New Developments in Quantum Algorithms

Andris Ambainis [⋆]

Faculty of Computing, University of Latvia, Raina bulv. 19, Riga, LV-1586, Latvia,
andris.ambainis@lu.lv

Abstract. In this talk, we describe two recent developments in quantum algorithms.

The first new development is a quantum algorithm for evaluating a Boolean formula consisting of AND and OR gates of size N in time $O(\sqrt{N})$. This provides quantum speedups for any problem that can be expressed via Boolean formulas. This result can be also extended to *span problems*, a generalization of Boolean formulas. This provides an optimal quantum algorithm for any Boolean function in the black-box query model.

The second new development is a quantum algorithm for solving systems of linear equations. In contrast with traditional algorithms that run in time $O(N^{2.37\ldots})$ where N is the size of the system, the quantum algorithm runs in time $O(\log^c N)$. It outputs a quantum state describing the solution of the system.

1 History of quantum algorithms

1.1 First quantum algorithms

Quantum computing (and, more broadly, quantum information science) is a new area at the boundary of computer science and physics. It studies how to apply quantum mechanics to solve problems in computer science and information processing. The area of quantum computing was shaped by the discoveries of two major quantum algorithms in mid-1990s.

The first of the these two discoveries was Shor's polynomial time quantum algorithm for factoring and discrete logarithms. Factoring and discrete logarithm are very hard number theoretic problems. The difficulty of these problems has been used to design cryptosystems (such as RSA and Diffie-Helman key exchange) for secure data transmission over an insecure network (such as Internet). The security of data transmission is based on the assumption that it is hard to factor (or find discrete logarithm of) large numbers. Until recently, this assumption was not in doubt. Mathematicians had tried to devise an efficient way of factoring large numbers for centuries, with no success.

In 1994, Shor [56] discovered a fast algorithm for factoring large numbers - on a quantum mechanical computer. This shook up the foundations of cryptography.

[⋆] Supported by FP7 Marie Curie Grant PIRG02-GA-2007-224886 and ESF project 1DP/1.1.1.2.0/09/APIA/VIAA/044.

If a quantum mechanical computer is built, today's methods for secure data transmission over the Internet will become insecure.

Another, equally strikingly discovery was made in 1996, by Lov Grover [34]. He invented a quantum algorithm for speeding up exhaustive search problems. Grover's algorithm solves a generic exhaustive search problem with N possible solutions in time $O(\sqrt{N})$. This provides a quadratic speedup for a range of search problems, from problems that are in P classically to NP-complete problems.

Since then, each of the two algorithms has been analyzed in great detail. Shor's algorithm has been generalized to solve a class of algebraic problems that can be abstracted to *Abelian hidden subgroup problem* [39]. Besides factoring and discrete logarithm, the instances of Abelian HSP include cryptanalysis of hidden linear equations [18], solving Pell's equation, principal ideal problem [35] and others.

Grover's algorithm has been generalized to the framework of *amplitude amplification* [21] and extended to solve problems like approximate counting [23, 47] and collision-finding [22].

1.2 Quantum walks and adiabatic algorithms

Later, two new methods for designing quantum algorithms emerged: quantum walks [4, 41, 9, 54, 60] and adiabatic algorithms [31].

Quantum walks are quantum generalizations of classical random walks. They have been used to obtain quantum speedups for a number of problems. The typical setting is as follows. Assume that we have a classical Markov chain, on a state-space in which some states are special (marked). The Markov chain starts in a uniformly random state and stops if it reaches a marked state. If the classical Markov chain reaches a marked state in expected time T, then there is a quantum algorithm which can find it in time $O(\sqrt{T})$, assuming some conditions on the Markov chain [58, 44, 42].

This approach gives quantum speedups for a number of problems: element distinctness [5], search on a grid [13, 59], finding triangles in graphs [45], testing matrix multiplication [19] and others.

Another application of quantum walks is to the "glued trees" problem [26]. In this problem, we have a graph G with two particular vertices u, v, designed as the entrance and the exit. The problem is to find the vertex v, if we start at the vertex u. There is a special exponential size graph called "glued trees" on which any classical algorithm needs exponential time to find v but a quantum algorithm can find v in polynomial time [26].

