Algorithm for k-distinctness with Prior Knowledge on the Input.* arXiv:1108.3022v1 (2011)

[7] Brassard, G., Høyer, P., Tapp, A.: *Quantum algorithm for the collision problem.* In Proc. 3rd LATIN, LNCS **1380**, 163—169, arXiv:quant-ph/9705002 (1998)

[8] Brassard, G., Høyer, P., Mosca, M., Tapp, A.: *Quantum amplitude amplification and estimation.* Quantum Computation and Quantum Information Science, AMS Contemporary Mathematics Series **305**, 53—74, arXiv:quant-ph/0005055 (2002)

[9] Buhrman, H., Durr, C., Heiligman, M., Hoyer, P., Magniez, F., Santha, M., de Wolf, R.: *Quantum algorithms for element distinctness.* SIAM Journal on Computing, **34**, 1324—1330 (2005)

[10] Grover, L.: *A fast quantum mechanical algorithm for database search.* Proc. ACM STOC, 212–219, arXiv:quant-ph/9605043 (1996)

[11] Høyer, P., Lee, T., Špalek, R.: *Negative weights make adversaries stronger.* In Proc. 39th ACM STOC, 526–535 (2007)

[12] Lee, T., Mittal, R., Reichardt, B., Špalek, R., Szegedy, M.: *Quantum query complexity of the state conversion problem.* In Proc. 52nd IEEE FOCS (2011)

[13] Magniez, F., Nayak, A., Roland, J., Santha, M.: *Search via quantum walk.* In Proc. 39th ACM STOC, 575—584, arXiv:quant-ph/0608026v4 (2007)

[14] Magniez, F., Santha, M., Szegedy, M.: *Quantum Algorithms for the Triangle Problem.* In Proc. SODA'05, 1109—1117, arXiv:quant-ph/0310134v3 (2005)

[15] Reichardt, B.: *Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function.* arXiv:0904.2759v1 (2009)

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

[17] Reichardt, B., Špalek, R.: *Span-program-based quantum algorithm for evaluating formulas.* In Proc. 40th ACM STOC, 103—112, arXiv:0710.2630 (2008)