QEDS: Um Simulador Clássico para Distinção de Elementos Quântico

QEDS: A Classical Simulator for Quantum Element Distinctness

Alexandre Santiago de Abreu [1, 2]
Matheus Manzoli Ferreira [3, 4]
Luis Antonio Brasil Kowada [3, 5]
Franklin de Lima Marquezino [1, 6, 7]

Data de submissão: 11/06/2016, *Data de aceite: 02/10/2016*

Resumo: O problema de decidir se todos os N elementos em uma lista são distintos requer $Omega(N)$ consultas no modelo clássico. Um algoritmo quântico baseado em caminhada quântica em um gráfico de Johnson melhora este limite para $O(N^{2/3})$ consultas. O algoritmo quântico para a distinção de elementos executa vários cálculos, cada um envolvendo superposições não triviais de estados. Por esta razão, é difícil estudar o algoritmo sem ferramentas apropriadas. Neste trabalho, apresentamos um simulador numérico para o algoritmo de distinção de elementos e analisamos seu desempenho. O objetivo principal de nosso simulador é servir como uma ferramenta educacional. No entanto, como um software livre de código aberto, ele pode ser facilmente estendido para uso profissional.

***P*alavras-chave:** algoritmos quânticos, simulações, caminhadas quânticas, distinção de elementos, computação quântica, educação

[1]Programa de Engenharia de Sistemas e Computação (PESC), Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa em Engenharia (COPPE), Universidade Federal do Rio de Janeiro (UFRJ) - Rio de Janeiro, RJ, Brasil.
[2]{santiago@cos.ufrj.br}
[3]Instituto de Computação (IC), Universidade Federal Fluminense (UFF) - Niterói, RJ, Brasil.
[4]{matheusmanzoli@id.uff.br}
[5]{luis@ic.uff.br}
[6]Núcleo Multidisciplinar de Pesquisa em Computação (NUMPEX-Comp), Polo Avançado de Xerém, Universidade Federal do Rio de Janeiro(UFRJ) - Duque de Caxias, RJ, Brasil.
[7]{franklin@cos.ufrj.br}{franklin@xerem.ufrj.br}

Abstract: The problem of deciding whether all N elements in a list are distinct requires $\Omega(N)$ queries in the classical setting. A quantum algorithm based in quantum walk on a Johnson graph improves this bound to $O(N^{2/3})$ queries. The quantum algorithm for element distinctness performs several steps of calculations each involving nontrivial superpositions of states. For this reason, it is difficult to study the algorithm without appropriate tools. In this work, we present a numerical simulator for the element distinctness algorithm and analyze its performance. The main purpose of our simulator is to serve as an educational tool. However, as a free open-source software, it can be easily extended for professional usage.

***K*eywords:** quantum algorithms, simulations, quantum walks, element distinctness, quantum computing education

1 Introduction

Building a general purpose quantum computer is still a technological challenge nowadays, even though important progress has been made in quantum hardware in the last few years. For instance, Harvard researchers used a D-Wave quantum computer to solve the protein folding problem [1], and the IBM Quantum Experience[8] now provides free access to IBM's experimental cloud-enabled quantum computing platform. However, both approaches were very different, and there is not a general agreement among experimentalists on which technology will be most suitable to control a large amount of qubits for the time necessary to perform useful computations. Nevertheless, understanding how quantum computers and quantum algorithms work is essential for the development of new quantum algorithms that outperform their classical counterparts. The simulation of quantum algorithms by classical computers is a subject studied by many authors [2][3][4][5][6][7] as a tool for understanding important concepts of quantum computing, focusing on at least two main goals: (i) providing an educational tool to learn how the algorithms would behave under certain conditions, and (ii) studying certain aspects of the algorithms where analytic results are very hard to obtain.

The problem of simulating quantum mechanics in classical computers usually have exponential complexity, both in time and space. Even so, many simulation tools have been developed to study the behavior of quantum systems, aiming at a variety of applications such as quantum circuits [3], quantum stabilizer formalism [8], quantum walks on graphs [4], quantum Hamiltonians [9] and so on. Those tools are very useful for the design of quantum circuits and quantum algorithms, and also improves the understanding of the power and limitations of the quantum paradigm.

