UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

The Quantum Production System

Luís Domingues Tomé Jardim Tarrataca

Supervisor: Doctor Andreas Miroslaus Wichert

Thesis approved in public session to obtain the PhD Degree in
Information Systems and Computer Engineering
Jury final classification: Pass With Merit

Jury

Chairperson: Chairman of the IST Scientific Board

Members of the Committee:

Doctor Mingsheng Ying
Doctor Paulo Alexandre Carreira Mateus
Doctor António Ferreira Pereira
Doctor Andreas Miroslaus Wichert
Doctor Nikola Paunkovic

2013

UNIVERSIDADE DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

The Quantum Production System

Luís Domingues Tomé Jardim Tarrataca

Supervisor: Doctor Andreas Miroslaus Wichert

Thesis approved in public session to obtain the PhD Degree in
Information Systems and Computer Engineering

Jury final classification: Pass With Merit

Jury

Chairperson: Chairman of the IST Scientific Board

Members of the Committee:

Doctor Mingsheng Ying, Distinguished Professor, Faculty of Engineering and Information Technology, University of Technology, Sydney;
Doctor Paulo Alexandre Carreira Mateus, Professor Associado do Instituto Superior Técnico, da Universidade Técnica de Lisboa;
Doctor António Ferreira Pereira, Professor Auxiliar da Universidade de Aveiro;
Doctor Andreas Miroslaus Wichert, Professor Auxiliar do Instituto Superior Técnico, da Universidade Técnica de Lisboa;
Doctor Nikola Paunkovic, Especialista na área em que se insere a tese.

Funding Institutions
Fundação para a Ciência e Tecnologia

2013

To all of those who contributed to my education.

Title Quantum Production System

Name Luís Domingues Tomé Jardim Tarrataca

PhD Degree in Information Systems and Computer Engineering

Supervisor Doutor Andreas Miroslaus Wichert

Abstract

Production system theory is well suited to represent tree search space examination. Quantum computation enables a quadratic speedup in generic search procedures relatively to their classical counterparts. This work presents a model for a quantum production system, which was developed with hierarchical search spaces in mind and that examines: (1) unitary operator development, (2) the implications of different types of branching factors; (3) the consequences of developing a heuristic that can be incorporated into the model. These concepts are then used to present a set of formal definitions for a quantum production system. The model is also compared against quantum random walks on a graph. It is shown that, in a worst-case scenario, both approaches differ in terms of nodes evaluated by a factor of two. An iterative extension to the model is developed and an application to the quantum Halting problem is described. This extension allows for (1) a detection of halt states without interfering with the final result of a computation; (2) the possibility of non-terminating computation and (3) an inherent speedup to occur during computations susceptible of parallelization. Details are presented of how such a model can be employed in order to simulate classical Turing machines.

Keywords: reversible computation, quantum computation, unitary operator development, heuristic function, tree search, quantum production system, quantum random walks, iterative deepening, quantum entanglement detection, periodic search.

Título Sistema de Produção Quântico

Nome Luís Domingues Tomé Jardim Tarrataca

Doutoramento em Engenharia Informática e de Computadores

Orientador Doutor Andreas Miroslaus Wichert

Resumo

A teoria que suporta os sistemas de produção é particularmente relevante para realizar pesquisa em árvore. A computação quântica permite uma melhoria de performance quadrática em mecanismos genéricos de procura. Este trabalho apresenta um modelo para um sistema de produção quântico desenvolvido tendo em vista espaços de procura hierárquicos e que examina: (1) o desenvolvimento de operadores unitários, (2) as implicações de diferentes tipos de factores de ramificação, e (3) as consequências associadas ao desenvolvimento de uma função heurística incorporável no modelo. Estes conceitos são utilizados para apresentar um conjunto de definições formais para um sistema de produção quântico. O modelo é comparado com caminhadas quânticas aleatórias, sendo demonstrado que ambas as abordagens diferem por um factor de dois no número de nós avaliados. É também desenvolvida uma extensão iterativa e é descrita uma aplicação do modelo no contexto do problema da paragem quântica. Esta extensão permite: (1) a detecção de estados de paragem sem interferir com o resultado final da computação; (2) a possibilidade de não terminação da computação; e (3) um aumento de performance inerente em computações susceptíveis de paralelização. São também apresentados os detalhes de como o modelo pode ser utilizado para simular máquinas de Turing clássicas.

Palavras-chave: computação reversível, computação quântica, desenvolvimento de operadores unitários, função heurística, procura em árvore, sistema de produção quântico, caminhadas aleatórias quânticas, procura iterativa, detecção de entreleçamento quântico, procura periódica.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of artificial intelligence can be divided into a vast array of key topics, covering natural language processing, automatic programming, automatic theorem proving, robotics, machine vision, intelligent data retrieval systems, etc [165]. Each of these areas is so extensive that it would be an insurmountable task to delve exhaustively into such matters. However, a large part of these problems can be represented through some type of symbolical state description. This internal representation of the problem can be evolved into other states by a set of specific rules. An algorithmic procedure examines such states and decides which rule should be applied in order to obtain some final result. Typically, algorithmic procedures aim to deliver such final states correctly, and to do it whilst minimizing some type of performance measure (*e.g.* time, space or error).

For some problems there may exist algorithms that are able to deduce which of the allowable rules is best. However, for many other computational problems, at any given point in time there may be multiple rules that can be applied and no single way of inferring which one is best when trying to minimize the performance measure. The set of all possible combinations of rules forms that which is commonly referred to as a search space. As a result, for this type of problems the simplest algorithmic strategy for finding a solution consists in systematically enumerating and evaluating every element of the search space until goal states are found. Usually, search algorithms reflect a means of determining a sequence of rules, composed of condition-action pairs, leading from an initial state to a goal state. The process of determining such a sequence of actions is generically labelled as search. All that is required to formulate a search problem is a tuple consisting of a set of states alongside subsets describing initial and goal states, and a set of actions mapping states into successor states [94]. Search algorithms are a central notion to many tasks in artificial intelligence.

As a motivating example please consider the graph presented in Figure 1.1, which represents a form of state space search usually referred to as tree search. Tree search can be seen as a subset of elementary graph theory representing acyclic connected graphs where each node has zero or more children nodes and at most one parent node [63]. The binary tree presented showcases the nodes reached from a root node A by applying one of two possible unconditional rules whose actions are labeled p_0 and p_1. The cardinality of the set of applicable actions is referred to as the branching factor b. It is not difficult to see that for non-unary branching factors the search space grows exponentially fast. As a result, a total of b^d leaf nodes exist at a search depth level d. Each leaf node is reached after having applied d actions, *e.g.* node I is reached after applying actions p_0, p_0 and p_1. The set of actions leading to a leaf node is often labeled as the path taken during the tree search. Tree search algorithms play a crucial role in many applications, *e.g.* production systems (please refer to [178], [162], [160], [66], [16], [131] and [130]) game playing programs [69] [111] [112] [38] and robot control systems [187] [186].

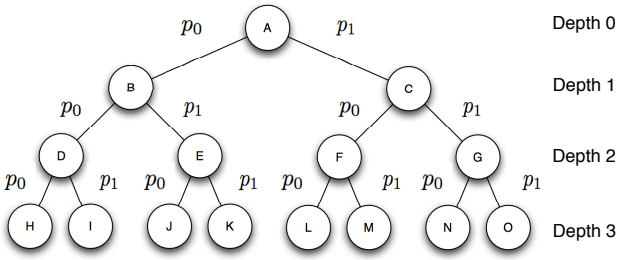


Figure 1.1: The possible paths for a binary search tree of depth 3.

The simplest search strategies rely on a "brute-force" approach, in which an exhaustive examination of all possible sequences of actions is performed until goal states are reached. The search through the state space systematically checks if the current state is a goal state. If a non-goal state is found then the current state is expanded by applying a successor function, generating a new set of children states. The choice of which state to expand is determined by a search strategy [183]. Search strategies that are only able to distinguish between goal states and non-goal states, without knowing if one state is more promising than another are called uninformed search strategies. Examples of uninformed search strategies include the well known breadth first search [155], depth-first search [104], and also the iterative deepening search [193]. In general, uninformed search methods can only solve small problem instances. This is due to the exponential growth of the search space experienced by such methods [77], and the consequent implications on time complexity. Unfortunately, most problems of interest cannot be be handled in an uninformed search fashion.

Informed search strategies arose naturally from this context. They employ problem specific knowledge beyond the definition of the problem itself and are able to find solutions more "efficiently" than uninformed search strategies. The term efficiently is employed carefully as shall later be described. Typically, informed search strategies employ an evaluation function $f(n)$ which considers a cost function $g(n)$ alongside a heuristic function $h(n)$. Function $g(n)$ can be interpreted as representing the cost to reach node n whilst $h(n)$ represents an estimate on the cost to reach a leaf node from node n. From a practical point of view the performance of search algorithms is greatly improved by the use of a heuristic evaluation function [125]. Traditionally, the node with the lowest evaluation is selected for expansion. Examples of some of the best known informed search strategies include greedy search [161] and A* search [92]. Other examples of informed strategies, with a strong emphasis on adversarial search, include minimax [209] and also minimax with $\alpha - \beta$ pruning [94]. Subsequent advances on informed strategies have focused on limiting the amount of memory used. Concrete examples include IDA* [124] and RBFS [126], [127]. A time and space comparative assessment between the various strategies is presented in Table 1.1. Although informed search strategies allowed for an increase in terms of average performance, the worst-case computational complexities of these methods still grew in an exponential manner. These limitations led to the development of a new investigation field in the late 1970's focusing on parallel search.

Search	**Reference**	**Strategy**	**Time**	**Space**
Breadth-first	[155]	Uninformed	$O(b^{d+1})$	$O(b^{d+1})$
Depth-first	[104]	Uninformed	$O(b^m)$	$O(bm)$
Iterative-deepening	[193]	Uninformed	$O(b^d)$	$O(bd)$
Greedy	[161]	Informed	$O(b^m)$	$O(b^m)$
A*	[92]	Informed	$O(b^d)$	$O(b^d)$
IDA*	[124]	Informed	$O(b^d)$	$O(bd)$
RBFS	[126]	Informed	$O(b^d)$	$O(bd)$
Minimax	[209]	Informed/Adversarial	$O(b^m)$	$O(bm)$
Minimax $\alpha - \beta$ pruning	[94]	Informed/Adversarial	$O(b^m)$	$O(b^{\frac{1}{2}}m)$

Table 1.1: Tree Search Algorithm Comparison (b - branching factor, d - depth of a solution, m - maximum depth).

The remainder of this chapter is organised in such a way as to continue to present the main reasons behind this work. As a result, Section 1.1 introduces some of the main concepts supporting classical parallel algorithms in the context of tree search. This section is used to reinforce the idea that despite the algorithmic advances, parallel search strategies are not without its flaws. Namely, they fail to deliver any type of exponential speedup due to several restrictions that will later be described. This inability to yield a significant performance increase is used as a motivation to introduce in Section 1.2 Grover's algorithm, a method based on quantum computation capable of delivering a quadratic improvement. This section also presents a set of questions on what is there to be gained by exploiting such concepts from a quantum perspective. Tree search is also a key component in production system

theory. This model is well suited to a quantum environment in which a superposition state can be perceived as representing multiple path alternatives. The main concepts of classical production system theory are presented in Section 1.3. These notions are then combined in Section 1.4 in order to present the formal definition of this thesis' statement. The same section also list some of the main questions surrounding the development of a quantum production system.

1.1 Classical Parallel Search Algorithms

Presently, parallel computation has assumed mainly two forms, namely multi-core architectures and parallel computers communicating through some type of network infrastructure. All of these processing elements need to collaborate in order to find a solution for a specific problem. Typically, this collaboration process has performance costs. Evidently, the main objective of employing parallel computation is to produce parallel algorithms that dramatically improve the performance of the sequential ones. However, as drawn attention by [169], the total amount of work performed by parallel algorithms, *i.e.* the number of computational steps executed by each processing element summed over all processing elements can be no smaller than the time complexity of the best sequential algorithm. Logically, any parallel algorithm can be simulated by a sequential one that does the same amount of work. In the context of tree search, the advent of parallel computation had the possibility to increase drastically the number of nodes evaluated in a given amount of time. A higher number of evaluated nodes could potentially provide optimal solutions, or improve decision quality [71]. Traditional approaches to parallelizing search algorithms include:

- Parallelize the processing of individual nodes, such as move generation and heuristic evaluation [62], *e.g.* heuristic functions employing weighting schemes where each term can be individually processed allowing for more complex evaluation functions. Generally, a speedup achievable in this scheme is limited, however, by the degree of parallelism available in move generation and evaluation.
- Parallel Aspiration [20] works exclusively for the minimax algorithm with $\alpha-\beta$ pruning. This approach requires multiple processors calculating non-overlapping $\alpha-\beta$ intervals. This approach is severely limited in speedup due to the obligatory expensive minimax verifications that need to be performed.
- Concurrent subtree evaluation.

Most discussions of parallel search have focused on concurrent examination of independent subtrees (please refer to [146], [71], [110] and [69]). The key challenge with parallel search

resides in how best to utilize multiple processing units when the communication cost can be perceived as high, since it may result in idle processor time. Typically, it is a difficult task to separate neatly into chunks a great deal of problems. Even then, each chunk requires processing and may thus require communication between different computational resources to exchange intermediate results or synchronize status. Unfortunately, there is no known optimal method for parallelizing search amongst N processors. Generally, a N-fold increase in the number of nodes evaluated is not possible because of two key overheads, namely [146]:

- Communication Overhead - some intercommunication between processors is always necessary either through some message strategy or through a shared data structure. This overhead may result in idle processor time.
- Search Overhead - if independent subtrees are searched concurrently, it is likely that redundant nodes will be examined because the best bounds are not always available. Accordingly, parallel search strategies may perform some redundant verifications that would not be required by their sequential versions.

It is also noteworthy to mention that these overheads do not act independently of each other. For example, significant communication overheads may be incurred between parallel searches, resulting in improved data sharing. This data can potentially result in better cutoff decisions, thus reducing the total amount of search overhead. But the inverse is also true, *i.e.* the increased data sharing may actually not provoke better cutoff decisions. In general, communication overhead is inversely related to search overhead [146]. As a consequence, classical parallel search algorithms will likely never yield the full speed-up potential of applying N processors. Due to the intricacies of classical parallel search algorithms these overheads are unavoidable, although clearly some strategies may be devised focusing in either overheads or on trying to strike a balance between both of the them.

In sequential search algorithms, potential performance increases arise from subtree cutoff decisions. These decisions are based on the accumulated information obtained until a given point in the search. Given the general nature of tree decomposition methods and the incurring search overheads associated with them, most authors have focused on parallelizing branch-and-bound algorithms such as minimax with $\alpha - \beta$ pruning. Branch-and-bound algorithms attempt to prune large sections of a search space in order to increase performance. Table 1.2 presents some of the best known parallel algorithms alongside the respective speedups (representing a scale factor of the additional states considered) relatively to their single-processor sequential version, where n is the number of processors employed.

Curiously, the parallel tree search field seems to have had its investigation heyday in the mid 1990's with IBM's Deep Blue initiative in 1996 and 1997 [38]. Deep Blue is the chess

Algorithm	Reference	Hardware	Speedup Obtained
Parallel Aspiration Search	[20]	Simulation	≤ 6 (for large n)
PV-Split	[148]	Sun 3 Network	3.75 (for $n = 5$)
Waycool	[70]	Intel Hypercube	101 (for $n = 256$)
Bound-and-Branch	[71]	Intel Hypercube	12 (for $n = 32$)
Delayed Branch Tree Expansion	[110]	Simulation	350 (for $n = 1000$)
Jamboree	[129]	CM-5	50 (for $n = 512$)
Dynamic Multiple PV-Split	[147]	AP-1000	32 (for $n = 64$)
APHID	[33]	Sparc 2 Network	6 (for $n = 16$)

Table 1.2: Parallel tree search comparison (n - number of processors, Source: [32]).

machine that defeated then-reigning World Chess Champion Garry Kasparov in a six-game match in 1997. Amongst the major factors that contributed to its success was a massively parallel system with multiple levels of parallelism. Previous research in game tree search typically dealt with systems that searched order of magnitude fewer positions than Deep Blue [38].

1.2 The Quantum Search Algorithm

Grover's algorithm was first proposed in [82] and subsequently published as a letter in [83]. The procedure, also known as the quantum search algorithm, performs a search using the principles of quantum computation. Whereas classical search algorithms require $O(N)$ time for data sets containing N elements. Grover's algorithm requires $O(\sqrt{N})$ time, providing a quadratic speedup. The algorithm performs a search for a binary string amongst what can be perceived as a database of binary strings. The algorithm focuses on verifying if a state belongs to a solution set. Grover's search algorithm employs features uniquely found in quantum computation, such as quantum superposition, in order to query many elements of the search space simultaneously, alongside entanglement and quantum interference. These elements allow quantum computation to provide for a new paradigm of parallel computation.

Subsequently, it was proved that any procedure that evaluates states and marks the ones representing solution in a black-box fashion will always require at least $\Omega(\sqrt{N})$ time. This result was first shown in [26], with earlier versions of the manuscript appearing before Grover's algorithm [22], and later revised on [29]. Grover's algorithm was experimentally demonstrated in [46]. In [220] the quantum search algorithm was shown to be optimal in the sense that it gives the maximal possible probability of finding a solution. Examples of possible extensions to Grover's original work include returning a solution with certainty (please refer to [219], [30], [141] and [113]) and assessing the impact of having multiple solutions [29]. Grover and Radhakrishnan [80] considered the speedup achievable if one was only interested in determining the first m bits of a n bit solution string. In practice, their approach proceed

with analysing different sections of the quantum search space. The authors prove that it is possible to obtain a speedup, however, as m grows closer to n the computational gains obtained disappear [80]. This speedup was then improved in [123] and [122] and an extension to multiple solutions was presented in [45]. In [81] the authors show that well known quantum algorithms for the collision and element distinctness problems, respectively [31] and [36], do not take advantage of the underlying structure to go beyond the standard quantum search algorithm. A number of extensions have been proposed since Grover's original work in [87] and [88]. These procedures essentially targeted reduced time complexity bounds for non-query operations and overall robustness. For a number of several novel search related applications please refer to [84] [85] [86]. Others approaches to quantum search have been detailed in [103], [100], [101] and [102].

Grover's original idea focused on searching a collection of binary strings. The author's work did not have hierarchical search mechanisms in mind. Accordingly, the question arises: can a mapping be devised for Grover's algorithm allowing for a tree search procedure to be performed? An initial mapping could take into consideration an initial state representation, *i.e.* the root node, alongside a tree path, *i.e.* the sequence of actions to apply in a manner reminiscent of depth-first search. Such an approach would give rise to a number of questions, namely:

- What is there to be gained by employing such a method? And consequently, what are the limitations?
- What is the best strategy for performing tree search in quantum fashion? Is the quantum search algorithm the only viable solution? If other alternatives exist how do they compare against each other?
- Is it possible to develop a quantum equivalent for informed search strategies in order to influence the probability distribution amongst tree paths? If so, what are the mechanisms available for doing so?
- Based on the ideas supporting tree search, namely that of performing search space decomposition is it possible to develop alternative search algorithms? Namely, is it possible to develop such procedure using existing, but unexploited quantum features, which would in principle yield significant performance increases over amplitude amplification schemes? If so, what are the main requirements of such approaches and how do they fare against Grover's original proposition?
- Tree search is typically depth limited in order to ensure that an answer is obtained within a certain temporal limit. However, iterative depth increases are also allowed. In the context of quantum computation what is the best way to determine the length

of the sequence of actions to apply? Is it possible to develop any iterative quantum tree search process? If so, what are the associated requirements and corresponding performance? Are there any broader implications for quantum computation besides only performing tree search?

Some of the possible relationships between artificial intelligence and quantum computation have been established in [215]. However, this approach mostly aimed at providing an introductory text to the artificial intelligence community into the field of quantum computation.

1.3 Production System

In terms of human cognition, tree search strategies are also commonly referred to as problem-solving procedures. Such procedures model in abstract terms the techniques employed by humans when trying to solve problems. Knowledge allows human beings to develop sets of actions allowing for goals to be attained in complex environments. Additionally, ones knowledge is not expected to remain static, but to change and grow over time as a person learns. Post proposed a computational formalism known as the production system [178] that has proved relevant in providing a connection between search algorithms and human problem solving modeling. Post's work was applied to model the principles of intelligence in the SOAR system [163], used in programming languages such as OPS5 [74], and also provided the theoretical foundations for expert systems such as Mycin [35]. Some of the best known examples of human cognition-based production systems include the General Problem Solver [162] [160] [66] [163], ACT [16] and SOAR [131] [130]. The orderly distinction between knowledge and control brought upon by production systems makes them desirable for experimentally modeling problem-solving behaviour [143]. In the field of artificial intelligence, one of the possible applications of production system theory consists in determining a sequence of actions leading to a desired state.

The formal behaviour of a production system can be described as performing a search where sequences of rules are evaluated in order to determine if a given set of goal states is reached. A production system is composed of condition-action pairs, *i.e.* if-then rules, which are also called productions. A computation is performed with the aid of productions through the transformation of an initial state into a desired state. The state description at any given time is also referred to as working memory. A rule is applied when the conditional part is recognized to be part of a given state. The action describes the respective problem-solving behaviour. Applying an action results in the state of the problem instance changing accordingly. On each cycle of operation, productions are matched against the working memory

of facts. At any given point, more than one production might be deemed to be applicable. This subset of productions represents the conflict set. A conflict resolution may be employed to this subset in order to determine an appropriate production. However, the system may also exist in a state where all the appropriate productions are simultaneously applied. This induces the type of behaviour that is characteristic of tree search. The operational cycle is brought to a close when a goal state is reached or when no more rules can be triggered. This type of behaviour is almost indistinguishable from the one describing tree search. Production system theory is therefore uniquely suited for detailing the dynamics of tree search. The general architecture described in the previous paragraphs is illustrated in Figure 1.2.

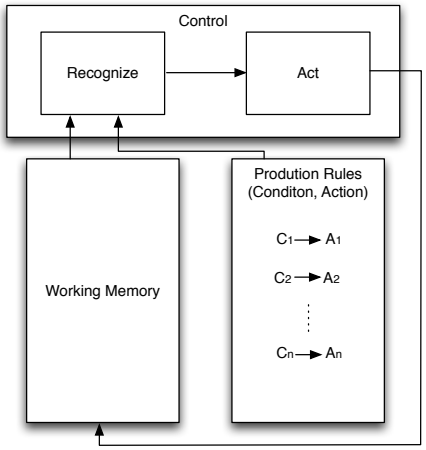


Figure 1.2: General architecture for a production system (adapted from [143]).

Traditionally, production systems require translating the existing knowledge into a database of symbolic rules, which in turn require adequate encoding mechanisms. When these rules are applied to a given problem the strategy employed to search through the database allows for two basic behavioural settings, namely forward- and backward- chaining. The former allows the system to demonstrate why a given assertion was reached. The latter enables the system to answer how an assertion was established [213]. Comprehensibly, this kind of systems can also take advantage of parallel search algorithms. *i.e.* parallel production systems where more than one production rule can fire simultaneously [75]. Regrettably, as was discussed in Section 1.1 this kind of approaches have their own set of specific limitations.

In addition, the production system was shown to be equivalent in power to other well known computational models such as the Turing Machine [205], Church's lambda-calculus [48], Post's production system [178], and Markov algorithms [145]. Please refer to [2], [53] and [54] for additional details on these equivalences. In practice this means that any function

that is computable by a given model is also computable by all the other models. This is equivalent to stating that production systems are comparable in power to a Turing machine. However, it is my opinion that specifying the set of transitions performed by a tree search on a Turing machines looses some of the elegance and simplicity provided by production system theory.

1.4 Thesis Statement

The quantum computation community has placed a lot of effort and emphasis on examining quantum equivalents of traditional computational methods such as the Turing machine. This research focus has resulted in important contributions to the field. However, less known methods may also yield significant results. The production system is one specific example that may have potential implications for quantum computation. Accordingly, a unique opportunity arises, namely to study the environment surrounding tree search from the perspective of a quantum computational model that is specifically designed with such a task in mind.

Immediately a number of questions can be posed, namely: can a modular approach to a production system be developed? *I.e.* is it possible to express through unitary operators, and operator composition a logical separation between the standard computational components of data, operations and control? If so, how does the operator evolve in relation with the set of production rules? How can the transitions of a production system be expressed in a probabilistic manner in order to allow for an adequate quantum amplitude mapping? How to formalize a theoretical model of a quantum production system? Also, what are the mechanisms allowing for symbolical rules to be adequately encoded? Do these encodings have an impact on performance? Additionally, how do classical search concepts such as branching factor, heuristic evaluation and iteration influence the model? Is it possible to extend existing quantum methods to obtain similar behaviour? If such extensions are possible, what are the associated requirements and how do the available mechanisms compare against each other? Finally, can alternative search methods be developed, which are not based on Grover's algorithm, and that take into account the concepts considered in this work?

The following chapters provide answers to these questions. The results herein contained form contributions that lend support to the following thesis:

Thesis: The quantum production system model represents a simple and elegant approach to tree search with an inherent quadratic speedup. Furthermore, the model is also well-suited when dealing with the possibility of non-terminating computation.

Some of the investigation tasks performed in the context of this research included, but were not restricted to: developing computational algorithms; elaborating adequate adaption strategies from a classical to a quantum context; evaluating the computational complexity, time- and space-wise of the proposed methods; evaluating specific behavioural dynamics associated with quantum computation models; evaluating potential performance bottlenecks; developing quantum space decomposition methods.

1.5 Organisation

The remainder of this thesis is organised as follows: Chapter 2 presents some of the key features associated with reversible state evolution, which represents a key concept in quantum computation. Chapter 3 then presents the fundamental concepts of quantum computation alongside some simple algorithmic examples. Given that this research essentially deals with questions surrounding quantum search some of the main theoretical details surrounding Grover's algorithm will be presented. Both these chapters mostly serve to present the main concepts that will later be employed to develop some of the results that are presented in this work. Chapter 4 then builds on the circuitry knowledge discussed in the previous chapters to develop an approach towards a quantum production system capable of solving instances of the n-puzzle; Chapter 5 reflects on some of the issues of employing such an approach and considers the requirements associated with informed search strategies; Chapter 6 generalizes this approach in order to provide a model of quantum computation based on production system theory; Chapter 7 considers tree search from a graph perspective alongside existing quantum methods focusing on graph search; Chapter 8 presents an iterative extension to the initial quantum production system in order to contemplate for the possibility of non-terminating computation. Chapter 9 considers alternative quantum search models, albeit not as efficient as Grover's method, based on (1) the principles of quantum entanglement detection alongside a recursive oracle definition; and (2) the notion of search space periodicity. The overall conclusions alongside important reflections are presented in Chapter 10.

Throughout this work an explicit effort was made to provide the appropriate context for all the results in a self-contained manner and also to make these easily comprehensible. In addition, most of the results presented in this thesis were published elsewhere. The majority of the results presented in Chapter 4 focusing on unitary operator development and composition were decomposed into two papers that were published in [198] and [197]. The method for performing quantum tree search through Grover's algorithm discussed in Chapter 5 was first published in [199]. The quantum production system proposed in Chapter 6 was presented in [201]. The results concerning quantum random walks on trees introduced

in Chapter 7 were published in [202]. The application of the iterative deepening extension to the halting problem discussed in Chapter 8 was published in [196]. The results and definitions regarding the quantum entanglement detection scheme discussed in Chapter 9 were published in [200]. Because each publication incorporated the problem that was being tackled, a small state of the art survey and the proposed solutions a choice was made to represent each publication as an individual chapter with the closest form possible to the original manuscript. However, some of the common material that was presented in these publications, namely regarding motivation, research questions and general problem definition, was fused into this introductory chapter, which became responsible for presenting the broad research points.

Chapter 2

Reversible Logic Fundamentals

2.1 Introduction

In classical computer science, information is represented by signals. Traditionally, the signals employed use just two discrete values and are therefore said to be binary. Information is represented in digital computers through binary strings (*i.e.* a sequence of binary variables) and adequate encoding mechanisms. This binary information is manipulated through hardware components, also known as digital circuits. A circuit may involve any number of inputs and output bits, wires and logical gates which transmit information and perform some type of processing.

The remainder of this chapter focuses on presenting some of the main connections between irreversible and reversible computation. Reversibility is a key characteristic of the type of operators employed to describe quantum computation. The reasons behind this requirement will be examined in Chapter 3. Accordingly, this chapter is organised as follows: Section 2.2 presents the main details of classical circuit logic; Section 2.3 describes how these elements can be described through linear matrix operators; and Section 2.4 describes the principles of reversible computation.

2.2 Classical Circuit Logic

A logical gate is a function $f : \{0,1\}^k \rightarrow \{0,1\}^l$ from some fixed number k of input bits to some fixed number l of outputs bits [144]. Typically, each gate performs a specific logical operation. The outputs of gates are applied to inputs of other gates to form a composite digital circuit. The number of inputs associated with each gate describes its Fan-in and the number of outputs describes its Fan-out.

Binary logic is then applied to each binary variable. Associated with the binary variables are logical operations such as NOT, OR, AND, XOR and equivalent complement operations, respectively, NOR, NAND and XNOR. The truth tables for these operations are presented in Table 2.1. The table lists all possible combinations of values for two variables and the results of the applying each aforementioned operation. Also, an additional operation is considered, respectively NOP (drawing from assembly language "no operation" instruction), that effectively maintains the state of the bit. It is important to draw attention to the fact that the NAND and the NOR gates are considered to be universal. This universality stems from the fact that every boolean function can be built by employing only this type of gates [144].

X	Y	NOP X	NOT X	X AND Y	X OR Y	X NAND Y	X NOR Y	X XOR Y
0	0	0	1	0	0	1	1	0
0	1	0	1	0	1	1	0	1
1	0	1	0	0	1	1	0	1
1	1	1	0	1	1	0	0	0

Table 2.1: Truth table for the most common logical operations.

These logical operations may then be combined into a single Boolean function. A Boolean function consists of a binary variable denoting the function and an algebraic expression formed by using binary variables. Consider for example the Boolean function presented in Expression 2.1. Recall that a boolean function expresses the logical relationship between binary variables. Accordingly, it is evaluated by determining the binary value of the expression for all possible combinations of values for the variables. Table 2.2 lists all possible input combinations, and respective outputs, for function F. A Boolean function can also be transformed from an algebraic expression into a circuit diagram composed of logic gates. The logic circuit diagram for F is shown in Figure 2.1.

$$F = \bar{X} + YZ \tag{2.1}$$

X	Y	Z	$F = \bar{X} + YZ$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 2.2: Truth table for $F = \bar{X} + YZ$.

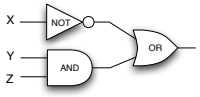


Figure 2.1: Logic Circuit Diagram for $F = \bar{X} + YZ$.

2.3 Matrix Perspective

As was previously seem, it is possible to forecast a circuit's output by applying each individual gate from left to right to the original input bits. A gate application acts on a specific subset of the binary variables. Intuitively, the different bit values each variable has at given point in time can be perceived as the system's state. The state of any particular variable at a given point in a circuit is just the value of the bit on that wire, respectively 0 or 1. This is the classical behaviour expected from a deterministic process. However, there exist certain probabilistic processes for which it is not possible to know for certain the state of the system [99]. Typically, in these cases what is known is actually the probability distribution of the states [212]. In this case the initial description for a circuit, *i.e.* that a bit value is either 0 or 1 is clearly not enough. This section reviews the classical circuit model of computation in terms of vectors and matrices.

A motivating example consists of a single bit that is in state 0 with probability p_0 and in state 1 with probability p_1. By employing some simple linear algebra knowledge it becomes possible to translate this information into a 2-dimensional vector of probabilities, as illustrated by Expression 2.2 [118].

$$\begin{pmatrix} p_0 \\ p_1 \end{pmatrix} \tag{2.2}$$

This representation is also capable of translating the deterministic behaviour previously mentioned. Namely, a deterministic circuit whose final state is 0 is equivalent to say that $p_0 = 1$ and since probabilities need to sum up to one, $p_1 = 0$, as depicted in Expression 2.3.

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.3}$$

A circuit whose state is 1 implies a $p_1 = 1$ and a $p_0 = 0$, which can be represented by the vector illustrated in Expression 2.4.

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.4}$$

What are the implications of employing a vector form to represent binary variables? Linear algebra can be employed to evolve a state. A process which is usually done through matrix operators. Accordingly, any logical gate can now be perceived to assume a matrix form. Accordingly, each gate must specify exactly what happens to its inputs. In linear algebra, a function f that maps a domain $\mathbb{R}^n$ into codomain $\mathbb{R}^m$ with $m = n$ is called an operator [17]. The following sections detail the matrices for the NOP and NOT gate, respectively Section 2.3.1 and Section 2.3.2. Section 2.3.3 presents the matrix representation for a previously unmentioned operation, respectively the CNOT gate. Section 2.3.4 details how the CNOT gate can simulate the Fan-out operation.

2.3.1 Nop Matrix

The NOP operation explicit purpose is not to change the state of the bits. Intuitively, from a linear algebra perspective there is a relationship between a state maintaining operator and the identity matrix I, as illustrated by Expression 2.5.

$$\text{NOP}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \text{NOP}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.5}$$

In order for the above statements to be verifiable it is easy to check that the NOP operator is simply a 2×2 identity matrix, as illustrated by Expression 2.6.

$$\text{NOP} = I_{2\times 2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.6}$$

It should also be clear from the aforementioned behaviours that in order to apply the NOP gate it is only necessary to conduct a simple matrix multiplication by the state vector, as depicted in Expression 2.7.

$$\text{NOP} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} \tag{2.7}$$

2.3.2 Not Matrix

The NOT gate requires a bit more effort but it is still a relatively simple operation. Take into account the logical NOT gate, a possible matrix representation should always observe the behaviour presented in Expression 2.8. This implies that the NOT operator can be represented by the matrix presented in Expression 2.9.

$$\text{NOT} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \text{NOT} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.8}$$

$$\text{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.9}$$

2.3.3 Cnot Matrix

In order to obtain equivalent matrix representations for the remaining elementary logic gates Expression 2.2 needs to be extended so that it can accommodate two bits. As a result, the overall state consists of bits p and q. Again, let k_i represent the probability of bit k being in state i. A natural observation is that at any given point in time it is possible to have a number of combinations, namely:

- **Combination 1** - bit p may be in state 0 with probability p_0 while bit q may be in state 0 with probability q_0;
- **Combination 2** - bit p may be in state 0 with probability p_0 while bit q may be in state 1 with probability q_1;
- **Combination 3** - bit p may be in state 1 with probability p_1 while bit q may be in state 0 with probability q_0;
- **Combination 4** - bit p may be in state 1 with probability p_1 while bit q may be in state 1 with probability q_1.

Following this reasoning it should come as no surprise that this information can also be encoded as a vector, this time with dimension 4, as illustrated by Expression 2.10 [118].

$$\begin{pmatrix} p_0q_0 \\ p_0q_1 \\ p_1q_0 \\ p_1q_1 \end{pmatrix} \tag{2.10}$$

Recall that in a deterministic circuit a bit can be either in state 0 or in state 1. This implies that the probabilities for a given bit being in a certain state will be either 0 or 1, namely the possible combinations for bits p and q will be:

- If bit p is in state 0 then $p_0 = 1$, and accordingly $p_1 = 0$;
- If bit p is in state 1 then $p_1 = 1$, and accordingly $p_0 = 0$;
- If bit q is in state 0 then $q_0 = 1$, and accordingly $q_1 = 0$;
- If bit q is in state 1 then $q_1 = 1$, and accordingly $p_0 = 0$.

Expression 2.10 can be combined with the previously mentioned outcomes in order to obtain appropriate vector representations, namely:

- Combination 1 $\overset{\substack{p_0=1,p_1=0\\q_0=1,q_1=0}}{\rightarrow}$ $\begin{pmatrix} p_0q_0 \\ p_0q_1 \\ p_1q_0 \\ p_1q_1 \end{pmatrix} = \begin{pmatrix} 1\times1 \\ 1\times0 \\ 0\times1 \\ 0\times0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$
- Combination 2 $\overset{\substack{p_0=1,p_1=0\\q_0=0,q_1=1}}{\rightarrow}$ $\begin{pmatrix} p_0q_0 \\ p_0q_1 \\ p_1q_0 \\ p_1q_1 \end{pmatrix} = \begin{pmatrix} 1\times0 \\ 1\times1 \\ 0\times0 \\ 0\times1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$
- Combination 3 $\overset{\substack{p_0=0,p_1=1\\q_0=1,q_1=0}}{\rightarrow}$ $\begin{pmatrix} p_0q_0 \\ p_0q_1 \\ p_1q_0 \\ p_1q_1 \end{pmatrix} = \begin{pmatrix} 0\times1 \\ 0\times0 \\ 1\times1 \\ 1\times0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$
- Combination 4 $\overset{\substack{p_0=0,p_1=1\\q_0=0,q_1=1}}{\rightarrow}$ $\begin{pmatrix} p_0q_0 \\ p_0q_1 \\ p_1q_0 \\ p_1q_1 \end{pmatrix} = \begin{pmatrix} 0\times0 \\ 0\times1 \\ 1\times0 \\ 1\times1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$

Intuitively, each of the aforementioned combinations can be perceived as being encoded in a unique vector state, reflecting a binary coding. Accordingly, for four possible combinations four vector states can be built that uniquely identify the originating combination. Expres-

sion 2.10 makes it difficult to obtain equivalent matrix representations for the remaining elementary logic gates. Intuitively, this can be perceived as a consequence of the remaining gates starting out with two input bits and outputting a single bit. This kind of gates are labeled as many-to-one functions. The output bit of these gates is neither of the input bits, and so would require the introduction of an additional bit r to Expression 2.10.

