Quantum Information Processing manuscript No.
(will be inserted by the editor)

Intricacies of Quantum Computational Paths

Luís Tarrataca · Andreas Wichert.

Received: date / Accepted: date

Abstract Graph search represents a cornerstone in computer science and is employed when the best algorithmic solution to a problem consists in performing an analysis of a search space representing computational possibilities. Typically, in such problems it is crucial to determine the sequence of transitions performed that led to certain states. In this work we discuss how to adapt generic quantum search procedures, namely quantum random walks and Grover's algorithm, in order to obtain computational paths. We then compare these approaches in the context of tree graphs. In addition we demonstrate that in a best-case scenario both approaches differ, performance-wise, by a constant factor speedup of two, whilst still providing a quadratic speedup relatively to their classical equivalents. We discuss the different scenarios that are better suited for each approach.

Keywords quantum computational paths · quantum random walks · quantum search

Mathematics Subject Classification (2000) 68Q05 · 68Q12 · 81P40

L. Tarrataca
GAIPS/INESC-ID
Department of Computer Science, Instituto Superior Técnico
Technical University of Lisbon
Avenida Professor Cavaco Silva
2780-990 Porto Salvo, Portugal
Tel.: +351 21423517
E-mail: luis.tarrataca@ist.utl.pt

A. Wichert
GAIPS/INESC-ID
Department of Computer Science, Instituto Superior Técnico
Technical University of Lisbon
Avenida Professor Cavaco Silva
2780-990 Porto Salvo, Portugal
Tel.: +351 214233231
E-mail: andreas.wichert@tagus.ist.utl.pt

1 Introduction

Graph theory assumes a pivotal dimension in computer science, where it is usually employed in order to depict transitions between different computational states. In its most simple form a graph depicts a set of points, referred to as vertexes, that may or may not be interconnected through edges [30]. A generic graph G can be represented as a pair $G = (V, E)$ where V and E represent, respectively, the sets of vertexes and edges. The number of edges at a particular vertex is referred to as the degree of a vertex. Depending on the particular problem being solved, the degree may or may not be constant.

Graph search problems can be represented as a tuple (V, E, S_i, S_g) comprising, respectively, (i) the set of all possible vertexes representing system states; (ii) the set of possible edges of the form (x, y) representing transitions from state x to state y; (iii) a set of initial states with $S_i \subseteq V$ and (iv) a set of goal states with $S_g \subseteq V$. Elementary graph algorithms traditionally focus on searching a graph, *i.e.* systematically following the edges of the graph so as to visit the vertices of the graph [15]. The objectives of the search can be various but typically include discovering specific details about the structure of the graph, determining the presence of certain vertexes or can focus on the specific details regarding possible sequences of edges.

In our case we are particularly interested in determining a sequence of edges P, also known as path, capable of leading the system from an initial state to a goal state, as illustrated by Expression 1, where $d \in \mathbb{Z}^+$ is referred to as the solution depth and $1 \leq k \leq d$. Determining P is important because it reflects a type a logical process of transitions responsible for performing an adequate system evolution. Representative examples include determining the sequence of moves in a game of chess leading to a victory [22], solving Rubik's cube, or developing general mechanisms for problem solving [26].

$$P := ((x_1, x_2), (x_2, x_3), \cdots, (x_{d-1}, x_d) | x_k \in V, x_1 \in S_i, x_d \in S_g) \tag{1}$$

This procedure is illustrated in Figure 1, where we define vertex A to be an initial state and vertex E to be a goal state. Accordingly, possible examples of computational paths leading from A to E include the sequences of edges $((A, B), (B, E))$ and $((A, C), (C, D), (D, E))$.

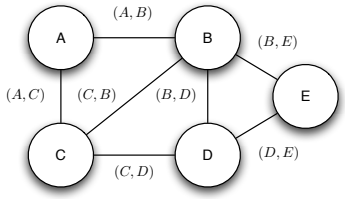

Fig. 1: A graph with five vertexes and seven edges.

1.1 Current Approaches to Quantum Search

Quantum computation has enabled for important performance improvements in search procedures. There are two main quantum procedures capable of performing graph search, namely quantum random walks on graphs and Grover's algorithm. Both of which can deliver a quadratic speedup, *i.e.* $O(\sqrt{N})$ time when the overall objective of the search method resides in evaluating N states and obtaining a goal state. Equivalent classical approaches to search require $O(N)$ time.

Quantum random walks are the quantum equivalents of their classical counterparts ([23] provides an excellent introduction to the area) and were initially approached in [2], [28] in one-dimensional terms, *i.e.* walk on a line. These concepts were then extended to quantum random walks on graphs in [17], [21], and [1]. Quantum random walks can also provide a probabilistic speedup relatively to their classical parts, namely the hitting time for some specific graphs, *i.e.* the time it takes to reach a certain vertex B starting from a vertex A, can be shown to be exponentially smaller [11]. However, these approaches only focus on graph transversal through a simultaneous selection of all possible edges at any given vertex via the superposition principle.