Adiabatic computation is a physics-based paradigm for quantum algorithms. In this paradigm, we design two quantum systems:

- H_{sol} whose lowest-energy state $|\psi_{sol}\rangle$ encodes a solution to a computational problem (for example, a satisfying assignment for SAT).
- H_{start} whose lowest-energy state $|\psi_{start}\rangle$ is such that we can easily prepare $|\psi_{start}\rangle$.

We then prepare $|\psi_{start}\rangle$ and slowly transform the forces acting on the quantum system from H_{start} to H_{sol}. Adiabatic theorem of quantum mechanics guarantees that, if the transformation is slow enough, $|\psi_{start}\rangle$ is transformed into a state close to $|\psi_{sol}\rangle$ [31].

The key question here is: what is "slowly enough"? Do we need a polynomial time or an exponential time to transform H_{start} to H_{sol} (thus solving SAT by a quantum algorithm)? This is a subject of an ongoing debate [31, 29, 2].

Adiabatic computation has been used by D-Wave Systems [30] which claims to have built a 128-bit adiabatic quantum computer. However, the claims of D-Wave have been questioned by many prominent scientists (see e.g. [1]).

1.3 Most recent algorithms

Two most recent discoveries in this field are the quantum algorithms for formula evaluation [32] and solving systems of linear equations [36]. Both of those algorithms use the methods from the previous algorithms but do it in a novel, unexpected way. Formula evaluation uses quantum walks but in a form that is quite different from the previous approach (which we described above). Quantum algorithm for formula evaluation uses *eigenvalue estimation* [46] which is the key technical subroutine of Shor's factoring algorithm [56] and the related quantum algorithms. But, again, eigenvalue estimation is used in a very unexpected way.

These two algorithms are the main focus of this survey. We describe them in detail in sections 2 and 3.

2 Formula evaluation

2.1 Overview

We consider evaluating a Boolean formula of variables $x_1, \ldots, x_N$ consisting of ANDs and ORs, with each variable occuring exactly once in the formula. Such a formula can be described by a tree, with variables x_i at the leaves and AND/OR gates at the internal nodes. This problem has many applications because Boolean formulas can be used to describe a number of different situations. The most obvious one is determining if the input data $x_1, \ldots, x_N$ satisfy certain constraints that can be expressed by AND/OR gates.

For a less obvious application, we can view formula evaluation as a black-box model for a 2-player game (such as chess) if both players play their optimal strategies. In this case, the game can be represented by a game tree consisting of possible positions. The leaves of a tree correspond to the possible end positions of the game. Each of them contains a variable x_i, with $x_i = 1$ if the 1^{st} player wins and $x_i = 0$ otherwise. If an internal node v corresponds to a position in which the 1^{st} player makes the next move, then v contains a value that is OR of the values of v's children. (The 1^{st} player wins if he has a move that leads to a position from which he can win.) If v is a node for which the 2^{nd} player makes the next move, then v contains a value that is AND of the values of v's children. (In this case, the 1st player wins if he wins for any possible move of the 2nd player.)

The question is: assuming we have no further information about the game beyond the position tree, how many of the variables x_i do we have to examine to determine whether the 1st player has a winning strategy?

Classically, the most widely studied case is the full binary tree of depth d, with $N = 2^d$ leaves. It can be evaluated by looking at $\Theta(N^{.754\cdots})$ leaves and this is optimal [54, 53, 57]. A natural question was whether one could achieve a better result, using quantum algorithms. This question was a well known open problem in the quantum computing community since mid-1990s. Until 2007, the only known result was that $\Omega(\sqrt{N})$ quantum steps are necessary, for any AND-OR tree [3, 14].