However, Expression 2.10 is not without its use. Namely, assume that one is trying to build a two-bit input gate responsible for outputting two bits. The input bits are labeled as control bit c and target bit t. The gate always outputs bit c unaltered (the reasons for this will become clearer in the next section). Besides outputting c the gate applies the NOT operation to t if c is set to 1, *i.e.* $c \oplus t$. Otherwise the gate performs nothing. This behaviour is extremely similar to the standard XOR gate. The single main difference resides in the fact that an extra bit is outputted. This operation is commonly referred to as controlled-NOT gate, or CNOT. A graphical representation for the CNOT operation is depicted in Figure 2.2.

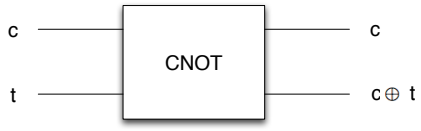


Figure 2.2: CNOT gate.

The CNOT gate requires two input bits, thus a total of four possible combinations can be fed into it. By using the previous method for mapping each combination into a state vector it is possible to describe the overall computation as a simple matrix multiplication. Based on the previous description, a matrix representation for the CNOT gate should be in accordance with the behaviour presented in Expression 2.11.

$$\begin{aligned} \text{CNOT}\begin{pmatrix}1\\0\\0\\0\end{pmatrix} &= \begin{pmatrix}1\\0\\0\\0\end{pmatrix}, \quad \text{CNOT}\begin{pmatrix}0\\1\\0\\0\end{pmatrix} = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \\ \text{CNOT}\begin{pmatrix}0\\0\\1\\0\end{pmatrix} &= \begin{pmatrix}0\\0\\0\\1\end{pmatrix}, \quad \text{CNOT}\begin{pmatrix}0\\0\\0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \end{aligned} \tag{2.11}$$

The aforementioned behaviour implies that the CNOT gate can be represented through the matrix presented in Expression 2.12.

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.12}$$

2.3.4 Fan-out Operation

The Fan-out operation previously described basically allows for individual bits to be copied in order to provide input to other circuit gates. This operation can be performed by simply using the CNOT gate alongside an auxiliary bit set to 0. In order to see this consider the output $c \oplus t$. Assume that t is set to 0 then $c \oplus t \overset{t=0}{\rightarrow} c \oplus 0 = c$, *i.e.* by setting the input bit t to 0 a copy of bit c is obtained. This situation combined with the CNOT gate always outputting bit c effectively results in two copies of bit c being produced. This behaviour is illustrated in Figure 2.3. This kind of reasoning, *i.e.* translating the inputs onto the outputs alongside the introduction of auxiliary bits will be key to fully understanding the mechanism behind reversible computation.

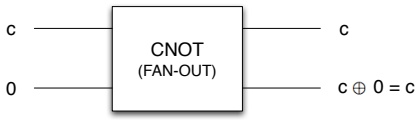


Figure 2.3: Fan-out operation using a CNOT gate.

2.4 Reversible Computation

Reversible computation was a matter throughly investigated on the seminal work on the subject by Bennett [23]. For a detailed analysis on the history of reversible computation and its interactions with other scientific fields please refer to [24] and [25]. The core ideas behind reversible computation can be formulated in linear algebra terms. Bit states and gate operators can be described through vectors and matrices [118]. Also, reversible gate operators can be mathematically defined through unitary matrices. This linear algebra framework for reversible computation provides a stable and proven basis for the reversible computation process.

Traditional logical gates employed in digital circuits can be classified according to their reversibility. A gate is said to be reversible if it is always possible to uniquely recover the input, given the output [98]. The NOP and NOT operators are examples of reversible logic gates because, given the output of each gate, it is possible to infer what the input must have been. In the case of the NOP operation there is no change to undo. Regarding the NOT gate, traditional boolean algebra enables one to see that by just applying a second NOT operation it is possible to obtain the original input, *i.e.* $\overline{\overline{X}} = X$.

This reversibility does not stand for other logical gates such as the universal NAND gate. When a transition is performed to standard two-bit gates of classical computation, reversibility is lost. This loss of reversibility can be seen as a result of elementary gates starting out with two bits and producing a single bit as output, *i.e.* $f : \mathbb{R}^2 \to \mathbb{R}$. This information loss results in an irreversible process. If a logic gate is irreversible, then some of the information input to the gate is lost irretrievably when the gate operates, *i.e.* some of the information has been erased by the gate (please refer to [133], [23] and [132]). In a reversible computation, no information is ever erased because the input can always be recovered from the output. Thus, saying that a computation is reversible is equivalent to saying that no information is erased during the computation [1]. Also if a function f is reversible then there also exists an inverse function of f, respectively labeled as f^{-1} . The inverse function f^{-1} is responsible for mapping f's outputs into the corresponding input states.

Earlier, it was mentioned that energy consumption in computation plays a crucial role in reversible computation. So it is natural for one to question what is the connection between energy consumption and irreversibility in computation? The theoretical foundations connecting energy consumption and irreversibility were established by Rolph Landauer in 1961 as Landauer's principle, namely [133]:

- **Landauer's principle**: The amount of energy dissipated into the environment when a bit of information is erased is at least $k_B T \log_2 2$, where k_B is a universal constant known as Boltzmann's constant, and T is the temperature of the environment of the computer. The factor $k_B T \log_2 2$ should be interpreted as the heat equivalent of one bit of entropy. This values corresponds to the amount of heat dissipated by any computer operating at temperature T per elementary binary operation performed.

Accordingly, any kind of reversible process has to avoid irreversible operations that ultimately lead to loss of information from any given state of the process. Associated with these irreversible operations (many-to-one data operations), is always an incurred thermodynamic penalty. One the other hand, reversible operations (one-to-one data operations) do not incur such a cost [23]. An important observation to perform is that if all computations could be done reversibly then no bits would be erased and as such no lower bound would exist on the

amount of energy dissipated by the computer [164].

Operations such as NOP and NOT are intrinsically reversible. Yet, the remaining logical operations are irreversible. Accordingly, the question naturally arises if an irreversible circuit model can be transformed into a reversible one? Toffoli provided a mathematical abstraction for reversible computation based on circuit design containing only reversible elements, by making use of a reversible gate known as the Toffoli gate [204]. The following sections are organised as follows: Section 2.4.1 presents the main details associated with the Toffoli gate; Section 2.4.2 concludes by describing how these type of reversible gates can be employed in order to develop more complex reversible circuits.

2.4.1 Toffoli Gate

The Toffoli gate has three input bits, a, b and c. Bits a and b are known, respectively, as the first and second control bits, while c is the target bit. Besides outputting bits a and b, the gate effectively flips the target bit, if and only if the control bits are are both state 1, *i.e.* $c \oplus (ab)$. Otherwise, the Toffoli gate does not perform any operation. If this sounds remarkably similar to the CNOT gate, that is because it is. In fact, it can be shown that the Toffoli gate is a generalization of the CNOT gate [18]. By outputting bits a and b the Toffoli gate allows one to determine the original value for input bit c from expression $c \oplus (ab)$. This way no information is lost, guaranteeing a reversible mechanism for the Toffoli gate. The set consisting of just the Toffoli gate is universal for classical computation [204]. The truth table for the Toffoli gate is presented in Table 2.3 [204].

Inputs			Outputs		
a	b	c	a	b	$c \oplus (ab)$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Table 2.3: Truth table of the Toffoli Gate.

The Toffoli gate representation is illustrated in Figure 2.4. How can the Toffoli be employed in order to perform reversible computation? With some careful manipulation of the Toffoli gate inputs, through the use of auxiliary bits, it becomes possible to construct reversible versions of elementary logic gates. Although the NOP and NOT operations are by themselves already reversible, it is still possible to employ the Toffoli gate to obtain equivalent gates. Can the Toffoli gate simulate other elementary gates besides the NOP and NOT operations? The next sections present the details surrounding the Toffoli's gate potential for simulation.

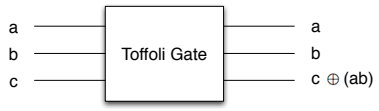


Figure 2.4: General purpose Toffoli Gate.

Toffoli–NOP Gate

What are the required inputs for the Toffoli gate to simulate the NOP gate on bit a. In order to answer this question take into account output $c \oplus (ab)$, which is an expression with two operands, respectively c and ab, and XOR operator $\oplus$. The second operand depends on the value of bit c. Also by setting bit b in ab the value a is obtained, *i.e.* $(ab) \overset{b=1}{\rightarrow} a$. The value of bit c needs to be chosen in such a way that expression $c \oplus a$ evaluates to a. This can be done by setting bit c to 0, *i.e.* $c \oplus a \overset{c=0}{\rightarrow} a$. The output from the NOP is on the target output bit. This situation is illustrated in Figure 2.5.

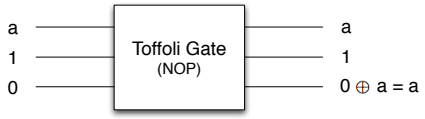


Figure 2.5: NOP gate using a Toffoli Gate.

Toffoli-Fan-Out Gate

Before advancing any further, it should be drawn into attention a fact that might have gone unnoticed. The Toffoli gate simulating the NOP operation presented in the previous section besides outputting bit a in effect also performs a copy of bit a. In reality, for all practical purposes there is no distinction between both operations. This situation is depicted in Figure 2.6.

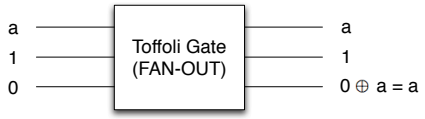


Figure 2.6: Fan-out operation using a Toffoli Gate is equivalent to a Toffoli gate simulating a NOP gate.

Toffoli-NOT Gate

The same line of reasoning can be applied in order to determine appropriate inputs for the Toffoli gate simulating the NOT gate behaviour. As in the previous analysis, $(ab) \overset{b=1}{\rightarrow} a$. Accordingly, c needs to be chosen such that a's complement is obtained. By setting c to 1 the desired behaviour is obtained, *i.e.* $c \oplus a \overset{c=1}{\rightarrow} \overline{a}$ as illustrated by Figure 2.7. Again, the output from the NOT is on the target output bit.

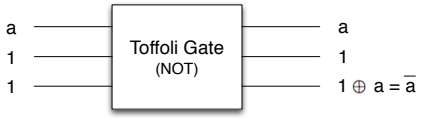


Figure 2.7: NOT gate using a Toffoli Gate.

Toffoli–NAND Gate

Can the Toffoli gate be employed in order to perform more advanced operations? It turns out that the Toffoli gate can be employed to perform universal computation by simply simulating the universal NAND operation. Suppose a NAND is to be performed with bits a and b. The second operand from expression $c \oplus (ab)$, respectively ab, already performs an AND operation. It would be useful to somehow be able to perform a NOT operation, *i.e.* to negate, this expression. Fortunately, by setting bit c to 1 the NAND operation can be obtained, *i.e.* $c \otimes (ab) \overset{c=1}{\rightarrow} 1 \otimes (ab) = \overline{ab}$. The output from the NAND is on the target bit. The gate representation for the reversible NAND gate using a Toffoli gate is presented in Figure 2.8. This behaviour is illustrated in the line entries of Table 2.3 where the the control bit c is set to 1, respectively lines 2,4,6 and 8.

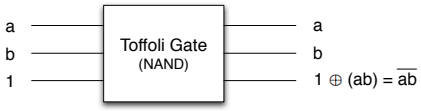


Figure 2.8: Reversible NAND gate using a Toffoli Gate.

Since the NAND gate is universal for classical computation and the Toffoli gate can be used to simulate the NAND gate, the Toffoli gate is also a universal gate for classical computation [204]. It is also important to draw attention to the fact that by simply replacing all the irreversible components with their reversible counterparts, a reversible version of a circuit is obtained. If the circuit is executed backwards, by replacing each gate by its inverse, and the output is provided as input then the original input is obtained.

Toffoli Matrix

How can one obtain a matrix representation for the Toffoli gate? The Toffoli gate has three inputs, so it is necessary to extend the previously presented vector representations for 1 and 2 bits, respectively presented in Expression 2.2 and Expression 2.10. This can easily be achieved by the introduction of a third bit. Accordingly, let the Toffoli gate inputs be labeled as a, b and c and let k_i denote the probability of bit k being in state i. Again at any given point in time a number of possible combinations may exist. The set of possible results combining all the possible states for the three bits is presented in Table 2.4.

Combination	a	b	c
1	a_0	b_0	c_0
2	a_0	b_0	c_1
3	a_0	b_1	c_0
4	a_0	b_1	c_1
5	a_1	b_0	c_0
6	a_1	b_0	c_1
7	a_1	b_1	c_0
8	a_1	b_1	c_1

Table 2.4: Possible combinations of results for three bits.

Recall from the previous discussions that the information contained in Table 2.4 regarding the set of possible states can be encoded as a vector. In this case the vector has an 8×1 dimension as illustrated by Expression 2.13.

$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} \tag{2.13}$$

However, the circuit also needs to be deterministic, *e.g.* when bit a is in state 0 then $a_0 = 1$ and since the probabilities need to sum up to one this automatically implies that $a_1 = 0$. The same holds true for the remaining bits. As a result, it is now possible to apply the specific details surrounding each combination of Table 2.4 to Expression 2.13, respectively:

- Combination 1 $\overset{\substack{a_0=1,a_1=0\\ b_0=1,b_1=0\\ c_0=1,c_1=0}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 1\times1\times1 \\ 1\times1\times0 \\ 1\times0\times1 \\ 1\times0\times0 \\ 0\times1\times1 \\ 0\times1\times0 \\ 0\times0\times1 \\ 0\times0\times0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Combination 2 $\overset{\substack{a_0=1,a_1=0\\ b_0=1,b_1=0\\ c_0=0,c_1=1}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 1\times1\times0 \\ 1\times1\times1 \\ 1\times0\times0 \\ 1\times0\times1 \\ 0\times1\times0 \\ 0\times1\times1 \\ 0\times0\times0 \\ 0\times0\times1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Combination 3 $\overset{\substack{a_0=1,a_1=0\\ b_0=0,b_1=1\\ c_0=1,c_1=0}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 1\times0\times1 \\ 1\times0\times0 \\ 1\times1\times1 \\ 1\times1\times0 \\ 0\times0\times1 \\ 0\times0\times0 \\ 0\times1\times1 \\ 0\times1\times0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Combination 4 $\overset{\substack{a_0=1,a_1=0\\ b_0=0,b_1=1\\ c_0=0,c_1=1}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 1\times0\times0 \\ 1\times0\times1 \\ 1\times1\times0 \\ 1\times1\times1 \\ 0\times0\times0 \\ 0\times0\times1 \\ 0\times1\times0 \\ 0\times1\times1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Combination 5 $\overset{\substack{a_0=0,a_1=1\\ b_0=1,b_1=0\\ c_0=1,c_1=0}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 0\times1\times1 \\ 0\times1\times0 \\ 0\times0\times1 \\ 0\times0\times0 \\ 1\times1\times1 \\ 1\times1\times0 \\ 1\times0\times1 \\ 1\times0\times0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Combination 6 $\overset{\substack{a_0=0,a_1=1\\ b_0=1,b_1=0\\ c_0=0,c_1=1}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 0\times1\times0 \\ 0\times1\times1 \\ 0\times0\times0 \\ 0\times0\times1 \\ 1\times1\times0 \\ 1\times1\times1 \\ 1\times0\times0 \\ 1\times0\times1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

- Combination 7 $\overset{\substack{a_0=0,a_1=1\\ b_0=0,b_1=1\\ c_0=1,c_1=0}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 0\times0\times1 \\ 0\times0\times0 \\ 0\times1\times1 \\ 0\times1\times0 \\ 1\times0\times1 \\ 1\times0\times0 \\ 1\times1\times1 \\ 1\times1\times0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

- Combination 8 $\overset{\substack{a_0=0,a_1=1\\ b_0=0,b_1=1\\ c_0=0,c_1=1}}{\rightarrow}$
$$\begin{pmatrix} a_0b_0c_0 \\ a_0b_0c_1 \\ a_0b_1c_0 \\ a_0b_1c_1 \\ a_1b_0c_0 \\ a_1b_0c_1 \\ a_1b_1c_0 \\ a_1b_1c_1 \end{pmatrix} = \begin{pmatrix} 0\times0\times0 \\ 0\times0\times1 \\ 0\times1\times0 \\ 0\times1\times1 \\ 1\times0\times0 \\ 1\times0\times1 \\ 1\times1\times0 \\ 1\times1\times1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

It is also important do draw attention to a few facts regarding the behaviour of the Toffoli gate and respective inputs originating from the set of possible combinations, namely:

- Combinations 1 through 6 imply that at least one of the control bits, *i.e.* a or b, is set to 0. Accordingly, by the Toffoli's gate definition no operation should be performed;

- Combination 7 has both control bits set to 1 and target bit c is in state 0. Accordingly, the Toffoli gate should flip the value of the target bit, *i.e.* c should be set to 1, and also output the control bits a and b. This is equivalent to a mapping between Combination 7 into Combination 8, since the values of the control bits are maintained and bit c is set;

- The same reasoning process applied immediately above can be reused for analyzing Combination 8. Combination 8 has both control bits set to 1 and target bit c is in state 1. Accordingly, the Toffoli gate should flip the value of bit c to 0, and again output the control bits a and b. This is equivalent to a mapping between Combination 8 into Combination 7, since the value of the control bits is maintained and bit c is set to 0.

Based on the aforementioned behavioural description, a matrix representation for the Toffoli gate should be in accordance with the following behaviour:

$$\text{TOFFOLI}\begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}=\begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{pmatrix},\quad \text{TOFFOLI}\begin{pmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{pmatrix}=\begin{pmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

$$\text{TOFFOLI}\begin{pmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{pmatrix}=\begin{pmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{pmatrix},\quad \text{TOFFOLI}\begin{pmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{pmatrix}$$

$$\text{TOFFOLI}\begin{pmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{pmatrix},\quad \text{TOFFOLI}\begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}$$

$$\text{TOFFOLI}\begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{pmatrix},\quad \text{TOFFOLI}\begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{pmatrix}$$

With the above rules in mind it is possible to obtain the matrix of the Toffoli gate, T, as presented in Expression 2.14. The first 6 lines of the Toffoli matrix effectively maintain the probability distributions of those inputs whose control bits are both not set to state 1. Otherwise, the last two rows are responsible for performing a flip on target bit c while maintaining the values for the control bits.

$$T = \begin{pmatrix} T|0\rangle & T|1\rangle & T|2\rangle & T|3\rangle & T|4\rangle & T|5\rangle & T|6\rangle & T|7\rangle \\ |0\rangle & |1\rangle & |2\rangle & |3\rangle & |4\rangle & |5\rangle & |7\rangle & |6\rangle \end{pmatrix}$$

$$= \begin{array}{c} \begin{matrix} T|0\rangle & T|1\rangle & T|2\rangle & T|3\rangle & T|4\rangle & T|5\rangle & T|6\rangle & T|7\rangle \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \tag{2.14}$$

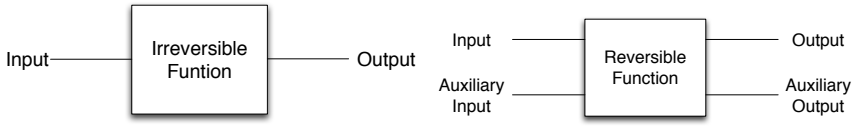
2.4.2 Reversible Circuits

Recall that a key incentive supporting reversible computation stems from a need for higher energy efficiency. Traditionally, classical computation is seen as an irreversible process, a direct consequence from the use of thermodynamically costly many-to-one functions. The reversible computation paradigm presents an alternative solution with logical elements such as the Toffoli gate providing the support for reversibility. It was also described how the Toffoli gate could be employed in order to simulate irreversible operations, such as the universal NAND gate.

Generally, any irreversible circuit can be made reversible by substituting each irreversible gate by an equivalent reversible gate [204]. In doing so a certain constant number of inputs and outputs is introduced to each irreversible gate. It is this additional information that provides for reversible computation. By assigning specified values to some input bits the desired gate behaviour is obtained. This situation is illustrated in Figure 2.9.

In [118] the authors draw attention to the fact that given an irreversible function f, a reversible mapping can be constructed with the form illustrated in Expression 2.15, where x is a set of bits, also known as a register, and c is an auxiliary control bit.

$$(x, c) \mapsto (x, c \oplus f(x)) \tag{2.15}$$



(a) (b)

Figure 2.9: It is possible to transform an irreversible function 2.9a into a reversible function 2.9b. This mapping can be performed with the introduction of a number of constants and auxiliary input and output bits. (Source: [203])

Toffoli draws attention to a number of facts resulting from a physical implementation perspective, namely [204]:

- Signals are encoded in some form of energy;
- Each auxiliary constant input incurs an associated energy cost and each auxiliary output implies the removal of energy, with the associated thermodynamic penalty;

2.5 Conclusions

Energy consumption in computation is deeply linked to the reversibility of the computation. Operations resulting in the loss or erasure of information incur a thermal penalty. Operators incorporating concepts such as the ones described by the Toffoli gate allow for reversible circuits to be obtained from traditionally irreversible circuits. A process which, in principle, can always be performed. This conversion procedure can be performed by adding a moderate number of resources.

Chapter 3

Quantum Computation Overview

3.1 Introduction

Quantum computation is a field resulting from the combination of exploring quantum mechanics alongside computer science. The underlying physical rules that govern computation are changed to a quantum setting. This modification allows the model to have far-reaching consequences on how computation is performed and perceived. Since quantum computation represents such a fundamental shift from classical computation it is important to consider what are the potential implications? Namely, what are the requirements associated with quantum computation? Do the same principles that govern classical computation apply to a quantum context? If not, how should computation be approached from a quantum perspective? Additionally, what are the key differences between both models of computation? Does quantum computation provide any meaningful advantage over the classical model? This chapter provides answers to these questions by presenting an overview of the basic concepts surrounding quantum computation. In order to do so, this chapter first starts by describing a probabilistic perspective of computation in Section 3.2. This approach is then extended in Section 3.3 in order to accommodate for complex amplitudes. This model is then employed to present some of the fundamental notions surrounding quantum computation in Section 3.4. The set of results presented there are fundamental in order to explain Grover's algorithm in Section 3.5, which is a fundamental result for the remaining work presented in this thesis.

3.2 Probabilistic Computation

The mathematical framework laid down in previous sections combined into a single hybrid approach a matrix perspective alongside a probabilistic point of view. These factors were responsible for producing a vector notation conveying not only the state of individual bits but also the probability for each possible bit state combination (as illustrated by Expression 2.2, Expression 2.10 and Expression 2.13). However, this hybrid approach is usually applied to translate the deterministic behaviour usually seen in classical computation, *i.e.* the probability that an individual bit is in a given state is either 0 or 1. Although this seems a rational approach, it is nonetheless a restriction on the general probability theory. If individual bit probabilities are allowed to strain beyond 0 or 1, *i.e.* $p \in \mathbb{R}$ and $0 \leq p \leq 1$, then it is possible to develop a new set of theoretical extensions.

3.2.1 Probabilistic Bit

In order to better understand probabilistic computation it is useful to first take into consideration a simple example consisting of a single bit a and let a_0 denote the probability that bit a is in state 0, and a_1 the probability that it is in state 1. Accordingly, it is possible to build an adequate notation reflecting this probability distribution for bit a, as illustrated by Expression 3.1. In addition, both a_0 and a_1 should be greater or equal than zero, *i.e.* $a_0 \geq 0$ and $a_1 \geq 0$. Also, both probabilities should some up to one, *i.e.* $a_0 + a_1 = 1$.

$$a_0 \text{ State}_0 + a_1 \text{ State}_1 \tag{3.1}$$

3.2.2 Multiple Probabilistic Bits

In this section the notation previously presented will be extended in order to accommodate any number of bits. Suppose a two bit system under consideration, respectively labeled as a and b. Also, as previously defined, let k_i represent the probability of bit k being in state i. Now, instead of being limited to 0 or 1 probabilities, it becomes possible to employ any kind of initial probability distribution. As a concrete example suppose that $a_0 = 0.7$ and $b_0 = 0.25$. Since probabilities need to sum up to one, then $a_1 = 0.3$ and also $b_1 = 0.75$. Table 3.1 illustrates the probabilities for each possible combination, *i.e.* state.

Table 3.1 also illustrates the close association between a state and its respective probability. Although in a probabilistic system the state of the system is not known for certain, it is possible to have a practical understanding of the probabilistic distribution of the states [99].

Combination	a	b	Individual Probabilities	Probability
1	0	0	a_0b_0	$0.7 \times 0.25 = 0.175$
2	0	0	a_0b_1	$0.7 \times 0.75 = 0.525$
3	0	1	a_1b_0	$0.3 \times 0.25 = 0.075$
4	0	1	a_1b_1	$0.3 \times 0.75 = 0.225$

Table 3.1: Probability distribution for three bits with probability distribution $a_0 = 0.7, b_0 = 0.25, a_1 = 0.3, b_1 = 0.75$.

This situation is illustrated in Expression 3.2, with the sum of the probabilities normalized to 1, *i.e.* $\sum_{i,j} a_i b_j = 1$. The probability distribution of Table 3.1 can also be presented as illustrated by Expression 3.3.

$$a_0b_0 \text{ State}_{00} + a_0b_1 \text{ State}_{01} + a_1b_0 \text{ State}_{10} + a_1b_1 \text{ State}_{11} \tag{3.2}$$

$$0.175 \text{ State}_{00} + 0.525 \text{ State}_{01} + 0.075 \text{ State}_{10} + 0.225 \text{ State}_{11} \tag{3.3}$$

Expression 3.2 can be simplified introducing the mathematical "ket" symbol $|\cdots\rangle$. For the moment $|a\rangle$ will simply denote state a, in order to avoid having to explicitly write the word "State". Expression 3.2 can thus gain a new form, as illustrated by Expression 3.4.

$$a_0b_0|00\rangle + a_0b_1|01\rangle + a_1b_0|10\rangle + a_1b_1|11\rangle \tag{3.4}$$

The notation presented in Expression 3.4 can easily be extended to accommodate any number of states and accordingly any number of bits. As mentioned in [99], let $x_1, \cdots, x_n$ denote the set of possible states with associated probabilities $p_1, \cdots, p_n$ then the probability distribution over states x_k can be represented as illustrated by Expression 3.5. Again $\forall_i p_i \geq 0$ and $\sum_i p_i = 1$. This probability distribution is also commonly referred to as a mixed state, with individual states x_k labeled as pure states.

$$p_1|x_1\rangle + \cdots + p_n|x_n\rangle \tag{3.5}$$

It is important to mention that the distribution presented in Expression 3.5 might be viewed as a vector with non-negative coordinates that sum up to 1 in an n-dimensional real vector space having $|x_1\rangle, \cdots, |x_n\rangle$ as basis vectors.

3.2.3 Time Evolution

A probabilistic system is capable of undergoing discrete time evolution [99]. Although the specific state towards which the system will evolve is not known it is possible to have a certain understanding regarding the evolution of any specific state x_k. Subsequent system evaluations of each state x_k should continue to reflect the system's probabilistic nature. Accordingly, any state x_k can evolve to state x_l with probability p_{kl}. The overall state evolution is illustrated by Expression 3.6. This notation also reflects a probability distribution regarding state x_k thus is should also be clear that $p_{k1} + \cdots + p_{kn} = 1$.

$$|x_k\rangle \mapsto p_{k1}|x_1\rangle + \cdots + p_{kn}|x_n\rangle \tag{3.6}$$

By combining the system's probability distribution (Expression 3.5) with individual state evolution (Expression 3.6) allows one to study the system's overall discrete time evolution. This situation is illustrated in Expression 3.7 [99], where $p'_i = p_1 p_{1i} + \cdots + p_n p_{ni}$.

$$\begin{aligned}
&p_1|x_1\rangle + \cdots + p_n|x_n\rangle \mapsto \\
&\mapsto p_1\left(p_{11}|x_1\rangle + \cdots + p_{1n}|x_n\rangle\right) + \cdots + p_n\left(p_{n1}|x_1\rangle + \cdots + p_{nn}|x_n\rangle\right) = \\
&= \left(p_1 p_{11} + \cdots + p_n p_{n1}\right)|x_1\rangle + \cdots + \left(p_1 p_{1n} + \cdots + p_n p_{nn}\right)|x_n\rangle = \\
&= p'_1|x_1\rangle + \cdots + p'_n|x_n\rangle
\end{aligned} \tag{3.7}$$

A probabilistic system with the aforementioned time evolution is commonly labeled as a Markov chain [179]. Expression 3.7 showcases the fact that in the end another probability distribution is obtained. Accordingly, $p'_1 + \cdots + p'_n = 1$. It should be drawn into attention that this is an important conclusion, *i.e.* the time evolution sends each system's probability distribution into a combination of all the states with non-negative coefficients that sum up to 1 [99]. This behaviour is to be expected and allows for further extensions. These extensions will be responsible for a different kind of interaction resulting in a profound impact in a system's discrete time evolution.

3.3 Extending the Non-Deterministic Framework

Succinctly, the previous section described: (1) how useful a probabilistic approach can be when characterizing a computation; and (2) how a probabilistic system evolves during dis-

crete time steps. The mathematical framework previously presented not only provided a non-deterministic perspective towards computation but also a simple and elegant notation for doing so. So the question naturally arises: How might this mathematical framework be extended whilst maintaining the elegance of the probabilistic approach? In order to answer this question it becomes necessary to stray beyond the real-number theory set that has been employed far. A natural observation to perform is that the set of complex numbers $\mathbb{C}$ contains the ordinary real numbers $\mathbb{R}$ so theoretically this might be a good place to start. Accordingly, Expression 3.5 can be extended in order to contemplate complex numbers α_i (known as amplitudes [99]) as illustrated by Expression 3.8.

$$\alpha_1|x_1\rangle + \cdots + \alpha_n|x_n\rangle \tag{3.8}$$

Although this notation already carries a significant resemblance with the probabilistic approach previously presented it can still be further improved. Namely, by requiring the vector $(\alpha_1, \cdots, \alpha_n)$ to be unit-length preserving it becomes possible to ensure that $|\alpha_1|^2 + \cdots + |\alpha_n|^2 = 1$. As a result, $|\alpha_i|^2$ may be perceived as translating the probability of observing state x_i. This approach fits nicely with the previously mentioned probability framework.

3.3.1 Time Evolution

Understanding the time evolution of such a system can be performed by adding the complex amplitudes requirement to Expression 3.6, as illustrated by Expression 3.9, which allows one to study system's evolution in discrete time steps.

$$|x_k\rangle \mapsto \alpha_{k1}|x_1\rangle + \cdots + \alpha_{kn}|x_n\rangle \tag{3.9}$$

This situation is illustrated in Expression 3.10 [99] where individual state evolution was applied, as expressed by Expression 3.9 against the system's state notation, as presented in Expression 3.8, where $\alpha'_i = \alpha_1\alpha_{1i} + \cdots + \alpha_n\alpha_{ni}$. This behaviour ensures that the operations performed during the evolution operation are length preserving, *i.e.* $|\alpha_1|^2 + \cdots + |\alpha_n|^2 = |\alpha'_1|^2 + \cdots + |\alpha'_n|^2$.

$$\begin{aligned}
&\alpha_1|x_1\rangle + \cdots + \alpha_n|x_n\rangle \mapsto \\
&\mapsto \alpha_1\left(\alpha_{11}|x_1\rangle + \cdots + \alpha_{1n}|x_n\rangle\right) + \cdots + \alpha_n\left(\alpha_{n1}|x_1\rangle + \cdots + \alpha_{nn}|x_n\rangle\right) = \\
&= \left(\alpha_1\alpha_{11} + \cdots + \alpha_n\alpha_{n1}\right)|x_1\rangle + \cdots + \left(\alpha_1\alpha_{1n} + \cdots + \alpha_n\alpha_{nn}\right)|x_n\rangle = \\
&= \alpha'_1|x_1\rangle + \cdots + \alpha'_n|x_n\rangle
\end{aligned} \tag{3.10}$$

Expression 3.11 can be derived from Expression 3.10, which can be presented in matrix form, relating amplitudes $\alpha_1, \cdots, \alpha_n$ and $\alpha'_1, \cdots, \alpha'_n$ as illustrated by Expression 3.12 [99].

$$\begin{cases}
\alpha'_1 = \alpha_1\alpha_{11} + \cdots + \alpha_n\alpha_{n1} \\
\vdots \\
\alpha'_n = \alpha_1\alpha_{1n} + \cdots + \alpha_n\alpha_{nn}
\end{cases} \tag{3.11}$$

$$\begin{pmatrix} \alpha'_1 \\ \vdots \\ \alpha'_n \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \tag{3.12}$$

This kind of linear, length-preserving and smooth operators are known in linear algebra terms as unitary matrices. 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 [99]. This is equivalent to say that $A^{-1} = A^\dagger$ and consequently $A^{-1}A = A^\dagger A = I$. The unitary requirement also implies that a quantum state evolution is always invertible, thus the close relationship with reversible computation.

3.3.2 Assessing The Key Differences

How well does this latest approach fare when compared with the initial real-valued probabilistic framework? In order to understand clearly the key differences it is useful to proceed with a couple of well known examples taken from [99]. First, consider the example of a fair coin.

Example 1 - Classical Approach Tossing a far coin will give head h or tail t with a probability of $\frac{1}{2}$ (for the specific case of this example it is important to mention that only non-negative real values are considered, *i.e.* $\mathbb{R}_0^+$). Accordingly, independently of whether the coins starts in the heads or tails position, the overall state of the

system after a single time evolution step can be described as illustrated, respectively, by Expression 3.13 and Expression 3.14.

$$|h\rangle \mapsto \frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle \tag{3.13}$$

$$|t\rangle \mapsto \frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle \tag{3.14}$$

The initial states $|h\rangle$ and $|t\rangle$ both produce the same state after a single step. Subsequent time steps on the system's state can now be determined, as illustrated by Expression 3.15.

$$\begin{aligned}\frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle &\mapsto \frac{1}{2}\left(\frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle\right) + \frac{1}{2}\left(\frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle\right) \\ &= \frac{1}{4}|h\rangle + \frac{1}{4}|t\rangle + \frac{1}{4}|h\rangle + \frac{1}{4}|t\rangle \\ &= \frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle \end{aligned} \tag{3.15}$$

Since all the probabilities are always non-negative real numbers and since probabilities need to sum up to 1 the final configuration will be equal to the initial one.

Example 2 - Non-Classical Approach Assume that a system with a similar behaviour to that of the fair coin of the previous example is being employed. However, this time the system's state description will be restricted to amplitudes, *i.e.* $\mathbb{C}$, instead of probabilities, *i.e.* $\mathbb{R}_0^+$. The mapping between probabilities and amplitudes can be easily performed since the probability of observing a certain state x_i with amplitude α_i is $|\alpha_i|^2$. Accordingly, for probability $\frac{1}{2}$ individual states can have amplitudes $\pm\frac{1}{\sqrt{2}}$. Suppose that the system's time evolution is provided by Expression 3.16 and Expression 3.17.

$$|h\rangle \mapsto \frac{1}{\sqrt{2}}|h\rangle + \frac{1}{\sqrt{2}}|t\rangle \tag{3.16}$$

$$|t\rangle \mapsto \frac{1}{\sqrt{2}}|h\rangle - \frac{1}{\sqrt{2}}|t\rangle \tag{3.17}$$

Both of the above expressions are consistent with the desired probabilistic behaviour of the previous example but built-upon a different theoretical foundation. First, it is

useful to examine an additional time evolution step regarding the initial state $|h\rangle$ as described in Expression 3.18.

$$\begin{aligned}\frac{1}{\sqrt{2}}|h\rangle + \frac{1}{\sqrt{2}}|t\rangle &\mapsto \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|h\rangle + \frac{1}{\sqrt{2}}|t\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|h\rangle - \frac{1}{\sqrt{2}}|t\rangle\right) \\ &= \frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle + \frac{1}{2}|h\rangle - \frac{1}{2}|t\rangle \\ &= |h\rangle \end{aligned} \tag{3.18}$$

The same procedure can be applied to initial state $|t\rangle$ as depicted in Expression 3.19.

$$\begin{aligned}\frac{1}{\sqrt{2}}|h\rangle - \frac{1}{\sqrt{2}}|t\rangle &\mapsto \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|h\rangle + \frac{1}{\sqrt{2}}|t\rangle\right) - \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|h\rangle - \frac{1}{\sqrt{2}}|t\rangle\right) \\ &= \frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle - \frac{1}{2}|h\rangle + \frac{1}{2}|t\rangle \\ &= |t\rangle \end{aligned} \tag{3.19}$$

These two examples are crucial for obtaining a clear understanding regarding the main differences between the classical and non-classical approaches. As previously stated, the time evolution of the real-valued approach illustrated in Expression 3.5 sends each basis vector again into a combination of all the basis vectors with non-negative coefficients that sum up to 1. In contrast, the complex-valued amplitudes of Expression 3.8 allow the system to evolve very differently. More precisely, when the combination of amplitudes for a given state cancel each other out the process is known as destructive interference. Naturally, destructive interference does not occur when dealing with probabilities since all the coefficients are always non-negative real numbers [99]. On the other hand, when the amplitudes for a given state amplify each other, the process is called constructive interference. Quantum interference between different computational basis in order to enhance correct outcomes, and in the process eliminate incorrect states, is a key feature of quantum computation [49]. Finally, it is important to mention that the joint operation described by Expression 3.16 and Expression 3.17 is known as the Hadamard transform, which has the matrix form described in Expression 3.20.