Grover's algorithm [19] represents another alternative to quantum search. After performing $O(\sqrt{N})$ amplitude amplifications the algorithm is capable of evaluating a superposition of N states and deliver, with high probability, a goal state upon measurement. This approach later inspired an equivalent $O(\sqrt{N})$ quantum random walk on graphs [29]. Both approaches rely on building a superposition over all possible bit strings and require that goal states are marked through an oracle and apply in some form Grover's iterate. However, some important differences also exist, namely the quantum random walk algorithm [29]: (1) can only be approximately mapped onto a two-dimensional subspace; (2) final superposition state contains trace contributions of the neighbours of a goal state and (3) applies Grover's iterate to a space of dimension n whilst Grover's algorithms evaluates a space of dimension 2^n.

1.2 Problem

Both quantum random walks and Grover's algorithm only focus on obtaining goal states, and do not consider the dynamics associated with determining the corresponding computational path P. Given that this is such a crucial task in many computational problems, the question naturally arises of how P can be determined in a quantum fashion? Namely, how can these methods be adapted such that P is obtained? More specifically, what are the mechanisms available, alongside respective requisites, limitations, and how do these fare against each other, not only from a computational perspective but also performance-wise.

Furthermore, both approaches to search involve intrinsically different concepts that were initially focused on performing search in general terms. This broadness in purpose can sometimes make it difficult to analyse exactly what stands to be gained from a graph perspective. In order to facilitate such an analysis we choose

to focus on graphs whose structure we can predict. This is the case of graphs whose structural form resembles an inverted tree. Tree search distinguishes itself from ordinary graph search by the exponential-growth of the number of vertexes that need to be examined with each additional depth-layer. Accordingly, it would also be interesting to determine if one of these methods is better suited for, not only determining P, but also to perform an exhaustive analysis of the search space.

1.3 Paper Organisation

The following sections review and describe how to adapt existing quantum search methods and are organised as follows: Section 2 presents the main results regarding quantum random walks on graphs and extends the existing framework such that the computational path is obtained; Section 3 describes Grover's algorithm and what adaptations are required in order to obtain P; Section 4 compares both methods and illustrates the key differences between them from a tree search perspective; Section 5 provides some additional insight regarding the dynamics and impacts associated with set E; the overall conclusions of this work are presented in Section 6.

2 Paths on Quantum Random Walks

There are two types of quantum random walks, namely, discrete- and continuous-time. Both approaches perform a random walk without intermediate measurements. In classical models of computations such as the Turing machine a computational procedure is specified through a set of discrete transitions associating input states to output states. A discrete and finite space is also important when such systems need to be simulated on a classical finite computer [25]. Accordingly, we choose to focus on discrete-time quantum random walks since they allow for a simple mapping between concepts. The next following sections are organised as follows: Section 2.1 provides the necessary details for understanding quantum random walks on graphs; Section 2.2 describes how to extend these concepts in order to store the computational path performed.

2.1 Original Quantum Random Walk on Graphs

The simplest discrete-time quantum random walk on a graph $G = (V, E)$ can be described by a unitary operator U acting upon a Hilbert space $\mathcal{H} = \mathcal{H}_S \otimes \mathcal{H}_C$, where $\mathcal{H}_S$ represents the space associated with vertexes of the graph whilst H_C is associated with a "coin space" [29]. The coin operation owns its name from the classical random walk where a destination state would be chosen according to some probabilistic distribution. Operator U can be described as presented in Expression 2 [1], which is obtained through a composition of operators S and C. Each step of

the quantum walk can be perceived as consisting of two operations, namely [10]: (1) building a superposition over the appropriate neighbour states; and (2) moving the system state to the new target destination. Additionally, operator U employs states of the form $|j\rangle|k\rangle$, where $(j,k) \in E$.

$$U = SC \tag{2}$$

Operator C is responsible for building a superposition in register $|k\rangle$ spanning the neighbours of j. This behaviour is presented in Expression 3 (adapted from [32]), where $deg(j)$ represents the degree of vertex j, *i.e.* the number of edges at vertex j [30]. Let $P_{j,k}$ represent the probability of making a transition from j to k, then in order to preserve unit-length normalization required by quantum states $\sum_{k=1}^{|V|} P_{j,k} = 1$. Assuming a uniform distribution amongst the neighbours of a vertex j this is equivalent to $\sum_{m:(j,m)\in E} \frac{1}{deg(j)} = 1$.