The first quantum algorithm for element distinctness was first developed by Ambai-

[8] http://www.research.ibm.com/quantum/

nis [10] and later improved for particular cases by Belovs *et al* [11, 12]. There are alternative formulations. For instance, it is possible to apply Szegedy's method directly on the Johnson graph [13, 14]. However, we describe the original formulation in this paper. Since Ambainis' algorithm involves several steps of calculations with nontrivial superpositions of states, the task of analytically studying the evolution of the quantum state becomes quite difficult even for moderate-sized input lists. Moreover, Ambainis' algorithms is based on a quantum walk over a very particular graph, namely a Johnson graph, which makes other simulators such as QWalk [4] unsuitable for this simulation. The Johnson graph $J(d, k)$, for $2 \leq 2d \leq k$, is the graph where the vertex set is composed by all subsets of $\Omega_k = (1, ..., k)$ of order d, for any positive integer k. A pair of vertices are adjacent if their intersection has order $d - 1$ [15].

Ambainis' algorithm uses important concepts and techniques in quantum computing—such as quantum walks, and search in graphs [16]. Notice that quantum walks had been successfully applied as a strategy for the development of several quantum algorithms besides the element distinctness algorithm [17]. Moreover, a remarkable recent result in the field is the proof that quantum walks are a universal model for quantum computing [18], which implies that the implementation of quantum walks may also contribute for the development of quantum hardware in general [19]. Thus, it is fundamental that every student in the field of quantum computing work through these concepts and techniques very carefully.

In this work, we develop a numeric simulator for the algorithm for element distinctness and analyze its performance. To the best of our knowledge, there is no other simulator for Ambainis' algorithm available in the literature. The main purpose of our simulator is to serve as an educational tool, simulating this particular algorithm and showing how the quantum state evolves at each step. Since the source code of our simulator is freely available under a GNU GPL license, it can be easily extended for other than didactic purposes, such as the simulation of decoherence or noisy operations.

This paper is structured as follows. In Sec. 2, we introduce the notation that will be used throughout the paper and review the problem of element distinctness. We also give a detailed explanation of Ambainis' algorithm [10]. In Sec. 3, we describe technical aspects of our simulator and explain how it can be used to study each step of the quantum algorithm for element distinctness. In Sec. 4, we show results of simulations performed with our tool and give a report describing how memory consumption and elapsed time scale with the size of the input list. Finally, in Sec. 5, we present our conclusions and discuss possibilities of future work. We also include an appendix with the Ambainis' algorithm.

2 Preliminary Notions

The mathematical background required for quantum computing is basically linear algebra. Quantum mechanics can still be very frightening for a beginner. However, since we

are only considering the *application* of quantum mechanics to computing, we can summarize quantum mechanics to only four postulates. The first postulate claims that every quantum state should be described by a unitary vector in the Hilbert space $\mathcal{H}$. The second postulate claims the the time evolution of a closed quantum system is described by a unitary matrix. The third postulate claims that the composition of two or more quantum states is described by the tensor product operation. Finally, the fourth postulate claims that the outcomes of the measurement of a quantum system are described by the eigenvalues of a Hermitean matrix, and that the state after the measurement is irreversibly collapsed to the corresponding eigenspace. A more detailed review can be found, for instance, in Reference [20].

In this context, suppose we have a list of N elements, we want to know if all these elements are distinct. In the classical setting, it is not difficult to find that a lower bound for this problem is $\Omega(N)$, since any algorithm should read all entries at least one time before making a decision. In fact, assuming without loss of generality that every element x in the list is bounded, $x_{\min} \leq x \leq x_{\max}$, then one can simply use a linear time search method such as Bucket Sort, for instance, in order to solve the element distinctness problem in time $O(N)$. In the quantum setting, Ambainis [10] proposed an algorithm that decides in $O(N^{k/k+1})$ steps if an input list has k non-distinct elements. In our case, we have $k = 2$, resulting in $O(N^{2/3})$ steps to solve this problem.

Let $[N]$ denote $\{1, ..., N\}$ and $r = \lfloor N^{2/3} \rfloor$. We define a bipartite graph, as in Fig. 1, with $\binom{N}{r} + \binom{N}{r+1}$ vertices, where each vertex corresponds to subsets of $[N]$, as follows: (i) vertices v_s correspond to sets $S \subseteq [N]$ of size r, and (ii) vertices v_t correspond to sets $T = S \cup \{i\}$, where $i \in [N]$. A vertex v_s is connected by an edge to another vertex v_t if the corresponding set of v_s is a subset of corresponding set of v_t. A vertex v_s is marked if and only if the elements of the list indexed by the elements of the set of v_s are repeated.