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.20}$$

3.4 Quantum Computation Fundamentals

As previously stated, irreversible computational models are responsible for thermal entropy of the system and its surroundings [15]. Reversible computation plays a pivotal duty in the quest to develop ever more powerful and energy efficient computers. Curiously, reversibility is also a key requirement quantum computation [99]. This reversibility, which translates into the quantum evolution postulate, is a key feature of quantum physics, and is thus an intricate part of quantum computation. The following sections present some of the key concepts behind quantum computation by establishing the parallels between quantum computation and the complex-valued non-deterministic approach previously presented. Namely: Section 3.4.1 presents the postulates that govern quantum computation; Section 3.4.2 describes the quantum equivalent of the classical bit; Section 3.4.3 focuses on the concept of quantum parallelism; Section 3.4.4 describes a simple quantum procedure, respectively Deutsch's algorithm.

3.4.1 Quantum Requirements

In this section some of the key features of a finite state quantum system are described in state-vector formalism [99] [118].

Hilbert Space

The linear algebra notation used in quantum computation is the Dirac notation, which was invented by Paul Dirac [59] [60]. In the Dirac notation, the symbol identifying a vector is written inside a "ket", *i.e.* $|\cdots\rangle$. This is the main reason why the ket symbol was previously employed to describe states of a system. The vector spaces considered are restrained to complex numbers $\mathbb{C}$. In quantum mechanics the state of a system with n qubits is represented as a unit-length vector in an 2^n-dimensional complex vector space H_{2^n}. Such finite-dimensional vector spaces are labeled as Hilbert spaces. These states can be represented as finite column vectors in a given basis [99]. Each computational basis of such an Hilbert space can be represented through an n-length binary string encoding mechanism. Each one of the 2^n basis vectors can be associated to a column vector. The standard way to perform this association is depicted in Expression 3.21 [118].

$$|\underbrace{0\cdots00}_{n}\rangle = \left.\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}\right\} 2^n, |0\cdots01\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix}, \cdots, |1\cdots11\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \tag{3.21}$$

General System State

The state of a quantum system is a unit-length vector expressed in terms of the chosen computational basis, as depicted in Expression 3.22. These general states are called superpositions of basis states. Also, since the vector is unit-length, then $|\alpha_1|^2 + \cdots + |\alpha_n|^2 = 1$. Thus, Expression 3.22 also induces a probability distribution in the following manner: when observing a general state such as the one depicted in Expression 3.22, then a state x_i (in quantum parlance a property) is observed with probability $|\alpha_i|^2$.

$$\alpha_1|x_1\rangle + \cdots + \alpha_n|x_n\rangle \tag{3.22}$$

Compound Systems

Additionally, if two quantum systems interact, respectively an n-level system in space H_n and an m-level system in H_m, then the state space of the combined system is described by the tensor product (also known as the product) $H_m \otimes H_n$.

System Evolution

Finally, changes occurring to a quantum state can be described using the language of quantum computation. Analogous to the way a classical computer is built from an electrical circuit containing wires and logic gates, a quantum computer is built from a quantum circuit containing wires and elementary quantum gates to carry around and manipulate the quantum information [58]. In quantum computation, the operations are described through reversible operators, expressed in terms of matrices. This approach closely resembles that of the previously presented reversible computation. The crucial requirement is for the state of quantum systems to change via unitary transformation. The unitarity requirement provides further support for the reversible computation approach.

As a concrete example the unitarity of the Toffoli matrix can be verified, as presented in Expression 2.14, and also of the Hadamard transform, presented in Expression 3.20. Let $H^\dagger$

denote the complex transpose of matrix H then it is easy to verify the result illustrated in Expression 3.23, where the symbol '$\times$' depicts matrix multiplication.

$$H \times H^{\dagger} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \times \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = I_{2\times 2} \tag{3.23}$$

The same procedure can also be applied to verify that the Toffoli gate is also a unitary operator. Accordingly, let U_T denote the Toffoli matrix and $U_T^{\dagger}$ be the complex transpose of matrix U_T. Expression 3.24 illustrates that matrix U_T is unitary. Accordingly, both the Toffoli gate and the Hadamard transform are legitimate quantum operators.

$$U_T \times U_T^{\dagger} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} = I_{8\times 8} \tag{3.24}$$

3.4.2 The quantum bit

Computation is usually perceived as an abstract mathematical process. However, no matter how abstract the computational process is, it must still be performed at a physical level. Classical computers work by manipulating entities known as bits that exist in one of two states, respectively 0 or 1. Typically, these discrete states are represented by voltage ranges, direction of magnetization and also transistor conductivity. The bit is thus the crucial element supporting classical computation and information. The bit concept inspired an analogous fundamental concept in quantum computation and information, the quantum bit, a.k.a. qubit.

Qubit

What is a qubit? Analogously to the classical bit a qubit also has a state. In fact, two possible states for a qubit are the states $|0\rangle$ and $|1\rangle$. However, there is a fundamental different concept between bits and qubits. A qubit can also be in a state other than $|0\rangle$ or $|1\rangle$. A qubit can also be in a superposition, or put more simply, as a linear combination of the computational

basis $|0\rangle$ and $|1\rangle$. This situation is illustrated in Expression 3.25, which closely resembles that of a general quantum system, respectively presented in Expression 3.22, where $\alpha, \beta \in \mathbb{C}$. This behaviour is strange since a classical bit acts like a coin: either heads or tails up. A qubit's superposition ability is counterintuitive to the classical understanding of the physical world, since it can "exist" as a combination of $|0\rangle$ and $|1\rangle$ [1].

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{3.25}$$

The quantum laws also impose a set of seemingly strange rules, since they run counter to our everyday understanding of the word. Namely, it is not possible to determine the quantum state of a qubit as presented in Expression 3.25. When a qubit is observed, or in quantum vocabulary measured, either the result 0 or the result 1 is obtained. More concretely, the probability of measuring 0 is $|\alpha|^2$ whilst the probability of measuring 1 is $|\beta|^2$. Quantum mechanics provides no explanation for this kind of random collapse into 0 or 1.

Multiples qubits

The aforementioned concept of a qubit can also be easily extended in order to accommodate any number of qubits. For example a two qubit system has four computational basis, namely $|00\rangle, |01\rangle, |10\rangle$, and $|11\rangle$. Accordingly a two qubit system can be expressed as a linear combination of these computation basis, as illustrated by Expression 3.26.

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{3.26}$$

3.4.3 Quantum parallelism

In quantum computation, it is common to employ computational structures known as oracles to verify if an argument belongs to a certain language. This type of unitary operators are typically represented as U_g or O depending on the literature in question. Oracles are employed in order to indicate which of the values present in an amplitude register corresponds to the searched ones. Function $g(x)$ has the form presented in Expression 3.28. This process can be performed by adding an additional input bit c to a unitary operator and performing a XOR operation, as illustrated by Expression 3.27.

[1] It is important to mention that qubits are mostly dealt as abstract mathematical objects, although they can be physically implemented *e.g.* [140] [175]. However, by choosing to treat qubits as mathematical entities, the quantum computation community has been able to focus on developing new theories in a separate fashion from the concrete details of a physical implementation

$$U_g : |x\rangle|c\rangle \mapsto |x\rangle|c \oplus g(x)\rangle \tag{3.27}$$

$$g(x) = \begin{cases} 1 & \text{if } x \text{ is a solution} \\ 0 & \text{otherwise} \end{cases} \tag{3.28}$$

Recall that one of the key features of quantum computation is the ability for a qubit to be in a superposition of states. Accordingly, take into account the circuit presented in Figure 3.1 [164], which applies U_f to an input in the superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$. This initial input superposition fed to the circuit can be obtained through the Hadamard gate and the computational basis state $|0\rangle$. In order to help understand the analysis of the circuit behaviour it is useful to introduce in Figure 3.1 two auxiliary states, respectively $|\psi_0\rangle$ and $|\psi_1\rangle$. State $|\psi_0\rangle$ describes the composite input system whilst $|\psi_1\rangle$ details the output.

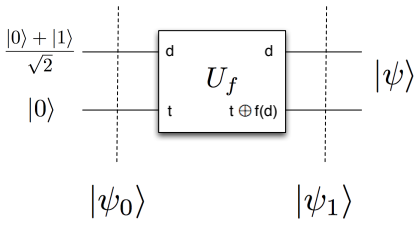


Figure 3.1: U_f evolves inputs $|d\rangle|t\rangle$ to $|d\rangle|t \oplus f(d)\rangle$. (Source: [164])

U_f acts on a composite system consisting of superposition $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and an auxiliary qubit set to $|0\rangle$, respectively $|\psi_0\rangle$ as illustrated by Expression 3.29.

$$|\psi_0\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)|0\rangle \tag{3.29}$$

The evolution of state ψ_0 to ψ_1 that results from applying unitary operator U_f is illustrated in Expression 3.30.

$$
\begin{aligned}
|\psi_1\rangle &= U_f|\psi_0\rangle \\
&= U_f\left[\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)|0\rangle\right] \\
&= U_f\left(\frac{|0\rangle|0\rangle+|1\rangle|0\rangle}{\sqrt{2}}\right) \\
&= \frac{U_f|0\rangle|0\rangle+U_f|1\rangle|0\rangle}{\sqrt{2}} \\
&= \frac{|0\rangle|0\oplus f(0)\rangle+|1\rangle|0\oplus f(1)\rangle}{\sqrt{2}} \\
&= \frac{|0\rangle|f(0)\rangle+|1\rangle|f(1)\rangle}{\sqrt{2}} \\
&= \frac{|0\rangle|f(0)\rangle}{\sqrt{2}}+\frac{|1\rangle|f(1)\rangle}{\sqrt{2}}
\end{aligned}
\tag{3.30}
$$

Before advancing any further it is important to assess the true impact of Expression 3.30. To the system's initial state, respectively $|\psi_0\rangle$, operator U_f was applied once, which resulted in state $|\psi_1\rangle$ being obtained. State $|\psi_1\rangle$ is itself a superposition containing information about $|f(0)\rangle$ and $|f(1)\rangle$. Ergo, with a single application of U_f two values of $f(d)$ were simultaneously evaluated. This feature is known as quantum parallelism and contrasts with classical parallel evaluation, which would require two individual circuits to perform the same operation. However, due to the quantum collapse measurement effect it is only possible to obtain one of those composite states, *i.e.* composite state $|0\rangle|f(0)\rangle$ is obtained with probability $\frac{1}{2}$ (recall that $p=|\alpha|^2$), or state $|1\rangle|f(1)\rangle$ with probability $\frac{1}{2}$. Apparently, this parallelism is not without its consequences. Fortunately, this is not the end of the story as Deutsch proved in [57].

3.4.4 Deutsch Algorithm

The Deutsch algorithm exemplifies how quantum interference can be employed alongside quantum parallelism to extract information about more than one value of $f(d)$ from superposition states like the one in Expression 3.30. The circuit responsible for implementing Deutsch's algorithm is presented in Figure 3.2. In a similar fashion to the circuit presented in Figure 3.1 the superposition principle will be employed by employing two Hadamard gates. Again, several auxiliary states need to be introduced in order to aide in the understanding of the circuit, respectively $|\psi_0\rangle, |\psi_1\rangle, |\psi_2\rangle$ and $|\psi_3\rangle$. The input state $|\psi_0\rangle$ is illustrated in Expression 3.31.

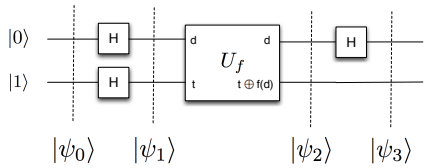


Figure 3.2: Circuit implementing Deutsch algorithm. (Source: [164])

$$|\psi_0\rangle = |0\rangle|1\rangle \tag{3.31}$$

State $|\psi_1\rangle$ translates the two Hadamard transforms that are applied in order to obtain the superpositions of the input lines. This situation is illustrated in Expression 3.32.

$$\begin{aligned} |\psi_1\rangle &= H|\psi_0\rangle H|\psi_1\rangle \\ &= \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \end{aligned} \tag{3.32}$$

State $|\psi_2\rangle$ translates the evolution the system undergoes when the unitary operator U_f is applied to the superpositions. However, before analyzing state $|\psi_2\rangle$ be mindful of what happens when U_f is applied to a general data state $|d\rangle$ and the superposition $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, this situation is illustrated in Expression 3.33.

$$
\begin{aligned}
U_f\left[|d\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right] &= U_f\left(\frac{|d\rangle|0\rangle-|d\rangle|1\rangle}{\sqrt{2}}\right)\\
&= \frac{U_f|d\rangle|0\rangle - U_f|d\rangle|1\rangle}{\sqrt{2}}\\
&= \frac{|d\rangle|0\oplus f(d)\rangle - |d\rangle|1\oplus f(d)\rangle}{\sqrt{2}}\\
&= \frac{|d\rangle|f(d)\rangle - |d\rangle|1\oplus f(d)\rangle}{\sqrt{2}}\\
&= \begin{cases} \frac{|d\rangle|0\rangle-|d\rangle|1\oplus 0\rangle}{\sqrt{2}} & \text{if } f(d)=0\\ \frac{|d\rangle|1\rangle-|d\rangle|1\oplus 1\rangle}{\sqrt{2}} & \text{if } f(d)=1 \end{cases}\\
&= \begin{cases} \frac{|d\rangle|0\rangle-|d\rangle|1\rangle}{\sqrt{2}} & \text{if } f(d)=0\\ \frac{|d\rangle|1\rangle-|d\rangle|0\rangle}{\sqrt{2}} & \text{if } f(d)=1 \end{cases}\\
&= \begin{cases} \frac{|d\rangle(|0\rangle-|1\rangle)}{\sqrt{2}} & \text{if } f(d)=0\\ \frac{-|d\rangle(|0\rangle-|1\rangle)}{\sqrt{2}} & \text{if } f(d)=1 \end{cases}\\
&= (-1)^{f(d)}|d\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \qquad (3.33)
\end{aligned}
$$

Since this is the more delicate part of the analysis it is useful to continue by stages. As previously mentioned, state $|\psi_2\rangle$ translates the evolution of the superpositions, as depicted by Expression 3.34.

$$
\begin{aligned}
|\psi_2\rangle &= U_f|\psi_1\rangle\\
&= U_f\left[\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right] \qquad (3.34)\\
&= U_f\left[\frac{|0\rangle}{\sqrt{2}}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{|1\rangle}{\sqrt{2}}\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right]\\
&= \frac{1}{\sqrt{2}}U_f\left[|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right]+\frac{1}{\sqrt{2}}U_f\left[|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right]\\
&= \frac{1}{\sqrt{2}}(-1)^{f(0)}|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{f(1)}|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \qquad (3.35)
\end{aligned}
$$

Expression 3.35 is a direct application of the result presented in Expression 3.33. In order to advance recall that $f(x)$ outputs a binary value. Accordingly, Expression 3.35 needs to be evaluated for all possible combinations, as depicted in Expression 3.36.

$$\frac{1}{\sqrt{2}}(-1)^{f(0)}|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{f(1)}|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)=$$

$$=\begin{cases}\frac{1}{\sqrt{2}}(-1)^{0}|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{0}|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=0, f(1)=0\\ \frac{1}{\sqrt{2}}(-1)^{0}|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{1}|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=0, f(1)=1\\ \frac{1}{\sqrt{2}}(-1)^{1}|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{0}|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=1, f(1)=0\\ \frac{1}{\sqrt{2}}(-1)^{1}|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)+\frac{1}{\sqrt{2}}(-1)^{1}|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=1, f(1)=1\end{cases} \tag{3.36}$$

$$=\begin{cases}\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=0, f(1)=0\\ \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=0, f(1)=1\\ \left(\frac{-|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=1, f(1)=0\\ \left(\frac{-|0\rangle-|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=1, f(1)=1\end{cases} \tag{3.37}$$

Therefore, applying U_f to $|\psi_1\rangle$ leaves the state in one of two possibilities, as illustrated by Expression 3.38 [164].

$$|\psi_2\rangle=\begin{cases}\pm\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=f(1)\\ \pm\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)\neq f(1)\end{cases} \tag{3.38}$$

Finally, state $|\psi_3\rangle$ merely reflects the application of the Hadamard transform on the first qubits, as depicted in Expression 3.39 [164].

$$|\psi_3\rangle=\begin{cases}\pm|0\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)=f(1)\\ \pm|1\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(0)\neq f(1)\end{cases} \tag{3.39}$$

What extra information does state $|\psi_3\rangle$ provide? If Expression 3.39 is read carefully it is possible to see that both possibilities differ only on the value of the first qubit, respectively either $|0\rangle$ or $|1\rangle$. Therefore, if a measurement is performed on the first qubit and 0 is obtain then automatically $f(0) = f(1)$, *i.e.* the function is said to be constant. Otherwise, if the measurement of the first qubit is 1 then $f(0) \neq f(1)$, *i.e.* the function is labeled as balanced. Both these conclusions can be learned with single quantum evaluation of function f. Classically, determining if a function is balanced would require two different function evaluations.

3.5 Grover's algorithm

Grover's algorithm [82] builds a superposition representing the search space and employs a specifically built oracle alongside an iterative amplitude amplification scheme. The iterate is responsible for systematically increasing the amplitude of solution states and performing the adequate unitary amplitude reduction of non-solution states. The algorithm executes in $O(\sqrt{N})$ time. This speedup is identical to the one delivered by the approach described in [190]. Once the search algorithm terminates, the superposition state is measured and a solution state is obtained, if one exists, with high probability. This type of oracle-based amplitude amplification schemes was shown to be optimal in [26], *i.e.* no fewer than $O(\sqrt{N})$ queries can be performed in order to obtain solution states with high probability.

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. This procedure alongside setting $|a\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ 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 applies a phase shift of -1 to all states orthogonal to state $|0\rangle$. Operator $U_{0^\perp}$ can thus be written as presented in Expression 3.40. This means that operator $HU_{0^\perp}H$ can also be expressed as depicted in Expression 3.41.

$$U_{0^\perp} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{pmatrix} = 2|0\rangle\langle 0| - \mathbb{I} \tag{3.40}$$

$$HU_{0^\perp}H = H(2|0\rangle\langle 0| - \mathbb{I})H = 2H|0\rangle\langle 0|H^\dagger - H \times H = 2|\psi\rangle\langle\psi| - \mathbb{I} \tag{3.41}$$

Both of these expressions are important in order to prove that operator $HU_{0^\perp}H$ inverts about the mean. More precisely, given a superposition state $|\phi\rangle = \sum_x \alpha_x|x\rangle$ it is possible to show that $HU_{0^\perp}H|\phi\rangle = \sum_x(2\mu - \alpha_x)|x\rangle$ where $\mu = \frac{1}{N}\sum_x \alpha_x$ is the mean value of the amplitudes. This demonstration is shown in Expression 3.42.

$$
\begin{aligned}
HU_{0^\perp}H|\phi\rangle &= (2|\psi\rangle\langle\psi| - \mathbb{I})\sum_x \alpha_x|x\rangle \\
&= 2|\psi\rangle\left(\frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}\alpha_x\langle y||x\rangle\right) - \sum_x \alpha_x|x\rangle \\
&= 2|\psi\rangle\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\alpha_x - \sum_x \alpha_x|x\rangle \\
&= 2\frac{1}{\sqrt{N}}\sum_{y=0}^{N-1}|y\rangle\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}\alpha_x - \sum_x \alpha_x|x\rangle \\
&= 2\sum_{y=0}^{N-1}|y\rangle\frac{1}{N}\sum_{x=0}^{N-1}\alpha_x - \sum_x \alpha_x|x\rangle \\
&= 2\sum_{y=0}^{N-1}|y\rangle\mu - \sum_x \alpha_x|x\rangle \\
&= \sum_{x=0}^{N-1}(2\mu - \alpha_x)|x\rangle
\end{aligned} \tag{3.42}
$$

Expression 3.42 means that each application of G adds roughly $\frac{2}{\sqrt{N}}$ 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[118]. 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. When multiple solutions k exist the algorithm's complexity can be restated as $O\left(\sqrt{\frac{N}{k}}\right)$. For more details concerning the algorithm please refer to the original work [82] as well as the texts [164] and [118].

Notice that superposition $|\psi\rangle$ is built with a polynomial amount of resources but generates a superposition with an exponential number of states. It should also be emphasized that the exponential parallelism that results from evaluating $|\psi\rangle$ does not automatically imply an exponential increase in performance. In [3] the author emphasizes the main reasons behind such behaviour. Namely, the nature of quantum computation requires that once the computation is finished a measurement needs to be performed on the superposition state, which causes a random collapse amongst the exponentially many states. As a result, one loses all the gains provided by the extreme parallelism. In order to gain a quantum advantage it is necessary to combine parallelism alongside another feature from quantum computation, respectively quantum interference. The interference mechanism allows for some states to cancel each other out. Accordingly, the main difficulty resides in determining

specific interference patters that allow for solution states to be obtained upon measuring state $|\psi\rangle$.

3.6 Conclusions

Algorithms based on the principles of quantum mechanics have allowed for speedups to be obtained relatively to their classical counterparts. In the context of space search, Grover's algorithm if of particular relevance since it provides a quadratic speedup relatively to classical brute force approaches.

Chapter 4

An n-puzzle quantum production system model

4.1 Introduction

Chapter 3 stated that changes occurring to a quantum state can be described by reversible circuitry capable of manipulating quantum information. In addition, it also established how to build basic reversible logic operations and how these could be mapped into unitary operators. From a computer science perspective it would be coherent and advisable to focus on developing an initial approach capable of tackling tree search, which is characteristic of production system behaviour, alongside the specific details surrounding the development of the corresponding unitary operator. The circuit model presents itself as a more logical approach to computation than Turing machines. Nevertheless, circuits represent a rigid behavioural model. Accordingly, in opting for a circuit approach some of the powerful computational abstractions provided by Turing machines are sacrificed.

In order to develop a quantum production system focus must first be given on how to perform the tree procedure illustrated in Figure 1.1, alongside the requirement to be able to "generate" multiple states by applying an action to a parent node. In quantum computation state evolution is expressed through mathematical operators known as unitary matrices. This kind of operators corresponds to bijective functions, *i.e.* one-to-one mappings. One possible approach is to take into account not only the state but also the set of actions involved in an N-level depth search. Also, in order to gain a quantum advantage the quantum superposition principle needs to be employed. Consequently, special care has to be taken in

order to maximize the probability of obtaining a solution and not just a random projection. Associated with each quantum state in a superposition exists an amplitude value. The probability of observing a state reflects the associated amplitude. Amplitude amplification processes such as the one described by Grover [82] can be employed to increase the probability of observing a solution state.

All of the above issues will be key features of the quantum production system proposition, which will be explored in the remainder of this chapter. Section 4.2 will present an in-depth analysis of the theoretical details surrounding this proposal for problem-specific quantum production system. The the key ideas of this model will be illustrated using the sliding block puzzle using the sliding block puzzle. Section 4.3 then describes how the developed model can be extended in order to accommodate the requirements of other problems. Section 4.4 presents the conclusions of this chapter.

4.2 Sliding block puzzle

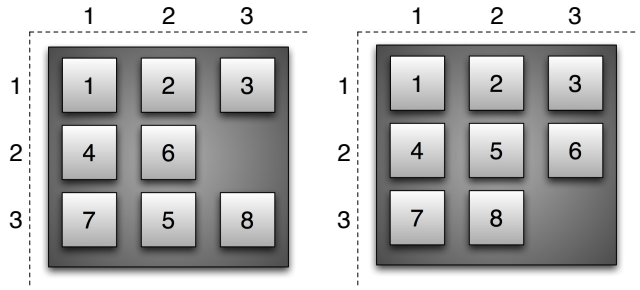
The sliding block puzzle is a familiar problem commonly approached in the artificial intelligence community, which conveniently showcases core notions. The next couple of sections are organised as follows. Section 4.2.1 will start by describing a classical production system for the sliding block 8-puzzle. Section 4.2.2 will then build on these results and develop a more quantum-suitable sliding block 3-puzzle. Since the work presented in this chapter intends to be a simple approach to a problem-specific quantum production the major components of the system will be presented on a gradual fashion. The general idea of such an approach is to provide some experience when dealing with reversibility and unitary operator construction. The knowledge secured in this chapter also serves as initial basis for a more formal definition of a general-purpose quantum production system.

4.2.1 Sliding block 8-puzzle

Many artificial intelligence applications involve composing a sequence of operations. The search space generated by an 8-puzzle sliding block puzzle is both complex enough to be interesting and small enough to be tractable. It also lends itself to solution using a production system [143]. A sliding block puzzle challenges a player to shift pieces around on a board without lifting them to establish a certain end-configuration. This non-lifting property makes finding moves, and the paths opened up by each move important parts of solving sliding block puzzles [105]. It is also important to mention that both the initial- and end-configuration might be chosen randomly, as illustrated by Figure 4.1. However, the end-configuration

typically reflects some sort of logical arrangement, as exemplified in Figure 4.1b. The final logical arrangement is specific to individual problem instances of sliding block puzzles. The set of possible elements for the 8-puzzle, $S_{8-puzzle}$, is presented in Expression 4.1.

$$S_{8-puzzle} = \{One, Two, Three, Four, Five, Six, Seven, Eight, Blank\} \tag{4.1}$$



(a) Initial board configuration (b) End board configuration

Figure 4.1: A sliding block puzzle example with a board of dimension 3×3.

Typically, each sliding block puzzle has a blank cell, which can be perceived to move on a set of possible directions. This way generality is gained by thinking of "moving the blank cell" rather than moving a numbered tile. In the case of the sliding block puzzle exemplified in Figure 4.1 only diagonal movements are not allowed. Accordingly, the set of possible movements for the blank cell consists of actions *Up*, *Down*, *Right* and *Left*, as illustrated by Expression 4.2. A solution to the problem space is an appropriate sequence of moves, such as "'move blank cell up, move blank cell left, ..., etc'"". Clearly, not all possible actions are applicable to all positions within the board. In fact, only position $(2, 2)$ is able to execute the full range of motions. If the blank cell is in any of the remaining positions of the board then only 2 to 3 moves can be executed.

$$\text{Possible Actions} = \{Up, Down, Left, Right\} \tag{4.2}$$

It is important to focus on a few points regarding the complexity of solving sliding block puzzles. More precisely, is there any way of determining if an end-configuration is obtainable from an initial configuration? If so, what is the minimum set of movements for achieving the desired state? Apparently, short of trial and error, it is impossible to present an answer to these questions [76]. This apparent inability to solve efficiently sliding block puzzles stems from computational complexity theory. Sliding-block puzzles have been shown to belong to

a class of problems known as PSPACE-complete [96] [95]. PSPACE consists of all decision problems that can be solved by employing a polynomial amount of space.

As previously stated, in order to solve a problem using a production system a set of items must be specified, namely a working memory, the productions set and the control strategy. Suppose that a working memory is configured with the board configuration depicted in Figure 4.1a and the state illustrated in Figure 4.1b is target board configuration. It is possible to define an illustrating set of production rules as presented in Table 4.1. The only remaining issue is due to the control strategy employed, which might be defined as [143]

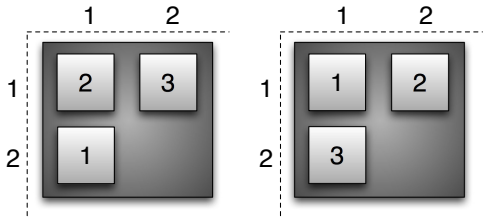
1. Try each production in order;
2. Do not allow loops;
3. Stop when goal state is reached.

Condition		Action
goal state in working memory	→	halt
blank is not on the top edge	→	move the blank up
blank is not on the right edge	→	move the blank right
blank is not on the bottom edge	→	move the blank down
blank is not on the left edge	→	move the blank left

Table 4.1: Production rules set for the 8-puzzle. (Source: [143])

4.2.2 Sliding block 3-puzzle

The development of the quantum production system will concentrate on a board with dimension 2×2, *i.e.* a 3-puzzle, merely for explanatory reasons. However, as will later be demonstrate, the model can be easily extended in order to accommodate any N-puzzle, with $N \geq 3$.



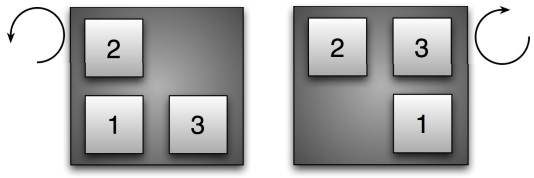
(a) Initial board configuration (b) Target board configuration

Figure 4.2: A 3-puzzle example.

Figure 4.2 depicts a 2×2 sliding block puzzle with an initial board configuration (Figure 4.2a)

and a target configuration (Figure 4.2b). The set of possible movements for the blank element remains the same as in the case of the 8-puzzle, respectively illustrated in Expression 4.2. However, for the 3-puzzle, at any given position only two movements are deemed possible to be executed. Since the blank cell always occupies a corner position, its movement can be perceived as performing a clockwise or counter-clockwise movement. This kind of binary movement is illustrated in Figure 4.3. The set of possible elements for the 3-puzzle, $S_{3-puzzle}$, is presented in Expression 4.3.

$$S_{3-puzzle} = \{One, Two, Three, Blank\} \tag{4.3}$$



(a) Counter-clockwise movement (b) Clockwise movement

Figure 4.3: Movement example for the blank cell given the board configuration depicted in Figure 4.2a.

The various components and overall behaviour of the approach will be presented in Section 4.2.2. Section 4.2.2 establishes the parallels between production systems and hierarchical search mechanisms. Section 4.2.2 discusses the quantum superposition principle in the context of the production system. The reversible circuit conversion is presented in Section 4.2.2.

Building the reversible circuit for the 3-puzzle

Since the overall purpose is to develop a reversible circuit representing the production system for the 3-puzzle, a proper binary representation for the board configuration is required. Each possible board configuration incorporates four elements, *i.e.* $|S_{3-puzzle}| = 4$. Logic dictates that a total of $\log_2 |S_{3-puzzle}| = 2$ bits are required in order to represent each element of $S_{3-puzzle}$.[1] Let Table 4.2 depict the bit encoding for each possible element.

[1] Another possible strategy would consist in encoding each of the $|S_{3-puzzle}|! = 4! = 24$ possible board configurations. This strategy would require $\lceil \log_2 24 \rceil = 5$ bits, allowing for three bits to be saved. However, such an encoding mechanism would also translate into a lack of clarity in what refers to determining, which element is at each position of the board. In this case an option was made favouring clarity instead of the aforementioned encoding.

Board configurations can now be perceived as a binary string of length 8 containing the encodings for each position of the board. This process is illustrated in Table 4.3. The binary representation for board configurations will be crucial to the development of the reversible circuit.

b_1	b_2	Element
0	0	Blank
0	1	One
1	0	Two
1	1	Three

Table 4.2: Binary encoding for elements of a 3-puzzle

Board Position	$(1,1)$		$(1,2)$		$(2,1)$		$(2,2)$	
Bits	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
Initial Board Configuration	1	0	1	1	0	1	0	0
Target Board Configuration	0	1	1	0	1	1	0	0

Table 4.3: Binary strings depicting the board configurations presented in Figure 4.2a and Figure 4.2b.

In order to proceed with the analysis a few design concepts must be taken into account. Conceptually, in order to develop a production system capable of tackling a sliding block puzzle the following abilities are required:

Requirement 1 Determine if a given board configuration is a target board configuration.

Requirement 2 Given a board configuration and a production rule determine the new board configuration generated by applying the production;

When dealing with reversible computation, it is helpful to first envisage the desired operational behaviour in terms of a classical gate. In doing so, useful insight is gained into the inputs and outputs of the reversible operator.

Focusing on the first requirement involves developing a gate capable of receiving as an argument a binary string depicting the state of the board to be tested. In classical computation, a single bit would be used for output, having value 1 if the board presented was the target board configuration and 0 otherwise. This computational process can be represented as function f illustrated in Expression 4.4.

$$f(\underbrace{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8}_{\text{board configuration}}) = \begin{cases} 1 & \text{if target board configuration} \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

Expression 2.15 requires that the inputs should also be part of the output, *i.e.* the board configuration 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. Accordingly, the reversible gate will have a total of 9 input and output bits, 8 of which are required for representing the board and 1 bit serving as control. This gate, labeled as the target board unitary operator, is illustrated in Figure 4.4 [2]. Table 4.4 showcases the gate's behaviour for a few board configurations, where $f(b)$ denotes $f(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8)$.

It is important to point out the fact that when the gate determines that a board is a target board configuration it effectively switches the control bit, as highlighted in Table 4.4. This is expected behaviour in reversible computation and a key part of some quantum algorithms such as the well known Shor's factorization algorithm [191] and Grover's search [82].

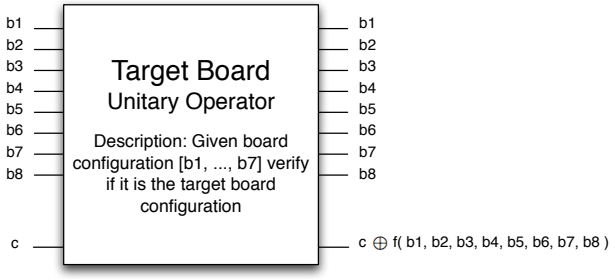


Figure 4.4: The target board unitary operator.

Inputs									Outputs								
b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	c	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	$c \oplus f(b)$
0	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	1	0	0	0	1	1	0	1	1	1
0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	0	1	0	0	0	1	1	1	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
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	0	1	1	0	1	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
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 4.4: A selected number of results from the truth table of the target board unitary operator.

From a mathematical point of view, how can the inner-workings of the reversible circuit illustrated in Figure 4.4 be expressed? The construction of simple unitary operators was described in Chapter 2. Accordingly, the set of column permutations needs to be specified. Let T denote the unitary operator responsible for implementing the behaviour of function

[2] This gate could be extended in order to receive as an input the desired target board configuration. This addition would be carried out at a cost of 8 additional input and output bits. However, since the main goal is design simplicity, the target board configuration will be "hard coded" into the gate. In doing so, generality is lost but a simpler design is gained.

f. T is a matrix with dimensions $2^9 \times 2^9$. From Table 4.4 it should be clear that only two input states map onto other states rather than themselves. Namely, $T|216\rangle \rightarrow |217\rangle$ and $T|217\rangle \rightarrow |216\rangle$. Accordingly, the 217^{th} column of T should permute to $|217\rangle$, and the 218^{th} column map to state $|216\rangle$. All other remaining states would continue to map onto themselves.

Focusing on the second requirement implies that the new gate should also output a new board configuration, *i.e.* the result. Also, in the context of a production system the move blank cell unitary operator should only be applied if and only if the inputted board configuration is not a target board configuration. Otherwise, the production system would systematically discard any potential solutions found. This process can be performed by including a reference to function f in the new function g's definition.

The only issue is due to how to output the new board configuration in a reversible manner. As was previously discussed in Chapter 2 given an irreversible function f, a reversible mapping can be constructed with the form illustrated in Expression 2.15, which illustrated how this process could be performed for a single result bit. In this specific case 8 result bits are required for representing the new board configuration. Developing a reversible version of the gate requires the addition of 8 control bits, that should be included as input. Expression 2.15 can be easily extended in order to accommodate any number of control bits, as illustrated by Expression 4.5 where x is an input register, c_i are control bits, and $f(x) = (y_1, y_2, \cdots, y_n)$.

$$(x, c_1, c_2, \cdots, c_n) \mapsto (x, c_1 \oplus y_1, c_2 \oplus y_2, \cdots, c_n \oplus y_n) \tag{4.5}$$

With Expression 4.5 in mind it is now possible to define a function g responsible for producing the new board configuration. Function g should receive as inputs the current board configuration and a bit m indicating whether the blank cell should perform a clockwise ($m = 1$) or counter-clockwise movement ($m = 0$). By systematically checking for the position of the blank cell and with the movement described in bit m it is possible to generate the new board.