$$C|j\rangle|k\rangle \rightarrow |j\rangle \frac{1}{\sqrt{deg(j)}} \sum_{m:(j,m)\in E} |k \oplus m\rangle \tag{3}$$

The state $|j,k\rangle$ obtained after applying the coin operator indicates to operator S that the system should be moved from state j to state k. Subsequently, operator S performs a conditional shift based on the state of the coin space, as illustrated by Expression 4. The overall effect of Expression 2 emphasizes the notion that the quantum random walk takes places on the edges of the graph.

$$S|j\rangle|k\rangle \rightarrow |k\rangle|j\rangle \tag{4}$$

In [29] the authors proposed an approach capable of obtaining goal states when searching N elements in time $O(\sqrt{N})$. Given that $N = |V|$ this can be restated as $O(\sqrt{|V|})$. Initially, the goal states are marked through an oracle operator $O|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$, where $f(x)$ is a function that return 1 when x is a solution and 0 otherwise. Oracle operators can be perceived as simple verification procedures that verify language membership. This procedure effectively results in an amplitude flipping of the marked states when $|y\rangle$ is initialized in the superposition $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. The authors then proceed by reformulating the the coin operator in order to perform Grover's diffusion operator [19] which was first proposed in [27]. The reformulated coin operator C is presented in Expression 5.

$$C|j\rangle|k\rangle \rightarrow |j\rangle \left(\frac{2}{\sqrt{deg(j)}} \sum_{m:(j,m)\in E} |k \oplus m\rangle\langle k| - \mathbb{I} \right) \tag{5}$$

The quantum random walk search algorithm first initializes the system state $|j\rangle|k\rangle$ to a uniform superposition $|\psi_i\rangle$ covering all vertexes in a graph G and the respective neighbours, as illustrated by Expression 6. State $|\psi_i\rangle$ can be obtained by applying n Hadamard gates to state $|0\rangle$, *i.e.* $H^{\otimes n}|0\rangle$, where n is the number of bits required to encode the elements in $\mathcal{H}$. This procedure assumes that the number of

states employed is a power of two, which simplifies analysis. However, for the remaining cases this procedure may have an adverse impact on system performance since a larger than necessary search space needs to be examined [31]. The algorithm then proceeds by applying the oracle followed by unitary operator SC. This process is repeated for $\frac{\pi}{2}\sqrt{2^n}$ times allowing for a superposition state $|\psi_f\rangle$ to be obtained that is primarily composed of the marked state. Each application of SC effectively increases the amplitude of the goal state [29]. However, as the authors point out, $|\psi_f\rangle$ also possesses small contributions from the closest neighbours to the solution state [29]. The procedure is concluded by performing a measurement on the quantum register, yielding with high probability a solution state.

$$|\psi_i\rangle = \frac{1}{\sqrt{|V|}} \sum_{v \in V} |v\rangle \frac{1}{\sqrt{deg(v)}} \sum_{m:(v,m) \in E} |m\rangle \tag{6}$$

2.2 Adapting Quantum Random Walks

In order for the computational path performed by the walk to be stored an auxiliary operator needs to be introduced, respectively R_t. The operator is responsible for copying the edge transition performed at time step t of the walk to an additional memory register $|m\rangle$. This register should have an adequate length in order to store the appropriate sequence of transitions. Accordingly, since the algorithm requires $O(\sqrt{|V|})$ time this means that $|m\rangle$ should have length $d = \sqrt{|V|}$, *i.e.* $|m\rangle = |m_0, m_1, \cdots, m_{d-1}\rangle$. This behaviour is illustrated in Expression 7 where m_k represents an individual auxiliary memory slot with $n = \lceil \log_2 |E| \rceil$ bits. As a result the overall memory register $|m\rangle$ will require $d \times n$ bits. After each step t of the quantum walk, the application of operator R_t ensures that, for each element of the superposition, the associated transition will be stored in register $|m\rangle$. This procedure is performed by applying the reversible action $m_t \oplus (j, k)$.

$$R_t|j,k\rangle|m_0, m_1, \cdots, m_d\rangle \rightarrow |j,k\rangle|m_0, m_1, \cdots, m_t \oplus (j,k), \cdots, m_{d-1}\rangle \tag{7}$$

Operator R_t is unitary since it performs a bijection between input and output states. Namely, any irreversible logical function f can be made reversible by introducing a constant number of auxiliary input and output bits to a logical gate [24]. This procedure produces as a result a unitary operator $U_f|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$. Since the original input states $|j\rangle$, $|k\rangle$, and the auxiliary memory slots m_k are part of the outputs it immediately follows that R_t is unitary. The original version of the algorithm would require an operator of the form $(SC)^d$. However, by introducing operator R_t we need to be careful in order to guarantee that index t is properly updated. The overall operator sequence for a quantum walk requiring d steps is described in Table 1. Once the final measurement on superposition $|\psi_f\rangle$ is performed the computational state will translate not only the goal state obtained but also the computational path leading to it.