2.2 The model

By standard rules from Boolean logic (de Morgan's laws), we can replace both AND and OR gates by NAND gates. A NAND gate $NAND(y_1, \ldots, y_k)$ outputs 1 if $AND(y_1, \ldots, y_k) = 0$ (i.e., $y_i = 0$ for at least one $i \in \{1, \ldots, k\}$) and 0 otherwise. Then, we have a tree with $x_1, \ldots, x_N$ at the leaves and NAND gates at the internal vertices. The advantage of this transformation is that we now have to deal with just one type of logic gates (instead of two - AND and OR).

We work in the quantum query model. In the discrete-time version of this model [6, 20], the input bits $x_1, \ldots, x_N$ can be accessed by queries O to a black box.

To define O, we represent basis states as $|i, z\rangle$ where $i \in \{0, 1, \ldots, N\}$. The query transformation O_x (where $x = (x_1, \ldots, x_N)$) maps $|0, z\rangle$ to $|0, z\rangle$ and $|i, z\rangle$ to $(-1)^{x_i}|i, z\rangle$ for $i \in \{1, ..., N\}$ (i.e., we change phase depending on x_i, unless $i = 0$ in which case we do nothing).

Our algorithm can involve queries O_x and arbitrary non-query transformations that do not depend on $x_1, \ldots, x_N$. The task is to solve a computational problem (e.g., to compute a value of a NAND formula) with as few queries as possible.

2.3 Results

In 2007, in a breakthrough result, Farhi et al. [32] showed that the full binary AND-OR tree can be evaluated in $O(\sqrt{N})$ quantum time in continuous-time counterpart of the query model.

Several improvements followed soon. Ambainis et al. [27, 8, 12] translated the algorithm of [32] to the conventional discrete time quantum query model and extended it to evaluating arbitrary Boolean formulas with $O(N^{1/2+o(1)})$ quantum queries.

Soon after, Reichardt and Špalek [52] discovered a far-reaching generalization of this result. Namely, the quantum algorithm was generalized to evaluating span programs. A span program is an algebraic model of computation, originally invented for proving lower bounds on circuit size [40].

In a span program, we have a *target* vector v in some linear space. We also have other vectors $v_1, \ldots, v_m$, each of which is associated with some condition $x_i = 0$ or $x_i = 1$. The span program evaluates to 1 on an input $x_1, \ldots, x_n$ if v is equal to a linear combination of vectors $v_{i_1}, \ldots, v_{i_k}$ which are associated with conditions that are true for the given input $x_1, \ldots, x_n$. Otherwise, the span program evaluates to 0.

Here is an example of a span program. We have a two dimensional linear space, with the following vectors:

$$v = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, v_1 = \begin{pmatrix} 1 \\ a \end{pmatrix}, v_2 = \begin{pmatrix} 1 \\ b \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ c \end{pmatrix}$$

where a, b, c are any three distinct non-zero numbers. Vectors v_1, v_2, v_3 are associated with conditions $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, respectively.

Given any two of v_1, v_2, v_3, we can express any vector in two dimensions (including v) as their linear combination. Thus, this span program computes the majority function $MAJ(x_1, x_2, x_3)$ which is 1 whenever at least 2 of variables x_1, x_2, x_3 are equal to 1.

Logic formulae can be embedded into span programs. That is, if we have two span programs computing functions $f_1(x_1, \ldots, x_N)$ and $f_2(x_1, \ldots, x_N)$, we can combine them into span programs for f_1 AND f_2 and f_1 OR f_2 in a fairly simple way.

Reichardt and Špalek [52] invented a complexity measure, *witness size* for span programs. This measure generalizes formula size: a logic formula of size S can be transformed into a span program with witness size S. [52] gave a quantum algorithm for evaluating a span program of witness size S with $O(\sqrt{S})$ queries. This is a very interesting result because it allows to evaluate formulas with gates other than AND and OR by designing span programs for those gates and them composing them into one big span program. The next step was even more interesting.

The next step was even more surprising. Reichardt [48, 49, 51] discovered that the span program approach is optimal, for any Boolean function $f(x_1, \ldots, x_N)$. That is [51], if $Q(f)$ is the minimum number of quantum queries for evaluating f (by any quantum algorithm), then there is a span program with witness size $O(Q^2(f))$. Thus, a span-program based algorithm can evaluate f with $O(Q(f))$ queries, within a constant factor of the best possible number of queries.