One could use Grover's algorithm to search for this marked vertex and find it after examining $O(1/\sqrt{\varepsilon})$ vertices, where ϵ is the fraction of marked elements. We can assume that there is a k-tuple with k different indexes $i_1, ..., i_k \in [N]$ such that $x_{i_1} = ... = x_{i_k}$. For a S chosen randomly, we have $|S| = N^{k/k+1}$, and the probability of v_s be a marked vertex is given by

$$Pr[i_1 \in S]Pr[i_2 \in S|i_1 \in S] + \cdots + Pr[i_{k-1} \in S| \bigcap_{j=1}^{k-2} i_j \in S]Pr[i_k \in S| \bigcap_{j=1}^{k-1} i_j \in S]$$
$$= \frac{N^{k/k+1}}{N} \frac{N^{k/k+1} - 1}{N - 1} + \ldots + \frac{2}{(N - N^{k/k+1} + 2)} \frac{1}{(N - N^{k/k+1} + 1)}. \quad (1)$$

It follows that the probability of v_s be a marked vertex is

$$O\left(\frac{N^{2k/k+1}}{N^2}\right) = O\left(\frac{1}{N^{2/k+1}}\right), \quad (2)$$

thus, $1/\sqrt{\epsilon} = O(N^{1/k+1})$. Then, we would have to check if the resulting vertex is the marked one by querying $N^{k/k+1}$ items x_i, for $i \in S$, resulting in the final query complexity being $O(N^{1/k+1}N^{k/k+1}) = O(N)$. Therefore, Grover's algorithm gives no speedup when compared to the classical method. The idea of Ambainis' algorithm is to combine quantum search in graphs with quantum walks. By doing that, we can re-use the information from previous queries reducing the search space.

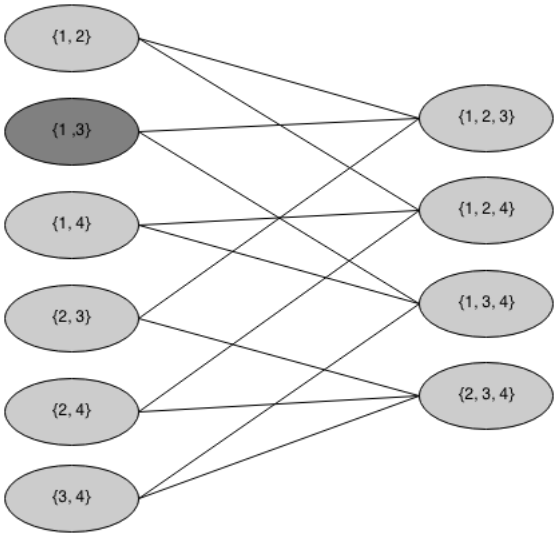

Figure 1. An example of Johnson graph built from the list $L = \{22, 34, 22, 55\}$.

Let us consider the values $x_1, ..., x_n \in [M]$. Each partition of the graph represents two Hilbert spaces, $\mathcal{H}$ and $\mathcal{H}'$, where the vertices v_s and v_t are respectively vertices in these spaces. The space $\mathcal{H}$ has dimension $\binom{N}{r}M^r(N-r)$ and the basis states for this space are given by $|S, x, y\rangle$, where $S \subseteq [N], |S| = r, x \in [M]^r, y \in [N]\backslash S$. On the other hand, the space $\mathcal{H}'$ has dimension $\binom{N}{r+1}M^{r+1}(r+1)$ with basis given by $|S, x, y\rangle$, where $S \subseteq [N], |S| = r+1, x \in [M]^{r+1}, y \in S$.

Let us describe each step of Ambainis' algorithm. We present an example of quantum state and we evolve it while describing each step. For simplicity, let us consider a single state $\alpha|1, 2\rangle|3\rangle|x_1, x_2\rangle$, that represents the vertex $\{1, 2\}$ in Fig.1, of the superposition of states, where $|S\rangle = |1, 2\rangle$, $|y\rangle = |3\rangle$, $|x\rangle = |x_1, x_2\rangle$, and α is the quantum phase amplitude of this state. In each step, we use r queries in all $x_i, i \in S$ for the starting vertex v_s. After that, we start a loop that checks if the vertex is marked without using any query—because we already know all the elements in this vertex. The next step consists in using a quantum walk to move

the walker from this vertex to an adjacent vertex v_t, by querying $x_i, i \in T \backslash S$. After that, we know all the elements in T. This algorithm alternates two kinds of transformations: changing the phase if the current vertex is marked, and performing a quantum walk.