Let $g : \{0,1\}^9 \rightarrow \{0,1\}^8$ with $g(b, m) = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$, where b denotes a valid board configuration[3] $(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8)$, and m the type of movement. The computational behaviour of g is presented in Expression 4.6.

[3] A valid board configuration means that a board configuration must be an unordered collection of size four composed of distinct elements taken from Table 4.2.

$$g(b,m) = \begin{cases} (b_5, b_6, b_3, b_4, b_1, b_2, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_1 = b_2 = 0 \text{ and } m = 0 \\ (b_3, b_4, b_1, b_2, b_5, b_6, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_1 = b_2 = 0 \text{ and } m = 1 \\ (b_3, b_4, b_1, b_2, b_5, b_6, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_3 = b_4 = 0 \text{ and } m = 0 \\ (b_1, b_2, b_7, b_8, b_5, b_6, b_3, b_4) & \text{if } f(b) = 0 \text{ and } b_3 = b_4 = 0 \text{ and } m = 1 \\ (b_1, b_2, b_3, b_4, b_7, b_8, b_5, b_6) & \text{if } f(b) = 0 \text{ and } b_5 = b_6 = 0 \text{ and } m = 0 \\ (b_5, b_6, b_3, b_4, b_1, b_2, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_5 = b_6 = 0 \text{ and } m = 1 \\ (b_1, b_2, b_7, b_8, b_5, b_6, b_3, b_4) & \text{if } f(b) = 0 \text{ and } b_7 = b_8 = 0 \text{ and } m = 0 \\ (b_1, b_2, b_3, b_4, b_7, b_8, b_5, b_6) & \text{if } f(b) = 0 \text{ and } b_7 = b_8 = 0 \text{ and } m = 1 \\ (b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) & \text{otherwise} \end{cases} \tag{4.6}$$

With function g defined it becomes relatively simple to develop the corresponding reversible gate. As before, special attention needs to be devoted in order to endow the gate with reversibility. Accordingly, the unitary operator has to incorporate the following characteristics:

- 8 input and output bits for the current board configuration;
- 1 input and output bit for describing the type of movement;
- 8 control and result bits in order to account for the new board configuration.

The reversible gate, respectively labelled as the move blank cell unitary operator M, is illustrated in Figure 4.5. M is a matrix of dimension $2^{8+1+8} \times 2^{8+1+8}$, which can be built in a similar way to T, *i.e.* from the corresponding truth table determine the mappings between states. These mappings can then be translated as columns permutations.

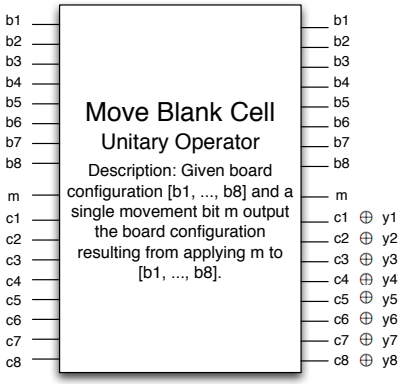


Figure 4.5: The move blank cell unitary operator.

With both the target board and the move blank cell gates designed it is now possible to continue with the development of the reversible production system. Recall that a production system has the following elements: working memory, set of production rules and a control strategy. The auxiliary bits employed by each of the previously defined gates can be perceived as the working memory of the system. Function f and g can be seen as enforcing the policies of a control and act cycle specific to the problem at hand. For now the set of production rules will be deliberately missing. Instead, focus will be given on certain features of the working memory and control strategy.

Generally speaking, for the production system to work it is fundamental to have the ability to verify if a target board configuration has been reached after applying a production rule. The move blank cell operator M already incorporates in its design a test for determining if the gate should be applied or not. Accordingly, it is only required to verify if the final board configuration corresponds to a target board configuration. This process is illustrated in Figure 4.6 [4] where res has the value presented in Expression 4.7.

$$res = c_9 \oplus f(c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7, c_8 \oplus y_8) \tag{4.7}$$

Notice that the concept of working memory is contemplated through the initial board configuration bits $b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$ and the result bits $c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7, c_8 \oplus y_8$. Also, the circuit design translates the control strategy (although a very primitive one), *i.e.* if it is possible move the blank cell and test if the new board is a target board. It is worthy to draw attention to the fact that Figure 4.6 illustrates the application of a single movement operator M alongside a target board operator T. Algebraically, this operation can be expressed as presented in Expression 4.8, where $I^{\otimes 9} = I \otimes I \otimes \cdots \otimes I$ repeated 9 times, since operator T should only take into consideration bits $c_1, c_2, \cdots, c_9$. The unitary operator presented in Expression 4.8 would act 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 board configuration bits $b = b_1, b_2, \cdots, b_8$, H_m is the Hilbert space spanned by the basis states employed to represent the set of productions, and H_c is the Hilbert space spanned by the auxiliary control bits.

$$(I^{\otimes 9} \otimes T)M|b_1, b_2, \cdots, b_8, m, c_1, c_2, \cdots, c_8, c_9\rangle \tag{4.8}$$

The above strategy can be extended in order to apply any number of movement operators,

[4] Alternatively, one could try to first employ the target board unitary operator and then redirect the output containing the result to a modified move blank cell gate. The modified version of the movement gate would employ that outcome in order to determine if the input board was in a target configuration or not. However, this would result in an unnecessary level of complexity to be added to the solution.

where the output of a movement gate is provided as input to another movement operator. In doing so, a guarantee is added that, if possible, another production rule is applied to the initial board configuration. This process is illustrated in Figure 4.7 where two movement gates, *i.e.* productions, are applied to an initial board configuration $b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$. Accordingly, two movement bits are required, namely m_1 and m_2. The former is fed as input to a first movement gate M_1, whilst the latter is provided as input to a second movement gate M_2. Consequently, res has the value presented in Expression 4.9. Applying M_2 requires that its application be "shifted" by a total of 9 positions, whilst operator T should be applied after input bit 18. The circuit behaviour can be described as presented in Expression 4.10.

$$res = c_{17} \oplus f(c_9 \oplus y_9, c_{10} \oplus y_{10}, c_{11} \oplus y_{11}, c_{12} \oplus y_{12}, c_{13} \oplus y_{13}, c_{14} \oplus y_{14}, c_{15} \oplus y_{15}, c_{16} \oplus y_{16}) \quad (4.9)$$

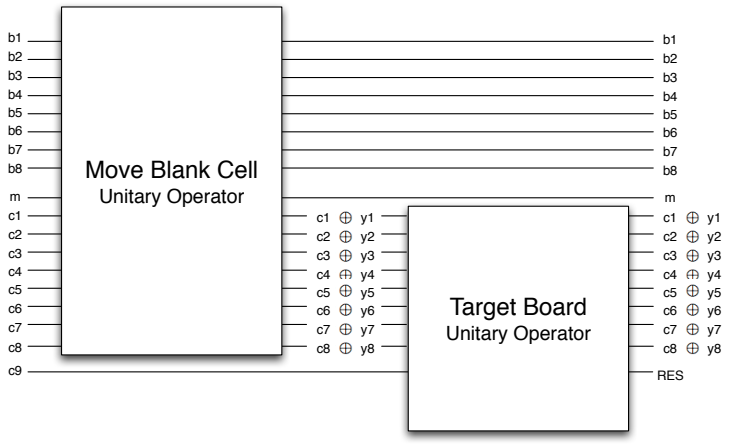


Figure 4.6: A reversible circuit incorporating the application of a single production rule for the 3-puzzle and a test condition in order to determine if the final board is a target configuration board.

$$(I^{\otimes 18} \otimes T)(I^{\otimes 9} \otimes M)M|b_1, b_2, \cdots, b_8, m_1, c_1, \cdots, c_8, m_2, c_9, \cdots, c_{16}, c_{17}\rangle \quad (4.10)$$

How can this result be expanded in order to accommodate any n-puzzle? Let E be the set of possible elements for an n-puzzle, then the number of bits required to encode each element is $e = \lceil \log_2 |E| \rceil$. This implies that the number of bits required to encode a board configuration is $b = |E|e$. Additionallly, let P be the set of possible productions for the same n-puzzle. Accordingly, $p = \lceil \log_2 |P| \rceil$ bits will be required for each production. Each

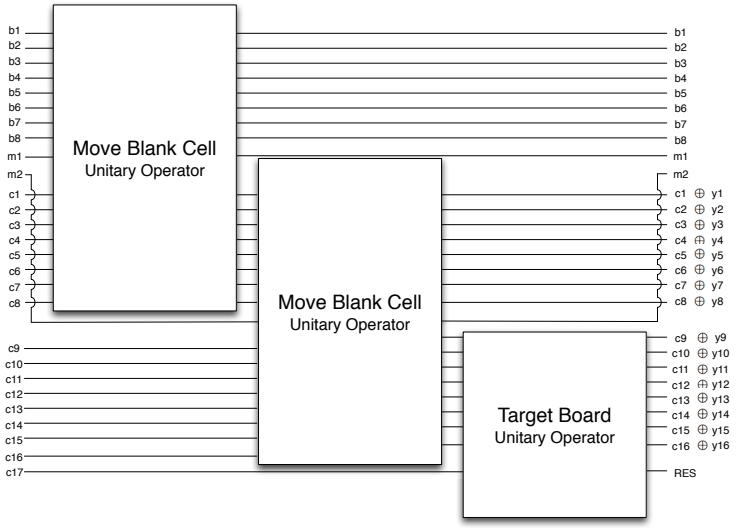


Figure 4.7: A reversible circuit illustrating the application of two move blank cell operators for the 3-puzzle and a target board gate in order to determine if the final board is a target configuration board.

movement operator M will require a total of $b + p + b = 2b + p$ input and output bits, and each target operator T will require a total of $b + 1$ input and output bits. How many bits will be required by the circuit? Suppose the M operators need to be applied a total of m times. The first operator M_1 requires $2b + p$ bits. Since a part of M_1 outputs will be provided as input to M_2 an additional $b+p$ bits will be added to the circuit. This means that $2b + p + (m - 1)(b + p)$ bits of the circuit are required just to move the blank element. Since operator T requires a single control bit this implies that the overall circuit employs a total of $n = 2b+p+(m-1)(b+p)+1$ bits. Of these n bits $c = n-(b+mp) = mb+1$ bits are control, or auxiliary, bits. These control bits can be perceived as the working memory of the production system. Furthermore, the sequence of bit indexes after which a movement operator M should be applied is $V = \{0, b+p, 2(b+p), 3(b+p), \cdots, (m-1)(b+p)\}$. Based on these statements it is possible to describe a general formulation for an n-puzzle circuit C employing adequate M and T operators. This process is presented in Expression 4.11. Unitary operator C would act on an input register $|x\rangle$ encompassing the initial board configuration, the set of productions 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. Also, the number of movement operators T grows linearly with the depth of the search, which is a key component of the input register.

$$C = (I^{\otimes m(b+p)} \otimes T) \prod_{k \in V} \left(I^{\otimes k} \otimes M\right) \tag{4.11}$$

3-Puzzle search approach

It is rare for an artificial intelligence application to have the ability to choose the best production rule out of the set of available productions. This is to due to a lack of information that is inherent to most control strategies [165]. In the case of the 3-puzzle the situation is even more delicate since at any given time both kinds of movements, clockwise and counter-clockwise, are possible. Consequently, it is not uncommon for control strategies to perform a systematic combination of available productions, until a sequence yielding a goal configuration of the working memory is discovered.

This process is remarkably familiar with classical hierarchical search strategies such as tree search. Tree search is utilized whenever decisions must be made that are based on complex knowledge, which cannot be implemented directly on a machine. The simplest, and by some measures, the most successful, applications have as their basis a "brute-force" search, in which an exhaustive examination of all possible sequences of actions are performed until goal states are reached [69]. The reversible circuit presented in Figure 4.6 applies a single movement gate and can thus be perceived as performing a depth-limited search of a single level. On the other hand, the two movement gates belonging to the circuit presented in Figure 4.7 enable it to perform a search of depth 2.

Quantum superpositions

In order to proceed with the analysis take into account the circuit presented in Figure 4.6. Conceptually, it is possible to differentiate between three inputs, respectively:

- the board configuration bits, $b_1, b_2, \cdots, b_8$, refered to as an 8-bit register $|b\rangle$.
- the movement bit m, which is basically a one bit register $|m\rangle$;
- the control bits, $c_1, c_2, \cdots, c_{17}$, refered to as a 17-bit register $|c\rangle$.

Let U_g refer to the unitary operator characterizing the reversible circuit presented in Figure 4.6. The behaviour of U_g can be described as illustrated by Expression 4.12.

$$U_g : |b\rangle|m\rangle|c\rangle \mapsto |b\rangle|m\rangle|c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7, c_8 \oplus y_8, res\rangle \tag{4.12}$$

Input register $|b\rangle|m\rangle|c\rangle$ can then be initialized to a specified value and by applying U_g it becomes possible to obtain the respective result. However, at first sight, the added overhead required for reversible computation of never destructing information does not seem to provide an improvement over a classical computation. Gaining a quantum advantage through Grover's algorithm requires: (1) building a uniform superposition state spanning a multitude of possible configurations and (2) applying an amplitude amplification process. With the assistance of a Hadamard gate, whose behaviour is presented in Expression 4.13, it is possible to build a uniform superposition of 2^n states, where n is the number of bits, represented by $|\psi\rangle$.

$$H^{\otimes n}|0\rangle = |\psi\rangle = \underbrace{\frac{1}{\sqrt{2^n}}}_{\text{amplitude}} \sum_{x=0}^{2^n-1} |x\rangle \tag{4.13}$$

State $|\psi\rangle$ will be used in conjunction with unitary operator U_g in order to evaluate a superposition containing all possible board configurations and productions. The control bits do not need to be placed in a superposition since they are only employed in order to assist the overall process. Accordingly, the control bit register is initialized to $|0\rangle^{\otimes c}$, where c is the number of control bits. Let $|\psi_b\rangle$ denote the superposition for the board configurations, $|\psi_m\rangle$ the superposition of the productions and $|\psi\rangle$ denote the combined superposition, which has the form presented in Expression 4.14.

$$\begin{aligned} |\psi\rangle &= |\psi_b\rangle|\psi_m\rangle|0\rangle^{\otimes c} \\ &= \frac{1}{\sqrt{2^8}} \sum_{x=0}^{2^8-1} |x\rangle \frac{1}{\sqrt{2^1}} \sum_{x=0}^{2^1-1} |x\rangle|0\rangle^{\otimes c} \\ &= \frac{1}{\sqrt{2^9}} \sum_{x=0}^{2^9-1} |x\rangle|0\rangle^{\otimes c} \end{aligned} \tag{4.14}$$

Where each $|x\rangle$ should be interpreted as a state of the combined input register $|b\rangle|m\rangle$. It is now possible to apply unitary operator U_g to the superposition register. In practice this process means that all values present in the superposition register are processed simultaneously. This operation is illustrated in Expression 4.15.

$$U_g|\psi\rangle = \frac{1}{\sqrt{2^9}} \sum_{x=0}^{2^9-1} U_g|x\rangle|0\rangle^{\otimes c} \tag{4.15}$$

However, due to the effects of a process known as quantum decoherence only one of the states conveying the answer can be obtained. This collapse from a multitude of states into a single one takes into account the probabilities associated with each state [99]. Accordingly, states with a higher probability are more likely to be obtained, whilst states with smaller probabilities are less likely to be obtained. This does not imply that only those states with higher probabilities will be obtained.

Oracle development

Ideally, in order to take advantage of Grover's algorithm the reversible circuit being developed up until this moment should mimic the behaviour illustrated in Expression 3.27 as opposed to the one presented in Expression 4.12. In order to carry on with such a mapping it is important to make a simple observation, namely that Expression 3.27 effectively means that all the inputs, excluding bit c, should also be part of the outputs.

Accordingly, the circuits presented in Figure 4.6 and Figure 4.7 should somehow undo their computation. As previously stated in Section 4.2.2, this operation can be performed by building a "mirror" circuit, 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 operations to the reversible circuit presented in Figure 4.6 is illustrated in Figure 4.8 whose unitary operator computes $U_g : |b\rangle|m\rangle|c\rangle \mapsto |b\rangle|m\rangle|c\rangle$. Logically, the overall operation results in the original state being obtained. Equivalently, it is possible to state this result in terms of the unitary operator C of Expression 4.11 as $C^{-1}C|x\rangle = |x\rangle$.

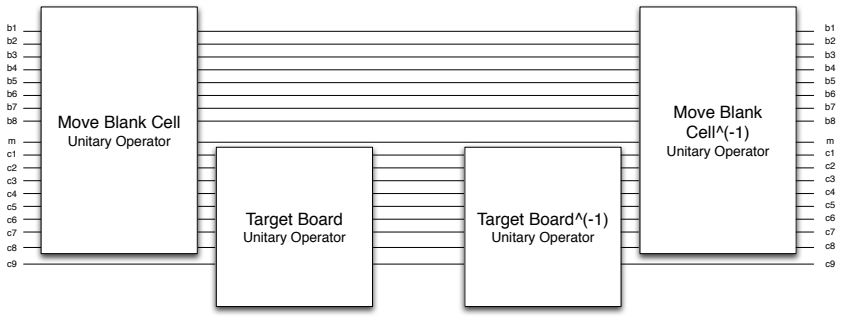


Figure 4.8: A reversible circuit showcasing the application of a single movement gate for the 3-puzzle, and then undoing the previously performed computations.

Consequently, an additional form of control has to be incorporated into the circuit design in order for the circuit's overall computation to be saved, respectively, the *res* value of

Expression 4.9. This operation can be performed with the introduction of another control bit alongside a controlled-NOT gate, denoted CNOT, which is a famous gate in quantum computation. The gate acts on two bits, labelled the control bit and the target bit. The control bit is always unaffected by the CNOT gate. The target bit is switched, *i.e.* applied the NOT operation, if the control bit is set to 1. Otherwise, if the control bit has value 0, the gate does nothing. The truth table for the CNOT gate is presented in Table 4.5.

Inputs		Outputs	
c	t	c	$c \oplus t$
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 4.5: Truth table for the CNOT gate.

The introduction of the CNOT gate allows the result to be saved in a reversible manner, which, as previously mentioned, is pre-requisite for describing operations in quantum computation. The combination of the reversible circuit presented in Figure 4.8 alongside the CNOT gate is shown in Figure 4.9. The circuits overall computation is presented in Expression 4.16 where res has the value shown in Expression 4.7. Let the input register $|b\rangle|m\rangle|c\rangle$ be referred to as $|x\rangle$ then Expression 4.16 is equivalent to Expression 3.27.

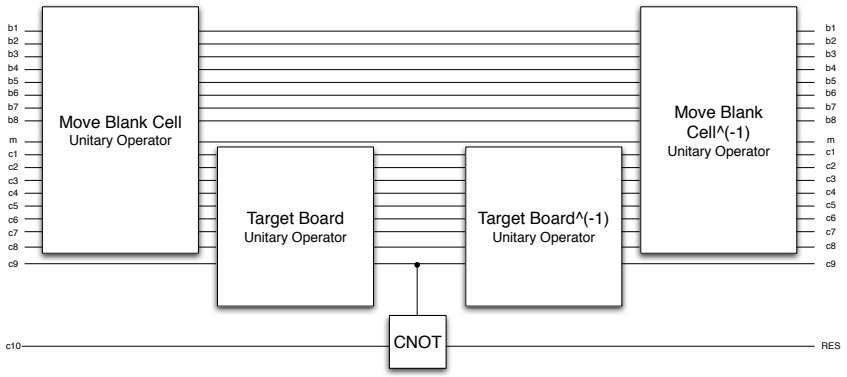


Figure 4.9: A reversible circuit showcasing the application of a single movement gate for the 3-puzzle whilst incorporating the principles of an oracle.

$$U_g : \underbrace{|b\rangle|m\rangle|c\rangle}_{\text{input}} \quad \underbrace{|c_{10}\rangle}_{\text{oracle's control bit}} \quad \mapsto |b\rangle|m\rangle|c\rangle|c_{10} \oplus res\rangle \tag{4.16}$$

Alternatively, it is possible to state this result in more general terms by employing unitary operator C, presented in Expression 4.11, as showcased by Expression 4.17.

$$O = C^{-1}(I^{\otimes 2b+p+(m-1)(b+p)}CNOT)C|b\rangle|m\rangle|c\rangle|c_{mb+2}\rangle \tag{4.17}$$

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 $\mathcal{H} = \mathcal{H}_b \otimes \mathcal{H}_m \otimes \mathcal{H}_c \otimes \mathcal{H}_{c_{mb+2}}$. The reversible circuit presented in Figure 4.9 alongside Grover's algorithm provide a quantum mechanism for performing a hierarchical search of depth level 1, *i.e.* for a given board configuration b and the set of all possible movements, verify if a target board configuration is reached by applying the respective production. This behaviour is equivalent in function to that of a classical production system.

4.3 Additional Reflections

The previously presented approach towards quantum hierarchical search focused on the theoretical details surrounding the 3-Puzzle. The reversible circuit proposed can be decomposed into two types of components: an operator responsible for recognizing which production should be applied to the working memory alongside with an operator capable of detecting target states. At each depth level of the search a new production operator is applied. Accordingly, the circuits overall width, *i.e.* the number of bits, will be mostly imposed by the production applying operator. Once all possible productions have been applied a test operator is applied in order to determine if a target state was reached. The specific details of both operators incorporate concepts such as a set of productions, a control-act strategy and a working memory.

The binary string employed encoded, excluding the control bits, the original board configuration alongside the set of productions to be applied. Afterwards, the reversible circuit was extended in order to perform a search of depth level N. The developed circuit used as an input argument a binary string with the form presented in Expression 4.18, where $m_1 m_2 \cdots m_N$ can be viewed as individual 1-bit registers.

$$\underbrace{b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8}_{\text{board configuration}} \ \underbrace{m_1 m_2 \cdots m_N}_{\text{productions}} \tag{4.18}$$

Recall that from a tree search perspective this process can be viewed as a depth-limited search where at each possible node two actions, *i.e.* productions, can be applied. Accordingly, an initial state is used as input, also known as the root, and for each action another node is reached, as illustrated in Figure 1.1. Each action can thus be viewed as leading to a

different subtree of the tree until a leaf node is reached. If after applying Grover's iterate the measurement outcome returns a solution, the state obtained represents a board configuration alongside the respective set of productions, *i.e.* path, yielding a target configuration. Classical search strategies require $O(b^d)$ time, where b is the branching factor and d the depth of a solution. A hierarchical search mapping to quantum computation employing Grover's algorithms allows this time to be reduced to $O(\sqrt{b^d})$, effectively cutting the depth factor in half.

This model can also be extended in order to accommodate non-binary trees. Suppose for a given problem instance, *e.g.* 16-puzzle, it is always possible to build a reversible circuit in oracle form capable of determining if a target configuration is reached after applying N productions. Assuming a constant branching factor b then $\lceil log_2 b \rceil$ bits are required. This means that an input register containing $m_1 m_2 \cdots m_N$ productions can be fed into an adequate reversible circuit and the corresponding unitary operator.

However, one delicate question remains: how many productions should be applied?, *i.e.* how deep should the search procedure go? It is not hard to see that this is a complex issue, which requires taking into account a series of factors. *e.g.* an initial configuration might yield a solution after applying one production, whilst others may require more. Theoretically, one can argue that it is possible to determine an upper bound (however large) after which all possible combinations have been exhausted. The reversible search circuit presented in Section 4.2 effectively stops applying productions as soon as a solution is found. Accordingly, the circuit could be extended in order to accommodate the upper-bound value. Yet, this is neither feasible nor practical, since the circuit would perform too many unnecessary computational steps. One should also mention the fact that it is possible to tweak the circuit in order to perform more useful computation according to specific needs. The reversible circuit presented receives a board configuration alongside a set of productions as input and tests if a target state is reached, namely it might be interesting to develop a gate that only receives a set of productions as input and has the initial desired board configuration is "hard coded" into the operational behaviour of the circuit.

4.4 Conclusions

In this chapter a possible model for a quantum production system for the n-puzzle was presented. 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 production system. By combining the concepts of these elements it is possible to search through the possible combinations of an n-puzzle quadratically faster than its classical counterparts. Also, this

proposition placed a strong emphasis on determining the set of productions leading up to a target node.

Chapter 5

Model Performance Analysis

5.1 Introduction

Chapter 4 described an initial approach towards a problem-specific production system. In this chapter an analysis is presented on how the concepts discussed for a Grover-based production system are influenced by traditional search concepts. The concepts supporting production system theory will be formalized from a quantum perspective through Grover's algorithm in Chapter 6.

For instance, what are the ramifications of a variable such as the branching factor? Would the system require the use of a constant branching factor? Clearly, this is not always the case for the complete set of problems that can potentially be addressed by search algorithms. When considering a non-constant branching factor, what would be the associated impacts in overall system performance? Additionally, traditional search strategies typically employ some kind of information to determine which states are more promising than others when trying to reach a goal state. Can a heuristic bounded procedure be incorporated into such an approach? If so, is there any significant advantage to be gained?

These and other questions will be addressed in the remaining sections of this chapter, namely: Section 5.2 focuses on assessing how such a system would perform from a branching factor perspective; Section 5.3 analyses the impacts of incorporating heuristic concepts into the quantum tree search method; Section 5.4 presents and discusses the parallels, alongside the differences, between the proposal for a quantum production system and another well-known kind of graph inspection tool, respectively the quantum random walk; Section 5.5 presents the conclusions of this chapter.

5.2 Branching factor ramifications

In theoretical computer science one possible way to measure a problem's complexity consists in assessing how long a given algorithm takes to find a solution. However, time performance is dependent on a multitude of hardware related factors. Accordingly, it is often more suitable to take appropriate steps to determine the total number of items that are to be evaluated. In the case of a classical tree search this equates to the number of nodes to take into account. From a classical tree search perspective complexity is expressed in terms of b, the branching factor or maximum number of successors of any node; d, the depth of the shallowest goal node; m, the maximum length of any path in the state space [183]. Section 5.2.1 focuses on the aspects surrounding a constant branching factor. The requirements for a non-constant branching factor are presented in Section 5.2.2. The impact of using a non-constant branching factor is presented in Section 5.2.3.

5.2.1 Constant branching factor

As previously stated, the proposal for quantum tree search system only relied on a constant branching factor b. This was mostly due to simplification reasons. However, a constant branching factor requirement is not feasible when considering potential applications of search algorithms. Since, at its essence, this system can be perceived as evaluating a superposition of all possible paths up to a depth level d, it is pivotal to determine the impact of a non-constant branching factor in this approach. Before proceeding, it is important to examine a couple of examples in order to have a clear understanding of the search process.

Figure 1.1 illustrates a search tree where at any given node it is always possible to apply two actions, which will be respectively refer to as a_0 and a_1. These actions can be encoded in a binary fashion. In order to do so, it is necessary to determine how many bits n are required. For an even number of actions calculating the base-2 logarithm suffices. However, an odd number of actions might be required. In this case the base-2 logarithm needs to be map into the next largest integer, which can be done through the ceiling function, *i.e.* $n = \lceil log_2|A| \rceil$ bits, where $|A|$ denotes the cardinality of the action set. The complete range of values allowed with n bits might not be used, *i.e.* $|A| < 2^n$. It is the unitary operator's responsibility to validate whether a binary string is an admissible action. In the case of the search tree illustrated in Figure 1.1 which possesses a branching factor $b = 2$ actions only a single bit is required. Accordingly, let value 0 denote a_0 and value 1 represent a_1. The binary strings encoding the paths leading to each leaf nodes of the search tree illustrated in Figure 1.1 are presented in Table 5.1.

Path to node	1^{st} Action	2^{nd} Action	3^{rd} Action
H	0	0	0
I	0	0	1
J	0	1	0
K	0	1	1
L	1	0	0
M	1	0	1
N	1	1	0
O	1	1	1

Table 5.1: Binary encoding for each possible path of the search tree illustrated in Figure 1.1

Suppose that a search up to depth level d is to be performed. Then it is possible to: (1) build a string of d elements, one for each possible depth, with each element requiring a binary representation using n bits. In total, the binary string will employ $n \times d$ bits; and (2) construct a quantum superposition $|\psi\rangle$ encompassing all the actions to be applied up to depth level d as illustrated by Expression 5.1. This superposition $|\psi\rangle$ can then be employed alongside an oracle operator and Grover's algorithm.

$$|\psi\rangle = \frac{1}{\sqrt{2^{n\times d}}} \sum_{x=0}^{2^{n\times d}-1} |x\rangle \tag{5.1}$$

5.2.2 Non-constant branching factor

Suppose that the tree presented in Figure 5.1 with an action set $A = \{a_0, a_1, a_2, a_3, a_4\}$ is being evaluated.

First, there no longer exists a constant branching factor at each node. In fact for this particular case it is convenient to distinguish between two types of branching factor, namely, the theoretical maximum branching factor $b_{max} = |A| = 5$, and the average branching factor of each node $b_{avg} = 2$, not including the leafs. In order to encode each of the possible actions $n = \lceil \log_2 |A| \rceil = 3$ bits are required. Let the encodings of each action be those presented in Table 5.2. The binary strings leading to each leaf node are illustrated in Table 5.3.

In order to employ Grover's algorithm the quantum superposition state $|\psi'\rangle$ would need to be similar to that presented in Expression 5.1. Again, state $|\psi\rangle$ would consist of those states with d elements, each of which with n bits. However, there is a crucial difference between both tree searches. Take into account the case illustrated in Figure 5.1 where a search to depth level $d = 3$ alongside $b_{max} = 5$ is performed. Superposition $|\psi'\rangle$ would contain all the quantum states belonging to the range $[0, 2^{d\times n} - 1] = [0, 2^9 - 1] = [0, 511]$, *i.e.* a total 512

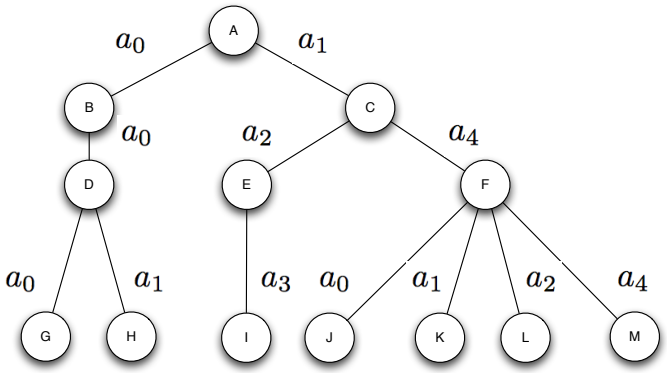


Figure 5.1: A search tree with a maximum branching factor $b_{max} = 5$ and an average branching factor $b_{avg} = \frac{2+1+2+2+1+4}{6} = 2$.

action	b_0	b_1	b_2
a_0	0	0	0
a_1	0	0	1
a_2	0	1	0
a_3	0	1	1
a_4	1	0	0
undefined	1	0	1
undefined	1	1	0
undefined	1	1	1

Table 5.2: Binary encoding for each possible path of the search tree illustrated in Figure 5.1

possible paths would be able to be encoded when in reality only those states presented in Table 5.3 would need to be encoded[1]. In reality, the vast majority of the states present in the superposition would contain inadmissible configurations of actions.

It is important to draw attention to the fact that Grover's algorithm provides a quadratic speedup $O(\sqrt{N})$ where N is the number of elements present in the superposition. By employing $n = \lceil \log_2 b_{max} \rceil$ bits, when in practice $n = \lceil \log_2 b_{avg} \rceil$ bits might have sufficed, the search space is being unnecessarily extended thus resulting in a loss of some of the speedup provided by Grover's algorithm. Naturally, the question arises: Considering the above encoding mechanism, and a $b_{avg} < b_{max}$ when does the quantum production system applied to tree search stop providing a speedup over classical approaches?

[1] An alternative approach would consist in encoding each possible state, instead of encoding in an binary fashion the sequence of actions. However, this would have a meaningful impact on the complexity and design of unitary operator U since each admissible input string would have to be mapped onto a predefined sequence of actions.

Path to node	1^{st} Action	2^{nd} Action	3^{rd} Action
G	000	000	000
H	000	000	001
I	001	010	011
J	001	100	000
K	001	100	001
L	001	100	010
M	001	100	100

Table 5.3: Binary encoding for each possible path of the search tree illustrated in Figure 5.1

5.2.3 Analysis

How should one proceed in order to analyze the problem depicted in the previous section? As with so many other fields it is usually easier to start out with an example and extrapolate from that. Accordingly, envisage the following scenario: a tree search is to be performed using the quantum search system up to depth level 10; the maximum branching factor, b_{max}, is 5, however on average only three actions can be performed, *i.e.* $b_{avg} = 3$. Does this approach still provides an advantage over classical search strategies?

In order to answer this question the complexities of classical search algorithms need to be considered. Traditionally, these methods experience some type of exponential growth in the number of leaf nodes that need to be assessed. Since the number of elements to be evaluated is a function of the branching factor b and the depth of the search d the associated complexity is typically of the form $O(b^d)$. Clearly, in the case of the scenario envisaged it necessary to differentiate between the two branching factors, respectively b_{max} and b_{avg}. Accordingly, if b_{max} is considered then a total of $O(b_{max}^d) = O(5^{10}) = 9765625$ nodes might potentially need to be evaluated. On the other hand, by employing b_{avg} a total of $O(b_{avg}^d) = O(3^{10}) = 59049$ nodes may be considered. These values differ by a factor of ≈ 165, which is considerable when in practice it is acceptable to perceive b_{avg} as the *de facto* branching factor.

How many times would one need to apply Grover's iterate? In this case there is no distinction to be made between b_{max} and b_{avg} since a quantum superposition state encoding all of the b_{max} actions up to a depth level d would still need to built. As previously stated this would result in a total of $n \times d = 3 \times 10 = 30$ bits being required, with $n = \lceil \log_2 b_{max} \rceil = \lceil \log_2 5 \rceil = 3$. Accordingly, for the above scenario Grover's iterate would need to be applied a total of $O(\sqrt{N}) = O(\sqrt{2^{30}}) = 32768$ times in order to obtain a solution. By comparing the number of states classically evaluated by using b_{avg} against the total number of iterations required by Grover's algorithm it is possible to see that both values differ by a factor of ≈ 1.8. Although a speedup is still obtainable over the classical approach it is severely lessened.

Intuitively, this behaviour can be perceived in the following manner:

- As b_{avg} grows closer to b_{max} the number of times to apply Grover's iterate will be optimal relatively to the classical approaches;
- As b_{avg} grows more distant to b_{max} the number of Grover iterations to apply will be closer to b_{avg}^{d}.

From the above reasoning it is a simple task to determine the number of times that Grover's iterate should be applied, respectively $|G|$, as illustrated by Expression 5.2.

$$\begin{aligned} |G| &= \sqrt{N} \\ &= \sqrt{2^{n \times d}} \\ &= \sqrt{2^{\lceil log_2 b_{max} \rceil \times d}} \\ &= 2^{\frac{\lceil log_2 b_{max} \rceil \times d}{2}} \end{aligned} \tag{5.2}$$

Determining where the threshold lies between the number of elements of a classical search, respectively b_{avg}^{d} and the total number of times to apply Grover's iterate $|G|$ can be formulated as presented in Expression 5.3, which when solved results in Expression 5.4.

$$b_{avg}^{d} = |G| \tag{5.3}$$

$$\Leftrightarrow b_{avg}^{d} = 2^{\frac{\lceil log_2 b_{max} \rceil}{2}^{d}}$$

$$\Leftrightarrow b_{avg} = 2^{\frac{\lceil log_2 b_{max} \rceil}{2}} \tag{5.4}$$

Accordingly, when $b_{avg} < 2^{\frac{\lceil log_2 b_{max} \rceil}{2}}$ then the total number of nodes evaluated in classical search will be less than the number of times to apply Grover's iterate, *i.e.* $b_{avg}^{d} < |G|$. Appropriately, when $b_{avg} > 2^{\frac{\lceil log_2 b_{max} \rceil}{2}}$ then the system will yield a speedup over classical search algorithms. The plot of Expression 5.4, for $b_{max} \in [2, 128]$, is presented in Figure 5.2. The shaded area indicates those values of b_{avg} that will produce better performance results classically over the quantum production system. For this specific case, the analysis focused in the worst-case scenario where a single solution exists. However, when the search space contains multiple solutions k, Grover's complexity can be restated as $O(\sqrt{N/k})$. This function grows slower than the original upper-bound limit. As a consequence, the number of required iterations would be lower, resulting in a decrease of the shaded area where classical search would prove more advantageous.

