A hierarchical sorting oracle

Luís Tarrataca* and Andreas Wichert

GAIPS/INESC-ID
Department of Computer Science, Instituto Superior Técnico
{luis.tarrataca,andreas.wichert}@ist.utl.pt

Abstract. Classical tree search algorithms mimic the problem solving capabilities traditionally performed by humans. In this work we propose a unitary operator, based on the principles of reversible computation, focusing on hierarchical tree search concepts for sorting purposes. These concepts are then extended in order to build a quantum oracle which, combined with Grover's quantum algorithm, can be employed as a quantum hierarchical search mechanism whilst taking advantage of a quadratic speedup. Finally, we show how the developed model can be extended in order to perform a N-level depth-limited search.

Key words: quantum search; tree search; artificial intelligence.

1 Introduction

Tree search algorithms assume a crucial role in artificial intelligence where they are employed to model problem solving behaviour. Typically, such problems can be described by a tuple (S_i, S_g, R) where S_i represents a finite set of initial states, R a finite set of actions and S_g a finite set of goal states. The objective of such algorithms consists in determining a sequence of actions leading from an initial state to a goal state. A wide range of problems has been formulated in terms of hierarchical search procedures *e.g.* game playing programs and robot control systems. Such behaviour requires the ability to determine what state is obtained after applying an action to a given state. This process is illustrated in Figure 1 where a set of possible actions, respectively $R = \{0, 1\}$, is applied to a root node A producing in the process a binary tree. The cardinality of the set of available actions is also referred to as the branching factor b. At a search depth level d there exist a total of b^d leaf nodes. Each leaf node translates into the state reached after having applied d actions, *e.g.* node I is reach after applying actions 0, 0 and 1. We will refer to set of actions leading to a leaf node as the path taken during the tree search.

Grover's quantum search algorithm [1] allows for a quadratic speedup to be obtained in search procedures. The algorithm performs a generic search for n-bit solutions amongst the 2^n possible combinations by employing the quantum

* This work was supported by Fundação para a Ciência e Tecnologia (FCT) (INESC-ID multiannual funding) through the PIDDAC Program funds and FCT grant DFRH - SFRH/BD/61846/2009.

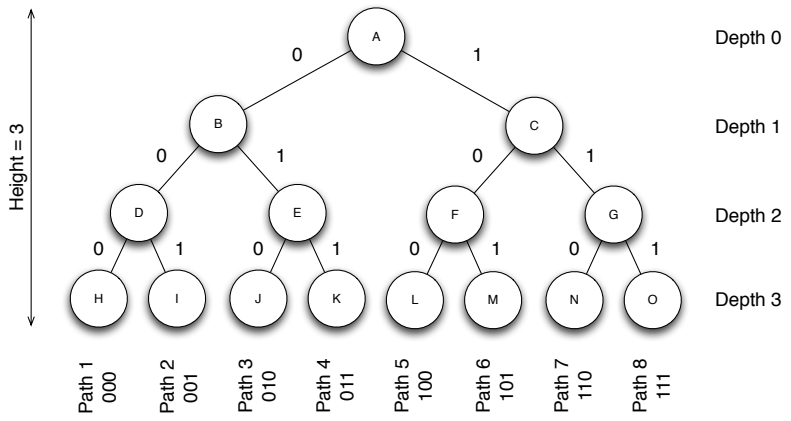

Fig. 1: The possible paths for a binary search tree of depth 3.

superposition principle alongside an oracle O in order to query many elements of the search space simultaneously. The oracle is responsible for determining which strings correspond to solutions and it should be able to do so in polynomial time. This behaviour is similar to the NP class of problems whose solutions are verifiable in polynomial time $O(n^k)$ for some constant k, where n is the size of the input's problem. Oracle O behaviour can be formulated as presented in Expression 1, where $|x\rangle$ is a n-bit query register, $|c\rangle$ is a single bit answer register where the output of $g(x)$ is stored. Function $g(x)$ is responsible for checking if x is a solution to the problem, outputting value 1 if so and 0 otherwise. Grover's original idea only focused on developing a generic search mechanism and did not have hierarchical search in mind. In this work we consider the impact of incorporating classical search concepts alongside Grover's algorithm into a hybrid quantum search system capable of solving instances of the hierarchical sorting problem.

$$O : |x\rangle|c\rangle \mapsto |x\rangle|c \oplus g(x)\rangle \tag{1}$$

The remainder of this work is organized as follows: Section 2 introduces the concepts of the hierarchical sorting problem; Section 3 presents the required reversible circuitry for our proposition alongside an oracle mapping capable of being integrated with Grover's algorithm; Section 4 discusses how such an oracle can be applied alongside Grover's algorithm and how our proposition differs from quantum random walks on graphs; Section 5 presents the conclusions of our work.