In the first step, the algorithm generates the uniform superposition of all vertices v_s,

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y \notin S} |S\rangle|y\rangle. \tag{3}$$

and after we query all x_i for $i \in S$, transforming the state to

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y \notin S} |S\rangle|y\rangle \bigotimes_{i \in S} |x_i\rangle. \tag{4}$$

At the end of these two steps, our walker will be in a superposition of all vertices of the partition of the space $\mathcal{H}$. The next step is composed by two loops. The outer loop will be repeated $O((N/r)^{k/2})$ times. First, the vertex v_s, which contains $x_{i_1} = ... = x_{i_k}$ for distinct $i_1, ..., i_k \in S$, is marked by a conditional phase flip, that will change the quantum phase amplitude of the marked vertex. After that, the inner loop will be repeated $O(\sqrt{r})$ times, starting the quantum walk—which will be presented in details later on. This step is responsible to increase the marked vertices' phase and decrease the phases of the others vertices. Finally, the measurement takes place. In this step, the marked vertex will have a higher probability of being measured.

The quantum walk is structured in six steps, where the first three steps are responsible for moving the walker from the space $\mathcal{H}$ to $\mathcal{H}'$, while the final three steps move it back from space $\mathcal{H}'$ to $\mathcal{H}$.

In the first step of the quantum walk, we apply a transformation that maps $|S\rangle|y\rangle$ to

$$|s\rangle \left(\left(-1 + \frac{2}{N-r} \right) |y\rangle + \frac{2}{N-r} \sum_{y' \notin S, y' \neq y} |y'\rangle \right) \tag{5}$$

on S and y registers of the state $\mathcal{H}$. For simplicity, we omitted the $|x\rangle$ register in this step, since it is not affected. In our example, this transformation maps the state $\alpha|1,2\rangle|3\rangle$ to $\alpha_1|1,2\rangle|3\rangle|x_1,x_2\rangle + \beta_1|1,2\rangle|4\rangle|x_1,x_2\rangle$, where α_1 and β_1 are quantum phases amplitudes.

In the second step of the quantum walk, the mapping between the spaces begins. It is done by adding y to S and changing x to a vector of length $k+1$ by introducing 0 in the location corresponding to y, increasing the current Hilbert space. In our example, the state $\alpha_1|1,2\rangle|3\rangle + \beta_1|1,2\rangle|4\rangle$ is transformed in $\alpha_1|1,2,3\rangle|3\rangle|x_1,x_2,0\rangle + \beta_1|1,2,4\rangle|4\rangle|x_1,x_2,0\rangle$, becoming a state in space $\mathcal{H}'$.

In the third step of the quantum walk, the $|S\rangle$ register has a new index for query, so the third step we query for the x_y and insert it into location of x corresponding to y. Thereafter, the walker will be in $\mathcal{H}'$ space. Now, our example state is $\alpha_1|1,2,3\rangle|3\rangle|x_1,x_2,x_3\rangle + \beta_1|1,2,4\rangle|4\rangle|x_1,x_2,x_4\rangle$.

In the fourth step of the quantum walk we apply a new transformation, modifying the vertices' phase. This transformation consists in mapping $|S\rangle|y\rangle$ to

$$|S\rangle\left(\left(-1+\frac{2}{r+1}\right)|y\rangle+\frac{2}{r+1}\sum_{y'\in S,y'\neq y}|y'\rangle\right). \tag{6}$$

In this step, our example state is transformed in $|1,2,3\rangle(\alpha_2|3\rangle+\alpha_3|2\rangle+\alpha_4|1\rangle)|x_1,x_2,x_3\rangle+|1,2,4\rangle(\beta_2|4\rangle+\beta_3|2\rangle+\beta_4|1\rangle)|x_1,x_2,x_4\rangle$, where α_i,β_i are quantum phase amplitudes.