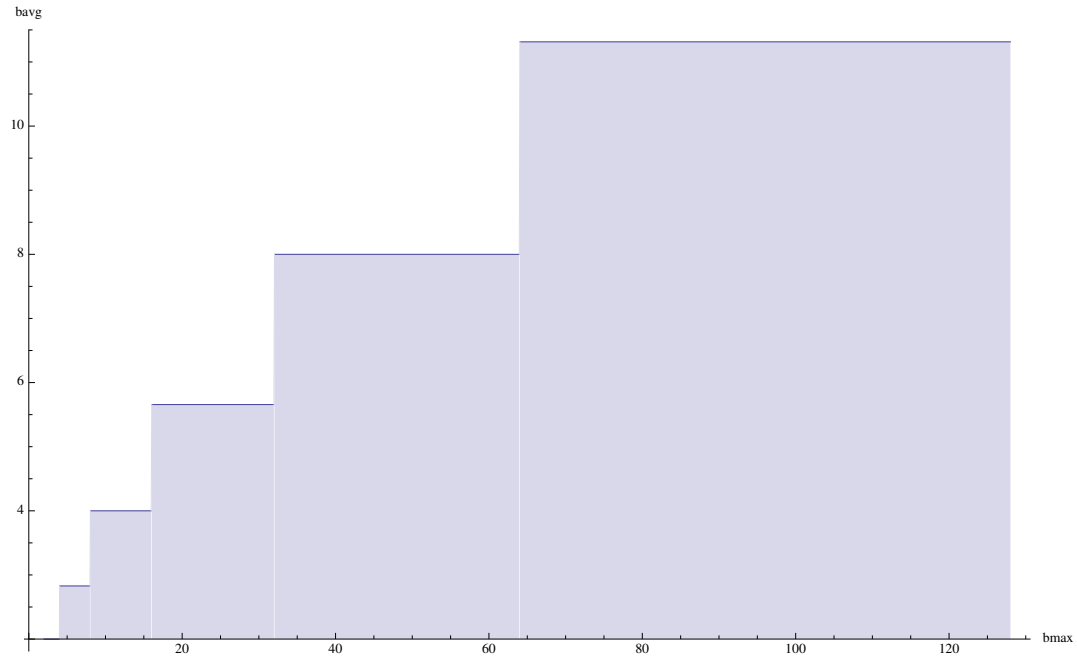


Figure 5.2: The area plot of $b_{avg} \leq 2^{\frac{\lceil log_2 b_{max} \rceil}{2}}$ for $b_{max} \in [2, 128]$. The shaded area indicates those values of b_{avg} that will produce better performance results over the quantum production system.

Figure 5.2 also showcases the characteristic ladder effect of functions employing the ceiling function required by the branching factor binary coding mechanism. In practice this means that there will always be ranges of values for b_{max} where the number of iterations, $|G|$, will remain the same. Consequently, the average branching factor, b_{avg}, for the boundary condition presented in Expression 5.3 will also remain constant in those intervals. This happens despite the fact that b_{max} is growing in the associated range.

It is also important to examine in more detail each b_{max} and the associated range of b_{avg} values. In order to encode in a binary fashion any b_{max} value a total of $n = \lceil log_2 b_{max} \rceil$ bits are required. The use of the ceiling function effectively forces certain ranges of b_{max} values to require the same number of n bits. As was previously seen, the number of Grover iterations is a function of the total number of bits required. It is easy to see how this influences b_{avg} by taking Expression 5.4 into account. Clearly, for those b_{max} values requiring the same number of bits, the associated b_{avg} values will remain the same. It is only when the required number of bits for b_{max} changes that an impact will be felt regarding b_{avg}. Accordingly, it is interesting to try to determine what happens when a transition is performed from, for example, n to $n+1$ bits. In this case the b_{avg} value grows from $2^{\frac{n}{2}}$ to $2^{\frac{n+1}{2}}$, which differ by a

factor of $\sqrt{2}$. Let b_{avg}^{n} denote the average branching factor b_{avg} associated with an interval requiring n bits. Then, it is possible to express b_{avg}^{n+1} as a function of b_{avg}^{n}. This recurrence behaviour is illustrated in Expression 5.5.

$$b_{avg}^{n+1} = \sqrt{2}\, b_{avg}^{n} \tag{5.5}$$

Expression 5.5 can be improved if the use of the ceiling function employed in Expression 5.4 is dropped.

In doing so, the ladder effect presented in Figure 5.2 is deliberately being lost. This new formulation is presented in Expression 5.6.

$$b_{avg} = 2^{\frac{log_2 b_{max}}{2}} = \sqrt{b_{max}} \tag{5.6}$$

As a result, a direct mapping of Expression 5.5 is impossible. Instead, it is possible to define a function, b'_{avg}, depicting the new upper-range values of b_{avg}, as illustrated in Expression 5.7. This $\sqrt{2}$-constant growth factor is depicted in Figure 5.3.

$$b'_{avg} = \sqrt{2}\, b_{avg} = \sqrt{2 b_{max}} \tag{5.7}$$

5.3 Heuristic Perspective

Determining which action should be applied is an important part of the overall search process. In a great deal of occasions an artificial intelligence application does not possess the adequate level of knowledge allowing for full differentiation amongst the set of possible actions [165]. At any given point in time during the search process it would be useful to somehow know which action might produce a state that is closer to a goal state. Intuitively, this process may be described as trying to determine the quality of a path of actions with an optimal solution having the lowest path cost among all solutions [183]. A key component of these systems, and many other algorithms in artificial intelligence, consists of a heuristic function $h(n)$ responsible for presenting an estimate of the distance that a given state n is relatively to a goal state. Function $h(n)$ is typically employed alongside a function $g(n)$, which reflects the search cost incurred to reach state n. Traditionally, the conjunction of both these functions is incorporated within a single evaluation function $f(n)$ as illustrated by Expression 5.8.

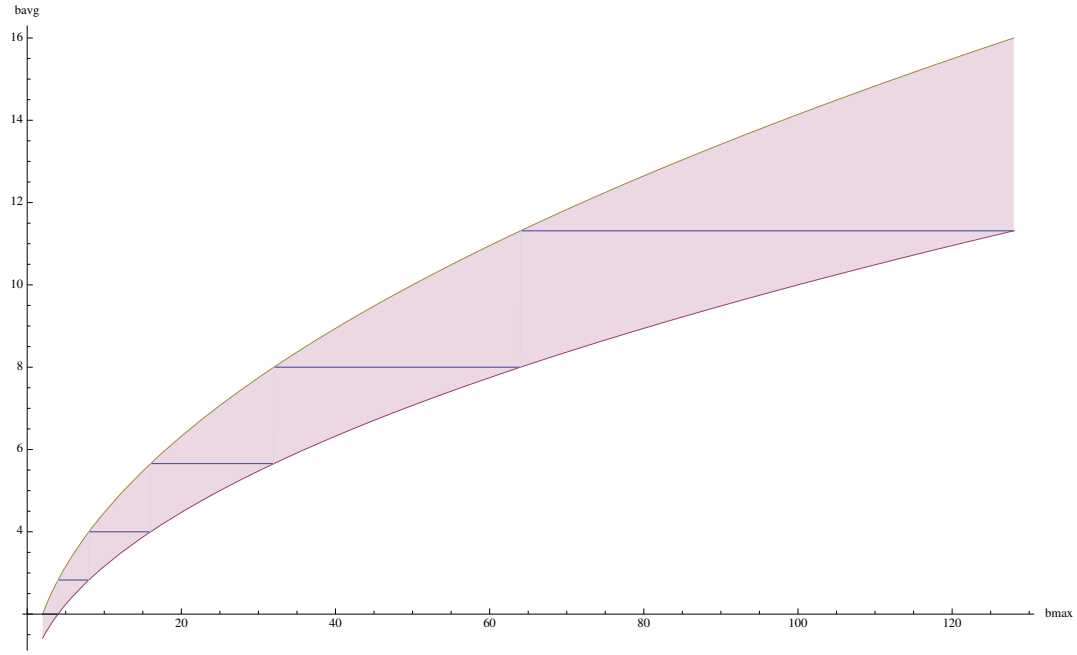


Figure 5.3: The $\sqrt{2}$-constant growth between b'_{avg} and b_{avg} superimposed on the original $b_{avg} = 2^{\frac{\lceil log_2 b_{max} \rceil}{2}}$ function for $b_{max} \in [2, 128]$.

$$f(n) = g(n) + h(n) \tag{5.8}$$

Function $h(n)$ can have a number of different definitions depending on the specifics of the problem at hand. In a sense, heuristic functions are responsible for providing extra-information about how-well a search is performing. However, search procedures need not depend on the use of heuristics. Systems which do not take into consideration any kind of auxiliary information are referred to as uninformed strategies. In the case of "uninformed" search systems the choice of which action to apply is performed at random. On the other hand, "informed" strategies enable search algorithms to select an appropriate rule.

The quantum production system model can be perceived as systematically trying to apply actions in an uninformed fashion. This operation is performed until a goal state is reached. Such behaviour resembles that of a standard blind depth-limited search process. Intuitively, the heuristic concepts incorporated into informed search strategies appear to be an adequate extension idea to the proposal. However, it remains to be seen if these concepts can be adequately mapped into the approach and if doing so provides an advantage. The remainder of this section is organised as follows: Section 5.3.1) considers how to incorporate the

use of a heuristic the unitary operator; Section 5.3.2) provides an analysis of the quantum computation procedure incorporating the use of a heuristic. Section 5.3.3) builds on these results to present an extended quantum heuristic mechanism.

5.3.1 The Quantum Heuristic

As previously mentioned, at its core, the system can be perceived as evaluating a superposition of all possible paths. Expression 5.1 illustrated this perspective, where a quantum superposition state is composed by an amplitude value and a multitude of sub-states, *i.e.* the computational basis. From a quantum mechanics perspective it is important to reinforce the idea that the mechanisms allowing quantum states of a superposition to communicate with the other states are complex and restricted in what they achieve. Those algorithms that are able to provide a meaningful speedup do so by employing, and determining, some type of global property *e.g.* Grover's algorithm is able to determine the mean amplitude A of a quantum superposition and perform an inversion about the mean [82]. Also Shor's algorithm for factoring numbers is able to efficiently determine the period of periodic quantum states [191]. This fact immediately poses a problem: traditionally, the use of heuristics is employed to choose among possible tree paths, ideally producing an optimal sequence of actions. However, with the system in question it is not possible to quantum mechanically perform such a comparison. Ergo, is there any way to incorporate any heuristic concepts into the system? In [30] the authors argue that many classical problems can be executed in $o(\sqrt{N})$ time by employing heuristic functions, which would therefore make amplitude amplification schemes less useful. The authors are able to show that these heuristics can be incorporated into the amplitude amplification process. However, the heuristic function employed represents a probabilistic algorithm, running in polynomial time, that outputs a solution state with some non-negligible probability.

This contrasts with the non-probabilistic route that will be followed in this chapter. Namely from a simplified point of view, a heuristic function outputs a value estimating the distance to a goal. Optionally, it is possible to take into account only those states whose f value is below a certain threshold T. However, deliberately ignoring these same states may later on result in a loss of optimal solution paths. Incorporating the heuristic function into the search process requires that the adequate modification are performed on the corresponding unitary operator. Grover's algorithm requires that the unitary operator U employed has an oracle form where the amplitudes of the solution states are flipped. Expression 5.9 reflects both of the previous requirements, where $|n\rangle$ represents the current node being processed and a_i an action taken at depth i and f stands for the function evaluation of the arguments, *i.e.* $f(n, a_1, a_2, \cdots, a_d)$.

$$U|n\rangle|a_1a_2\cdots a_d\rangle = \begin{cases} - & |n\rangle|a_1a_2\cdots a_d\rangle \quad \text{if} f \leq T \\ & |n\rangle|a_1a_2\cdots a_d\rangle \quad \text{otherwise} \end{cases} \tag{5.9}$$

In a similar way as to deciding which of two possible heuristics to employ, the matter of deciding the threshold value T could be left to statistical studies, or even informed intuition based on hands-on experience [165]. However, in no way did this circumvent the problem of comparing multiple paths. In conclusion, it is possible with such a method to incorporate some of the concepts surrounding the use of heuristics, but not all of them.

5.3.2 Interpretation

What happens when a unitary operator U is employed with the form presented in Expression 5.9 with Grover's algorithm? *I.e.* is there anything to be gained by incorporating heuristic concepts into the system? The following sections tackle this question, namely: the first section illustrates how superposition $|\psi\rangle$ can be decomposed into two components; the second section provides a graphical illustration of the impacts of performing search up to different depth levels.

Decomposing the superposition state

Answering this question requires extending the analysis of what happens when unitary operator U is applied to superposition $|\psi\rangle$, *i.e.* $U|\psi\rangle$. Suppose superposition $|\psi\rangle$ takes the form illustrated in Expression 5.10, where n represents the number of bits.

$$|\psi\rangle = \sqrt{\frac{1}{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \tag{5.10}$$

Before advancing any further, it is important to emphasize that in an uniform superposition, associated with each state $|x\rangle$ there is an amplitude $\alpha \in \mathbb{C}$, which in this case is $\sqrt{\frac{1}{2^n}}$ for all states. Let α_i denote the amplitude associated with state $|i\rangle$. Quantum computation requires the norm of amplitudes to be unit-length, *i.e.* $\sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1$, at all times.

Note that the initial quantum superposition $|\psi\rangle$ can be decomposed into two parts [118]. One part will contain all those states that are solutions, which will respectively be labeled as set X_{good}. Another part will include the non-solutions states, respectively labeled as set X_{bad}. Also, assume that k solutions exist. This implies that $|X_{good}| = k$ and $|X_{bad}| = 2^n - k$. Accordingly, an uniform superposition of the states belonging to X_{good} would set an equal

amplitude among its k elements, *i.e.* $\sqrt{\frac{1}{k}}$. By the same reasoning, an uniform superposition of the states in X_{bad} would impose an amplitude values of $\sqrt{\frac{1}{2^n-k}}$. Accordingly, it is possible to define the superposition states $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$, respectively presented in Expression 5.11 and Expression 5.12.

$$|\psi_{good}\rangle = \sqrt{\frac{1}{k}} \sum_{x \in X_{good}} |x\rangle \tag{5.11}$$

$$|\psi_{bad}\rangle = \sqrt{\frac{1}{2^n - k}} \sum_{x \in X_{bad}} |x\rangle \tag{5.12}$$

Superposition $|\psi\rangle$ can now be expressed in terms of the subspaces $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$. This process is illustrated in Expression 5.13, which is indispensable to the rest of the analysis.

$$|\psi\rangle = \sqrt{\frac{k}{2^n}} |\psi_{good}\rangle + \sqrt{\frac{2^n - k}{2^n}} |\psi_{bad}\rangle \tag{5.13}$$

The requirement that the norm must be preserved at all times induces a probabilistic behaviour. More specifically, state $|\psi\rangle$ whose norm is $\left|\sqrt{\frac{k}{2^n}}\right|^2 + \left|\sqrt{\frac{2^n-k}{2^n}}\right|^2 = 1$, *i.e.* a sum of values which sum up to 1 similarly to a probability distribution. Accordingly, the probability of obtaining state $|\psi_{good}\rangle = \left|\sqrt{\frac{k}{2^n}}\right|^2$ and state $|\psi_{bad}\rangle = \left|\sqrt{\frac{2^n-k}{2^n}}\right|^2$. Applying Grover's iterate effectively changes the amplitudes, maximizing the probability of obtaining a solution contained in the subspace spawned by $|\psi_{good}\rangle$. As a result, the amplitude associated with state $|\psi_{good}\rangle$ will increase. Since the norm of state $|\psi\rangle$ must be preserved, employing Grover also implies that the amplitude of $|\psi_{bad}\rangle$ will be decreased.

Heuristic Impact

In order to continue the analysis of such a heuristic method consider the ideal scenario where an admissible heuristic is possessed which always eliminates candidate states with each additional level of depth. Although this is a rather optimistic strategy the heuristic's behaviour can be viewed as ideal. This best case scenario to demonstrate the system's potential performance. Given a sufficiently high depth level d the heuristic will have to eventually produce the exact number of solutions k. Let k_d denote the number of solutions at depth level d. Then, with such a heuristic would result in $k_1 < k_2 < \cdots < k_d \leq k$.

However, fewer than k solutions can never expected to be produced, since that would violate basic assumptions about the search space of the problem.

The above process can be visualized geometrically. The initial superposition $|\psi\rangle$ containing k solutions can be viewed as vector with two components $(|\psi_{good}\rangle, |\psi_{bad}\rangle) = (\sqrt{\frac{k}{2^n}}, \sqrt{\frac{2^n-k}{2^n}})$. Therefore, state $|\psi\rangle$ can be mapped it into a two dimensional plane with axis $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$. This mapping process is illustrated in Figure 5.4. From a quantum computation perspective, this process can be best understood with the use of different states $|\psi_{k_d}\rangle$ reflecting a search up to depth level d. State $|\psi_{k_d}\rangle$ is a simple reformulation in terms of k_d of state $|\psi\rangle$, as presented in Expression 5.14.

$$|\psi_{k_d}\rangle = \sqrt{\frac{k_d}{2^n}}|\psi_{good}\rangle + \sqrt{\frac{2^n - k_d}{2^n}}|\psi_{bad}\rangle \tag{5.14}$$

Clearly, as the number of solutions k_d approaches k, the $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$ components will tend towards the values of the original state vector $|\psi\rangle$. This behaviour is illustrated in Expression 5.15.

$$\lim_{k_d \to k} \left(\sqrt{\frac{k_d}{2^n}}, \sqrt{\frac{2^n - k_d}{2^n}} \right) = \left(\sqrt{\frac{k}{2^n}}, \sqrt{\frac{2^n - k}{2^n}} \right) \tag{5.15}$$

As the depth of the search increases the corresponding state vector $|\psi_{k_d}\rangle$ gets closer to the original $|\psi\rangle$, as can be perceived from Figure 5.4. Additionally, each state $|\psi_{k_d}\rangle$ can be understood as forming an angle θ whose tangent is shown in Expression 5.16.

$$\tan\theta_{k_d} = \frac{\sqrt{\frac{k_d}{2^n}}}{\sqrt{\frac{2^n-k_d}{2^n}}} = \sqrt{\frac{k_d}{2^n - k_d}} \tag{5.16}$$

Using this approach it is possible to perform a comparison between the angles of different search depth levels. If a comparison of how the search advanced between depth levels d_1 and d_2 is to be performed, then it possible to define an operator $\Delta\theta_{k_{d_1}, k_{d_2}}$ with the behaviour defined in Expression 5.17.

$$\Delta\theta_{k_{d_1}, k_{d_2}} := \underbrace{\arctan \sqrt{\frac{k_{d_2}}{2^n - k_{d_2}}}}_{\theta_{k_{d_2}}} - \underbrace{\arctan \sqrt{\frac{k_{d_1}}{2^n - k_{d_1}}}}_{\theta_{k_{d_1}}} \tag{5.17}$$

Operator $\Delta\theta_{k_{d_1}, k_{d_2}}$ provides a mechanism for determining the operational success of adding

$(d_2 - d_1)$ extra levels of depth relatively to d_1. Accordingly, small $\Delta\theta_{k_{d_1},k_{d_2}}$ values can be understood as not contributing in a significant manner to changing the system's overall state. Conversely, high $\Delta\theta_{k_{d_1},k_{d_2}}$ values reveal that the system's state significantly shifted towards $|\psi\rangle$. Ultimately, one can only expect to get as far as state $|\psi\rangle$. From this point on Grover's usual $O(\sqrt{N})$ iterations will still be required in order to perform a rotation of $|\psi\rangle$ towards $|\psi_{good}\rangle$, leaving the algorithm overall complexity unchanged.

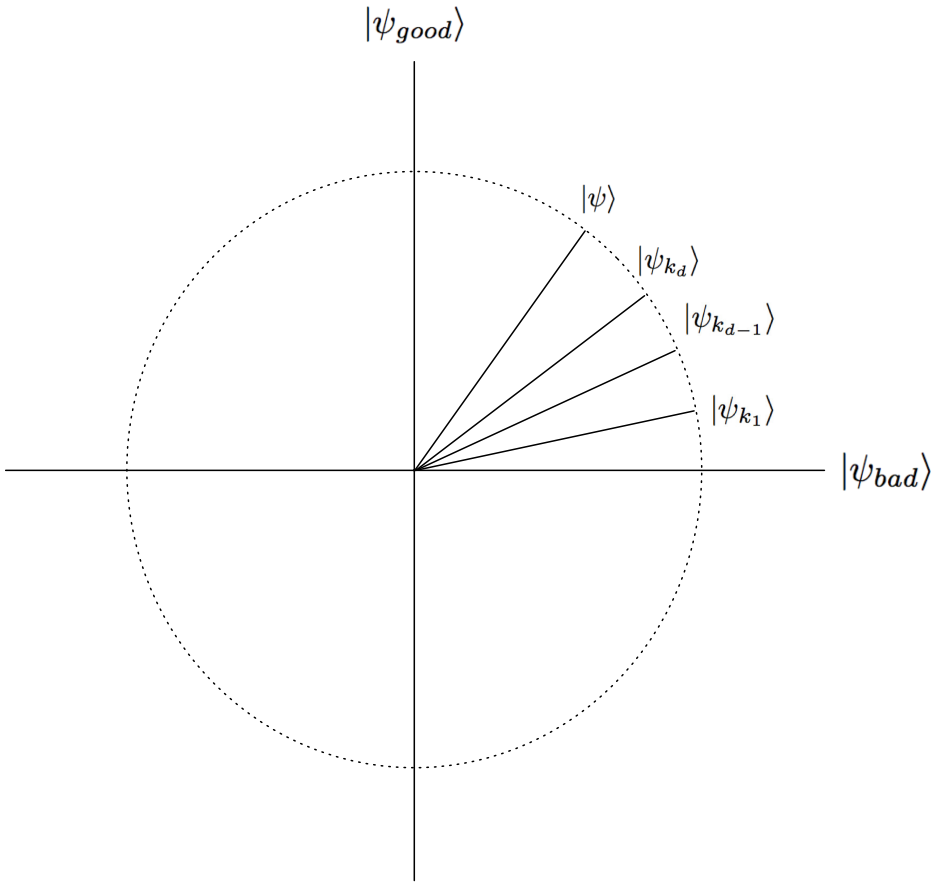


Figure 5.4: The geometric interpretation of the original state vector $|\psi\rangle$ alongside different state vectors $|\psi_i\rangle$ reflecting the state attained by performing a search to depth-level i. Deeper searches will be closer to the original $|\psi\rangle$ state. All states are unit-length vectors.

5.3.3 Extending the quantum heuristic

Can the concepts of the quantum heuristic be extended in such a way as to perform some type of useful task? In order to answer this question assume that the heuristic function employed

has a probabilistic distribution. The following sections review some principles surrounding heuristic distributions and describe how to incorporate these concepts into the tree search method.

Heuristic distributions

Assume that the heuristic function employed has a probabilistic distribution, *i.e.* the probability of each output value is between 0 and 1. This behaviour can be written as $0 \leq P(X = x) \leq 1$ where X is a random variable representing an output event, and x is a possible value of X. Additionally, a random variable may be either discrete or continuous depending on the values that it can assume. Random variables whose set of possible values belong to $\mathbb{Z}$ are said to be discrete. On the other other hand, continuous variables map events to an uncountable set such as $\mathbb{R}$. For a discrete random variable X, the sum of the set containing all possible probability values is 1 as illustrated by Expression 5.18.

$$\sum_{i=1}^{n} P(X = x_i) = 1 \tag{5.18}$$

The sliding block puzzle discussed in Chapter 4 can also be employed when trying to study heuristic distributions. It is possible to define a simple heuristic function for this problem, namely h_1, which simply calculates the number of misplaced tiles between a board and a target board configuration. For the 8-puzzle, function h_1 outputs values belonging to the range $[0, 9]$ and consequently can be classified as discrete. The discrete probability distribution for function h_1 is presented in Figure 5.5.

If X is a continuous random variable then there exists a nonnegative function $P(X)$, the probability density function of the random variable X. The density function $P(X = c)$ is defined as the ratio of the probability that X falls into an interval around c, divided by the width of the interval, as the interval width goes to zero [55]. This process is illustrated in Expression 5.19. Additionally, the density function must be nonnegative for all arguments and must obey the behaviour presented in Expression 5.20 [183].

$$P(X = c) = \lim_{dx \to 0} P(c \leq X \leq c + dx)/dx \tag{5.19}$$

$$\int_{-\infty}^{+\infty} P(X)dx = 1 \tag{5.20}$$

The n-puzzle search problem is also helpful when trying to develop a continuous heuristic

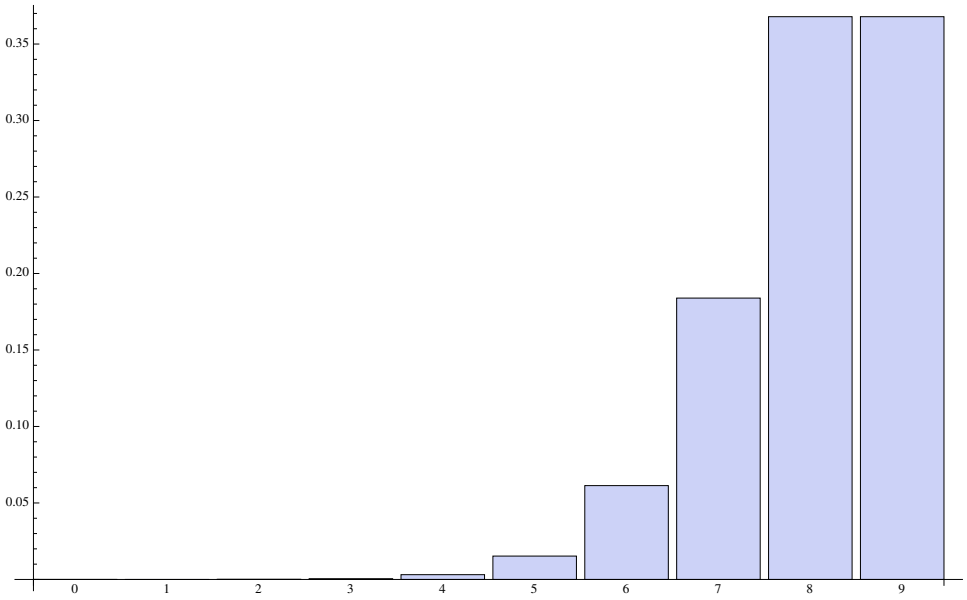


Figure 5.5: Discrete probability distribution for h_1 when applied to the 8-puzzle

function. What is only required is to determine a heuristic function h_2 mapping values into a real codomain. This can be done by employing a metric such as the euclidean distance. For instance, h_2 can be defined in such a fashion as to calculate the euclidean distance for all the corresponding elements of a board and a target board configuration. *I.e.*, $h_2 = \sum_{\forall \text{ tiles}} d(l_{\text{tile}}, l'_{\text{tile}})$, where d is the euclidean distance, l_{tile} the location of a tile in a board configuration and l'_{tile} the location of the corresponding tile in the target configuration. The probability density function for h_2 regarding the 8-puzzle is presented in Figure 5.6.

The probability that a random variable X takes on a value that is less than or equal to x is referred to as the cumulative distributive function F and has the form $F(x) = P(X \leq x)$. The cumulative distribution function $F(a)$ for a discrete random variable is a simple sum of the values up to element a. This behaviour is illustrated in Expression 5.21. Similarly, a cumulative probability density function $F(a)$ can be defined for a continuous random variable as shown in Expression 5.22.

$$F(a) = \sum_{\text{all } x \leq a} P(X = x) \tag{5.21}$$

$$F(a) = \int_{-\infty}^{a} P(x)dx \tag{5.22}$$

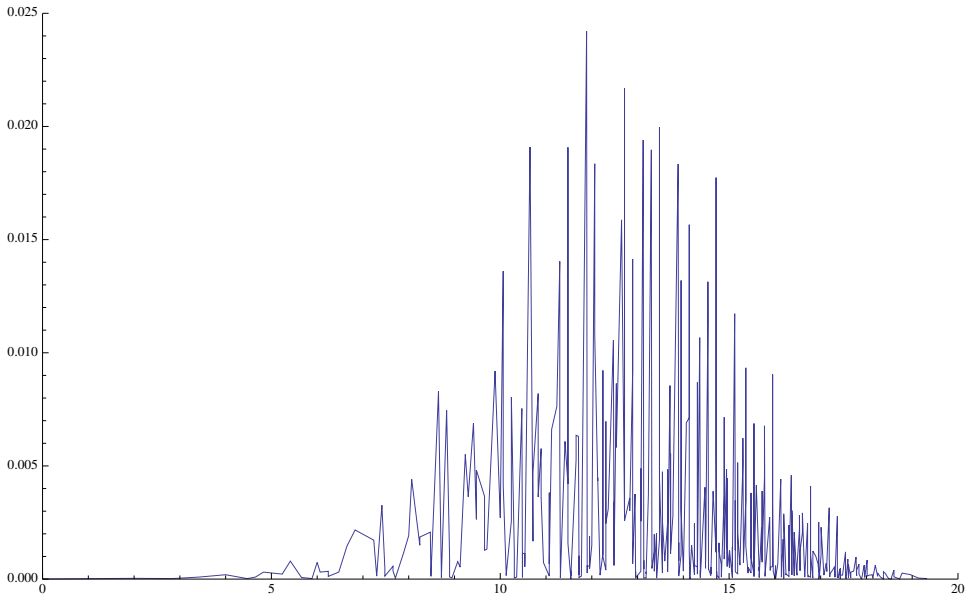


Figure 5.6: Continuous probability density function for h_2 when applied to the 8-puzzle

It may also be important to determine when, *i.e.* for which values, the cumulative distribution function equals a certain probability p, *i.e.* $F(x) = p$. In probability theory, this mapping is known as the quantile function of a cumulative distribution function $F(x)$ and is expressed as $F^{-1}(p) = x$. Care must be taken in order to differentiate between discrete and continuous random variables. In the former, there may exist gaps between values in the domain of the cumulative distribution function, accordingly F^{-1} is defined as presented in Expression 5.23 where inf denotes the infimum operator [79].

$$F^{-1}(p) = inf\{x \in \mathbb{R} : p \leq F(x)\} \tag{5.23}$$

In the case of a continuous random variable obtaining a clear expression for the quantile function is not so trivial since it requires determining the inverse of an integral. Notwithstanding, it is still possible to determine such expressions, for instance, the quantile function of a normal distribution with mean μ and standard deviation σ is illustrated in Expression 5.24, where erf is the Gauss error function [79].

$$F^{-1}(p, \mu, \sigma^2) = \mu + \sqrt{2}\sigma erf^{-1}(2p - 1), \quad p \in [0, 1] \tag{5.24}$$

Extended quantum heuristic

Naturally, the question arises, how can this process be combined with the quantum production system approach? It turns out that when a fourth of all possible states analyzed by Grover's algorithm are marked as solutions, *i.e.* $k = \frac{1}{4}.N$ then a single iteration is required in order to obtain with certainty one of the solution states [99]. Accordingly, it is possible to try to fine tune the behaviour of the unitary operator such that it outputs a solution for a fourth of all cases. This procedure can be performed with the assistance of the quantile function concept introduced in the previous section.

Envisage a scenario where it is important to obtain the states which are closest to a solution and a heuristic function $f : X \rightarrow Y$ is supplied. Then, the states that should be marked as solutions are those that produce the smallest values of codomain Y. Also, assume that the states which are closer to a goal node are those who tend to happen less times (as exemplified by Figure 5.5 and Figure 5.6). From a probabilistic point-of-view it is possible to check if the heuristic value is less than or equal to the quantile function output for a probability of 25%. This behaviour is illustrated in Expression 5.25, where f stands for the function evaluation of the arguments, *i.e.* $f(n, a_1, a_2, \cdots, a_d)$.

$$U|n\rangle|a_1 a_2 \cdots a_d\rangle = \begin{cases} - & |n\rangle|a_1 a_2 \cdots a_d\rangle \quad \text{if} f \leq F^{-1}(0.25) \\ & |n\rangle|a_1 a_2 \cdots a_d\rangle \quad \text{otherwise} \end{cases} \tag{5.25}$$

Ideally, by using this approach it is possible to obtain a superposition containing one fourth of the "closest" states to a goal configuration by applying a single iteration of Grover's algorithm. Additionally, the results of Expression 5.25 can be expanded in order to contemplate different sections of a probability distribution. For instance, it is possible to obtain in a single iteration of Grover's algorithm the states which lie between heuristic values $(F^{-1}(0.5) - F^{-1}(0.25))$. Similarly, a choice could also be made to obtain the states belonging to $(F^{-1}(0.75) - F^{-1}(0.5))$ or $(F^{-1}(1) - F^{-1}(0.75))$. More generally, let a and b denote two probability values such that $b - a = 0.25$, then it is possible to determine if a given heuristic value belongs to the range $[F^{-1}(a), F^{-1}(b)]$. This strategy for selecting 25% of the search space is presented in the unitary operator shown in Expression 5.26, where f stands for the function evaluation of the arguments, *i.e.* $f(n, a_1, a_2, \cdots, a_d)$.

$$U|n\rangle|a_1 a_2 \cdots a_d\rangle = \begin{cases} - & |n\rangle|a_1 a_2 \cdots a_d\rangle \quad \text{if} f \in [F^{-1}_{(a)}, F^{-1}_{(b)}] \\ & |n\rangle|a_1 a_2 \cdots a_d\rangle \quad \text{otherwise} \end{cases} \tag{5.26}$$

In a certain sense, employing this type of procedure allows for a kind of partial selection of the search space to be performed using a single Grover iteration. The selection is only partial

because upon measurement a random collapse amongst the marked states is obtained.

5.4 Final considerations

The proposal for a quantum production system incorporates classical search concepts, expressed through unitary operators alongside Grover's quantum search algorithm. The method relies on testing a sequence of actions in order to determine if a goal state was reached. However, other applications may not require this type of precise knowledge. For instance, some problems may be interested in determining the first m bits of a n bit solution string. Grover and Radhakrishnan were some of the first to purpose a possible approach to this problem in [80]. The main motivation of their work was the following: suppose a quantum search space containing a single solution is divided into l-blocks of equal size, and that the goal is to determine in which of the l-blocks the solution is, can this process be performed with fewer queries than the original Grover algorithm? In practice, this problem reduces to the one of determining the first m bits of the n bit computational basis containing the solution, with $m \leq n$. The authors proceed by analysing what happens when a variation of Grover's iterate for amplitude amplification is applied to each block. They conclude that it is indeed easier to determine the initial m bits. However, the speedup obtained does not change the overall complexity. In addition, as m grows closer to n the computational gains obtained disappear [80]. The original work focused exclusively on finding a single block with one solution. Subsequently, a generalization method was proposed in [45] considering multiple solution states equally spread amongst a number of solution blocks. Grover-Radhakrishnan's result was further improved in [123] when the authors developed a simple partial search algorithm targeting a large number of blocks. An additional optimization to Grover-Radhakrishnan's work was proposed in [122] improving the lower bound on the number of oracle queries. The motivation behind partial search allows the method to sacrifice precision for speed. Analogously to the intuition behind heuristics it is possible to envisage a quantum search system where different subtrees are tested for a solution using the principles of partial search. Such a system would require an adequate mapping mechanism in order for the different blocks present to convey information about the subtree, *i.e* a correspondence between the block and subtree concepts. Upon measuring, the collapsed state would contain valid information about the subtree where the solution can be found. This behaviour would be similar to that of an informed search.

5.5 Conclusions

In this chapter some of the ramifications of the quantum production system were examined. Two key aspects were used, namely: the use of a non-constant branching factor and the adoption of a heuristic point of view. Clearly, both of these cases are not without its flaws. However, the additional insight provided is valuable. In the case of the non-constant branching factor, deciding whether or not to employ Grover, or to proceed classically, is a process that should take into account the maximum and the average branching factor. Additionally, by evaluating the states that are within a given threshold it is possible to incorporate some classical heuristic concepts into the approach. These concepts were further extended with the use of probabilistic distribution functions allowing for a selection mechanism to be obtained. This mechanism enables specific ranges of quantum states to be obtained in an efficient manner.

Chapter 6

A quantum production model

6.1 Introduction

In this chapter the circuit-based approach presented in Chapter 4 is generalized in order to provide a model of quantum computation based on production system theory with a clear emphasis on problem-solving behaviour. Traditional approaches such as the quantum Turing machine are complex mechanisms oriented towards general purpose computation. A quantum production system model would be more suited to typical artificial intelligence tasks such as reasoning, inference and hierarchical search. From the onset it is possible to immediately pose some questions, namely: How should such a quantum production system model be developed? What are the requirements of quantum computation and its respective impact on the aforementioned model? How to develop the associated unitary operator? Additionally, what are the performance gains from employing quantum mechanics and how does such a proposition compare against similar strategies? Finally, are there any requirements that should be observed for those improvements?