2 Sorting

The sorting problem may be defined in terms of the application of a problem-specific set of actions with the objective of determining a sequence of actions that

produces a goal state. For some problems, the action set may convey an increasing element order, whilst for others the final arrangement may only be expressed through condition-action pairs. For some problems the only viable procedure consists in performing an exhaustive examination of all possible actions until goal states are found. *E.g.* suppose we wish to sort a list containing elements of an alphabet $\sum = \{a, b, c, d\}$ and that the dimension of the list, E, is fixed to four elements. In each computational step we can perform operation $S(x, y)$, responsible for switching the elements in position x and y. If repetitions are not allowed then is is possible to check that a total of $\binom{|\sum|}{2} = \binom{4}{2}$ possible combinations exist, where $|\sum|$ represents alphabet length. Accordingly, we are able to define the set of possible actions $R = \{S(0,1), S(0,2), S(0,3), S(1,2), S(1,3), S(2,3)\}$, and apply it an initial state, as illustrated in Figure 2.

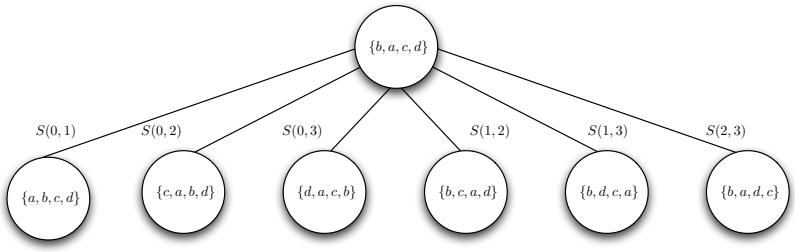

Fig. 2: A search of depth 1 with a branching factor $b = |R| = 6$ applied to an initial state $\{b, a, c, d\}$ and goal state $\{a, b, c, d\}$.

3 Oracle Development

Changes occurring to a quantum state can be described with quantum circuits containing wires and elementary quantum gates to carry around and manipulate quantum information [2]. Mathematically, state evolution can be expressed unitary operators. A matrix A is said to be unitary if A's transpose complex conjugate, denoted by A^{*^T}, or simply by $A^\dagger$, is also the inverse matrix of A [3]. In this notation each matrix column describes the transformation suffered at a specific column index, *i.e.* a permutation. These concepts are related to reversible computation theory, ergo our approach relies on developing a reversible circuit capable of sorting the 4-length list element presented in Section 1. Therefore, we need to represent the overall state in a binary fashion. More specifically, $\lceil \log_2 |\sum| \rceil = \lceil \log_2 4 \rceil = 2$ bits are required to encode the symbols of the alphabet, each of which can be represented as presented in Table 1. This implies that a total of 8 bits will be employed to represent each list. Let Table 2 represent the encodings for the root state and the goal state associated with the sorting example of Figure 2. Conceptually, our reversible circuit will require the ability

to: (1) determine if a state is a goal state; and (2) given a state and an action determine the new state obtained. These two requirements will be discussed, respectively, in Section 3.1 and Section 3.2. Section 3.3 presents the details of the overall circuit.

b_0 b_1	Element
0 0	a
0 1	b
1 0	c
1 1	d

Table 1: Binary encoding for each symbol of $\sum$

Position	0	1	2	3
Bits	b_0 b_1	b_2 b_3	b_4 b_5	b_6 b_7
$\{b, a, c, d\}$	0 1	0 0	1 0	1 1
$\{a, b, c, d\}$	0 0	0 1	1 0	1 1

Table 2: Binary encodings for the initial and goal states of Figure 2.