After that, the fifth step of the quantum walk continues the process of transforming the state vector back to $\mathcal{H}$ space. It is done by erasing the element of x corresponding to new y, by using it as input to query for x_y. Therefore, in our example, we now have the state

$$\begin{aligned}&\alpha_2|1,2,3\rangle|3\rangle|x_1,x_2,0\rangle+\alpha_3|1,2,3\rangle|2\rangle|x_1,0,x_3\rangle+\\&+\alpha_4|1,2,3\rangle|1\rangle|0,x_2,x_3\rangle+\beta_2|1,2,4\rangle|4\rangle|x_1,x_2,0\rangle+\\&+\beta_3|1,2,4\rangle|2\rangle|x_1,0,x_4\rangle+\beta_4|1,2,4\rangle|1\rangle|0,x_2,x_4\rangle.\end{aligned} \tag{7}$$

Finally, to conclude the mapping, the sixth step of the quantum walk removes the 0 component corresponding to y from x and removes y from S, resulting, in our example, the final state

$$\begin{aligned}&\alpha_2|1,2\rangle|3\rangle|x_1,x_2\rangle+\alpha_3|1,3\rangle|2\rangle|x_1,x_3\rangle+\\&+\alpha_4|2,3\rangle|1\rangle|x_2,x_3\rangle+\beta_2|1,2\rangle|4\rangle|x_1,x_2\rangle+\\&+\beta_3|1,4\rangle|2\rangle|x_1,x_4\rangle+\beta_4|2,4\rangle|1\rangle|x_2,x_4\rangle.\end{aligned} \tag{8}$$

With these six steps of the quantum algorithm the walker goes from partition S to partition T, and then back to partition S, completing one step of quantum walk.

Ambainis also described an algorithm to decide if all elements are distinct for the case with multiple k-collision, however, it is out of the scope of this paper. See Appendix A for the complete algorithm as originally described by Ambainis, and see Ref. [10] for extra information about the algorithm, including its proof of correctness and analysis of complexity.

3 The Quantum Element Distinctness Simulator

We developed a numeric simulator to provide an easier way to analyze each step of the algorithm for element distinctness. Analytically, performing each step of the algorithm

becomes quite difficult even for a moderate-sized input list because the algorithm involves several steps of calculations with nontrivial superpositions of states. In this work, we focused on developing a simulator for educational purposes, due to the fact that Ambainis' algorithm uses essential concepts for the development of new quantum algorithms. Since the source code of our simulator is freely available under a GNU GPL license, any user with knowledge of the C++ programming language can modify our code. The source code itself is a helpful object of study for undergraduate students beginning in this field.

The simulator was designed to work in any computer with a C++ compiler. Experiments were performed in a Windows system and in a Linux Ubuntu system with success (see Sec. 4) and without any substantial difference in time or memory performance. We estimate that the simulator requires less than 10MB of memory to run simulations with moderate-sized input list, as we can see in Sec. 4. However, for more interesting simulations, with larger input list, the RAM memory will be a limit. Our simulator does not use external programs, working independently.

To install the simulator in a Linux environment, one needs to download the source code[9] and save it at any folder. After that, one needs to compile the source code using the command *make* or any C++ IDE. After that, there will be an executable file that can be executed in the shell. The process of installing our simulator in a Windows environment is also very simple. One must download the installer and follow the instructions. After that, he or she can run the simulator by double-clicking the icon.

The program gives two options for the input list: generating random values, or manually defining the values. After choosing any of these options, in the second step the user must enter the list size. If the user had previously chosen the option of random input list, the list will be automatically generated and the simulation will start immediately. If the user had chosen the option for manually defining the input list, he or she will have to enter each element of the list before the simulation starts. The simulator will show the execution of Ambainis' algorithm step by step. At the end of the simulation, the software will answer the question of whether *there is a 2-collision* in the list, just like Ambainis' algorithm does.

While the simulator is running, it continuously generates a report where each algorithm's step is described. The software shows the title of the step, the current generalization of states, and each state's phase. Thus, it allows the student to see how each state evolves and how the phases vary over time.

It is important to pay attention to the amount of memory available on the computer. As previously explained, for large input lists and not enough memory, the simulation can start normally and then close without finishing.

[9]Available at `http://bit.ly/1TJ3LCA`

4 Experiments and Numerical Results

The testing environment is an Intel Core i7-2630QM CPU @ 2.00GHz (2.90 GHz - turbo max), with 2.84 GB of available RAM memory, 320 GB of secondary memory, Microsoft Windows 10 and Ubuntu Linux 15.10. We chose a modest configuration for our testing environment, corresponding to the equipment we expect an average student to have access to. As expected, the elapsed time and memory consumption grow exponentially. It happens because, in general, quantum simulations in a classical machine have exponential complexity, both in time and memory. Thus, the vector spaces grow exponentially as the input increases, which explains the memory growth.