By employing such an approach it is possible to (1) provide a detailed explanation of how to develop a quantum production system model; (2) assess the main differences between this proposition and its classical analog; and (3) provide an insight into better describing the power of quantum computation. However, it is not my intention to present an exact characterization of quantum computational models. Answering this question would have far-reaching consequences on complexity theory which are beyond the scope of this work.

The following sections are organised as follows: Section 6.2 presents the formal definitions of the proposition for a quantum production system; Section 6.3 presents an assessment

comparing the performance of classical production systems against the quantum proposition. The concluding remarks of this chapter are presented in Section 6.4.

6.2 Formal Definitions

In this section a modular approach to the quantum production system proposition is presented. Accordingly, a set of definitions is described in Section 6.2.1 incorporating traditional production system behaviour. General reversibility requirements associated with quantum computation are presented Section 6.2.2. These concepts are then employed to enumerate the characteristics of a probabilistic production system in Section 6.2.3. The probabilistic model will serve as a basis for the quantum production system, which will extend those concepts in Section 6.2.4.

6.2.1 Classical Production System

Any approach to a general quantum production system model needs to incorporate powerful computational abstractions, which are not bounded by input length, in a similar manner to the classical Turing machine [205] and its quantum counterparts. Accordingly, the following definitions are presented through set theory. As previously discussed each production system S consists of a set of production rules R and a control system C alongside a working memory W. The following definitions embody the production system incorporate discussed in Section 1.3, respectively:

Definition 1: Let Γ be a finite nonempty set whose elements are referred to as symbols. Additionally, let Γ^* be the set of finite strings over Γ.

Definition 2: The working memory W is capable of holding a string belonging to Γ^*. The working memory is initialized with a given string, who is also commonly referred to as the initial state γ_i.

Definition 3: The set of production rules R has the form presented in Expression 6.1.

$$\{(precondition, action) | precondition, action \in \Gamma^*\} \tag{6.1}$$

Each rules precondition is matched against the contents of the working memory. If the precondition is met then the action part of the rule can be applied, changing the contents of the working memory.

Definition 4: The formal definition of a production system S is a tuple (Γ, S_i, S_g, R, C) where Γ, R are finite nonempty sets and $S_i, S_g \subset \Gamma^*$ are, respectively, the set of initial and goal states. The control function C satisfies Expression 6.2.

$$C : \Gamma^* \rightarrow R \times \Gamma^* \times \{h, c\} \tag{6.2}$$

The control system C chooses which of the rules to apply and terminates the computation when a goal configuration, γ_g, of the memory is reached. If $C(\gamma) = (r, \gamma', \{h, c\})$ the interpretation is that, if the working memory contains symbol γ then it is substituted by the action γ' of rule r and the computation either continues, c, or halts, h. Traditionally, the computation halts when a goal state $\gamma_g \in S_g$ is achieved through a production, and continues otherwise.

6.2.2 Reversible Requirements

As stated in Chapter 3, discrete state evolution is achieved through mathematical maps known as unitary operators [164]. These maps correspond to injective and surjective functions, i.e. bijections. Bijections guarantee that every element of the codomain is mapped by exactly one element of the domain [28]. From a computational perspective, the bijection requirement can be obtained by employing reversible computation. Classical computation is an irreversible process since at its core the use of many-to-one binary gates makes it impossible to ensure a one-to-one and onto mapping. A computation is said to be reversible if given the outputs the inputs can be uniquely recovered [204] [203]. Irreversible computational processes can be made reversible by (1) substituting irreversible logic elements by the adequate reversible equivalents; or (2) by accounting for the information that is traditionally lost.

The emphasis in production system theory consists in determining what state is obtained after applying a production. Forward chaining is employed when moving from the conditions to the actions, *i.e.* an action is applied when all the associated conditions are met. Conversely, there may be a need for determining which state preceded the current state, *i.e.* a sort of backtrace mechanism from a given state up until another state. This mechanism allowing one to reverse the actions applied and thus obtaining the associated conditions is also commonly referred to as backward chaining. Although this behaviour seems fairly simple and intuitive it is possible to immediately pose a question regarding the system's nature, namely what are the requirements associated with a reversible production system?

It is possible to adapt Bennett's original set of definitions [23] in order to describe the

behaviour of a production system by a finite set of transition formulas also referred to as quadruples, in an allusion to the form of Expression 6.2. Each quadruple maps the present state of the working memory to its successor. By introducing the tuple terminology it becomes simpler to present the following set of definitions:

Definition 5: A production system can be perceived as being deterministic if and only if its quadruples have non-overlapping domains.

Definition 6: A production system is said to be reversible if and only if its quadruples have non-overlapping ranges.

Definition 7: A reversible and deterministic production system can be defined as a set of quadruples no two of which overlap either in domain or range.

These definitions contrast with Bennett's more elaborated model where information regarding the internal states of the control unit before and after the transition, alongside tape movement with the associated reading and writing information is maintained. In order to fully understand the exact impact of such requirements picture a production system responsible for sorting strings composed of letters *a*, *b*, *c*, *d*, and *e* based on [213]. The set of production rules is presented in Table 6.1. Whenever a substring of the original string matches a rule's condition the production is applicable. Applying a specific rule consists in replacing the original substring, *i.e.* precondition, by the action string. The sequence of rules that is applied when the working memory is initialized in state "edcba" is illustrated in Table 6.2, with the computation proceeding until the string is fully sorted.

Bennett [23] points to the fact that any irreversible computation can be made reversible by saving all the information that is typically erased. However, this reversible history needs to be saved into a resource. Reusing this resource would require the information to be erased or thrown away, merely postponing the problem. The solution relies on performing a computation, saving the intermediate information that is typically lost, and then using this information to backtrack to the original input. Since both forward and backward stages are done in a reversible manner, the overall process always preserves the original information.

However, before undoing the computation, care has to be taken in order to ensure that the output is preserved. This requires copying the output to an output register, an operation which has to be performed reversibly. Once the output copy has been completed it is possible to proceed with the backward stage, *i.e* reverse the consequences of each quadruple application. Eventually, the computation terminates, the production system returns to its original state and the result of the procedure is stored in the output medium. In Bennett's original work the reversible Turing machine is composed of three tapes, namely [23]:

Rule	Precondition	Action	Symbolic
R1	ba	ab	ba → ab
R2	ca	ac	ca → ac
R3	da	ad	da → ad
R4	ea	ae	ea → ae
R5	cb	bc	cb → bc
R6	db	bd	db → bd
R7	eb	be	eb → be
R8	dc	cd	dc → cd
R9	ec	ce	ec → ce
R10	ed	de	ed → de

Table 6.1: Rule set for sorting a string composed of letters *a*, *b*, *c*, *d*, and *e* (adapted from [143]).

Iteration Number	Working Memory	Conflict Set	Rule Fired	Continue?
0	edcba	$\{R1, R5, R8, R10\}$	R1	continue
1	edcab	$\{R2, R8, R10\}$	R2	continue
2	edacb	$\{R5, R3, R10\}$	R3	continue
3	eadcb	$\{R5, R8, R4\}$	R4	continue
4	aedcb	$\{R5, R8, R10\}$	R5	continue
5	aedbc	$\{R6, R10\}$	R6	continue
6	aebdc	$\{R8, R7\}$	R7	continue
7	abedc	$\{R8, R10\}$	R8	continue
8	abecd	$\{R9\}$	R9	continue
9	abced	$\{R10\}$	R10	continue
10	abcde	$\emptyset$	$\emptyset$	halt

Table 6.2: An example of the sequence of rules applied for sorting a string composed of letters *a*, *b*, *c*, *d*, and *e*.

- working tape - where the program's input is initially stored and computation is performed in order to obtain an output which is later reversed to the original input;
- history tape - where the information that is traditionally thrown away is kept, once the program's output has been copied the history information is used in order to revert the working tape to its original state;
- output tape - where the program's output is stored.

By observing Table 6.2 it is possible to see that in order to ensure that the original input is obtained, the sequence of rules leading from an initial state γ_i to a goal state γ_g needs to be accounted for. This sequence of rules can be used in order to "undo" each action. In doing so it is possible to obtain each precondition that led to a particular action being applied, up until an initial state $\gamma_i \in S_i$. The quadruples presented in Expression 6.2 effectively convey information about which production is applied when going from a certain condition to the appropriate action. Additionally, in production system theory there exists a strong emphasis on the sequence of rules leading up to a target state. This situation contrasts with the traditional interest of merely knowing the final state of the working memory. If Bennett's original definitions of the reversible Turing machine are adapted then it becomes possible to obtain a mapping for a reversible production system. This process can be performed by requiring that:

1. applying a production results in its addition to the history tape, instead of a new control-unit state. Since the quadruple and production rules are equivalent concepts the same transitional information employed by Bennett's model is being stored;
2. once the computation halts it is necessary to copy the contents of the history tape to the output tape, this contrasts with the original copying of the working tape. In order to do so the history tape's head needs to be place at the tape's beginning. Afterwards, the copy process from the history tape to the output tape can proceed.
3. upon the copying mechanism's conclusion, the output tape's head needs to be placed at the beginning. This process can be performed by shifting left the output tape until a blank symbol is found.

Table 6.3 illustrates this set of ideas for a reversible production simple based on the string sorting production system presented earlier (Table 6.1 and Table 6.2). As it is possible to verify the computation proceeds normally for iteration 0 through 10, also known as the forward computation stage. The only alteration to Bennett's model consists in adding the productions fired to the history tape. Once this stage has concluded the history tape's head needs to be properly placed at the beginning. This step is carried out in iteration 11. The position of a tape's head by an underbar. The system then proceeds in iteration 12 by

Iteration	Memory	Rule	History Tape	Output Tape
0	edcba	$R1$	{_}	{_}
1	edcab	$R2$	{$\underline{R1}$}	{_}
2	edacb	$R3$	{$R1$, $\underline{R2}$}	{_}
3	eadcb	$R4$	{$R1$, $R2$, $\underline{R3}$}	{_}
4	aedcb	$R5$	{$R1$, $R2$, $R3$, $\underline{R4}$}	{_}
5	aedbc	$R6$	{$R1$, $R2$, $R3$, $R4$, $\underline{R5}$}	{_}
6	aebdc	$R7$	{$R1$, $R2$, $R3$, $R4$, $R5$, $\underline{R6}$}	{_}
7	abedc	$R8$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $\underline{R7}$}	{_}
8	abecd	$R9$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $\underline{R8}$}	{_}
9	abced	$R10$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $\underline{R9}$}	{_}
10	abcde	$\emptyset$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $\underline{R10}$}	{_}
11	abcde	$\emptyset$	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}	{_}
12	abcde	$\emptyset$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $\underline{R10}$}	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $\underline{R10}$}
13	abcde	$\emptyset$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $\underline{R10}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
14	abcde	$\emptyset$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $\underline{R10}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
15	abced	$R10^{-1}$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $\underline{R9}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
16	abecd	$R9^{-1}$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $\underline{R8}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
17	abedc	$R8^{-1}$	{$R1$, $R2$, $R3$, $R4$, $R5$, $R6$, $\underline{R7}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
18	aebdc	$R7^{-1}$	{$R1$, $R2$, $R3$, $R4$, $R5$, $\underline{R6}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
19	aedbc	$R6^{-1}$	{$R1$, $R2$, $R3$, $R4$, $\underline{R5}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
20	aedcb	$R5^{-1}$	{$R1$, $R2$, $R3$, $\underline{R4}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
21	eadcb	$R4^{-1}$	{$R1$, $R2$, $\underline{R3}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
22	edacb	$R3^{-1}$	{$R1$, $\underline{R2}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
23	edcab	$R2^{-1}$	{$\underline{R1}$}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}
24	edcba	$R1^{-1}$	{_}	{$\underline{R1}$, $R2$, $R3$, $R4$, $R5$, $R6$, $R7$, $R8$, $R9$, $R10$}

Table 6.3: Operation of a reversible production system based on the example of Table 6.2 and Bennett's model for a reversible Turing machine. The underbar denotes the position of the head.

copying the contents of the history tape onto the output tape. Additionally, the output tape's head is placed at the beginning in iteration 13. The last stage of the computation consists in undoing each one of the applied productions, as illustrated from iteration 14 to 24. The inverse of a rule R mapping a precondition A into an action B, *i.e.* $R : A \rightarrow B$, is represented by R^{-1} such that $R^{-1} : B \rightarrow A$. By inverting the rules applied implies reversing the consequences of each associated quadruple.

6.2.3 Probabilistic Production System

Consider a production system whose control strategy chooses a rule to apply from set of production rules based on a probability distribution. This behaviour can be formalized with a simple reformulation of Expression 6.2 as illustrated by Expression 6.3, where $C(\gamma, r, \gamma', d)$ represents the probability of choosing rule r, substituting symbol γ with γ' and making a decision d on whether to continue or halt the computation if the memory contains γ.

$$C : \Gamma^* \times R \times \Gamma^* \times \{h, c\} \rightarrow [0, 1] \tag{6.3}$$

Additionally, it would have to be required that $\forall \gamma \in \Gamma$ Expression 6.4 be observed

$$\sum_{\forall (r,\gamma',d) \in R \times \Gamma \times \{h,c\}} C(\gamma, r, \gamma', d) = 1 \tag{6.4}$$

This modification to the deterministic production system allows the control strategy to yield different states with probabilities that must sum up to 1. In such a model, a computation can be perceived has having an associated probability, which is simply the multiplication of each production's probability. If the several possibilities are accounted for the overall computational process presents a tree form as the one illustrated in Figure 1.1.

6.2.4 Quantum Production System

A suitable model for a probabilistic production system enables a mapping between real-valued probabilities and complex-value quantum amplitudes. Specifically, the complex valued control strategy would need to behave as illustrated in Expression 6.5 where $C(\gamma, r, \gamma', d)$ provides the amplitude if the working memory contains symbol γ then rule r will be chosen, substituting symbol γ with γ' and a decision d made on whether to continue or halt the computation.

$$C : \Gamma^* \times R \times \Gamma^* \times \{h, c\} \to \mathbb{C} \tag{6.5}$$

The amplitude provided would also have to respect Expression 6.6, $\forall\ \gamma \in \Gamma$.

$$\sum_{\forall (r,\gamma',d) \in R \times \Gamma \times \{h,c\}} |C(\gamma, r, \gamma', d)|^2 = 1 \tag{6.6}$$

Is it possible to elaborate on the exact unitary form that C should take? Developing a classical computational gate for calculating Expression 6.2 would require the form illustrated in Figure 6.1a. Since multiple arguments could potentially map onto the same element such a strategy would not allow for reversibility. Theoretically, any irreversible production system can be made reversible by adding some auxiliary input bits and through the addition modulo 2 operation [203], a process formalized in Expression 6.7 and shown in Figure 6.1b. Since the inputs are now part of the outputs, this mechanism allows for a bijection to be obtained. Notice that Expression 6.7 does not output a complex value. However, the unitary matrix that can be derived from such an expression can be employed alongside unit-length complex-valued vectors to obtain an overall control strategy that is in accordance with Expression 6.5.

$$\underbrace{(\gamma, b_0, b_1, b_2)}_{\text{input vector } v_1} \xrightarrow{C} \underbrace{(\gamma, r \oplus b_0, \gamma' \oplus b_1, \{h, c\} \oplus b_2)}_{\text{output vector } v_2} \tag{6.7}$$

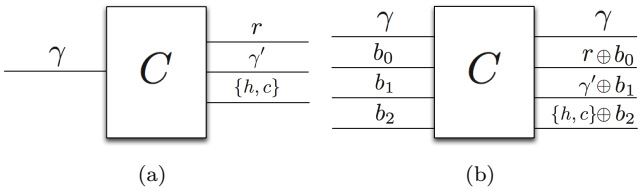


Figure 6.1: An irreversible control strategy 6.1a can be made reversible 6.1b through the introduction of a number of constants and auxiliary input and output bits.

The gate can be perceived as acting on vector v_1 and delivering v_2. Then such behaviour can be described as shown in Expression 6.8, where C is the required unitary operator.

$$C|v_1\rangle = |v_2\rangle \tag{6.8}$$

Based on Expression 6.7 and Expression 6.8 it becomes possible to develop a unitary operator C. Accordingly, C acts upon an input vector v_1 conveying specific information about the argument's state. From Expression 6.7 it is possible to verify that any input vector $|v_1\rangle$ should be large enough to accommodate γ, b_0, b_1 and b_2. Since b_0, b_1 and b_2 will be used for bitwise addition modulo 2 operations with, respectively, r, γ' and $\{h, c\}$, then the appropriate dimensions for a binary encoding of these elements need to be determined. Assume that:

- $\alpha = \lceil \log_2 |\Gamma| \rceil$, represents the number of bits required to encode the symbol set
- $\beta = \lceil \log_2 |R| \rceil$, represents the number of bits required to encode each one of the productions;
- δ is a single bit used to encode either h or c

If a binary string is employed to represent this information, then its length will be $\alpha + \beta + \delta$ bits, thus allowing for a total of $2^{\alpha+\beta+\delta}$ combinations. This information about the input's state can be conveyed in a column vector v_1 of dimension $2^{\alpha+\beta+\delta}$. The general idea being that the m^{th} possible combination can be represented by placing a 1 on the m^{th} row of such a vector. These same principles are still observed by v_2. The unitary operator's responsibility relies on interpreting such information and presenting an adequate output vector v_2. The overall requirements of unitarity alongside the dimensions of input and output vectors imply that unitary operator C will have dimension $2^{\alpha+\beta+\delta} \times 2^{\alpha+\beta+\delta}$.

A parallel can be established between C's behaviour and the truth table concept of classical gates. Truth tables are classical mechanisms employed to describe logic gates employed in

electronics. The tables list all possible combinations of the inputs alongside the respective results [144]. In a similar manner, it is possible to build a unitary operator C by going through all possible combinations and decoding the information present in each combination. This procedure is illustrated through pseudo-code in Procedure 1. Lines 1-3 are employed in order to determine the required number of bits for the encoding mechanism. These values can also be used to determine the dimension $2^{\alpha+\beta+\delta} \times 2^{\alpha+\beta+\delta}$ of unitary operator C. This operator is initialized in line 4 as a matrix with all entries set to zero.

The cycle *for* from lines 5-15 is responsible for going through all possible combinations. Line 6 of the code obtains a string S, which is the binary version of decimal combination λ, represented as $\lambda_{(2)}$ to illustrate base-2 encoding. Recall from Expression 6.1 that each input vector needs to convey information about γ, b_0, b_1 and b_2. Accordingly, for each λ the different elements of the string need to be parsed in order to determine those values. This process is illustrated through lines 7-10, which are responsible for obtaining the binary substrings. For any string S, $S[i, j]$ is the contiguous substring of S that starts at position i and ends at position j of S [91]. Line 11 is responsible for invoking function mapBinaryEncoding, which maps substring S_1 to a symbol $\gamma \in \Gamma$.

Once the input symbol γ has been determined it is possible to calculate the transition depicted in Expression 6.2. It is important to mention that the transition calculated in Line 12 through function C should not be confused with the associated unitary operator C of line 4. The next logical step consists in forming a binary string represented as $w_{(2)}$, which is simply the concatenation of elements $S_1, r_{(2)} \oplus S_2, \gamma'_{(2)} \oplus S_3$ and $d_{(2)} \oplus S_4$. Again, this step is done by employing the base-2 version of elements r, γ' and d. After the conclusion of line 13 all the information required to determine the corresponding mapping will have been determined, λ can be viewed as the decimal encoding of the input state, whilst ω can be interpreted as the new decimal state achieved. This behaviour can be adequately incorporated into the unitary operator by marking column λ and row ω with a one, a procedure realized in line 14.

Correctness Proof: In order to verify the correctness of Procedure 1 operator C needs to be confirmed as indeed performing a bijective mapping. At its core a bijection performs a simple permutation of all possible input state combinations. Accordingly, for a collision to occur, *i.e.* multiple arguments mapping into the same image, would require that several λ's produced the same ω. If the transition function employed in Line 12 is irreversible then it is conceivable that different γ's may produce the same output vector (r, γ', d). However, the new state w besides contemplating output (r, γ', d) through the addition modulo 2 elements $r_{(2)} \oplus S_2, \gamma'_{(2)} \oplus S_3$ and $d_{(2)} \oplus S_4$ also takes into consideration the original input symbol γ allowing for a differentiation of possible collision states. As a consequence, for a collision to still occur would require that function mapBinaryEncoding produced the same γ for

Algorithm 1 Pseudo code for building unitary operator C

1: $\alpha = \lceil \log_2 |\Gamma| \rceil$
2: $\beta = \lceil \log_2 |R| \rceil$
3: $\delta = 1$
4: $C = \text{zeros}[2^{\alpha+\beta+\delta}, 2^{\alpha+\beta+\delta}]$
5: **for all** integers $\lambda \in [0, 2^{\alpha+\beta+\delta}]$ **do**
6: $S = \lambda_{(2)}$
7: $S_1 = S[0, \alpha - 1]$
8: $S_2 = S[\alpha, \alpha + \beta - 1]$
9: $S_3 = S[\alpha + \beta, 2\alpha + \beta - 1]$
10: $S_4 = S[2\alpha + \beta, 2\alpha + \beta + \delta - 1]$
11: $\gamma = \text{mapBinaryEncoding}(\ \Gamma,\ S_1\ )$
12: $C(\gamma) = (r, \gamma', d)$
13: $\omega_{(2)} = S_1, r_{(2)} \oplus S_2, \gamma'_{(2)} \oplus S_3, d_{(2)} \oplus S_4$
14: $C_{\omega,\lambda} = 1$
15: **end for**

different binary strings. This same binary mapping behaviour can be easily avoided with proper management of an adequate data structure thus guaranteeing the correctness of such a procedure.

Unitary operator C is only responsible for applying a single production of the control strategy. This represents a best case scenario where a problem's solution can be found within the immediate neighbours, *i.e.* those nodes that can be reached by applying a single production. However, the production system norm relies on having to apply a sequence of rules before obtaining a solution state. This proposition can be easily extended in order to apply multiple steps. Such an extension would require developing a logical circuit employing elementary gates C alongside any necessary output redirection to the adequate inputs. Algebraically, such a procedure would require unitary operator composition acting upon the appropriate inputs, which would continue to guarantee overall reversibility. Additionally, it is important to emphasize that any potential unitary operator requires the ability to verify if the conditional part of a rule is met (determine if a string contains a substring), which can be achieved with simple comparison operators.

6.3 Classical vs. Quantum Comparison

Deutsch described a universal model of computation capable of simulating Turing machines with inherent quantum properties such as quantum parallelism that cannot be found in their classical counterparts [57]. However, the number of computational steps required by Deutsch's model grew exponentially as a function of the simulated Turing's machine run-

ning time. Subsequently, a more efficient model for a universal quantum Turing machine was proposed in [27]. In the same work the authors questioned themselves if a quantum Turing machine can provide any significant advantage over their classical equivalents. They proceeded by showing that a quantum Turing machine described in [56] is capable of efficiently solving the Fourier sampling problem. However, care was also employed in order to emphasize that their result did not prove that quantum Turing machines are more powerful than probabilistic Turing machines, since the latter can sample from a distribution within ϵ total variation distance of the desired Fourier distribution [27]. In [39] Yao shows that for any function computable in polynomial time by a quantum Turing machine there is an equivalent polynomial quantum circuit.

Naturally, the question arises: how does this quantum production system proposal fare against its classical counterpart? Namely, what is there to be gained by applying quantum computation? And what are the requirements associated to those improvements? In order to answer these questions take into account a unitary operator C, which is applied to an initial state $x \in S_i$. Additionally, assume that C needs to be applied a total of d times for a result to be obtained, where $d \in \mathbb{N}$ is chosen such that the computation is able to proceed until it stops. The result of applying C can be represented as $g(x)$, which in production system theory can be a simple output of the productions applied. As a consequence, the quantum register employed needs to convey information about the initial state and also be large enough to accommodate for $g(x)$. This requirement can be fulfilled by employing a unspecified length register $|z\rangle$. Accordingly, the initial state of the system can be represented by the left-hand side of Expression 6.9. The right-hand side represents the result obtained after unitary evolution.

$$C^d|x, z\rangle = |x, z \oplus g(x)\rangle \tag{6.9}$$

It is now possible to initialize register $|x\rangle$ as a superposition, $|\psi\rangle$, of all starting states, a procedure illustrated in Expression 6.10, where $S_i \subset \Gamma^*$ is the set of starting states. This procedure is also depicted in Figure 6.2, where multiple binary searches are performed simultaneously, with the dotted line representing initial nodes that, for reasons of space, are not shown, but are still present in the superposition. Now envisage a scenario where the production system definition only contemplates a single initial state, *i.e.* $|S_i| = 1$. Since it is not possible to explore the high levels of parallelism provided by the superposition principle, no significant advantage would be gained over the sequential procedure by applying $|\psi_n\rangle$. However, if the productions set cardinality is greater than one, then there exist several neighbour states, which can be employed as initial states thus circumventing the problem.

$$|\psi\rangle = \frac{1}{\sqrt{|S_i|}} \sum_{s \in S_i} |s\rangle \tag{6.10}$$

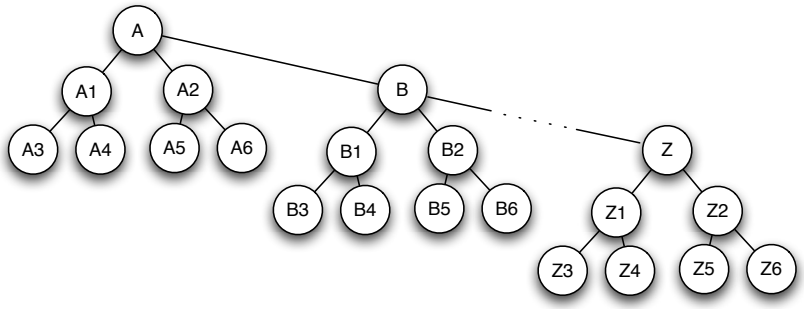


Figure 6.2: Parallel search with $S_i = \{A, B, \cdots, Z\}$ and $|\psi_n\rangle = \frac{1}{\sqrt{|S_i|}} \sum_{s \in S_i} |s\rangle$. The dotted lines represent the initial states belonging to superposition $|\psi_n\rangle$

This approach differs from other strategies of hierarchical search, namely [199] and [200], who, respectively, (1) evaluate a superposition of all possible tree paths up to a depth-level d in order to determine if a solution is present and (2) present an hierarchical decomposition of the quantum search space through entanglement detection schemes.

The following sections are organised as follows: Section 6.3.1 extends the concepts of Grover's algorithm in order to present a system combining the quantum production system proposal alongside the quantum search algorithm. Section 6.3.2 presents the concluding remarks by discussing the performance gains achieved over the classical production system equivalent.

6.3.1 Oracle Extension

In this section an extension to the oracle operator employed by Grover's algorithm is presented allowing it to be combined alongside the quantum production system proposal. As a result, it is important to determine what happens when two different functions f and g are combined into a single unitary evolution, as illustrated by Expression 6.11. In this specific case a choice was made to employ three quantum registers, namely $|x\rangle$, which is configured with the system's initial state, alongside registers $|y\rangle$ and $|z\rangle$ where, respectively, the output of functions $f(x)$ and $g(x)$ is stored. The original amplitude flipping process is a result of placing register $|y\rangle$ in the superposition state $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Accordingly, it is necessary to verify if the amplitude flip still holds with the oracle formulation of Expression 6.11 alongside $|y\rangle$'s

$$O|x\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}|z\rangle = \frac{1}{\sqrt{2}}\left(|x\rangle|f(x)\rangle|z\oplus g(x)\rangle - O|x\rangle|1\oplus f(x)\rangle|z\oplus g(x)\rangle\right) \tag{6.12}$$

$$= \begin{cases} \frac{1}{\sqrt{2}}\left(|x\rangle|0\rangle|z\oplus g(x)\rangle - |x\rangle|1\rangle|z\oplus g(x)\rangle\right) & \text{if } f(x)=0 \\ \frac{1}{\sqrt{2}}\left(|x\rangle|1\rangle|z\oplus g(x)\rangle - |x\rangle|0\rangle|z\oplus g(x)\rangle\right) & \text{if } f(x)=1 \end{cases}$$

$$= \begin{cases} |x\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}|z\oplus g(x)\rangle & \text{if } f(x)=0 \\ |x\rangle\frac{|1\rangle-|0\rangle}{\sqrt{2}}|z\oplus g(x)\rangle & \text{if } f(x)=1 \end{cases}$$

$$= (-1)^{f(x)}|x\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}|z\oplus g(x)\rangle \tag{6.13}$$

superposition initialization. This behaviour is shown in Expression 6.12. From Expression 6.13 it is possible to verify that despite the new oracle formulation the amplitude flipping continues to occur.

$$O|x,y,z\rangle = |x,y\oplus f(x),z\oplus g(x)\rangle \tag{6.11}$$

6.3.2 Performance Analysis

In order to proceed with the performance analysis picture a production system whose definitions are incorporated into a unitary operator C combining the results of Expression 6.9 and Expression 6.11. Accordingly, C will have the form presented in Expression 6.14, where $|x\rangle$ is initialized with a superposition of the production system starting states. In addition, an auxiliary register $|z\rangle$ with an unspecified length will be employed in order to accommodate for the productions applied, *i.e* the output growth of function $g(x)$.

Such a formulation of a production system C allows one to employ it alongside Grover's algorithm in order to speedup the computation. In this particular case $f(x)$'s definition needs to be changed in order to check if a goal state $s \in S_g$ is achieved after having applied d productions. *E.g.* assume that state M shown in Figure 1.1 is a goal state, then, assuming no backtracking occurs, such state can be reached by applying productions p_1, p_0 and p_1. As a consequence, such a state evolution as $C^3|A,0,\underline{\mathbf{0}}\rangle = |x,1,\{p_1,p_0,p_1\}\rangle$, where $\underline{\mathbf{0}}$ represents a vector of zeros. Function f new definition is presented in Expression 6.15. The state of the system is described by a unit vector in a Hilbert space $\mathcal{H}_{2^m} = \mathcal{H}_{2^n}\otimes\mathcal{H}_2\otimes\mathcal{H}_{2^p}$.

$$C^d|x,y,z\rangle = |x,y\oplus f(x),z\oplus g(x)\rangle \tag{6.14}$$

$$f(x) = \begin{cases} 1 & \text{, if } C^d|x\rangle \in S_g \\ 0 & \text{, otherwise} \end{cases} \tag{6.15}$$

Grover's original speedup was dependent on superposition $|\psi\rangle$ and the associated number of possible states. More concretely, the dimension of the space spanned is dependent on the dimension of the query register $|x\rangle$ employed. However, by applying an oracle C whose behaviour mimics that of Expression 6.14 the elements present in superposition $|\psi\rangle$ will interact with registers $|y\rangle$ and $|z\rangle$. Typically, register $|y\rangle$ is ignored when evaluating the running time, producing an overall superposition $|\xi\rangle$, which will no longer span the original 2^n possible states but 2^{n+p}. From an algebraic perspective, the interaction process is due to the tensor product employed to describe the overall state between $|x\rangle$, $|y\rangle$ and $|z\rangle$. As a result, it is possible to pose the following question: what can be said about the growth of $|z\rangle$ and its respective impact on overall system performance?

Assume that a solution state can always be found after d invocations, either by indeed finding a goal state or by applying a heuristic function to determine an appropriate state selection. Classically, an evaluation function would be invoked once for each one of the initial states, *i.e.,* $C = |S_i|d$ times. Is it possible to do any better with the proposed model? Answering this question requires determining appropriate boundary conditions on the exact dimensions of $|z\rangle$ for which it is still possible to obtain a speedup over classical procedures.

Employing Grover's algorithm results in a search procedure that will span the dimension of $|\xi\rangle$, which varies between $[2^n, 2^{n+p}]$. Accordingly, in the very unlikely best case scenario, the procedure will execute in $O(\sqrt{|S_i|})$ time. The number of quantum invocations will be $\sqrt{|S_i|}\,d$. Comparing both values allows one to conclude that C and Q differ by a factor of $\sqrt{|S_i|}$, effectively favoring the quantum proposal. Typically two things can be verified, namely: (1) d grows linearly and (2) $|S_i|$ generates an exponential growth search space. Such a ratio does not take into account the dimension of register $|z\rangle$. Therefore, it is important to determine what happens when $|z\rangle$ grows and how it affects overall performance. Let m denote the number of bits employed by registers $|x\rangle$ and $|z\rangle$, then the number of quantum iterations will be $Q = \sqrt{2^m}$. Accordingly, the $\frac{C}{Q}$ ratio can be restated in terms of m, as depicted in Expression 6.16, which effectively conveys the notion that each additional bit added to $|z\rangle$ impacts the ratio negatively by a factor of $\frac{1}{\sqrt{2}}$.

$$\frac{C}{Q} = \frac{|S_i|}{\sqrt{2^m}} \tag{6.16}$$

Additionally, it is also interesting to determine when is the number of quantum iterations Q smaller than the number of classical iterations C, as shown in Expression 6.17.

$$\begin{aligned} & Q & < & C & (6.17) \\ \Leftrightarrow\ & 2^m & < & |S_i|^2 & \\ \Leftrightarrow\ & m & < & \log_2 |S_i|^2 & (6.18) \end{aligned}$$

Expression 6.18 needs to be further refined since $m \in \mathbb{N}$ but the right-hand side may produce a value belonging to $\mathbb{R}$. This output is a consequence of having to deal with initial state sets S_i whose cardinality is not a power of 2. Notice that the measurement of performance chosen, respectively, the ratio C/Q will eventually be 1 when $m = \log_2 |S_i|^2$. Accordingly, if a larger number of bits is employed it effectively yields $C/Q < 1$, which will no longer translate into a speedup by the quantum version. That being the case, it is possible to choose to restrict the model to those cases where $m < \lfloor \log_2 |S_i|^2 \rfloor$. Furthermore, m should also be large enough to contain the set of possible binary encodings of S_i, *i.e.* $m \geq \lceil \log_2 |S_i| \rceil$. The general boundary conditions are presented in Expression 6.19.

$$\lceil \log_2 |S_i| \rceil \leq m \leq \lfloor \log_2 |S_i|^2 \rfloor \tag{6.19}$$

Figure 6.3 illustrates the three-dimensional plot of Expression 6.16 as a function of a number of initial nodes in the range $[1, 2^{13}]$ alongside the required boundary conditions described by Expression 6.19. The plot presents the characteristic ladder effect associated with employing logarithmic functions in conjunction with functions that map real domains to the integer set. As a consequence, a plateau is reached for some combinations where a number of different cardinality S_i sets can be mapped by the same number of bits, thus presenting the same C/Q ratio. Relinquishing the floor and ceiling functions allows one to obtain a crude comparison between the lower and upper limits of Expression 6.19. More concretely, it is possible to verify that these limits differ by a $\log_2 |S_i|$ factor. This means that the system, besides requiring $\lceil \log_2 |S_i| \rceil$ bits for register $|x\rangle$, can still employ an additional $\log_2 |S_i|$ bits to encode g's output and in the process still perform better than its classical counterpart.

On the growth of g's output

Consider a production system with a constant branching factor where a set of productions is applied then there will exist a total of $|R|^d$ possible tree paths at depth level d, who will require $\lceil \log_2 |R|^d \rceil$ bits for an adequate encoding. Clearly, if $\lceil \log_2 |R|^d \rceil \leq \lceil \log_2 |S_i| \rceil$ then g's output can encode the sequence of productions applied. If this is not the case there is also the possibility to encode an unspecified number of productions applied according to some

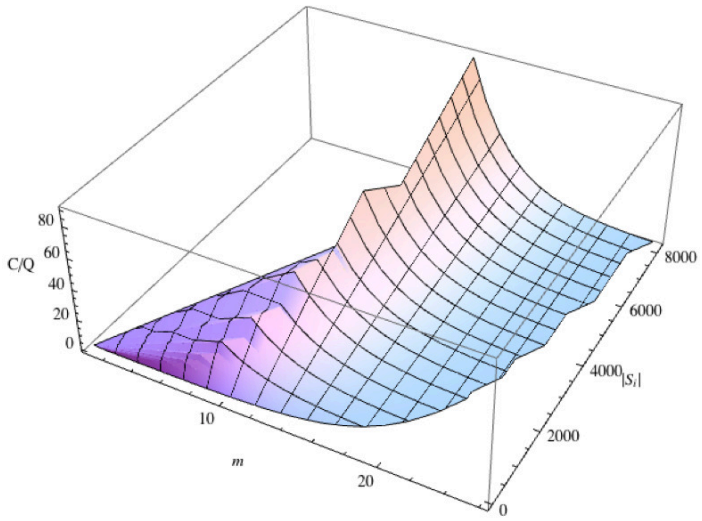


Figure 6.3: The performance measurement ratio C/Q for $|S_i| \in [1, 2^{13}]$ illustrating the logarithmic growth $\lceil \log_2 |S_i| \rceil \leq m \leq \lfloor \log_2 |S_i|^2 \rfloor$ alongside the associated $\frac{p}{\sqrt{2}}\sqrt{|S_i|}$ decrease in performance.

previously chosen strategy. As a consequence, this proposal may be more appropriate when dealing with large S_i sets since this would automatically imply that a large set of working bits would be required. Even if this is not the case it is still possible to employ as initial states the set of nodes that can be found at a depth d.