3.1 First Requirement

Tackling the first requirement requires developing a gate capable of receiving as an argument a binary string representing the state and testing if it corresponds to a goal state. This computational behaviour can be represented through an irreversible function f, as illustrated in Expression 2. It is possible to obtain a reversible mapping of an irreversible function f with the form presented in Expression 3, where x represents the input and c an auxiliary control bit [4].

$$f(\underbrace{b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7}_{\text{state}}) = \begin{cases} 1 \text{ if state } \in S_g \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

$$(x, c) \mapsto (x, c \oplus f(x)) \tag{3}$$

From Expression 3 we know that the inputs should also be part of the outputs. The only issue is due to the result bit, which requires that a single control bit be provided as an input. Therefore, any potential gate would require 9 input and output bits, 8 of which are required for representing the state and 1 bit serving as control. This gate, which we will label as the goal state unitary operator, is illustrated in Figure 3. Table 3 showcases the gate's behaviour for a selected number of states, where $f(b)$ denotes $f(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$. Notice that when the gate determines that the input state $\in S_g$ it effectively switches the control bit, as highlighted in Table 3. Mathematically, we need to specify the

set of column permutations. Let T denote the unitary operator responsible for implementing the behaviour of function f. T is a matrix with dimensions $2^9 \times 2^9$. From Table 3 it should be clear that only two input states map onto other states rather than themselves. Namely, $T|54\rangle \rightarrow |55\rangle$ and $T|55\rangle \rightarrow |54\rangle$. Accordingly, the 54^{th} column of T should permute to state $|55\rangle$, and the 55^{th} column map to state $|54\rangle$. All other remaining states would continue to map onto themselves.

Inputs									Outputs								
b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	c	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	$c \oplus f(b)$
0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	1
0	0	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	0	0
0	1	1	0	1	1	0	0	1	0	1	1	0	1	1	0	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	0	0	1	0	0	0	1	1	1	0	0	1	0	0	0
1	1	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1

Table 3: A selected number of results from the truth table of the goal state unitary operator.

3.2 Second Requirement

The second requirement combined alongside with Expression 3 implies that the new state should be presented alongside the original one. Additionally, we are interested in applying a switch action if and only if the input state $\notin S_g$. As a consequence, we can opt to develop a new function g which includes in its definition a reference to function f. Our main concern resides in how to output the new state in a reversible manner since we are interested in having 8 result bits representing the new state. Expression 3 can be extended in order to accommodate any number of control bits, as illustrated by Expression 4 where c_i are control bits, and $f(x) = (y_0, y_1, \cdots, y_{n-1})$ with $y_i \in \{0, 1\}$. Function g is responsible for producing the new state by taking into account the current state and four bits, respectively (m_0, m_1) and (m_2, m_3), representing, respectively, the arguments x and y of the switching function $S(x, y)$. Accordingly, let $g : \{0, 1\}^{12} \rightarrow \{0, 1\}^8$ with $g(b, m) = (y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$, where b denotes the input state $(b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$, m the positional bits (m_0, m_1, m_2, m_3) and $(y_0, y_1, y_2, y_3, y_4, y_5, y_6, y_7)$ the resulting state. Then, g's behaviour has the form presented in Expression 5. The corresponding gate therefore has (1) 8 input and output bits for the current state; (2) 4 input and output bits describing the switch positions; and (3) 8 control and result bits in order to account for the new

state. The reversible gate, which we will refer to as the switch element operator M, is depicted in Figure 3. The corresponding unitary operator M is a matrix of dimension $2^{8+4+8} \times 2^{8+4+8}$ which can be built in a similar way to T.

$$(x, c_0, c_1, \cdots, c_{n-1}) \mapsto (x, c_0 \oplus y_0, c_1 \oplus y_1, \cdots, c_{n-1} \oplus y_{n-1}) \tag{4}$$

$$g(b,m) = \begin{cases} (b_2, b_3, b_0, b_1, b_4, b_5, b_6, b_7) \text{ if } f(b) = 0 \text{ and } m = (0,0,0,1) \\ (b_4, b_5, b_2, b_3, b_0, b_1, b_6, b_7) \text{ if } f(b) = 0 \text{ and } m = (0,0,1,0) \\ (b_6, b_7, b_2, b_3, b_4, b_5, b_0, b_1) \text{ if } f(b) = 0 \text{ and } m = (0,0,1,1) \\ (b_0, b_1, b_4, b_5, b_2, b_3, b_6, b_7) \text{ if } f(b) = 0 \text{ and } m = (0,1,1,0) \\ (b_0, b_1, b_6, b_7, b_4, b_5, b_2, b_3) \text{ if } f(b) = 0 \text{ and } m = (0,1,1,1) \\ (b_0, b_1, b_2, b_3, b_6, b_7, b_4, b_5) \text{ if } f(b) = 0 \text{ and } m = (1,0,1,1) \\ (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7) \text{ otherwise} \end{cases} \tag{5}$$