This fact linked two lines of research: quantum formula evaluation algorithms and "quantum adversary" lower bounds. "Quantum adversary" (invented in [3]) is a method for proving lower bounds on the number of queries to evaluate $f(x_1, \ldots, x_N)$ by a quantum algorithm. Several progressively stronger versions of "quantum adversary" have been invented [7, 15, 43, 38], with the strongest being the "negative adversary" method of [38].

Finding the best lower bound for quantum algorithms provable via "negative adversary" method is a semidefinite program (SDP). Reichardt [48, 49, 51] considered the dual of this SDP and showed that the dual SDP gives the span program with the smallest witness size. Thus, the span programs are optimal (in terms of query complexity) for any Boolean function $f(x_1, \ldots, x_N)$. (Finding the optimal span program, however, requires solving a semidefinite program of size 2^N.)

As a by-product, this gave a better algorithm for formula evaluation, improving the complexity from $O(N^{1/2+o(1)})$ in [12] to $O(\sqrt{N}\log N)$ in [50].

2.4 Algorithmic ideas

We now give a very informal sketch the simplest version of formula evaluation algorithm. We augment the formula tree with a finite "tail" of length L as shown in Figure 1. We then consider a quantum walk on this tree. At the leaves of the tree, the transformations that are performed depend on whether the leaf holds $x_i = 0$ or $x_i = 1$. (This is achieved by querying the respective x_i and then performing one of two transformations, depending on the outcome of the query.)

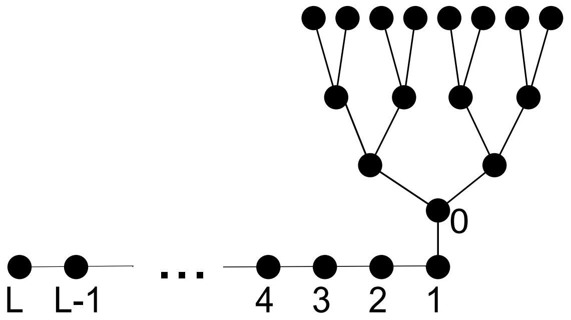

Fig. 1. A formula tree augmented with a finite "tail"

The starting state is an appropriately chosen quantum state $|\psi_{start}\rangle = \sum_i \alpha_i |i\rangle$ consisting of the states $|i\rangle$ in the tail. If the quantum walk is set up properly, an amazing thing happens! Whenever the formula evaluates to 0, the state $|\psi_{start}\rangle$ remains almost unchanged. Whenever the formula evaluates to 1, after $O(N^{1/2+o(1)})$ steps, the state is almost completely different from $|\psi_{start}\rangle$. This means that we can distinguish between the two cases by running the walk for $O(N^{1/2+o(1)})$ steps and measuring whether the state is still $|\psi_{start}\rangle$. Surprisingly, the behaviour of the walk only depends on the value of the formula and not on which particular variables $x_1, \ldots, x_N$ are 1.

The algorithm for evaluating span programs is essentially the same, except that the quantum walk is performed on a weighted graph that corresponds to the span program.

For more information on this topic, we refer the reader to the survey [10] and the original papers.

3 Linear equations

3.1 Overview

Solving large systems of linear equations is a very common problem in scientific computing, with many applications. We consider solving a system of N linear equations with N unknowns: $Ax = b$ where

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \dots & \dots & \dots & \dots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix}, x = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix}, b = \begin{pmatrix} b_1 \\ b_2 \\ \dots \\ b_N \end{pmatrix}.$$

A and b are given to us. The task is to find x.

The best classical algorithm for solving a general system $Ax = b$ runs in time $O(N^{2.37\dots})$. The reason for that was that even outputting the solution requires time $\Omega(N)$ because the solution contains values for N variables. Thus, it seemed that there was no hope for achieving more than a polynomial speedup by a quantum algorithm.