Walk length	Operator Sequence
1	$(R_0SC)\|\psi\rangle$
2	$(R_1SC)(R_0SC)\|\psi\rangle$
$\vdots$	$\vdots$
d	$(R_{d-1}SC)(R_{d-2}SC)\cdots(R_0SC)\|\psi\rangle$

Table 1: Operator sequence for the modified quantum random walk search algorithm.

3 Paths on Grover's algorithm

Grover's algorithm was first proposed in [19] and subsequently published as a letter in [20]. Grover's algorithm is able to search a superposition spanning N elements in time $O(\sqrt{N})$. This approach was later extended in order to search for only the first k bits of an n bit goal state in [18]. This upper bound of $O(\sqrt{N})$ queries was shown to be asymptotically optimal when using oracle operators. This result was first proven in [7], with earlier versions of the manuscript appearing before Grover's algorithm [6]. A simple closed-form formula for the probability of success after any given number of iteration was given in [8]. This result was used to proof that even if other quantum algorithms are devised that deliver a solution with non-random probability they are still not able to provide a meaningful speedup. Zalka then showed that $\frac{\pi}{4}\sqrt{N}$ oracle queries are required in order to output a goal state with maximum probability [33]. The following sections are organised as follows: Section 3.1 presents the basic details regarding Grover's algorithm; Section 3.2 explains how to adapt these concepts such that the computational path can be obtained upon the measurement.

3.1 Grover's algorithm

The quantum search algorithm employs a system state $|q\rangle|a\rangle$ representing, respectively, the query and answer register. Register $|q\rangle$ is configured with query elements belonging to a set S of size N and is placed in a superposition $|\psi\rangle = \sum_{x=0}^{2^{n-1}} \frac{1}{\sqrt{2^n}}|x\rangle$ where n is the number of bits required to encode the set of possible values contained in S, *i.e.* $n = \lceil \log_2 |S| \rceil$. The states present in the superposition are then marked by an oracle operator similar to the one employed by quantum random walks. This procedure effectively marks with an amplitude flip the goal states. The algorithm then proceeds by applying operator $G = HU_{0^\perp}HU_f$, where U_f, H and $U_{0^\perp}$ are, respectively, the oracle, the Hadamard gate, and an n-qubit phase shift operator. The latter inverts the amplitude of all states orthogonal to state $|0\rangle$. Each application of G adds close to $\frac{2}{\sqrt{N}}$ to the amplitude of marked states and slightly decreases the amplitude of the remaining states such that the overall unit-length norm is still preserved [24]. After performing $O(\sqrt{N})$ amplifications the algorithm is capable of evaluating a superposition of N states and deliver, with high probability, a goal state upon measurement.

3.2 Adapting Grover's algorithm

Whilst the appropriateness of quantum random walks for searching graphs seems fairly natural, Grover's application to the same problem is not immediately apparent. Mainly, this is because of the original formulation proposed by Grover in [19] which was not formulated in terms of graph search, *i.e.* with a set of states V and transitions E. Instead, the quantum search algorithm focused exclusively on analysing the elements of a set S. Accordingly, in order to obtain graph-like search behaviour from Grover's algorithm some of the information contained in a graph $G = (V, E)$ needs to be embedded into the verification procedure performed by oracle O.

Given that oracle U_f verifies if an input state is a goal state, one possible strategy resides in evaluating if a given initial input state alongside a sequence of transitions leads to a goal state, an approach proposed in [31]. This encoding is illustrated in Expression 8, where the query register is decomposed into two components, namely $|s\rangle$ where $s \in S_i$ and a sequence of edge transitions where each pair $(e_k, e_{k+1}) \in E$. Applying Grover's algorithm requires initializing the query register in a superposition state spanning all admissible possible path configurations for the corresponding state s. Consequently, the function employed by oracle U_f needs to be defined in order to reflect these changes. This operation is depicted in Expression 9 which can be performed in linear-time since it represents a simple verification procedure. Notice that with this approach the quantum superposition will span all the sequence of transitions up to depth d. This means that if we consider an average degree b then the upper-bound complexity will be $O(\sqrt{b^d})$.

$$|q\rangle|a\rangle = |s\rangle|(e_0, e_1), (e_1, e_2), \cdots, (e_{d-2}, e_{d-1})\rangle|a\rangle \tag{8}$$

$$f(s, (e_0, e_1), \cdots, (e_{d-2}, e_{d-1})) = \begin{cases} 1\ , \text{ if } ((e_0, e_1), \cdots, (e_{d-2}, e_{d-1})) \text{ applied to } S \text{ is in } S_g \\ 0\ , \text{otherwise} \end{cases} \tag{9}$$