The asymptotic complexity of our program is $O(2^N)$ because we use an algorithm to create the subsets that are represented by graph's vertices, and this algorithm is exponential. For small input lists, the simulator runs the algorithm for element distinctness very quickly. However, for big input lists, it takes a lot of time.

We started the tests with the lowest nontrivial value, using four elements in our list and then increased this value by adding new elements to the list. We varied the positions of the collisions and, as expected, it did not influence the results. In Fig. 2, we have the memory consumption in kilobytes (log scale) as function of the size of the input list used in the simulation. We could only simulate up to 24 (twenty-four) elements, since larger lists consumed more memory than the computer could handle. In Fig. 3, we have the elapsed time in seconds (log scale) as function of the size of the input list used in the simulation.

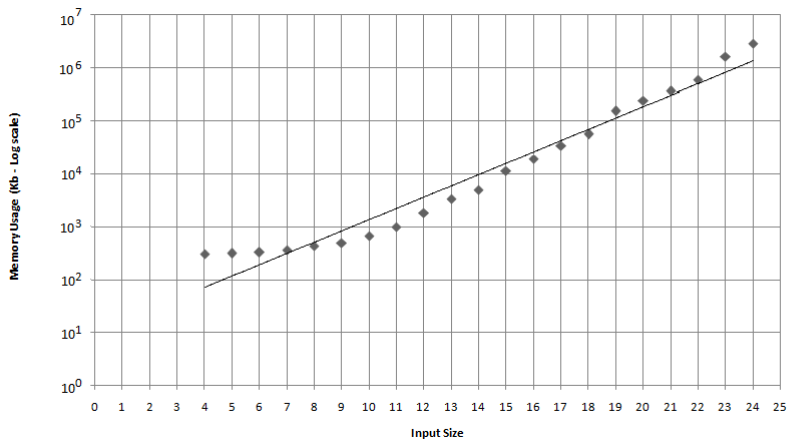

Figure 2. Memory consumption in kilobytes (log scale) as function of the size of the input list used in the simulation.

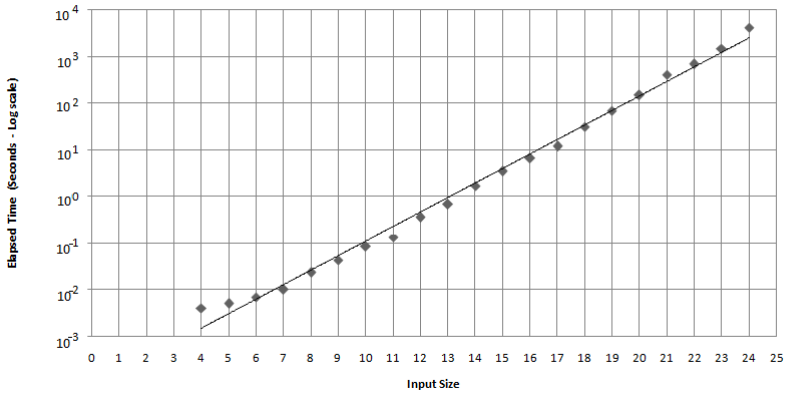

Figure 3. Simulation time in seconds (log scale) as function of the size of the input list used in the simulation.

Notice that the memory consumption increases as

$$M \approx 10^{\frac{N+2}{4}}, \tag{9}$$

and the elapsed time grows as

$$T \approx 10^{\frac{N-13}{3}}. \tag{10}$$

If we plot input size $N = 13$ in these equations, we estimate that the memory necessary to make a simulation with this entry is 10 MB while the elapsed time will be less than two seconds. On the other hand, if we just double this input size, setting $N = 26$, we estimate that the memory consumption will be about 4 GB and the simulation will spend over an hour.

In Tab. 1, we have the probability of finding the searched vertex after the measurement step, for every list size N that we tested.

Notice that running our simulation on a classical computer caused the elapsed time and the memory consumption to be exponentially larger than would be required of the quantum algorithm running directly on a quantum computer. This was the expected behavior—in fact, the Hilbert space used by Ambainis' algorithm grows exponentially with the input list size. In general, the time and memory required for classically simulating quantum systems grow exponentially with the number of particles being simulated [21], and that is the reason why the development of classical simulators of quantum algorithms is so challenging. To the best of our knowledge, there is no other simulator designed specifically for Ambainis' algorithm.