6.3.3 Comparison with Quantum Finite Automata

Some of the early results regarding quantum finite automata were discussed in [121, 154] and illustrated the consequences of applying complex amplitudes and transitions to the computational procedures. The propose presented in [154] requires: (1) building a system that is configure in a superposition of initial states; (2) performing unitary evolution and (3) measuring the system. Alternatively, the method described in [121] employs multiple measurements in order to determine whether the system is in an accepting, rejecting or non-halting state. This initial research allowed for a number of results that illustrated some significant differences from their classical counterparts, *e.g.:* quantum finite automata can have exponentially less states than classical automata recognizing the same language [13] and any periodic language with period n can be recognized with a quadratic improvement in the number of states [149]. There are clear similarities between the proposed model for a quantum production system and quantum finite automata. Namely, both models share a number of

similar definitions regarding the use of a finite set of states, accepting or rejecting subsets of states, input symbols and transition functions. However, there is a inherent different focus between both approaches since the production system model differentiates itself by opting to encode state transitions in a deterministic way through operator C, which is built in such a way to accommodate for amplitude amplification. This contrasts with the emphasis that is placed on amplitude induced state transitions by the finite automata models.

6.4 Conclusions

In this chapter a quantum computational model based on production system theory was presented. Quantum computation is an inherently reversible process and as a consequence the proposed model would also allow for a reversible decision process. Since production systems share some key characteristics with classical tree search the proposed model also allows for an hierarchical quantum search mechanism. The formalization of the theoretical foundations of this approach allowed for the enumeration of the reversible and quantum requirements. These requirements enabled the development of a method allowing for the construction of the associated unitary operator. This proposition was then extended in order to be combined with Grover's algorithm. Such an approach enabled: (1) an adequate study of the system performance to be performed; and (2) to enumerate those cases in which the model outperforms its classical counterpart. Although this proposition is able to compute faster, the $\frac{p}{\sqrt{2}}\sqrt{|S_i|}$ performance penalty associated with each additional bit required is expensive, favoring the choice of models that rely exclusively on exploiting the class of problems NP through polynomial time verifications and path superpositions.

Chapter 7

Quantum Random Walks on Trees

7.1 Introduction

Some of the graph dynamics considered in this work relate directly to the well-studied quantum random walks on graphs (for an introduction to this research area please refer to [10], [119] and [7]. Quantum random walks were developed as an extension to classical random walks in order to take advantage of quantum effects such as the superposition principle (please refer to [114] [214] for basic facts regarding random walks). Quantum random walks were initially approached in [5], [150], [159] and [12] in one-dimensional terms, *i.e.* walk on a line. The system is described in terms of a position n on the line and a direction d, *i.e.* $|n\rangle|d\rangle$ in the Hilbert space $\mathcal{H} = \mathcal{H}_n \otimes \mathcal{H}_d$ where $\mathcal{H}_n$ is the Hilbert space spanned by the basis vectors encoding the position and $\mathcal{H}_d$ the Hilbert space spanned by the vectors of the direction. The direction register, sometimes referred to as the coin space, is initialized to a superposition of the possible directions, in the case of a walk on a line, either left or right, and the position n updated based on the direction of the walk. The choice of which superposition to apply is also a matter investigated. Surprisingly, if the system is executed for t steps it behaves rather differently than its classical random walks. Specifically, the authors found that the system spreads quadratically faster over the line than its classical equivalent.

The initial approaches proposing quantum random walking on graphs can be found in [68], [101], [4] and [44]. Let $G(V, E)$ represent a d-regular graph, and let $\mathcal{H}_V$ be the Hilbert space spanned by states $|v\rangle$ where $v \in V$, and $\mathcal{H}_E$ be an Hilbert space of dimension d spanned

by basis states $|1\rangle, \cdots, |d\rangle$. Overall, the system can be described through basis states $|v\rangle|e\rangle$ for all $v \in V$ and $e \in E$. The common approach is to randomly select one of the edges e adjacent to v and to update the current position of the graph, *i.e.* $U|v\rangle|e\rangle \rightarrow |v'\rangle|e\rangle$, if e has edge points v and v'. The random selection may also be performed using a d-dimensional coin space. Perhaps more interesting is the hitting time, *i.e.* the time it takes to reach a certain vertex B starting from a vertex A. In [44] and [42] a graph example is presented where a classical random walk would take $\Omega(2^d)$ steps to reach B, where d is the depth of the graph. However, the quantum equivalent walk can reach B in $O(d^2)$ computational steps, providing for an exponential speedup. Other examples of quantum random walks include how to: (1) adapt the models to perform a search (please refer to [190], [7], [14], and [11]) and (2) perform quantum image processing (please refer to [208], [207] and [206]).

Due to the hard computational problems being asked, the vast majority of these approaches put a strong emphasis on actually determining if a node can be reached, and if so how fast. Consequently, questions about the path leading to a pre-specified node have not been properly addressed. For instance, this fact is explicitly pointed out in [42], namely: "Note that although our algorithm finds the name of the exit, it does not find a particular path from entrance to exit". For many artificial intelligence applications the ability to answer this question is a crucial one as it provides the basis for powerful inference mechanisms capable of knowledge deduction. The ability to obtain the full path open measurement is a key feature of the search system proposed in this work. Additionally, quantum random walks incorporate previous knowledge of a graph in the form of the relationship $G(V, E)$. This kind of knowledge may not always be available from start, *e.g.* systems where new nodes are generated based on the information accessible to a system at any given point in time. The quantum random walk approach differs drastically from the model discussed in the previous chapters.

Furthermore, quantum random walks can be employed in order to evaluate binary formulas, which represent a form of tree search. Recall that tree search is a subset of elementary graph theory. 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 [37], where N is the number of nodes in the tree. Ambainis proved that computing an AND of ORs requires time $\Omega(\sqrt{N})$ [6]. In addition, every NAND formula of size N can be evaluated in time $N^{\frac{1}{2}+o(1)}$ [40]. Ambainis then presented an $O(\sqrt{N})$ discrete query quantum algorithm for evaluating balanced binary NAND formulas [8], which was also proven to be optimal [9]. In [67] 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 [43] requiring $N^{\frac{1}{2}+o(1)}$ queries. These results were later employed to demonstrate how to evaluate minmax

trees with $N^{\frac{1}{2}+o(1)}$ queries [50].

The following sections review and describe how to adapt existing quantum search methods and are organised as follows: Section 7.2 presents the formal definition of the problem; Section 7.3 presents the main results regarding quantum random walks on graphs and extends the existing framework such that the computational path is obtained; Section 7.4 compares both methods and illustrates the key differences between them from a tree search perspective; Section 7.5 provides some additional insight regarding the dynamics and impacts associated with set E; the overall conclusions of this chapter are presented in Section 7.6.

7.2 Problem

Tree search can be seen as a subset of elementary graph theory, *i.e.* an acyclic connected graph where each node has zero or more children nodes and at most one parent node. 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 [192]. 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 [52]. 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 also focus on the specific details regarding possible sequences of edges.

One of the most important tasks resides 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 7.1, 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 [112], solving Rubik's cube, or

developing general mechanisms for problem solving [130].

$$P := ((x_1, x_2), (x_2, x_3), \cdots, (x_{d-1}, x_d) | x_k \in V, x_1 \in S_i, x_d \in S_g) \tag{7.1}$$

This procedure is illustrated in Figure 7.1, where vertex A is the initial state and vertex E the goal state. Accordingly, possible examples of computational paths leading from A to E include the sequences of edges $((A, B), (B, E))$ and $((A, C), (C, D), (D, E))$.

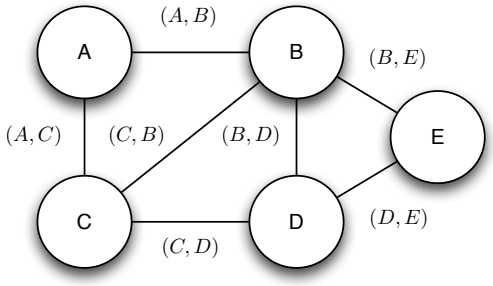


Figure 7.1: A graph with five vertexes and seven edges.

Graph search procedures based on quantum random walks only focus on obtaining goal states, and do not take into account 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? Namely, how can quantum random walks be adapted such that P is obtained? More specifically, what are the mechanisms available, alongside respective requisites and limitations. In addition, it would also be important to determine how such an approach would fare against our original proposition for performing tree search based on Grover's algorithm, not only from a computational perspective but also performance-wise.

7.3 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 [119]. As a result, the results presented in this chapter will focus on discrete-time quantum random

walks since they allow for a simple mapping between previously discussed concepts. The following sections are organised as follows: Section 7.3.1 provides the necessary details for understanding quantum random walks on graphs; Section 7.3.2 describes how to extend these concepts in order to store the computational path performed.

7.3.1 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" [190]. 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 7.2 [4], 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 [41]: (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 $(j, k) \in E$.

$$U = SC \tag{7.2}$$

Operator C is responsible for building a superposition in register $|k\rangle$ spanning the neighbours of j. This behaviour is presented in Expression 7.3 (adapted from [211]), where $deg(j)$ represents the degree of vertex j, *i.e.* the number of edges at vertex j [192]. 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 $\sum_{m:(j,m)\in E} \frac{1}{deg(j)} = 1$.

$$C|j\rangle|k\rangle \rightarrow |j\rangle \frac{1}{\sqrt{deg(j)}} \sum_{m:(j,m)\in E} |k \oplus m\rangle \tag{7.3}$$

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 7.4. The overall effect of Expression 7.2 emphasizes the notion that the quantum random walk takes places on the edges of the graph.

$$S|j\rangle|k\rangle \rightarrow |k\rangle|j\rangle \tag{7.4}$$

In [190] 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 membership for a given language $\mathcal{L}$. 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 coin operator in order to perform Grover's diffusion operator [82], which was first proposed in [153]. The reformulated coin operator C is presented in Expression 7.5.

$$C|j\rangle|k\rangle \rightarrow |j\rangle \left(\frac{2}{\sqrt{deg(j)}} \sum_{m:(j,m)\in E} |k \oplus m\rangle\langle k| - \mathbb{I} \right) \tag{7.5}$$

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 7.6. 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 [199]. 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 [190]. However, as the authors point out, $|\psi_f\rangle$ also possesses small contributions from the closest neighbours to the solution state [190]. The procedure is concluded by performing a measurement on the quantum register, yielding with high probability a solution state.

$$|\psi_i\rangle = \frac{1}{\sqrt{|V|}} \sum_{v\in V} |v\rangle \frac{1}{\sqrt{deg(v)}} \sum_{m:(v,m)\in E} |m\rangle \tag{7.6}$$

7.3.2 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 transi-

tions. 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 7.7 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 $m_t \oplus (j, k)$.

$$R_t|j,k\rangle|m_0, m_1, \cdots, m_d\rangle \rightarrow |j,k\rangle|m_0, m_1, \cdots, m_t \oplus (j,k), \cdots, m_{d-1}\rangle \tag{7.7}$$

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 [118]. 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, the introduction of operator R_t also requires that index t is properly updated. The overall operator sequence for a quantum walk requiring d steps is described in Table 7.1. 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 7.1: Operator sequence for the modified quantum random walk search algorithm.

7.4 Comparison against the quantum production system model

Although both quantum randoms walks and the quantum production system 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, how do these functions compare against each other? First, it is important to analyse

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, Grover's algorithm performance can be viewed as the standard to which to compare. This means that random walks will perform an additional effort relatively to the quantum search algorithm. Accordingly, it is possible to represent the supplementary effort performed as a function of the branching factor b and depth d, respectively $\xi(b,d)$, as illustrated in Expression 7.8. For $d > 0, \xi(b,d) > 1$ since the last term of the series is $\frac{b^d}{b^d} = 1$.

$$\xi(b,d) = \frac{\sum_{k=0}^{d} b^k}{b^d} = \frac{1}{b^d}\frac{1-b^{d+1}}{1-b} \tag{7.8}$$

Figure 7.2 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 $b^d = 2^d$ paths, 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.

Expression 7.8 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 Expression 7.2 is perceived as 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 7.9, where $p \in \mathbb{N}$ is chosen so that $d - p \geq 0$.

$$\xi(b,d,p) = \frac{\sum_{k=d-p}^{d} b^k}{b^d} = \frac{b^{d-p} + b^{d-p+1} + \cdots + b^{d-1} + b^d}{b^d} = \frac{1}{b^p} + \frac{1}{b^{p-1}} + \cdots + \frac{1}{b^1} + \frac{1}{b^0} \tag{7.9}$$

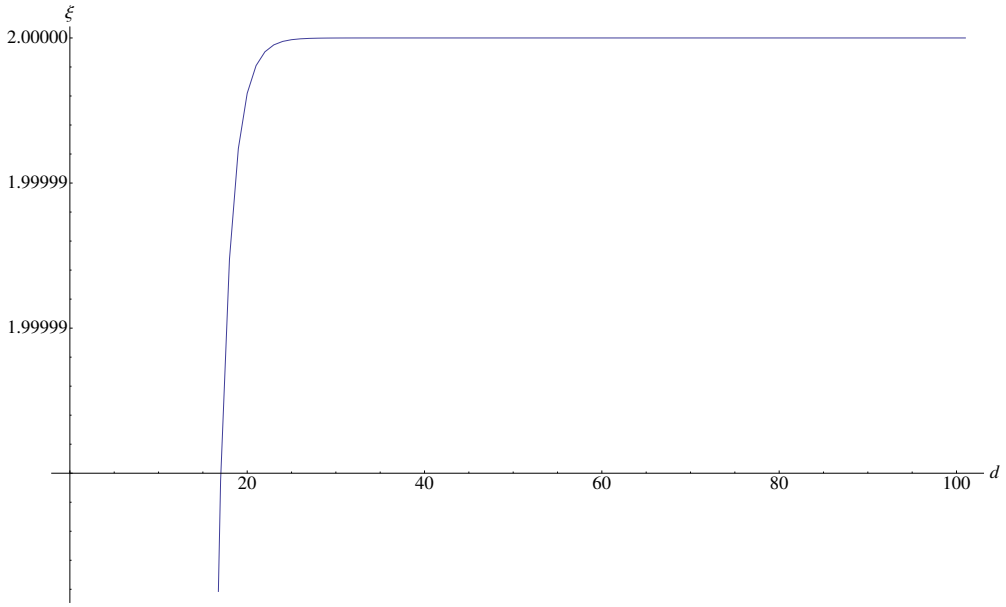


Figure 7.2: The additional effort performed by quantum random walks relatively to Grover's algorithm when searching a binary tree up to depth level 100.

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 7.2 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.

7.5 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, picture 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 6.1, 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 performs 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 $\gamma \in R$.

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 1.1. Suppose that node K in Figure 1.1 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 one can 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, it is possible 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 7.10.

$$P := (((x_1, x_2), \gamma_1), ((x_2, x_3), \gamma_2), \cdots, ((x_{d-1}, x_d), \gamma_d) | x_k \in V, x_1 \in S_i, x_d \in S_g, \gamma_i \in R) \tag{7.10}$$

Expression 7.10 can be simplified since it is a simple task to determine a successor state y given an input state x alongside a rule γ. Accordingly, it suffices 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 7.11.

$$P := (\gamma_1, \gamma_2, \cdots, \gamma_d | \gamma_i \in R) \tag{7.11}$$

Although both approaches achieve the same functional purposes of determining P they differ substantially on the costs required for obtaining the computational path. Namely, the use of rules allows one 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.

7.6 Conclusions

This chapter described how to adapt quantum random walks in order to store the paths leading to solutions states. Additionally, the results presented also described that the best result one should expect, 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.

The quantum production system method 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 state of the superposition represents individual computational paths.

Accordingly, with the current formulation 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.

Chapter 8

Quantum Iterative Deepening with an application to the Halting problem.

8.1 Introduction

Classically, the status of any computation can be determined through a halt state. The concept of the halting state has some important subtleties in the context of quantum computation. Ekert draws attention to this fact stating that there are two possibilities to circumvent such an issue, namely [65]: either run the computation for some predetermined number of steps or alternatively employ a halt flag. This flag is then employed by a computational model to signal an end of the calculation. Traditionally, such a flag is represented by a halt bit, which is initialized to 0 and set to 1 once the computation terminates. Accordingly, determining if a computation has finished is simply a matter of checking if the halt bit is set to 1, a task that can be accomplished through some form of periodic observation.

Furthermore, there is a class of computational problems, respectively undecidable problems (such as the famous *Entscheidungsproblem* challenge proposed by Hilbert in [97]), for which no solution can be found no matter how much time and space is provided to the algorithmic process. As a result, any computational model that is employed in order to try to solve these problems require the ability to proceed indefinitely. Classically, the recurrent observation of a halt bit that is required can be performed without affecting the overall result of the calculation. Undecidable problems are important because they demonstrate the existence of

a class of problems that does not admit an algorithmic solution no matter how much time or spatial resources are provided. This result was first demonstrated by Church [47] and shortly after by Turing [205].

8.1.1 Problem

Deutsch [57] was the first to suggest and employ such a strategy in order to describe a quantum equivalent of the Turing machine, which employs a compound system $|r\rangle$ expressed as a tensor of two terms, *i.e.* $|r\rangle = |w\rangle|h\rangle$, spanning a Hilbert space $\mathcal{H}_r = \mathcal{H}_w \otimes \mathcal{H}_h$. The component $|w\rangle$ represents a work register of unspecified length and $|h\rangle$ a halt qubit, which is used in an analogous fashion to its classical counterpart. However, Deutsch's strategy turned out to be flawed, namely suppose a unitary computational procedure C acting on input $|x\rangle$ is applied d times and let $d_{C,x}$ represent the number of steps required for a procedure C to terminate on input x. Then it may be possible that there exist i and j for which $d_{C,i} < d < d_{C,j}, \forall i \neq j$. Now, consider what happens when such a behaviour is observed and $|w\rangle$ is initialized as a superposition of the computational basis. Then those states which only require a number of computational steps less than or equal to d in order to terminate will have the halt qubit set to $|1\rangle$, whilst the remaining states will have the same qubit set to $|0\rangle$. This behaviour effectively results in the overall superposition state $|w\rangle|h\rangle$ becoming entangled as exemplified by Expression 8.1, where n represents the number of bits used by $|w\rangle$.

$$\underbrace{\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} C^d|x\rangle}_{|\psi\rangle} |0\rangle = \begin{cases} |\underbrace{00\cdots0}_{\text{n bits}}\rangle|0\rangle & \implies d_{C,00\cdots0} > d \\ |00\cdots1\rangle|1\rangle & \implies d_{C,00\cdots1} \leq d \\ \quad\vdots & \\ |11\cdots0\rangle|1\rangle & \implies d_{C,11\cdots0} \leq d \\ |11\cdots1\rangle|0\rangle & \implies d_{C,11\cdots1} > d \end{cases} \tag{8.1}$$

More generally, suppose that the compound system after the unitary evolution C^d is in the entangled state represented by the right-hand side of Expression 8.2. Also, assume that the probability of observing the halting qubit $|h\rangle$ with outcome k is $P(k) = \sum_{x=0}^{2^n-1} |\alpha_{x,k}|^2$. The projection postulate implies that a post observation state of the whole system is obtained as the one illustrated in Expression 8.3, where the system is projected to the subspace of the halting register and renormalized to the unit length [99].

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} C^d|x\rangle|0\rangle = \sum_{x=0}^{2^n-1} \sum_{j=0}^{1} \alpha_{x,j}|x\rangle|j\rangle \tag{8.2}$$

$$\frac{1}{\sqrt{P(k)}} \sum_{x=0}^{2^n-1} \alpha_{x,k}|x\rangle|k\rangle \tag{8.3}$$

Consequently, observing the halt qubit after d computational steps have been applied, will result in the working register containing either: (1) a superposition of the non-terminating states; or (2) a superposition of the halting states. Such behaviour has the to dramatically disturb a computation since: (1) a halting state may not always be obtained upon measurement due to random collapse, if indeed there exists one; and (2) any computation performed subsequently using the contents of the working register $|w\rangle$ may employ an adulterated superposition with direct consequences on the interference pattern employed. Roughly speaking, there is no way to know whether the computation is terminated or not without measuring the state of the machine, but, on the other hand, such a measurement may disturb the current computation.

8.1.2 Current approaches to the quantum halting problem

Ideally, one could argue that any von Neumann measurement should only be performed after all parallel computations have terminated. Indeed, some problems may allow one to determine $\max d_{C,|x\rangle}, \forall|x\rangle \in |\psi\rangle$, *i.e.* an upper-bound $d_{C,x}$ on the number of steps required for every possible input x present in the superposition. However, this procedure is not viable for those problems which, like the *Entscheidungsproblem*, are undecidable. Bernstein and Vazirani subsequently proposed a model for a universal quantum Turing machine in [27], which did not incorporate into its definition the concept on non-termination. Although their model is still an important theoretical contribution it is nonetheless only capable of dealing with computational processes whose different branches halt simultaneously or fail to halt at all. These same arguments were later employed by Myers in [158] who argues that it is not possible to precisely determine for all functions that are Turing-computable, respectively μ-recursive functions, the number of computational steps required for completion. Additionally, the author also states that the models presented in [57] and [27] cannot be qualified as being truly universal since they do not allow for non-terminating computation. The work described in [27] is also restricted to the class of quantum Turing machines whose computational paths are synchronized, *i.e.* every computational path is synchronized in the sense that they must each reach an halt state at the same time step. This enabled the authors to sidestep the halting problem.

Following Myers observation of the conflict between quantum computation and system observation a number of authors provided meaningful contributions to the question of halting in quantum Turing machines. Ozawa [167] [166] proposed a possible solution based on quantum nondemolition measurements, a concept previously employed for gravitational wave detection. Linden [139] argued that the standard halting scheme for Turing machines employed by Ozawa is unitary only for non-halting computations. Additionally, the author described how to build a quantum computer, through the introduction of an auxiliary ancilla bit that enabled system monitoring without spoiling the computation. However, such a scheme introduced difficulties regarding different halting times for different branches of computation. These restrictions essentially rendered the system classical since no useful interference occurred. In [168] expands the halting scheme described in [167] in order to introduce the notion of a well-behaved halting flag, which is not modified upon completion. The author showed that the output probability distribution of monitored and non-monitored flags is the same. Miyadera proved that no algorithm exists capable of determining if an arbitrarily constructed quantum Turing machine halts at different computational branches [152]. Iriyama discusses halting through a generalized quantum Turing machine that is able to evolve through states in a non-unitary fashion [117].

Measurement-based quantum Turing machines as a model for computation were defined in [171] and [170]. Perdrix explores the halting issue by introducing classically-controlled quantum Turing machines [173], in which unitary transformations and quantum measurements are allowed, but restricts his model to quantum Turing machines that halt. Muller shows the existence of a universal quantum Turing machine that can simulate every other quantum Turing machine until the simulated model halts, which then results in the universal machine halting with probability one [156, 157]. The author describes operators that do not disturb the computation as long as the original input employed halts the calculation process. This requires presenting a precise definition of the concept of halting state, resulting in parts of the domain being discarded since some of those requirements are not met.

In [61] a method is presented for verifying the correctness of measurement-based quantum computation in the context of the one-way quantum computer described in [180]. This type of quantum computation differs from the traditional circuit based approach since one-qubit measurements are performed on an entangled resource labeled as a cluster state in order to mold a quantum logic circuit on the state. With each measurement the entanglement resource is further depleted. These results are further extended in [181] in order to prove the universality of the computational model. Subsequently, in [34] these concepts were used in order to prove that one-way quantum computations have the same computational power as quantum circuits with unbounded fan-out. Perdrix [172] discusses partial observation of quantum Turing machines, which preserve the computational state through the introduction

of a weaker form of the original requirements of linear and unitary δ functions suggested by Deutsch in [57]. Recently, [151] proved that measurements performed on the (X, Z)-plane of the Bloch sphere over graph states is a universal measurement-based model of quantum computation.

The topic of quantum halting has also been approached from the perspective of a Markov chain model of concurrent quantum programs (please refer to [218, 216]). In addition, such an approach also allows for a characterization of reachable space and uniformly repeatedly reachable space. The authors derive a quantum version of a classical model of probabilistic concurrent programs presented in [93]. In classical concurrent program execution, each procedure is perceived as executing in sequence for a given amount of time. As a result, no two processes ever execute simultaneously [93]. The quantum extension is also developed with this type of sequential program execution in mind. Therefore, the question of how to represent parallel program execution through a superposition state does not arise. This behaviour allows for a measurement to be performed after each execution step which decided whether or not the computation should proceed without disturbing the computation. In [217] the authors propose a method to remove multiple measurements from quantum random walk which requires: (1) extending the Hilbert space by adding a register for step counting; and (2) modifying the evolution operator in order to accommodate for such an extension. Afterwards, the authors show that the extended version of the walk not only shares the original concurrent hitting time but also a one shot hitting time at specific steps. This behaviour is employed to produce an amplitude amplification technique that delivers a quadratic speedup over the original walk with success probability $O(1)$.

8.1.3 Objectives

In his seminal paper [57], Deutsch emphasizes that a quantum computer needs the ability to operate on an input that is a superposition of computational basis in order to be "fully quantum", When confronted with the halting issue Myers naturally raised the question if a universal quantum computer could ever be fully quantum? And how would such a computational model eventually function? This chapters aims to provide an answer to these questions by extending the quantum production system previously described. This extension is employed in order to gain additional insight into the matter of halting and universal computation from a different perspective than that of the standard quantum Turing machine.

As Miyadera stated, the notion of probabilistic halting in the context of quantum Turing machines cannot be avoided, suggesting that the standard halting scheme of traditional quantum computational models needs to be reexamined [152]. The proposal described in

this chapter is essentially different from the ones previously discussed since it imposes a strict notion of how the computation is performed and progresses in the form of the sequence of instructions that should be applied. The proposed method evaluates d-length sequences of instructions representing different branches of computation, enabling one to determine which branches, if they exist, terminate the computation. Underlying the proposed model will be Grover's algorithm in order to amplify the amplitude of potential halting states, if such states exist, and thus avoiding obtaining a random projection upon measurement. As a result, the computational complexity associated with such a model will be described in order to show that it does not differ from that of Grover's algorithm.

The work described in this chapter focuses on: (1) preserving the original principles proposed by Deutsch of linearity and unitary operators, in contrast with other proposals such as [172] and [117], which perform modifications to the underlying framework; (2) developing a model which considers all possible computational paths and (3) works independently of whether the computation terminates or not taking into account each possible computational path. Additionally, some of the implications of being able to circumvent the halting problem will also be considered. Computation universality is a characteristic attribute of several classical models of computation. For instance, the Turing machine model was shown to be equivalent in power to lambda calculus and production system theory. Accordingly, it would be interesting to determine what aspects of such a relationship are maintained in the context of quantum computation. Namely, it would be interesting to determine if it is possible to simulate a classical Turing machine given a quantum production system.

8.1.4 Organisation

The ensuing sections are organised as follows: Section 8.2 presents the details of a model capable of dealing with the halting issue; Section 8.3 demonstrates how such a model can be employed in order to coherently simulate a classical Turing machine. The conclusions are presented in Section 8.4.

8.2 Quantum Iterative Deepening

Universal models of computation are capable of calculating μ-recursive functions, a class of functions which allow for the possibility of non-termination. These functions employ a form of unbounded minimalization, respectively the μ-operator, which is defined in the following terms [138]: let $k \geq 0$, $c \in \mathbb{N}$, $m \in \mathbb{N}$ and $g \colon \mathbb{N}^{k+1} \rightarrow \mathbb{N}$, then the unbounded minimization of g is function $f \colon \mathbb{N}^{k+2} \rightarrow \mathbb{N}$ as illustrated in Expression 8.4, for any $\bar{n} = n_1, \cdots, n_k \in \mathbb{N}^k$.

$$f(g, \bar{n}, c) = \begin{cases} \text{the least } m \text{ such that } g(\bar{n}, m) = c & \text{, if such an } m \text{ exists} \\ 0 & \text{, otherwise} \end{cases} \tag{8.4}$$

The unbounded minimization operator can be perceived as a computational procedure responsible for repeatedly evaluating a function with different inputs m until a target condition $g(\bar{n}, m) = c$ is obtained [195]. However, as illustrated by Expression 8.4, there is no guarantee that the target condition will ever be met. Accordingly, it is possible to express the inner-workings of f as an iterative search that may never terminate, as illustrated in Algorithm 1. Notice that although μ-recursive functions employ a collections of variables belonging to the set of natural numbers, for practical purposes these values are restricted by architecture-specific limits on the number of bits available for representing the range of possible values.

Algorithm 1 The classical μ-operator (adapted from [195])

```
1: function f(g, n̄, c)
2:     m ← 0
3:     while g(n̄, m) ≠ c do
4:         m ← m + 1
5:     return m
```

From a quantum computation perspective, it is possible to perform a generic search for solution states through amplitude amplification schemes such as the one described by Grover in [82] and [80]. This section discusses how to combine production system theory alongside the quantum search algorithm in order to develop a new computational model better suited to deal with the halting issue.

The next sections are organised in the following manner: Section 8.2.1 proposes an oracle formulation of a the quantum production; Section 8.2.2 focuses on how to integrate these components into a single unified approach for a computational model based on production system theory capable of proceeding indefinitely without affecting the overall result of the computation; Section 8.3 presents a simple mapping mechanism of how the method described can be used to simulate a classical Turing machine.

8.2.1 Quantum Production System Oracle

A comparison of Expression 8.5 and Expression 3.27 allows one to reach the conclusion that oracle O performs a verification whilst C focuses on executing an adequate state evolution. Therefore, an alternate mechanism needs to developed that behaves as if performing a verification. This can be done by focusing on one of the main objectives of production system

theory, namely that of determining the sequence of production rules leading up to a goal state. Formally, it is of particular interest to establish if an initial state $i \in S_i$ alongside a sequence of d productions rules $\{r_1, r_2, \cdots, r_d\} \in R$ leads to a goal state $g \in S_g$. If the sequence of rules leads to a goal state, then the computation is marked as being in a halt state h, otherwise it is flagged to continue c. It is therefore possible to proceed with a redefinition of the control function presented in Expression 6.5, as illustrated in Expression 8.5, which closely follows the oracle definition presented in Expression 3.27.

$$C : \Gamma^* \times R^d \times \{h, c\} - \mathbb{C} \tag{8.5}$$

Recall that the oracle operator is applied to register $|r\rangle = |w\rangle|h\rangle$. Register $|w\rangle$ is represented as a tensor of two products, namely $|w\rangle = |s\rangle|p\rangle$, where $|s\rangle$ is responsible for holding the binary representation of the initial state and $|p\rangle$ contains the sequence of productions. Register $|h\rangle$ is used in order to store the status s of the computation. Additionally, the revised version of the quantum production system C with oracle properties should also maintain a unit-norm, as depicted by Expression 8.6, $\forall\gamma \in \Gamma^*$. For specific details surrounding the construction of such a unitary operator please refer to [198].

$$\sum_{\forall(r_1, r_2, \cdots, r_d, s) \in R^d \times \{h,c\}} |C(\gamma, r_1, r_2, \cdots, r_d, s)|^2 = 1 \tag{8.6}$$

Any computational procedure can be described in production system theory by specifying an appropriate set of production rules that are responsible for performing an adequate state evolution. This set of production rules can be applied in conjunctions with a unitary operator C incorporating the behaviour mentioned in Expression 8.5 and Expression 8.6. In doing so it is possible to obtain a derivation of a production system that can be combined with Grover's algorithm. From a practical perspective, $|p\rangle$ can be initialized as a superposition over a set $P_{R,d}$ representing the sequence of all possible production rules $\in R$ up to a depth-level d, as illustrated by Expression 8.7 and Expression 8.8. Implicit to these definitions is the assumption that set P has a total of b^d possible paths.

$$P_{R,d} := \{\text{sequence of all possible production rules} \in R \text{ up to a depth-level } d\} \tag{8.7}$$

$$|p\rangle = \frac{1}{\sqrt{b^d}} \sum_{\forall x \in P_{R,d}} |x\rangle \tag{8.8}$$

Traditionally, throughout a computation set S_i remains static in the sense that it does not

grow in size. However, variable d is constantly increased in order to generate search spaces covering a larger number of states. As a result, given a sufficiently large depth value the number of bits required for $P_{R,d}$ will eventually surpass the amount of bits required to encode set S_i. Accordingly, in the reasonable scenario where the number of bits required to encode the sequence of productions over $P_{R,d}$ is much larger than the number of bits required to encode the set of initial states S_i, *i.e.* $\log_2 |P_{R,d}| \gg \log_2 |S_i|$, then the most important factor to the dimension of the search space will be the number of productions.

8.2.2 General procedure

Any approach to a universal model of quantum computation needs to focus on two main issues, namely: (1) how to circumvent the halting problem and (2) how to handle computations that do not terminate without disturbing the result of the procedure. The following sections describe the general procedure. Initially, Section 8.2.2 will focus on the second requirement given that it provides a basis for model development by establishing the parallels between μ-theory and production system theory. Section 8.2.2 then describes how these arguments can be utilized in order to develop a computational model capable of calculating μ-recursive functions. Section 8.2.2 describes how such a proposal is essentially non-different, complexity-wise, from the original Grover algorithm employed.

Parallels between μ-theory and production system theory

Universal computation must allow for the possibility of non-termination, a characteristic that is achievable through the ability to calculate μ-recursive functions. Therefore, the question naturally arises if it is possible to develop a quantum analogue of the iterative μ-operator? By itself μ-recursive functions are not seen as a model of computation, but represent a class of functions that can be calculated by computational models. Accordingly, it would be interesting to determine if it is possible to develop a quantum computational model, namely by employing the principles of production system theory, capable of calculating μ-recursive functions without affecting the end result.

In order to answer this question it is important to first start by establishing some parallels between these concepts. Namely, consider the μ-operator presented in Algorithm 1 that receives as an argument a tuple $(g, \bar{n}, c)$ and a production system defined by the tuple (Γ, S_i, S_g, R, C). Accordingly, parameter g can be perceived as a control strategy C responsible for mapping a set of symbols Γ in accordance with a set of rules R. Variable $\bar{n}$ can be interpreted as an element of the set of initial states, *i.e.* $i \in S_i$. The target condition c can be understood as the set of goal states S_g. In addition, the unbounded minimization operator

employs a parameter m that represents the first argument where the target condition is met. Analogously, from a production system perspective, variable m can be viewed as the first depth d where a solution to the problem can be found. Finally, the condition $g(\bar{n}, m) \neq c$ of the while loop is equivalent to applying the control strategy C at total of d times, *i.e.* C^d, and evaluating if a goal state was reached.

Iterative Search

The fact that such mappings can be performed hints at the possibility of being able to develop a quantum equivalent of the μ-operator based on production system fundamentals. All that is required is a while loop structure, mimicking the iterative behaviour of the μ-operator, that exhaustively examines every possibility for d alongside C, until a goal state is found. The quantum production system oracle presented in Expression 8.5 can be combined alongside Grover's iterate for a total of $\sqrt{b^d}$ times in order to evaluate a superposition of all the available sequences of productions up to depth-level d, *i.e.* $P_{R,d}$. After applying Grover's algorithm, a measurement M can be performed on the superposition. If the state ξ obtained is a goal state, then the computation can terminate since a solution was found at depth d.

This process is illustrated in Algorithm 2, which receives as an argument a tuple (Γ, i, S_g, R, C), where i is an initial state, *i.e.* $i \in S_i$. The procedure is represented as a form of pseudocode that is in accordance with the conventions utilized in [52], namely: (1) indentation indicates block structure, *e.g.* the set of instructions of the while loop that begins on line 5 consists of lines 6-14; (2) the symbol $\leftarrow$ is used to represent an assignment of a variable; and (3) the symbol $\triangleright$ indicates that the remainder of the line is a comment.