Recently, Harrow, Hassidim and Lloyd [36] discovered a surprising quantum algorithm that allows to solve systems of linear equations in time $O(\log^c N)$ - in an unconventional sense. Namely, the algorithm of [36] generates the quantum state

$$|\psi\rangle = \sum_{i=1}^{N} x_i |i\rangle$$

with the coefficients x_i being equal to the values of variables in the solution $x = (x_1, x_2, \dots, x_N)$ of the system $Ax = b$.

What can we do with this quantum state? We cannot extract all the values x_i from it. If we measured this state, we would obtain one value i, with probabilities of different i proportional to $|x_i|^2$.

We can, however, estimate some quantities that depend on all of x_i. For example, if all variables in the solution had values 1 or -1, having the quantum state $|\psi\rangle$ would enable us to estimate the fraction of variables $x_i = -1$. Moreover, similar tasks appear to be hard classically. As shown by [36], a classical $O(\log^c N)$-time algorithm for computing any quantity of this type implies a polynomial time classical algorithm for simulating any quantum computation. Thus (unless P=BQP), this quantum algorithm provides a genuine speedup over the classical algorithms.

3.2 More details

In more detail, the running times of both classical and quantum algorithms for solving systems of linear equations actually depend on several parameters. One parameter is N, the number of equations (and variables). Another parameter is κ, the condition number of the system. κ is defined as $\frac{\mu_{max}}{\mu_{min}}$ where μ_{min} and μ_{max} are the smallest and the largest singular values of the matrix A [37, Chapter 5.8].

Intuitively, the condition number describes the closeness of the matrix A to a singular matrix. For a singular matrix, $\mu_{min} = 0$ and $\kappa = \infty$. Larger condition number means that the matrix A is closer to a singular matrix. In this case, small changes to input data A and b (or small numerical inaccuracies) can cause large changes to solution x. To compensate, if we have a matrix A with large κ, we have to perform the computation with a higher accuracy. This increases the running time of all algorithms for solving systems of linear equations but the exact increase varies greatly from algorithm to algorithm.

The main classical algorithms for solving systems of linear equations are:

1. LU-decomposition [33, Chapter 3] which runs in time $O(N^{2.376...} \log^c(\kappa))$ [24]. Here, $w = 2.376...$ is the constant from the running time $O(N^w)$ of the best matrix multiplication algorithm [28].
2. Conjugate gradient [33, Chapter 10], which runs in time $O(m\sqrt{\kappa})$ [55] where m is the number of non-zero entries in the matrix. If we know that each row of A contains at most s non-zero entries, this is at most $O(Ns\sqrt{\kappa})$.

The running time of the quantum algorithm [36] is $O(\kappa^2 T \log^c N)$ where T is the time necessary to implement the transformation e^{iA} on a quantum computer. T varies greatly, depending on A. For sparse A with at most s nonzero values in each row and each column, $T = O(s^2 \log N)$ [16]. Thus, in this case the running time of the quantum algorithm is $O(\kappa^2 s^4 \log^c N)$. This achieves an exponential speedup for the case when N is large and κ is relatively small (e.g., $\kappa = O(1)$ or $\kappa = O(\log N)$).

The key bottleneck is the dependence on κ which is actually worse than in the classical algorithms. We have been able to improve it to $O(\kappa^{1+o(1)} \log^c N)$ [11]. Unfortunately, further improvement is very unlikely. [36] have shown that an $O(\kappa^{1-\epsilon} \log^c N)$ time quantum algorithm would imply BQP=PSPACE.

For non-sparse A, one could use the algorithms of [25, 17] to simulate e^{iA}. The dependence on N is better than $O(N^{2.376...})$ in the classical LU decomposition but the speedup is only polynomial.

References

1. S. Aaronson. D-Wave Easter Spectacular. A blog post, April 7, 2007. `http://scottaaronson.com/blog/?p=225.`
2. B. Altshuler, H. Krovi, J. Roland. Anderson localization casts clouds over adiabatic quantum optimization. arxiv:0912.0746.
3. A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64:750-767, 2002. Also available as quant-ph/0002066.
4. A. Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1:507-518, 2003. Also quant-ph/0403120.
5. A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1): 210-239, 2007. Also FOCS'04 and quant-ph/0311001.
6. A. Ambainis. Quantum search algorithms (a survey). *SIGACT News*, 35(2):22-35, 2004. Also quant-ph/0504012.
7. A. Ambainis. Polynomial degree vs. quantum query complexity. *Journal of Computer and System Sciences*, 72(2): 220-238, 2006. Also quant-ph/0305028.