N	Pr(.)
4	59.46 %
5	61.58 %
6	55.56 %
7	57.14 %
8	53.22 %
9	79.53 %
10	90.63 %
11	92.74 %
12	78.66 %
13	82.32 %
14	82.41 %
15	73.25 %

Table 1. Table showing the probability of measuring the marked vertex given that the list has size N.

5 Discussions

In this paper we present a numerical simulator for the element distinctness algorithm [10] with the main purpose of being used as a tool to study the behavior of quantum computers and their algorithms. The problem of an analytic approach for learning the Ambainis' algorithm is that the amount of calculations usually scales exponentially with the input size, making this approach impractical for most cases. Simulating quantum algorithms on a classical computer is also a problem of exponential complexity, both in time and memory. However, this kind of simulation is very helpful for educational purposes. For this reason, many simulation tools have been developed although any of them directly addresses the specific case of the element distinctness algorithm.

The quantum algorithm for element distinctness, developed by Ambainis [10], uses important and complex concepts of quantum computing, such as quantum walks in a Johnson graph and quantum search on graphs. For this reason, studying Ambainis' algorithm can be insightful in the developing of new algorithms and techniques for quantum computing. We introduced a numeric simulator for this important algorithm with the purpose of serving as an educational tool, showing how the quantum state evolves at each step.

As future work, the simulator could be generalized for arbitrary k values and multiple k-collision, while keeping its pedagogical character. It would also be interesting to make it

compatible with HiPerWalk [10] for better performance.

Acknowledgments

A preliminary version of this work has been presented in the 5th Workshop-School in Quantum Information and Computation (WECIQ), Campina Grande, PB, Brazil [22]. A.S.A. thanks CNPq for financial support. F.L.M. acknowledges financial support from CAPES/AUXPE, CNPq/Universal and CNPq/PQ grants.

Contribuição dos autores:

- A.S. Abreu: Abordagem teórica do algoritmo de distinção de elementos, desenvolvimento do algoritmo para executar simulação, programação do simulador.

- M.M. Ferreira: Abordagem teórica do algoritmo de distinção de elementos, desenvolvimento do algoritmo para executar simulação, programação do simulador.

- L.A.B. Kowada: Abordagem teórica do algoritmo de distinção de elementos, desenvolvimento do algoritmo para executar simulação.

- F.L. Marquezino: Abordagem teórica do algoritmo de distinção de elementos, desenvolvimento do algoritmo para executar simulação.

References

[1] Alejandro Perdomo-Ortiz, Neil Dickson, Marshall Drew-Brook, Geordie Rose, and Alán Aspuru-Guzik. Finding low-energy conformations of lattice protein models by quantum annealing. *Scientific reports*, 2, 2012.

[2] Scott D Berry, Paul Bourke, and Jingbo B Wang. qwviz: Visualisation of quantum walks on graphs. *Computer Physics Communications*, 182(10):2295–2302, 2011.

[3] Alexandre de Andrade Barbosa. *Um simulador simbólico de circuitos quânticos*. PhD thesis, Universidade Federal de Campina Grande, 2007.

[4] Franklin L Marquezino and Renato Portugal. The Qwalk simulator of quantum walks. *Computer Physics Communications*, 179(5):359–369, 2008.

[5] George F Viamontes, Igor L Markov, and John P Hayes. Graph-based simulation of quantum computation in the density matrix representation. In *Defense and Security*, pages 285–296. International Society for Optics and Photonics, 2004.

[10] Available at `http://qubit.lncc.br/qwalk/`.

[6] George F Viamontes, Igor L Markov, and John P Hayes. *Quantum circuit simulation*. Springer, 2009.

[7] Juliana Kaizer Vizzotto and Bruno Crestani Calegaro. Qjava: A monadic java library for quantum programming. *Revista de Informática Teórica e Aplicada*, 22(1):242–266.

[8] Stephen D Bartlett, Barry C Sanders, Samuel L Braunstein, and Kae Nemoto. Efficient classical simulation of continuous variable quantum information processes. In *Quantum Information with Continuous Variables*, pages 47–55. Springer, 2002.

[9] Adam D Bookatz, Pawel Wocjan, and Lorenza Viola. Hamiltonian quantum simulation with bounded-strength controls. *New Journal of Physics*, 16(4):045021, 2014.

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

[11] Aleksandrs Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 207–216, Oct 2012.

[12] Aleksandrs Belovs, Andrew M. Childs, Stacey Jeffery, Robin Kothari, and Frédéric Magniez. *Automata, Languages, and Programming: 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, chapter Time-Efficient Quantum Walks for 3-Distinctness, pages 105–122. Springer, Berlin, Heidelberg, 2013.