3.3 General Circuit

By combining both the switch elements and the goal state gates we are now able to verify if a goal state has been reached after switching two elements. The switch elements operator M already incorporates in its design a test for determining if the gate should be applied or not. Accordingly, we only need to check if the final state obtained corresponds to a goal state. This process is illustrated in Figure 3 where a switch operator M is employed alongside a goal state operator T, where res has the value presented in Expression 6.

$$res = c_8 \oplus f(c_0 \oplus y_0, c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7) \tag{6}$$

Algebraically, the overall circuit behaviour can be expressed as presented in Expression 7, where $I^{\otimes(8+4)} = I \otimes I \otimes \cdots \otimes I$ repeated 12 times, since operator T should only take into consideration bits $c_0, c_1, \cdots, c_8$. The associated unitary operator, respectively presented in Expression 7, acts on Hilbert space $\mathcal{H} = H_b \otimes H_m \otimes H_c$, where H_b is the Hilbert space spanned by the basis states employed to encode the state configuration bits $b = b_0, b_1, \cdots, b_7$, H_m is the Hilbert space spanned by the basis states employed to represent the set of permutations, and H_c is the Hilbert space spanned by the auxiliary control bits.

$$(I^{\otimes 12} \otimes T)M|b_0, b_1, \cdots, b_7, m_0, m_1, m_2, m_3, c_0, c_1, \cdots, c_8\rangle \tag{7}$$

This strategy can be extended in order to apply any number of switch operators, where the output of a switch gate is provided as input to another switch operator. In doing so, we add a guarantee that, if possible, another element permutation is applied to the input state. More specifically, in order to represent each element of the alphabet we require $e = \lceil \log_2 |\sum| \rceil$ bits. Let E represent the element list to be sorted, then an adequate encoding for E will require $b = |E| \times e$ bits, where $|E|$ denotes the list size. Additionally, specifying a list position involves $p = \lceil \log_2 |E| \rceil$ bits. Each switch operator M will thus require a total of

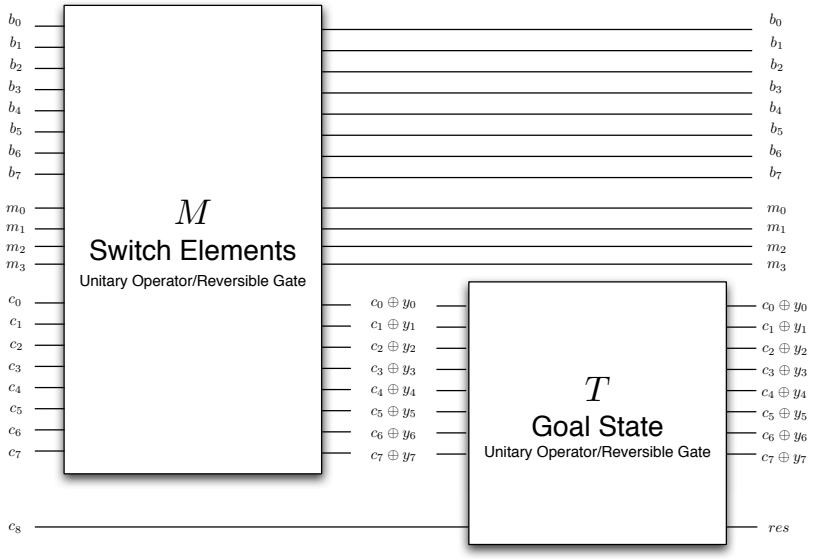

Fig. 3: The reversible circuit responsible for performing the depth-limited search of Figure 2.

$b + p + p + b = 2(b + p)$ input and output bits, and each goal state gate T will require a total of $b+1$ input and output bits. How many bits will be required by the circuit? Suppose we wish to apply m permutation, *i.e.* apply operator M a total of m times. The first operator M_1 requires $2(b+p)$ bits. Since a part of M_1 outputs will be provided as input to M_2 an additional $b+2p$ bits will be added to the circuit. If we extend this reasoning to m applications of M then it is possible to conclude that $2(b + p) + (m - 1)(b + 2p)$ bits will be required to perform the switching operations. Since operator T requires a single control bit this implies that the overall circuit employs a total of $n = 2(b+p) + (m-1)(b+2p)+1$ bits.