4 Comparing both methods

Although both procedures previously described deliver a quadratic speedup over their classical counterparts, they nonetheless still grow in an exponential manner, albeit expressed as different functions. Namely, quantum random walks execute in time $O(\sqrt{|V|})$ whilst Grover's algorithm upper-bound complexity is $O(\sqrt{b^d})$. Hence, it is important to ask the question of how do these functions compare against each other? Section 4.1 provides the necessary background details in order to analyse these approaches from a tree search perspective. We then analyse both methods through an effort function that considers the total number of states evaluated in Section 4.2.

4.1 Tree search

In order to answer this question we will focus on a particular type of graph, namely tree search graphs. Tree search problems can be perceived as a subset of elementary graph theory representing acyclic connected graphs where each vertex has zero or more children nodes and at most one parent node, as illustrated in Figure 2. Traditionally, tree search literature refers to a vertex as a node. In addition, the degree of a node is referred to as branching factor and is usually represented through variable b.

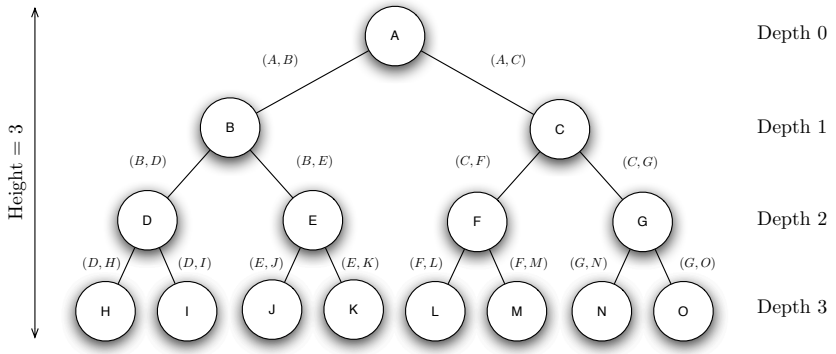

Fig. 2: A search tree with binary branching factor and depth 3 alongside all possible paths.

Quantum random walks can be employed in order to evaluate binary formulas, which represents a form of tree search. The original Grover's algorithm can be adapted to evaluate the logical OR of N bits. The quantum search algorithm can even be employed to evaluate AND-OR trees in time $O(\sqrt{N}\log N)$ as was demonstrated in [9], where N is the number of nodes in the tree. Ambainis proved that computing an AND of ORs requires time $\Omega(\sqrt{N})$ [3]. In addition, every NAND formula of size N can be evaluated in time $N^{\frac{1}{2}+o(1)}$ [13]. Ambainis then presented an $O(\sqrt{N})$ discrete query quantum algorithm for evaluating balanced binary NAND formulas [4] which was also proven to be optimal [5]. In [16] a continuous time quantum random walk algorithm was presented capable of evaluating a balanced AND-OR tree in time $O(\sqrt{N})$. A discrete version of this algorithm was later presented [12] requiring $N^{\frac{1}{2}+o(1)}$ queries. These results were later employed to demonstrate how to evaluate minmax trees with $N^{\frac{1}{2}+o(1)}$ [14].

4.2 Geometric growth

We will start by performing an analysis on how these functions behave in the context of tree search. Note that with tree search the dimension of set V grows

exponentially fast, since each additional level of depth adds b^d to V. As a result, the method based on quantum random walks will need to evaluate $b^0 + b^1 + \cdots + b^d$ states. In contrast, the approach based on Grover's algorithm will only evaluate the possible computational paths, which represent the number of leaf nodes, respectively b^d. In practice the latter should outperform the former since $b^d < b^0 + b^1 + \cdots + b^d$. But precisely how much of an improvement is obtained? Since quantum random walks need to examine more states we can view Grover's algorithm performance as the standard to which we wish to compare. This means that random walks will perform an additional effort relatively to the quantum search algorithm. Accordingly, we can choose to represent the supplementary effort performed to be represented as a function of the branching factor b and depth d, respectively $\xi(b, d)$, as illustrated in Expression 10. Notice that for $d > 0, \xi(b, d) > 1$ since the last term of the series is $\frac{b^d}{b^d} = 1$.

$$\xi(b, d) = \frac{\sum_{k=0}^{d} b^k}{b^d} = \frac{1}{b^d} \frac{1 - b^{d+1}}{1 - b} \tag{10}$$

Figure 3 illustrates the additional effort required to search a binary tree up to depth level $d = 100$. As it is possible to verify the function rapidly grows until approximately depth level $d = 25$ but then stabilizes when $\xi \simeq 2$. In practice, this means that for this specific case both approaches differ in complexity by a constant factor of two. Hence, if Grover's algorithm considers $b^d = 2^d$ states, the random walk approach will need to evaluate twice the number of nodes, *i.e.* $2 \times 2^d = 2^{d+1}$. Consequently, for binary tree search this is equivalent, performance-wise, to extending the search by an additional depth limit, which represents a significant hindrance. This result is also valid from a classical point of view.