[13] Mario Szegedy. Quantum speed-up of markov chain based algorithms. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 32–41. IEEE, 2004.

[14] Raqueline A. M. Santos. Cadeias de Markov Quânticas. Master's thesis, National Laboratory of Scientific Computing, Petrópolis, Brazil, 2010.

[15] Paul Terwilliger. The Johnson graph J(d, r) is unique if (d, r)$\neq$(2, 8). *Discrete Mathematics*, 58(2):175–189, 1986.

[16] Renato Portugal. *Quantum walks and search algorithms*. Springer, 2013.

[17] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(04):507–518, 2003.

[18] Andrew M Childs, David Gosset, and Zak Webb. Universal computation by multiparticle quantum walk. *Science*, 339(6121):791–794, 2013.

[19] Renato Portugal, Marcos Cesar de Oliveira, and Jalil Khatibi Moqadam. Staggered quantum walks with hamiltonians. *arXiv preprint arXiv:1605.02774*, 2016.

[20] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[21] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[22] Alexandre S. Abreu, Matheus M. Ferreira, Franklin L. Marquezino, and Luis A. B. Kowada. Numeric simulation of quantum algorithm for element distinctness. In *V Workshop-School of Quantum Information and Computation (WECIQ), Campina Grande/PB*, pages 1–5, 2015.

[23] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of Computing*, pages 212–219. ACM, 1996.

A The Element Distinctness Algorithm

In order to make this paper as self contained as possible, we include below the original element distinctness quantum algorithm as Ambainis proposed in his paper [10]. We focus on the single-solution algorithm, although Ambainis had also considered the multiple collision problem in his paper.

First, he gives an upper level description of his algorithm, which we reproduce as Algorithm 1. In this algorithm, we have the preparation of the initial state, the initial query, the outer loop and the final measurement. The detailed explanation for each of these steps were given in Sec. 2 of our paper.

Finally, Ambainis details the quantum walk step of his algorithm, which we reproduce as Algorithm 2. In this sub-routine, we have six steps that move the walker between the Hilbert spaces, i.e., between the partitions of graph. The detailed explanation for each of these steps were given in Sec. 2 of our paper.

Algorithm 1: Single-solution algorithm.

1 Generate the uniform superposition

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y \notin S} |S\rangle |y\rangle .$$

2 Query all x_i for $i \in S$. This transforms the state to

$$\frac{1}{\sqrt{\binom{N}{r}(N-r)}} \sum_{|S|=r, y \notin S} |S\rangle |y\rangle \bigotimes_{i \in S} |x_i\rangle .$$

3 $t_1 = O((\frac{N}{r})^{k/2})$ times repeat:
(a) Apply the conditional phase flip (the transformation $|S\rangle |y\rangle |x\rangle \rightarrow -|S\rangle |y\rangle |x\rangle$) for S such that $x_{i_1} = x_{i_2} = ... = x_{i_k}$ for k distinct $i_1, ..., i_k \in S$.
(b) Perform $t_2 = O(\sqrt{r})$ of the quantum walk (Algorithm 2).

4 Measure the final state. Check if S contains a k-collision and answer "there is a k-collision" or "there is no k-collision", according to the result.

Algorithm 2: One step of quantum walk.

1 Apply the transformation mapping $|S\rangle\,|y\rangle$ to

$$|S\rangle\left(\left(-1+\frac{2}{N-r}\right)|y\rangle+\frac{2}{N-r}\sum_{y'\notin S,y'\neq y}|y'\rangle\right).$$

on the S and y registers of the state in $\mathcal{H}$. (This transformation is a variant of "diffusion transformation" in [23])

2 Map the state from $\mathcal{H}$ to $\mathcal{H}'$, adding y to S and changing x to a vector of length $k+1$ by introducing 0 in the location corresponding to y.

3 Query for x_y and insert it into location of x corresponding to y.

4 Apply the transformation mapping $|S\rangle\,|y\rangle$ to

$$|S\rangle\left(\left(-1+\frac{2}{r+1}\right)|y\rangle+\frac{2}{r+1}\sum_{y'\in S,y'\neq y}|y'\rangle\right).$$

on the y register.

5 Erase the element of x corresponding to new y by using it as the input to query for x_y.

6 Map the state back to $\mathcal{H}$ by removing the 0 component corresponding to y from x and removing y from S.