Of these n bits $c = n - (b + m \times 2p) = mb + 1$ bits are control, or auxiliary, bits. Furthermore, the sequence of bit indexes after which a switch operator M should be applied is $V = \{0, b+2p, 2(b+2p), 3(b+2p), \cdots, (m-1)(b+2p)\}$. Based on these statements we can describe a general formulation for a sorting circuit C employing operators M and T , as illustrated in Expression 8. Unitary operator C would act on an input register $|x\rangle$ conveying information regarding the initial state, the set of permutations and also the auxiliary control bits. Accordingly, operator C would act upon a Hilbert space $\mathcal{H}$ spanned by the computational basis states required to encode x. Notice that this approach is equivalent to

performing a depth-limited search, one whose number of switch operators T would grow linearly with the depth.

$$C = (I^{\otimes m(2p+b)} \otimes T) \prod_{k \in V} \left(I^{\otimes k} \otimes M\right) \tag{8}$$

Expression 8 needs to be further refined in order to be in conformity with the oracle formulation of Expression 1. which effectively means that all the original inputs, excluding bit c, should also be part of the outputs. This means that the circuit presented in Figure 3 should somehow undo their computation and then store the overall conclusion in an output register, an operation which can be performed by employing a CNOT gate. This behaviour can be obtained by building a mirror circuit, C^{-1}, where each component is the inverse operation of original circuit. Then, with both circuits developed, it is just a matter of establishing the appropriate connections, *i.e.* the outputs of the original circuit are provided as inputs to the mirror. The application of these requirements to the reversible circuit of Figure 3 is presented in Figure 4. The circuits output is presented in Expression 9 where res has the value shown in Expression 6. If the input register $|b\rangle|m\rangle|c\rangle$ is relabeled as $|x\rangle$ then Expression 9 is equivalent to Expression 1.

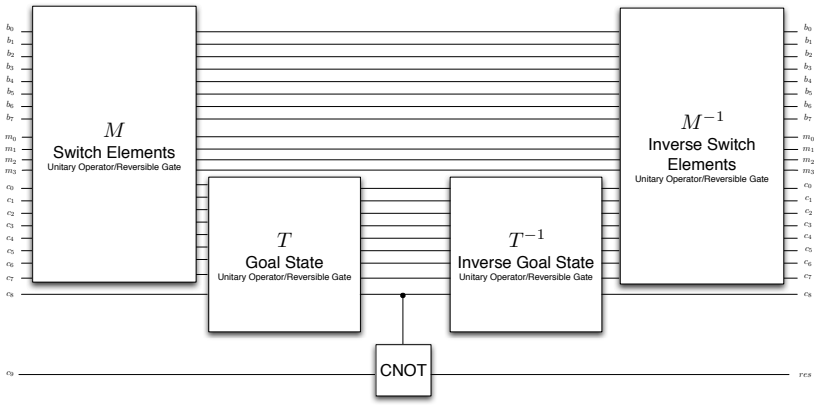

Fig. 4: The oracle formulation of the depth-limited search circuit of Figure 3.

$$O : \underbrace{|b\rangle|m\rangle|c\rangle}_{\text{input}} \quad \underbrace{|c_9\rangle}_{\text{oracle's control bit}} \quad \mapsto |b\rangle|m\rangle|c\rangle|c_9 \oplus res\rangle \tag{9}$$

Alternatively, we can state this result in more general terms by employing unitary operator C, presented in Expression 8, as showcased by Expression 10. In both cases the Hilbert space $\mathcal{H}$ of the input register is augmented with the

basis states required to encode the additional auxiliary control bit, accordingly $\mathcal{H} = \mathcal{H}_b \otimes \mathcal{H}_m \otimes \mathcal{H}_c \otimes \mathcal{H}_{c_{mb+2}}$.

$$O = C^{-1}(I^{\otimes 2(b+p)+(m-1)(b+2p)}CNOT)C|b\rangle|m\rangle|c\rangle|c_{mb+2}\rangle \tag{10}$$