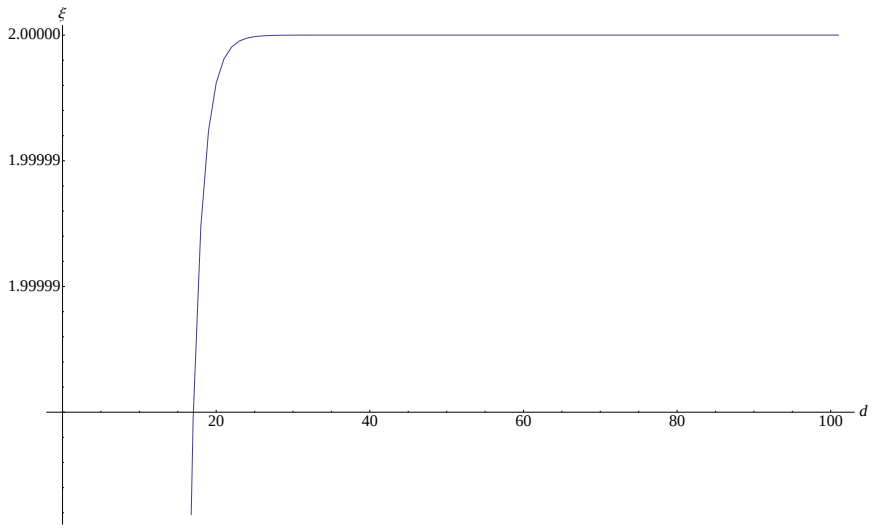

Fig. 3: The additional effort performed by quantum random walks relatively to Grover's algorithm when searching a binary tree up to depth level 100.

Expression 10 represents a geometric series with a common ratio of b which for non-trivial computation is always greater than one. As a result the series will never fully converge. This fact makes it difficult to predict exactly how ξ will behave as both b and d grow towards infinity. The analysis is somewhat simplified if we consider that we are dealing with a ratio of exponential growth functions. As a result, the last last p terms of the series can be perceived as being the ones that contribute the most to the effort performed. The initial $d-p$ terms drop off suddenly and stop contributing in a meaningful manner to the overall effort. This p-error approximation is presented in Expression 11, where $p \in \mathbb{N}$ is chosen so that $d-p \geq 0$.

$$\xi(b,d,p) = \frac{\sum_{k=d-p}^{d} b^k}{b^d} = \frac{b^{d-p} + b^{d-p+1} + \cdots + b^{d-1} + b^d}{b^d} = \frac{1}{b^p} + \frac{1}{b^{p-1}} + \cdots + \frac{1}{b^1} + \frac{1}{b^0} \tag{11}$$

By focusing on the p-error it becomes possible to effectively forgo the depth variable from the equation. As b grows towards infinity, and assuming a constant error p, the only relevant term will be the last one, respectively $\frac{b^d}{b^d}$. Hence, up to p-error, $\lim_{b \to +\infty} \xi(b,d,p) = 1$. The constant factor of two presented in Figure 3 thus illustrates an optimal performance gain, since any increases in branching factor and depth will only exacerbate the rate at which the remaining elements, other than the last one, stop contributing significantly to the effort.

5 Final Considerations

For some specific applications knowing in advance the precise form of sets V and E maybe problematic given the sheer number of possibilities to consider, and consequently, to specify. For example, consider a chess playing application where V may be represented in more general terms as $V := \{x | x \text{ is an admissible chess state}\}$. The same process could be applied to set E, since it would also be infeasible to completely specify all possible edges. In these situations, this issue is usually tackled through the use of symbolical rules γ belonging to a set R, which has the form presented in Expression 12, where Γ^* is a set of strings over a finite nonempty set Γ. The elements of Γ^* are an integral component of the overall state description. The application of any of these rules effectively verifies if an input state x meets certain conditions, and if so perform a computational action that results in a transition to state y, respectively represented by the tuple $((x,y),\gamma)$, where $x, y \in V$ and $\gamma \in R$.

$$R := \{(precondition, action) | precondition, action \in \Gamma^*\} \tag{12}$$

Notice that a single rule γ has the potential to: (1) be applied to multiple input states depending on whether or not the conditions are met by these; and (2) generate multiple output states depending on the original node. This process is represented in Figure 2 illustrating a binary tree. Suppose that node K in Figure 2 is a goal state, then the computational path leading to it can be perceived as

applying the sequence of rules $0, 1, 1$ which map, respectively, to the transitional tuples $((A, B), 0)$, $((B, E), 1)$ and $((E, K), 1)$.