8. A. Ambainis. A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. arxiv:0704.3628.
9. A. Ambainis. Quantum random walks - New method for designing quantum algorithms. *Proceedings of SOFSEM'08*, pp. 1-4.
10. A. Ambainis. Quantum algorithms for formula evaluation. *Proceedings of the NATO Advanced Research Workshop "Quantum Cryptography and Computing: Theory and Implementations"*, to appear.
11. A. Ambainis. Variable time amplitude amplification and a faster quantum algorithm for systems of linear equations. In preparation, 2010.
12. A. Ambainis, A. Childs, B. Reichardt, R. Spalek, S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *Proceedings of FOCS'07*, pp.363-372.
13. A. Ambainis, J. Kempe, A. Rivosh, Coins make quantum walks faster. *Proceedings of SODA'05*, pp. 1099-1108. Also quant-ph/0402107.
14. H. Barnum, M. Saks, A lower bound on the quantum complexity of read once functions. *Journal of Computer and System Sciences*, 69:244-258, 2004. Also quant-ph/0201007.
15. H. Barnum, M. E. Saks, M. Szegedy. Quantum query complexity and semi-definite programming. *IEEE Conference on Computational Complexity 2003*, pp. 179-193.
16. D. W. Berry, G. Ahokas, R. Cleve, B. C. Sanders. Efficient quantum algorithms for simulating sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2):359-371, 2007. Also quant-ph/0508139.
17. D. W. Berry, A. Childs. The quantum query complexity of implementing black-box unitary transformations. arxiv:0910.4157.
18. D. Boneh, R. Lipton. Quantum cryptanalysis of hidden linear functions (extended abstract). *Proceedings of CRYPTO'95*, pp. 424-437.
19. H. Buhrman, R. Špalek: Quantum verification of matrix products. *Proceedings of SODA'06*, pp. 880-889. Also quant-ph/0409035.
20. H. Buhrman, R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288:21-43, 2002.
21. G. Brassard, P. Høyer, M. Mosca, A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information Science*, AMS Contemporary Mathematics Series, 305:53-74, 2002. Also quant-ph/0005055.
22. G. Brassard, P. Hoyer, A. Tapp. Quantum cryptanalysis of hash and claw-free functions. *Proceedings of LATIN'98*, pp. 163-169. Also quant-ph/9705002.
23. G. Brassard, P. Høyer, A. Tapp. Quantum counting. *Proceedings of ICALP'98*, pp. 820-831. Also quant-ph/9805082.
24. J.R. Bunch, J.E. Hopcroft, Triangular factorization and inversion by fast matrix multiplication, *Mathematics of Computation*, 28:231-236, 1974.
25. A. Childs. On the relationship between continuous- and discrete-time quantum walk. *Communications in Mathematical Physics* 294:581-603, 2010. Also arXiv:0810.0312.
26. A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, D. A. Spielman, Exponential algorithmic speedup by quantum walk. *Proceedings of STOC'03*, pp. 59-68. Also quant-ph/0209131.
27. A. Childs, B. Reichardt, R. Špalek, S. Zhang. Every NAND formula on N variables can be evaluated in time $O(N^{1/2+\epsilon})$, quant-ph/0703015, version v1, March 2, 2007.
28. D. Coppersmith, S. Winograd, Matrix multiplication via arithmetic progressions, *Journal of Symbolic Computation* 9(3): 251-280, 1990.
29. W. van Dam, M. Mosca, U. Vazirani. How Powerful is Adiabatic Quantum Computation?. *Proceedings of FOCS'2001*, pp. 279-287.