4 Final considerations

Overall, our reversible circuit and the associated oracle O can be perceived as employing a binary string of the form $|b_1b_2b_3b_4b_5b_6b_7b_8 \quad r_1r_2\cdots r_N\rangle$, where r_i represent a sequence of permutations. Accordingly, we are now able to employ Grover's algorithm alongside oracle O and a superposition $|\psi\rangle$. The exact form of $|\psi\rangle$ depends on the specific task at hand, *e.g.* (1) we may be interested in only building a superposition of all possible permutations, a behaviour similar to the depth-limited search presented in Figure 2, or (2) we may set $|\psi\rangle = H^{\otimes k}|0\rangle^{\otimes k}$, where k is the number of bits employed by the input state $|b\rangle|m\rangle|c\rangle$, effectively allowing us to search all possible combinations of initial states and permutations simultaneously. After Grover's algorithm has been applied and upon measuring the superposition state we obtain a state containing the sequence of permutations leading up to a goal state. From a tree search perspective this process can be viewed as a depth-limited search. Classical search strategies require $O(b^d)$ time, where b is the branching factor and d the depth of a solution. If we only take into consideration the dimension of the search space then such a quantum hierarchical search strategy would allow this time to be reduced to $O(\sqrt{b^d})$, effectively cutting the depth factor in half. However, this is a best case scenario since it assumes that the bit encoding strategy always produces viable paths, which is not always true depending on the dimension of the search space or when non-constant branching factors are employed (please refer to [5] for more details).

Finally, from a graph perspective, it is possible to establish some links between the concepts discussed and quantum random walks on graphs. Quantum random walks are the quantum equivalents of their classical counterparts ([6] provides an excellent introduction to the area). Quantum random walks were initially approached in [7], [8], [9] and in one-dimensional terms, *i.e.* walk on a line. These concepts were then extended to quantum random walks on graphs in [10], [11], and [12]. 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 [13]. However, these approaches only focus on graph transversal through a simultaneous selection of all possible edges at any given node, a procedure which is applied through the superposition principle. In contrast, our approach focuses on a simultaneous evaluation of all possible path up to a depth level d with a focus on (1) finding states $\in S_g$ and (2) determining the path leading up to these states.

5 Conclusions

In this work we presented a possible model for a depth-limited search with an emphasis on sorting. The proposed model can be viewed as an hybrid between a pure quantum search mechanism, such as the one detailed in Grover's algorithm, and a classical search system. By combining these concepts we are able to hierarchically search through all possible combinations quadratically faster than its classical counterparts. Our proposal placed a strong emphasis on determining the set of actions leading up to a target node, since this a crucial task for many artificial intelligence applications. Our approach can be also perceived as performing hierarchical search by exploiting the NP class of problems.

References

1. 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, New York, NY, USA, ACM (1996) 212–219
2. Deutsch, D.: Quantum computational networks. In: Proceedings of the Royal Society of London A. Volume 425. (1989) 73–90
3. Hirvensalo, M.: Quantum Computing. Springer-Verlag, Berlin Heidelberg (2004)
4. Kaye, P.R., Laflamme, R., Mosca, M.: An Introduction to Quantum Computing. Oxford University Press, USA (2007)
5. Tarrataca, L., Wichert, A.: Tree search and quantum computation. Quantum Information Processing (2010) 1–26 10.1007/s11128-010-0212-z.
6. Hughes, B.D.: Random Walks and Random Environments. Volume Volume 1: Random Walks. Oxford University Press, USA (1995)
7. Aharonov, Y., Davidovich, L., Zagury, N.: Quantum random walks. Phys. Rev. A **48**(2) (Aug 1993) 1687–1690
8. Meyer, D.: From quantum cellular automata to quantum lattice gases. Journal of Statistical Physics **85**(5) (12 1996) 551–574
9. Nayak, A., Vishwanath, A.: Quantum walk on the line. Technical report, DIMACS Technical Report (2000)
10. Farhi, E., Gutmann, S.: Quantum computation and decision trees. Phys. Rev. A **58**(2) (Aug 1998) 915–928
11. Hogg, T.: A framework for structured quantum search. PHYSICA D **120** (1998) 102
12. Aharonov, D., Ambainis, A., Kempe, J., Vazirani, U.: Quantum walks on graphs. In: Proceedings of ACM Symposium on Theory of Computation (STOC'01). (July 2001) 50–59
13. Childs, A.M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., Spielman, D.: Exponential algoritmic speedup by quantum walk. In: Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003). (September 2003) 59–68