By employing set R we are able to obtain the same computational behaviour represented by the original set of transitions E, although through a much smaller specification. In order to accommodate for these requirements, the specific terms of the definition may be modified such that a stronger emphasis is placed on determining sequences of rules that lead to goal states. Accordingly, we can opt to represent such problems by a tuple (V, R, S_i, S_g), and where the overall objective consists in determining P, but this time reformulated in order to store sequence of rules, as illustrated in Expression 13.

$$P := (((x_1, x_2), \gamma_1), ((x_2, x_3), \gamma_2), \cdots, ((x_{d-1}, x_d), \gamma_d) | x_k \in V, x_1 \in S_i, x_d \in S_g, \gamma_i \in R) \tag{13}$$

Expression 13 can be simplified if we consider that given an input state x alongside a rule γ it is a simple task to determine the successor state y. Accordingly, we can opt to merely keep track of the sequence of rules performed, since the original edges of a path P can be reconstituted in linear time given an initial state $i \in S_i$, as presented in Expression 14.

$$P := (\gamma_1, \gamma_2, \cdots, \gamma_d | \gamma_i \in R) \tag{14}$$

Although both approaches achieve the same functional purposes of determining P they differ substantially on the costs required for obtaining the computational path. Namely, by utilising rules we are able to describe a set R that will impose a maximum branching factor b. As a result the computation would execute in $O(\sqrt{b^d})$ time. Alternatively, employing a traditional graph representation would require specifying b^d edges, which by itself would involve an exponential amount of time. Ultimately, such a procedure would yield an $O(b^d)$ time complexity since there is a "hidden cost" associated with generating V.

6 Conclusions

In this work we presented two adaptations to existing quantum search methods in order to store the paths leading to solutions states. Additionally, we showed that the best one should be expected, performance-wise, is a constant speedup of two that tends to dilute when the branching factor is increased. Although both approaches provide a quadratic speedup relatively to their classical counterparts, they are intrinsically different. The path approach based on quantum random walks performs in $O(\sqrt{|V|})$ time and upon measurement produces a path whose length grows as a function of the search space. This means that typical exponential growth search spaces will also deliver a computational path with an exponential number of transitions. With quantum random walks not only is the complexity exponential time-wise but the space required to store the path also grows exponentially fast.

Further, the path obtained may be non-optimal since nothing in quantum random walks theory prevents loops from occurring.

Grover's methods allows for a time of $O(\sqrt{b^d})$ with the associated path length expressed as a function of the search depth d, which grows linearly. In addition there is a greater control over what computational paths should be processed since one has the ability to build path-verifying oracle operators. Although this may first be perceived as advantageous over quantum random walks it is not clear how to perform node evaluation that requires non-linear transversal of the tree. Namely, it would be interesting to determine how to evaluate a node whose value is calculated as a function of its children, which is precisely the type of computational behaviour required by procedures such as the minimax algorithm. This difficulty arises since each basis state of the superposition employed represents individual computational paths.

Accordingly, with the current form of these procedures, deciding which of the two methods should be employed is a matter of elaborating a careful analysis of the problem being tackled. If no requisites exist involving the analysis of a node based on the forest of nodes starting at it, then the method based on Grover's algorithm is adequate. Otherwise, the approach based on random walks is the most suitable.

Acknowledgements

This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and FCT grant DFRH - SFRH/BD/61846/2009.

References

1. Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proceedings of ACM Symposium on Theory of Computation (STOC'01), pp. 50–59 (2001). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0012090
2. Aharonov, Y., Davidovich, L., Zagury, N.: Quantum random walks. Phys. Rev. A **48**(2), 1687–1690 (1993). DOI 10.1103/PhysRevA.48.1687
3. Ambainis, A.: Quantum lower bounds by quantum arguments. eprint arXiv:quant-ph/0002066 (2000)
4. Ambainis, A.: A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. ArXiv e-prints (2007)
5. Ambainis, A., Childs, A., Reichardt, B.: Any and-or formula of size n can be evaluated in time $n^{\frac{1}{2}+o(1)}$ on a quantum computer. In: Foundations of Computer Science, 2007. FOCS '07. 48th Annual IEEE Symposium on, pp. 363 –372 (2007). DOI 10.1109/FOCS.2007.57
6. Beals, R., Buhrman, H., Cleve, R., Mosca, M., de Wolf, R.: Quantum lower bounds by polynomials. J. ACM **48**, 778–797 (2001). DOI http://doi.acm.org/10.1145/502090.502097. URL http://doi.acm.org/10.1145/502090.502097
7. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing (1997). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9701001
8. Boyer, M., Brassard, G., Hoeyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik **46**, 493 (1998). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9605034