30. D-Wave Systems. `http://www.dwavesys.com`
31. E. Farhi, J. Goldstone, S. Gutman, M. Sipser. A quantum adiabatic algorithm applied to random instances of an NP-complete problem, *Science* 292:472-476, 2001. Also quant-ph/0104129.
32. E. Farhi, J. Goldstone, S. Gutman, A Quantum Algorithm for the Hamiltonian NAND Tree. *Theory of Computing*, 4:169-190, 2008. Also quant-ph/0702144.
33. G. Golub, C. van Loan. Matrix Computations, 3rd edition. John Hopkins University Press, 1996.
34. L. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of STOC'96*, pp. 212-219. Also quant-ph/9605043.
35. S. Hallgren. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. *Journal of the ACM*, 54:1, 2007.
36. A. Harrow, A. Hassidim, S. Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103:150502, 2008. Also arXiv:0907.3920.
37. R. Horn, C. Johnson. *Matrix Analysis*, Cambridge University Press, 1985.
38. P. Høyer, T. Lee, R. Špalek. Negative weights make adversaries stronger. *Proceedings of STOC'07*, pp. 526-535. Also quant-ph/0611054.
39. R. Jozsa. Quantum factoring, discrete logarithms, and the hidden subgroup problem. *Computing in Science and Engineering*, 3:34-43, 2001. Also quant-ph/0012084.
40. M. Karchmer, A. Wigderson. On Span Programs. *Structure in Complexity Theory Conference 1993*, pp. 102-111.
41. J. Kempe. Quantum random walks - an introductory overview. *Contemporary Physics*, 44 (4):307-327, 2003. Also quant-ph/0303081.
42. H. Krovi, F. Magniez, M. Ozols and J. Roland. Finding is as easy as detecting for quantum walks. *Proceedings of ICALP'2010*, to appear. Also arxiv:1002.2419.
43. S. Laplante, F. Magniez. Lower Bounds for Randomized and Quantum Query Complexity Using Kolmogorov Arguments. *SIAM Journal on Computing*, 38(1): 46-62, 2008. Also CCC'2004 and quant-ph/0311189.
44. F. Magniez, A. Nayak, J. Roland, M. Santha. Search via quantum walk. *Proceedings of STOC'07*, pp. 575-584. Also quant-ph/0608026.
45. F. Magniez, M. Santha, M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2): 413-424, 2007. Also quant-ph/0310134.
46. M. Mosca, A. Ekert. The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. *Proceedings of QCQC'98*, pp. 174-188. Also quant-ph/9903071.
47. A. Nayak, F. Wu. The quantum query complexity of approximating the median and related statistics. *Proceedings of STOC'99*, pp. 384-393. Also quant-ph/9804066.
48. B. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every boolean function. *Proceedings of FOCS'2009.* Also arXiv:0904.2759.
49. B. Reichardt. Span-program-based quantum algorithm for evaluating unbalanced formulas. arXiv:0907.1622.
50. B. Reichardt. Faster quantum algorithm for evaluating game trees. arXiv:0907.1623.
51. B. Reichardt. Reflections for quantum query algorithms. arxiv:1005.1601.
52. B. Reichardt, R. Špalek. Span-program-based quantum algorithm for evaluating formulas. *Proceedings of STOC'2008*, pp. 103-112. Also arXiv:0710.2630.
53. M. Saks, A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees, *Proceedings of FOCS'86*, pp. 29-38.
54. M. Santha. Quantum walk based search algorithms. *Proceedings of TAMC'2008*, pp. 31-46. Also arXiv:0808.0059.

55. J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

56. P. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. *Proceedings of FOCS'94*, pp. 124-134. Also quant-ph/9508027.

57. M. Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38: 69-82, 1985.

58. M. Szegedy. Quantum speed-up of Markov chain based algorithms. *Proceedings of FOCS'04*, pp. 32-41.

59. T. A. Tulsi. Faster quantum walk algorithm for the two dimensional spatial search, *Physical Review A*, 78:012310, 2008. Also arXiv:0801.0497.

60. S. E. Venegas-Andrade. *Quantum Walks for Computer Scientists.* Morgan and Claypool, 2008.