9. Buhrman, H., Cleve, R., Wigderson, A.: Quantum vs. classical communication and computation. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing, STOC '98, pp. 63–68. ACM, New York, NY, USA (1998). DOI 10.1145/276698.276713. URL http://doi.acm.org/10.1145/276698.276713
10. Childs, A.M.: LECTURE 14: Discrete-time quantum walk. University of Waterloo (2011). URL http://www.math.uwaterloo.ca/~amchilds/teaching/w11/l14.pdf
11. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.: Exponential algorithmic speedup by quantum walk. In: Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003), pp. 59–68 (2003)
12. Childs, A.M., Cleve, R., Jordan, S.P., Yonge-Mallo, D.: Discrete-query quantum algorithm for nand trees. Theory of Computing **5**(1), 119–123 (2009). DOI 10.4086/toc.2009.v005a005. URL http://www.theoryofcomputing.org/articles/v005a005
13. Childs, A.M., Reichardt, B.W., Spalek, R., Zhang, S.: Every NAND formula of size N can be evaluated in time $N^{\frac{1}{2}+o(1)}$ on a quantum computer. eprint arXiv:quant-ph/0703015 (2007)
14. Cleve, R., Gavinsky, D., Yonge-Mallo, D.: Quantum algorithms for evaluating min-max trees. In: Y. Kawano, M. Mosca (eds.) Proceedings of Theory of Quantum Computation, Communication, and Cryptography (TQC 2008), vol. 5106, pp. 11–15. Springer Berlin / Heidelberg (2008)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2/e. MIT Press (2001)
16. Farhi, E., Goldstone, J., Gutmann, S.: A quantum algorithm for the hamiltonian nand tree. Theory of Computing **4**(1), 169–190 (2008). DOI 10.4086/toc.2008.v004a008. URL http://www.theoryofcomputing.org/articles/v004a008
17. Farhi, E., Gutmann, S.: Quantum computation and decision trees. Phys. Rev. A **58**(2), 915–928 (1998). DOI 10.1103/PhysRevA.58.915
18. Grover, L., Rudolph, T.: How significant are the known collision and element distinctness quantum algorithms? QUANTUM INFORMATION AND COMPUTATION **4**, 201 (2004). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0309123
19. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 212–219. ACM, New York, NY, USA (1996). DOI http://doi.acm.org/10.1145/237814.237866
20. Grover, L.K.: Quantum mechanics helps in searching for a needle in a haystack. Physical Review Letters **79**, 325 (1997). URL doi:10.1103/PhysRevLett.79.325
21. Hogg, T.: A framework for structured quantum search. PHYSICA D **120**, 102 (1998). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9701013
22. Hsu, F.h.: Behind Deep Blue: Building the Computer That Defeated the World Chess Champion. Princeton University Press (2002)
23. Hughes, B.D.: Random Walks and Random Environments, vol. Volume 1: Random Walks. Oxford University Press, USA (1995)
24. Kaye, P.R., Laflamme, R., Mosca, M.: An Introduction to Quantum Computing. Oxford University Press, USA (2007)
25. Kempe, J.: Quantum random walks - an introductory overview. Contemporary Physics **44**, 307 (2003). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0303081
26. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial Intelligence **33**(1), 1–64 (1987)
27. Moore, C., Russell, A.: Quantum Walks on the Hypercube. eprint arXiv:quant-ph/0104137 (2001)
28. Nayak, A., Vishwanath, A.: Quantum walk on the line. Tech. rep., DIMACS Technical Report (2000). URL http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0010117
29. Shenvi, N., Kempe, J., Whaley, K.B.: Quantum random-walk search algorithm. Phys. Rev. A **67**(5), 052,307 (2003). DOI 10.1103/PhysRevA.67.052307
30. Sipser, M.: Introduction to the theory of computation. Computer Science Series. Thomson Course Technology (2006). URL http://books.google.pt/books?id=SV2DQgAACAAJ
31. Tarrataca, L., Wichert, A.: Tree search and quantum computation. Quantum Information Processing **10**(4), 475–500 (2011). URL http://dx.doi.org/10.1007/s11128-010-0212-z. 10.1007/s11128-010-0212-z

32. Watrous, J.: Quantum simulations of classical random walks and undirected graph connectivity. CoRR **cs.CC/9812012** (1998)
33. Zalka, C.: Grover's quantum searching algorithm is optimal. Physical Review A **60**, 2746 (1999). URL `http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9711070`