Line 8 is responsible for applying the oracle alongside an initial state and all possible sequences of productions. Recall that register $|h\rangle$ will be set if goal states can be reached. Line 9 is responsible for applying Grover's algorithm. If goal states are present in the superposition, then Grover's amplitude amplification scheme allows for one of them to be obtained with probability $|\sin[\frac{\theta}{2}(\frac{\pi}{2}\sqrt{\frac{b^d}{k}}+1)]|^2$ [164], where k represents the number of solutions and $\theta = 2\arccos(\sqrt{\frac{b^d-k}{b^d}})$. It is possible that state $|\psi_2\rangle$ contains a superposition of solutions. Therefore, measuring the system in Line 10 will result in a random collapse amongst these. If the measurement returns an halt state, then register $|p\rangle$ will contain a sequence of productions leading to a goal state. Once the associated sequence has been obtained one has only to apply each production of the sequence in order to determine precisely what was the goal state obtained [198] (Line 11). Otherwise, the search needs to be expanded to depth level $d+1$ and the production evaluation process repeated from the start. As a result, this

Algorithm 2 Quantum Iterative Deepening

```
1: function f(Γ, i, S_g, R, C)
2:     d ← 0
3:     ξ ← ∅
4:     |s⟩ ← i
5:     while true do
6:         |p⟩ ← 1/√(b^d) Σ_{∀x∈P_{R,d}} |x⟩      ▷ Build superposition of productions
7:         |h⟩ ← 1/√2 (|0⟩ − |1⟩)
8:         |ψ_1⟩ ← C^d |s⟩|p⟩|h⟩                  ▷ Mark if goal states exist at depth d
9:         |ψ_2⟩ ← G^{√(b^d)} |ψ_1⟩               ▷ Apply Grover's iterate
10:        ξ ← M|ψ_2⟩                            ▷ Measure the superposition
11:        if ξ ∈ S_g
12:            return ξ                          ▷ If a goal state was found terminate
13:        else
14:            d ← d + 1                         ▷ Otherwise, continue searching
```

procedure requires building a new superposition of productions $P_{R,d+1}$ each time a solution was not found in $P_{R,d}$.

Due to the probabilistic nature of Grover's algorithm there is also the possibility that the measurement will return a non halting state, even though $|\psi_2\rangle$ might have contained sequences of productions that led to goal states. This issue can be circumvented to a certain degree. Notice that the sequences expressed by $P_{R,d+1}$ also contain the paths $P_{R,d}$ as subsequences. This means that when $P_{R,d+1}$ is evaluated the iteration procedure has the opportunity to re-examine $P_{R,d}$. As a result, operator C would have the chance to come across the exact subsequences that had previously led to goal states but that were not obtained after the measurement. Therefore, the control strategy would need to be modified in order to signal an halt state as soon as a solution is found, *i.e.* the shallowest production, independently of the sequence length being analyzed. With such a strategy the probability of obtaining a non-halting state in each unsought iteration level d would be $1-|\sin[\frac{\theta}{2}(\frac{\pi}{2}\sqrt{\frac{b^d}{k}}+1)]|^2$.

Each iteration of Algorithm 2 starts by building a superposition $|p\rangle$ spanning the respective depth level. This means that the original interference pattern that was possibly lost upon measuring the system in the previous iteration is rebuilt and properly extended by the tensor product that is performed with the new productions. Because of this process the computation is able to proceed as if undisturbed by the measurement. Such a reexamination comes at a computational cost, which will be shown to be neglectable in Section 8.2.2. This behaviour contrasts with the original approach discussed by Deutsch where: (1) a computation would be applied to a superposition $|\psi\rangle$; (2) a measurement would eventually be made on the halt qubit collapsing the system to $|\psi\rangle'$ and (3) if a goal state had not been obtained the computation would proceed with $|\psi\rangle'$.

Complexity Analysis

Algorithm 2 represents a form of iterative deepening search, a general strategy employed alongside tree search, that makes it possible to determine an appropriate depth limit d, if one exists [183]. The first documented use of iterative deepening in the literature is in Slate and Atkin's Chess 4.5 program [193], a classic application of an artificial intelligence problem. Notice that up until this moment no upper-limit value for depth d was specified. This was done deliberately since the essence of μ-recursive functions relies in the fact that such a value may not exist. In general, iterative deepening is the preferred strategy when the depth of the solution is not known [183]. Accordingly, the while loop will execute forever unless the state ξ in line 11, obtained after the measurement, is a goal state.

Since Grover's algorithm is employed the measurement performed does not necessarily need to be applied to the halting register. Instead it is possible to perform a measurement on the entire Hilbert space of the system in order to verify if a final state is obtained. This type of a control structure is responsible for guaranteeing the same type of partial behaviour that can be found on the classical μ-operator. Consequently, Algorithm 2 also does not guarantee that variable d will ever be found, *i.e.* the search may not terminate. Line 8 of the algorithm uses the register $|r\rangle = |w\rangle|h\rangle = |s\rangle|p\rangle|h\rangle$ described in Section 8.2.1.

Quantum iterative deepening search may seem inefficient, because each time C^d is applied to a superposition spanning $P_{R,d}$, the states belonging to previous depth levels multiple times, $\forall d > 0$, are being re-evaluated. However, the bulk of the computational effort comes from the dimension of the search space to consider, respectively b^d, which grows exponentially fast. As pointed out in [124] if the branching factor of a search tree remains relatively constant then the majority of the nodes will be in the bottom level. This is a consequence of each additional level of depth adding an exponentially greater number of nodes. As a result, the impact on performance of having to search multiple times the upper levels is minimal. This argument can be stated algebraically by analysing the individual time complexities associated with each application of Grover's algorithm for the various depth levels. Such a procedure is illustrated in Expression 8.9, which gives an overall time complexity of $O(\sqrt{b^d})$ remaining essentially unchanged from that of the original quantum search algorithm.

$$\sqrt{b^0} + \sqrt{b^1} + \sqrt{b^2} + \cdots + \sqrt{b^d} = O(\sqrt{b^d}) \tag{8.9}$$

By employing the previously described proposal it becomes possible to develop a quantum computational model with an inherent speedup relatively to its classical counterparts. Notice that this speedup is only obtained when searching through a search space with a branching factor of at least 2 (please refer to [199] [198]). In addition, if the set of goal states is de-

fined to be the set of halt states, then Algorithm 2 can be used to circumvent the halting problem. The algorithm is able to do so since it can compute a result without the associated disruptions of Deutsch's model. Such a term is employed carefully, since it may be argued that the measurements performed during computation will inherently disturb the superposition. This is not a problem if a halt state is found. However, if such a goal state is not discovered, the algorithm proceeds to an extended superposition through $P_{R,d}$, representing an exponentially greater search space, where the states from the previous tree levels are included. Consequently, it becomes possible to recalculate the computation as if it had not been disturbed and without changing the overall complexity of the procedure.

8.3 Turing machine simulation

The approach proposed in this chapter allows for the possibility of non-termination, without inherently interfering with the results of the quantum computation. This hints at the possibility that the method described can be applied to coherently simulate classical universal models of computation such as the Turing machine. Specifically, what needs to be done for the previously described iterative quantum production system to simulate any classical Turing machine?

In order to answer this question it is useful to present a set of mappings between Turing machine and production system concepts in a manner analogous to the trivial mapping described in [75]. Both models employ some form of memory where the current status of the computation is stored. The Turing machine model utilises a tape capable of holding symbols. Each element of the tape can be referred to through a location. Tape elements are initially configured in a blank status, but their contents can be accessed and modified through primitive read and write operations. These operations are performed by a head that is able to address each element of the tape. As a result, the memory equivalent of the production system, respectively, the working memory should convey information regarding the current head position and the symbols, alongside the respective locations, on the tape. In addition, the tape employed in Turing's model has an infinite dimension. Consequently, the working memory must also possess an infinite character.

The Turing machine model utilises a δ function to represent finite-state transitions. The δ functions maps an argument tuple containing the current state and the input being read to tuples representing a state transition, an associated output and some type of head movement. This set of transitions can be represented as a table whose rows correspond to some state and where each column represents some input symbol. Each table entry contains the associated transition tuple representing the next internal state, a symbol to be written, and a head

movement. Notice, that this behaviour fits nicely into the fixed set of rules R employed by production systems. Namely, δ's argument and transition tuples can be seen, respectively, as a precondition and associated action of a certain rule. Accordingly, for each table entry of the original Turing transition function it is possible to derive an adequate production rule, thus enabling the obtention of R.

The only remaining issue resides in defining a control strategy C that mimics the behaviour presented in Expression 8.5. Consequently C needs to choose which of the rules to apply by accessing the working memory, determining the element that is currently being scanned by the head, and establishing if a goal state is reached after applying some specific sequence of R^d rules. Once this is done, it becomes possible to apply the iterative quantum production system to simulate the behaviour of a classical Turing machine. The δ-function conversion to an adequate database of productions is a simple polynomial-time procedure (please refer to [2] and [189] for additional details). In addition, it is important to mention that this approach will only provide a speedup if the Turing machine simulated allows for multiple computational branches. Otherwise, if the computation is not capable of being parallelized then nothing can be gained, performance-wise, from employing such an approach.

8.4 Conclusions

In this chapter an approach was presented for an iterative quantum production system with a built-in speedup mechanism and capable of the partial behaviour characteristic of μ-recursive functions. The proposal described makes use of a unitary operator C that can be perceived as mapping a total function since it maps for every possible input into a distinct output. However, operator C is employed in a quantum iterative deepening procedure that examines all path possibilities up to a depth level d until a solution is found, if indeed there exists one. Due to the probabilistic nature of Grover's algorithm there is always the possibility that, upon measurement, a non-terminating state is obtained. As a consequence, the procedure would iterate to an additional level of productions and could therefore fail to recognize a halting state. This issue can be sidestepped through the development of specific control strategies capable of signaling that an halting state has been found at the shallowest production yielding such a conclusion, independently of the sequence length being analyzed.

The model is able to operate independently of whether the computation terminates or not, a requirement associated with universal models of computation. As a result, it becomes possible for the model to exhibit partial behaviour that does not disturb the overall result of the underlying quantum computational process. This result is possible since: (1) Grover's algorithm effectively allows one to obtain halting states, if they exist, with high probability

upon system observation; and (2) the overall complexity of this proposition remains the same of the quantum search algorithm. This procedure enables the development of verification-based universal quantum computational models, which are capable of coherently simulating classical models of universal computation such as the Turing machine.

Chapter 9

Different Quantum Search Frameworks

9.1 Introduction

Computational algorithms encompass a large scope of tasks that range from sorting arrays, performing signal analysis and encrypting/decrypting information. A large part of these problems can be represented through some symbolical state. This internal representation of the problem can be evolved into other states by a set of specific rules. An algorithmic procedure examines such states and decides which rule should be applied in order to obtain some final result. Typically, algorithmic procedures aim to deliver such final states correctly, and to do it whilst minimizing some type of performance measure (*e.g.* time, space or error).

It is the task of algorithms to determine, and apply, the sequence of rules that will correctly lead to the best performance. As previously stated, for some problems there may be a way to deduce which rule is best. However, for many other computational problems, at any given point in time there may be multiple rules that can be applied and no single way of inferring which one is best when trying to minimize the performance measure. The set of all possible combinations of rules forms what is referred to as the search space. As a result, for this type of problems the simplest algorithmic strategy for finding a solution consists in systematically enumerating and evaluating every element of the search space until goal states are found. This type of procedures are usually referred to as search algorithms whose goal can be succinctly described as finding elements within a search space that fulfill certain

conditions.

It is also interesting to question if it is possible to develop alternatives to Grover's algorithm? Namely, what concepts can be explored besides amplitude amplification schemes? Also, what are the implications of these alternatives performance-wise? The following sections examine these questions from two distinct perspectives. Section 9.2 analyses a quantum search algorithm based on entanglement detection schemes when used in conjunction with purposely tune unitary-operators. Section 9.3 evaluates the consequences of employing periodic search spaces in order to perform search, an idea that tried to explore the performance gains delivered by the quantum factoring algorithm.

9.2 Quantum Entanglement and Partial Search

Computer scientists are often faced with the task of constructing algorithms capable of delivering a solution for a given problem. For some problems it is possible to engineer algorithms capable of producing a solution with a number of computational steps that is bounded by a polynomial n^k where n is the length of the input and k some constant. The class of problems for which a polynomial-time algorithm exists is known as P. Problems belonging to P are usually seen as being efficiently solvable. Class EQP represents the quantum equivalent of P.

For other problems it is possible to verify in polynomial-time if a given configuration is a solution, although there are no known methods for efficiently calculating a solution. For these type of problems, there is no alternative but to perform an exhaustive search of all possible configurations. The class NP consists of those problems whose possible configurations can be verified in polynomial-time. Clearly, P $\subseteq$ NP since the possibility of constructing a solution in polynomial time also implies that a solution can be verified efficiently. One of the outstanding questions in computer science consists in determining if the class NP is equivalent to the class P, *i.e.* P=NP? Traditionally, approaches to answering this question have focused at a subclass of NP, namely NP-complete problems. This subclass contains those problems which are both NP and NP-hard. A problem is said to be NP-hard if an algorithm capable of solving it can be translated into an adequate algorithm for any NP problem. An efficient solution for a problem in NP-complete implies that an efficient solution exists for all problems in NP.

In [56] some clues were given that some problems that are classically hard may have an efficient quantum solution. Shor's algorithm for efficient factorization [191] reinforced this idea. Later, Grover's search algorithm [82] provided an asymptotical quadratic speed-up over

classical strategies. The quantum search algorithm systematically increases the probability of obtaining a solution with each iteration. After the algorithm has concluded, a measurement is performed in a quantum superposition, in order to obtain a solution with high probability. The superposition state represents the set of all possible results. Grover's approach sparked interest by the scientific community on whether it would be possible to devise a faster search algorithm. Shortly after this result was published, a lower-bound of $\Omega(\sqrt{N})$ oracle queries for the search problem was proved in [26].

In this section an alternative search method based on the principles of tree search decomposition and quantum entanglement detection was presented. Unlike traditional approaches, this one does not rely on measuring a quantum superposition of possible values. Rather, the proposed method exploits the unitary operator that is applied to a quantum superposition in order to infer possible solutions with certainty. However, an implicit caveat exists associated to this quantum search proposal. Namely, the system implies a trade-off between speed and space that will become apparent in the following sections.

The next sections are organised as follows: Section 9.2.1 presents the hybrid approach, combining tree search decomposition alongside with quantum entanglement detection schemes. Section 9.2.2 presents the conclusions of this work.

9.2.1 Approach

How should an alternative approach to Grover's algorithm be developed? First, it is useful to envisage the following scenario: suppose a quantum system composed of a query register, $|q\rangle$, and the answer register, $|a\rangle$, acting on Hilbert space $\mathcal{H} = \mathcal{H}_q \otimes \mathcal{H}_a$. The query register is an n-qubit register where possible values for the binary variables of a specific problem instance will be setup. Typical examples include the SAT problem, which was the first problem ever shown to be NP-complete [51]. In order to gain a quantum advantage over classical computation $|q\rangle$ needs to be placed in a uniform superposition of the computation basis. This can be done efficiently by applying the Hadamard transform H a total of n times to the n-qubit state $|0\rangle$, *i.e.* $H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle$. Such a procedure enables the creation of a superposition containing an exponential number of states, each of which representing a possible tree path, by only employing a polynomial number of gates. The answer register contains a single qubit, which is initialized to state $|0\rangle$. The overall state of the system can thus be described as illustrated in Expression 9.1. Additionally, suppose that a quantum oracle O specific to the problem being considered is constructed with the form presented in Expression 3.27.

$$|q\rangle|a\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|0\rangle \tag{9.1}$$

If oracle O is applied to the combined state of Expression 9.1 a result like the one illustrated in Expression 9.2 may be obtained, where $|\psi'\rangle$ denotes the overall superposition evaluation. For simplification issues assume that there exists at least a solution. Naturally, some of the query values produce a solution, whilst others do not.

$$|\psi'\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} O|x\rangle|0\rangle = \begin{cases} |\underbrace{00\cdots0}_{\text{n bits}}\rangle|0\rangle \\ |00\cdots1\rangle|0\rangle \\ \quad\vdots \\ |11\cdots0\rangle|1\rangle \\ |11\cdots1\rangle|0\rangle \end{cases} \tag{9.2}$$

From this point on the system's state can no longer be expressed as a tensor product between query and answer register, *i.e.* the system becomes entangled. Quantum entanglement is a key feature of quantum mechanics, which details the connections between subsystems of compound quantum systems. It was a key aspect of the quantum world formalism proposed by von Neumann in 1932 [210]. Although the intriguing impacts of quantum inseparability were only later grasped by Einstein, Podolsky and Rosen [64] alongside Schrödinger [188]. Quantum entanglement is also a key resource in quantum information.

Mathematically, the state of each register can be described by tracing out the remaining register, through the partial trace mechanism. In this case the main object of interest is the state of the answer register. In order to obtain the partial trace of the answer register it is first required to calculate the density operator of the quantum state presented in Expression 9.2, respectively ϱ. The overall form for ϱ^a is illustrated in Expression 9.4

$$\begin{aligned} \varrho &= |\psi\rangle\langle\psi| \\ &= \tfrac{1}{\sqrt{2^n}}(|00\cdots0\rangle|0\rangle + \cdots + |11\cdots1\rangle|0\rangle) \\ &\quad \tfrac{1}{\sqrt{2^n}}(\langle00\cdots0|\langle0| + \cdots + \langle11\cdots1|\langle0|) \\ &= \tfrac{1}{2^n}|00\cdots0\rangle|0\rangle(\langle00\cdots0|\langle0| + \cdots + \langle11\cdots1|\langle0|) + \\ &\quad + \cdots + \\ &\quad \tfrac{1}{2^n}|11\cdots1\rangle|0\rangle(\langle00\cdots0|\langle0| + \cdots + \langle11\cdots1|\langle0|) \end{aligned} \tag{9.3}$$

$$\begin{aligned}\varrho^a &= \mathrm{Tr}_q(\varrho) \\ &= \tfrac{1}{2^n}(\langle 00\cdots 0||00\cdots 0\rangle|0\rangle\langle 0| + \langle 00\cdots 1||00\cdots 1\rangle|0\rangle\langle 0| + \\ &\quad + \cdots + \\ &\quad \langle 11\cdots 0||11\cdots 0\rangle|1\rangle\langle 1| + \langle 11\cdots 1||11\cdots 1\rangle|0\rangle\langle 0|) \\ &= \tfrac{1}{2^n}\left[(2^n-1)|0\rangle\langle 0| + |1\rangle\langle 1|\right]\end{aligned} \tag{9.4}$$

Generally, the result presented in Expression 9.3 can be improved if the number of solutions is taken into account. Accordingly, let k denote the overall number of solutions, then ϱ^a takes the form shown in Expression 9.5. The overall state is separable only when $k = 0$, *i.e.* no solution exists, or when $k = 2^n$, each value belonging to $[0, 2^n - 1]$ is a solution. Otherwise, the system is entangled. Thus, the problem of determining whether or not a solution to a problem exists can be reduced to the problem of determining whether the state is separable or entangled.

$$\varrho^a = \frac{1}{2^n}\left[(2^n-k)|0\rangle\langle 0| + k|1\rangle\langle 1|\right] \tag{9.5}$$

Quantum entanglement detection

The quantum separability problem consists in determining if a given a density matrix ϱ representing a quantum state is entangled or separable [78]. Efficiently deciding on the nature of such states has grabbed researchers attention and remains a problem of crucial importance to the fields of quantum computation and information [116]. Generally speaking, quantum entanglement is studied in accordance with a varied mix of properties (just to name a few of these: bipartite vs. multipartite systems, pure vs. mixed states, bound entanglement; for exhaustive reviews please refer to [128], [109] and [89]). The quantum separability question has been approached from both classical and quantum perspectives. These approaches typically consider the nature of the input (classical vs. quantum), and whether any required processing will be performed on a classical or quantum computer [115]. This problem was shown to be NP-hard classically [90]. However, as mentioned in [115] the processes involving both quantum input and processing have not been thoroughly investigated.

In the case of this specific approach it would suffice to restrict the analysis to bipartite quantum systems with mixed states. As pointed out in [106] the mixed state requirement stems from the fact that any potential laboratory demonstration of this approach would have to deal with mixed states rather than pure ones, due to the uncontrolled interactions with the environment. These requirements are present in one of the existing quantum detection

schemes, namely the one proposed in [108]. The method employed by the authors is experimentally viable and provides for a direct detection mechanism of quantum entanglement. Their approach is based on the theoretical foundations laid down in [106]. The method determines whether a state ϱ is separable or not, *i.e.* entangled, based on the mathematical properties of linear positive maps acting on matrices. More specifically [109], let $M_d \to M_d$ be the space of matrices of dimension d, a map $\Lambda : M_d \to M_d$ is called positive if it is Hermitian and has non-negative spectrum. Additionally, the map Λ is completely positive if and only if $I \otimes \Lambda$ is positive for identity map I on any finite-dimensional system. A state ϱ is separable if and only if the result presented in Expression 9.6 is observed for all positive but not completely positive maps $\Lambda : M_d \to M_d$.

$$[I \otimes \Lambda](\varrho) \geq 0 \tag{9.6}$$

Expression 9.6 cannot be directly used since it requires knowing state ϱ beforehand. Additionally, positive maps Λ cannot be directly implemented in laboratory [109]. It is possible to obtain a physically realizable map by mixing an appropriate proportion of $[I \otimes \Lambda]$ with a depolarizing map. This approach allows for a new map $\widetilde{[I \otimes \Lambda]}$ to be obtained, which have been referred to as structural physical approximations. For more on this subject please refer to [73]. The separability criterion can then be restated as follows [108]: ϱ is separable if and only if for all positive maps Λ the condition presented in Expression 9.7 is observed.

$$\widetilde{[I \otimes \Lambda]}(\varrho) \geq \frac{d^2\lambda}{d^4\lambda + 1} \tag{9.7}$$

Where λ corresponds to the most negative eigenvalue obtained when the induced map $[(I \otimes I) \otimes (I \otimes \Lambda)]$ acts on the maximally entangled state of the form $\frac{1}{d^2}\sum_{i=1}^{d^2} |i\rangle|i\rangle$. Accordingly, Expression 9.7 states that the lowest eigenvalue of the transformed state $\varrho' = \widetilde{[I \otimes \Lambda]}(\varrho)$ should be greater than $\frac{d^2\lambda}{d^4\lambda+1}$ for ϱ to be separable.

The authors devised a method which allows for an estimate of the lowest eigenvalue to be obtained efficiently and directly. It requires that a joint measurement be performed on N copies of state ϱ'. The overall input density operator of the estimation problem is $\varrho'^{\otimes N}$, which exists on the Nth tensor power $\mathcal{H}^{\otimes N}$ [120]. The error ϵ associated with the estimate of the lowest eigenvalue decreases exponentially with N. Such a measurement can be represented as a quantum network implementing projections on the symmetric and partially symmetric subspaces [108]. An efficient method addressing these questions was proposed in [19] requiring a number of auxiliary gates that grows quadratically with the dimension of the input, *i.e.* $O(n^2)$, where n is the number of bits. If ϱ' represents the state of an n

qubit register, then each additional tensor power will mean that another n bits should be taken into account. Consequently, an $\varrho'^{\otimes N}$ system will have a total of $N \times n$ bits, which means that the quantum network responsible for estimating the lowest eigenvalue will have $O(N^2 n^2)$ complexity.

Clearly, this approach is dependent on map Λ, which have not been operationally characterized so far [107]. As pointed out in [109] in general the set of positive but not completely positive maps is not characterized and it involves a hard problem in contemporary linear algebra. However, for low dimensional systems, namely those with dimension $2 \otimes 2$ or $2 \otimes 3$, the positive partial transpose map proposed in [174] can be employed as the Λ. In [107] the authors draw attention to the fact that: "Recently, the progress in this direction has been made [136] [137] which suggests that tests of separability based on positive maps will soon acquire practical meaning beyond the scope of two-qubit systems." Whether such a map Λ acting on $\mathcal{H}_d \otimes \mathcal{H}_d$ quantum systems can be determined remains an open question.

Subset entanglement inducing oracle

Grover's algorithm provides an $\Omega(\sqrt{N})$ lower bound when employing oracles searching on the full range of searchable items. It would be desirable to develop an alternative search approach not solely based on amplitude amplification schemes. In classical tree search it is a standard technique to start by analyzing subtrees and deciding whether these may eventually lead to a solution. Based on problem requirements it is possible to automatically exclude, *i.e.* prune, certain subtrees. Pruning may eventually be responsible for discarding large sections of the tree, and therefore allow the search to terminate faster. These classical search concepts can be used in order to develop an alternative quantum hierarchical search approach.

Quantum algorithms employing traditional oracles provide at most a polynomial advantage over classical algorithms for total functions, *i.e.* functions defined for the whole of $\{0,1\}^n$, where n is the number of bits. The oracle model contemplates superpolynomial advantage but only when partial functions are defined which operate on a subset of $\{0,1\}^n$ [21]. Classical search can be viewed as a procedure which evaluates subsets of an initial range. Since in quantum computation the oracle operator can be applied to a superposition of computational basis, evaluating subsets is equivalent to only evaluating specific ranges of the superposition. Accordingly, it is possible to develop an oracle responsible for evaluating only a certain subset of the initial range $[0, 2^n - 1]$ allowed with n qubits.

Although this approach focuses in evaluating a specific subset there are other alternatives for trying to decompose a quantum search space. For instance, Grover concluded in [80] that determining the first n bits of a solution by employing amplitude amplification schemes is

only slightly easier than determining the total bits.

This model for a range specific entanglement inducing oracle can be described as presented in Expression 9.8, which employs an auxiliary function $f_{[a,b]}(q)$ defined in Expression 9.9.

$$O_{[a,b]}|q\rangle|a\rangle = |q\rangle|a \oplus f_{[a,b]}(q)\rangle \tag{9.8}$$

$$f_{[a,b]}(q) = \begin{cases} 1 & \text{if } f(q) \text{ is a solution and } q \in [a,b] \\ 0 & \text{otherwise} \end{cases} \tag{9.9}$$

As was previously pointed out, the oracle evaluation process has the overall effect of entangling the quantum registers. By testing whether the oracle has induced, or not, quantum entanglement it is possible to check for the presence of a solution state in a given range. This mechanism allows for ranges containing solutions to be further decomposed. In contrast, the absence of a solution allows for a specific range to be pruned from the overall search procedure.

Ideally, the entanglement detection scheme should present some type of polynomial upper-bound behaviour such as the one described in the previous section. [1] If the state ϱ resulting from applying an oracle O with the form presented in Expression 9.9 is separable then the range evaluated can automatically be discarded. Discarding a wide range of potential candidates *en masse* can be understood as the classical tree search operation of pruning certain subtrees. On the other hand, if ϱ is entangled then it is possible to further decompose the associated range. Eventually, this sort of recursive branch and bound procedure, by constantly readjusting the range of oracle O, will "zoom in" on a solution. Additionally, it would be a relatively easy task to search problem spaces comprising of multiple solutions k. Namely, one would simply need to systematically focus on previously non-expanded but solution-bearing ranges.

This approach requires a new oracle to be defined with each iteration in terms of a specific subset that may be entangled. The set of oracles applied throughout the search can be viewed as a single "dynamic" oracle, which differs substantially from the standard "static" oracle applied in quantum search. Additionally, in contrast with Grover's algorithm, this approach does not rely on performing an amplitude amplification process. Rather, the method discussed is mainly concerned with decomposing the quantum search space.

[1] The entanglement detection approach described in [108] requires the overall bipartite system to be $d \otimes d$. Consequently, the answer register $|a\rangle$ should have the same dimension than $|q\rangle$, *i.e.* n bits. This requirement has no direct consequences in the overall oracle since unitary evolution can still be assured.

On the growth of the number of copies required

Clearly, one question still lingers: What can be said about the number of copies N of the system that are required? According to [118] any procedure that on input $|\psi_Z\rangle$ guesses whether state $Z = X$ or $Z = Y$ will guess correctly with probability of at most $1 - \epsilon = \frac{1}{2} + \frac{1}{2}\sqrt{1-\delta^2}$ where $\delta = |\langle\psi_x||\psi_y\rangle|$. In this case it is interesting to distinguish two cases, namely:

- $|\psi_{solution}\rangle$ which results from applying $U|\psi\rangle$ when *one* solution exists;
- $|\psi_{no-solution}\rangle$ which results from applying $U|\psi\rangle$ when *no* solution exists;

For search spaces of dimension L the initial amplitudes α_i associated with each computational basis i of the superposition $|\psi\rangle$ will have value $\frac{1}{\sqrt{L}}$. After having applied oracle O the two states remain exactly equal except for two computational basis where the amplitudes permuted. This means that when calculating the inner product the permuted computational basis will only slightly contribute to a general decrease. Accordingly, the inner product will sum the value $\frac{1}{\sqrt{L}}^2$ a total of $L-1$ times, as depicted in Expression 9.10.

$$\delta = \langle\psi_{solution}||\psi_{no-solution}\rangle = \frac{L-1}{L} \tag{9.10}$$

The approach devised in [108] requires N copies of the system. Generalizing, this means that the combined δ has the value $\delta^N = \left(\frac{L-1}{L}\right)^N$. The three-dimensional plot of δ^N as a function of $L \in [2^1, 2^{30}]$ and $N \in [2^1, 2^{30}]$ is illustrated in Figure 9.1.

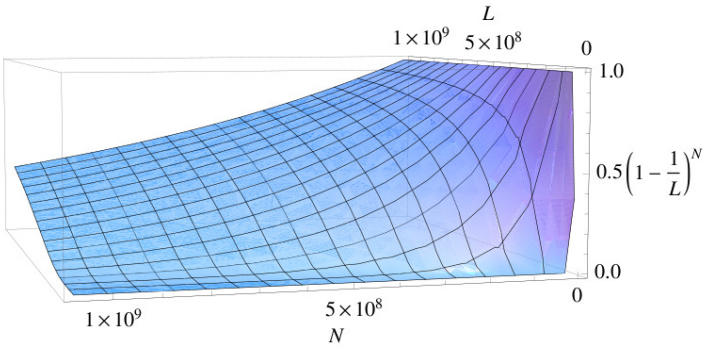


Figure 9.1: Three-dimensional plot of $\delta^{\otimes N}$ as a function of $L \in [2^1, 2^{30}]$ and $N \in [2^1, 2^{30}]$.

In order for these states to be distinguished with significant probability the inner product δ^N must be made small. However, in order to achieve this one needs to choose a number of

copies N that grows in accordance with the dimension of the search space L, *i.e.* $N = O(L)$. Consequently, this approach would not provide for any gains over classical search.

Consequences for efficient entanglement detection schemes

What would be the consequences if the number of system copies N was not a function of the search space? Suppose the proposed search procedure is executed on n-qubits placed on a superposition. Initially, the algorithm has to decompose the $[0, 2^n - 1]$ initial range. Assuming that any specific range being considered is split in half, then the procedure needs to verify if evaluating the elements in $[0, 2^{n-1} - 1]$ produces an entangled quantum state ϱ. If this is found to be true then subset $[0, 2^{n-1} - 1]$ can be also split in half and evaluated. Otherwise, subset $[2^{n-1} - 1, 2^n - 1]$ needs to be decomposed. Independently of what subset induces entanglement, the algorithm is able to prune half of the 2^n initial states, *i.e.* $2^n/2$. Accordingly, for iteration i, the oracle is able to focus on $2^n/2^i$ states. Clearly, when $i = n$ a single state is being considered and consequently a solution can be determined with certainty by employing $O(n)$ oracle queries. Associated with each oracle query is the quantum entanglement detection scheme bringing the overall complexity of the approach to $O(N^2 n^3)$.

It is my belief that it is not possible to efficiently detect quantum entanglement. Taking into account the simplicity of the search procedure designed in Section 9.2.1 then if such a method existed it would be possible to efficiently search, *i.e.* in quantum polynomial time, a problem space of dimension d. Accordingly, it is possible to define the following conjecture.

Quantum entanglement detection conjecture - It is not possible to efficiently detect quantum entanglement non-classically since this would automatically imply that a simple algorithm exists proving that NP=EQP.

The above conjecture stresses the notion that there appears to be a relationship between entanglement detection and search in terms of computational complexity, *i.e.* both problems appear to be equally difficult. Indeed, since quantum entanglement detection via classical methods was shown to be NP-hard [90] any polynomial classical algorithm capable of solving NP-hard problems would allow for efficient mappings capable of tackling both quantum entanglement detection as well as exponential-growth search problems. From a quantum computation perspective it appears that, by employing such an entanglement detection scheme, there exists a direct relationship where a trade-off between space and time occurs. Nonetheless, this approach can still be perceived as a form of quantum computation, although one requiring a careful examination of the total time and space resources employed.

9.2.2 Conclusions

Shor's algorithm provided a superpolynomial speed-up by exploiting a hidden structure of the problem [191]. However, traditional tree search mechanisms are employed when such an element of structure cannot be determined. Quantum computation provides at best a polynomial speed-up when oracles mapping total functions are employed. Superpolynomial speed-up is achievable but only if a subset of a functions domain is analyzed. This section focused on the dynamics of partial function unitary evolution alongside quantum entanglement detection schemes. The general characterization of positive but not completely positive linear maps Λ alongside quantum entanglement detection schemes and partial range entanglement inducing operators may eventually be responsible for producing efficient algorithmic solutions capable of searching exponential-growth search spaces. Although some research has already been carried out, further thorough analysis into the subject is still required. However, given that $N = O(L)$ current methods cannot be employed in order to speed up quantum search.

9.3 Periodic Search Space

Despite the quadratic improvement brought upon by quantum computation to classical search procedures the fact of the matter is that the overall time complexity of current approaches still grows in an exponential manner. Accordingly, it would be interesting to determine if more efficient approaches exist. Besides search, perhaps one of the best known applications of quantum computation is Shor's algorithm [191] for factoring numbers. Classically, finding the factors of an n-bit number is a non-tractable computational task that requires sub-exponential time as a function of n (please refer to [176], [177], [134] and [135]). The quantum factoring algorithm is capable of determining the factors in time that is polynomial in $\log n$. The procedure is able to deliver a superpolynomial speed-up relatively to the classical methods by reducing the original problem to one of calculating the period of a function, a problem that is thought to be hard in classical computation. In a quantum setting, determining the period of a function is an easy task that can be performed through the quantum Fourier transform. More recently, researchers have been able to develop a quantum processor capable of factoring a composite number into prime factors [142]. Given the ability of determining periods exponentially faster through the quantum Fourier transform, it is natural to raise the question of whether it is possible to incorporate the concepts of periodicity to search a state space?

In this section some of the ideas behind Shor's algorithm are explored in order to develop

a platform for search, which by itself raises some interesting questions, namely: how should the search problem be tackled in such a context? How should the procedure operate? What are the requirements and limitations associated with such a procedure? Finally, is there something to be gained performance-wise? Although the procedure can be used for search, there are some important limitations associated with it, namely concerning the specific composition of the search space. However, the limitations of trying to map the search problem into the period finding problem are still worthy to be analysed. More concretely, the degree of feasibility of developing an alternative method for performing quantum search based on periodicity alongside space decomposition are examined.

The following sections are organised as follows: Section 9.3.1 describes the general period finding strategy employed by the quantum Factoring algorithm; Section 9.3.2 elaborates on these concepts in order to develop an alternative quantum search method; Section 9.3.3 presents the overall conclusions of this work.

9.3.1 Quantum Period Finding

Shor's algorithm to factoring integers reduces the problem to one of determining the orders of integers modulo M. Specifically, given integers a and M such that the greatest common divisor between them is 1, the order of a mod M is the small positive integer r so that $a^r \equiv 1(\text{mod } M)$. Such an order r effectively translates as the period of sequence. The quantum factoring algorithm builds a superposition capable of encompassing several periodic occurrences of the function $f(x) = a^x \text{ mod } M$. Such a procedure results in a state with the form presented in Expression 9.11.

$$|\psi\rangle = \sum_{x=0}^{2^{\lceil \log_2 M^2 \rceil}-1} |x\rangle|a^x \text{ mod } M\rangle \tag{9.11}$$

Where the number of bits n employed is such that $2^n \geq 2^{r^2}$. In this case n was chosen such that $n = \lceil \log_2 M^2 \rceil$, in order to deliver a bound on the error capable of producing a correct period estimate with high probability [118]. The periodic analysis is simplified if a measurement is performed on the second register [182]. After performing such a measurement the first register will be in a superposition of periodic values, as depicted in Expression 9.12

$$|b\rangle + |b+r\rangle + |b+2r\rangle + \cdots + |b+zr\rangle, \forall z : 0 \leq b + zr < 2^n \tag{9.12}$$

Expression 9.12 represents a periodic superposition of states of the form presented in Expression 9.13, with period r, shift b, and m repetitions of the period. By subsequently applying the inverse quantum Fourier transform, denoted by QFT_m^{-1} in Expression 9.14, on the basis states $|0\rangle, |1\rangle, \cdots, |mr-1\rangle$, it is possible to able to obtain the superposition state represented by Expression 9.15 (please refer to [118] for additional details into quantum Fourier analysis).

$$|\phi_{r,b}\rangle = \frac{1}{\sqrt{m_b}} \sum_{z:0\leq b+zr<2^n} |b+zr\rangle \tag{9.13}$$

$$\mathrm{QFT}_{2^n}^{-1}|x\rangle \rightarrow \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{-2\pi i \frac{x}{2^n} y}|y\rangle \tag{9.14}$$

$$\mathrm{QFT}_{mr}^{-1}|\phi_{r,b}\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i \frac{b}{r} k}|mk\rangle \tag{9.15}$$

If a measurement is performed upon the superposition represented by Expression 9.15 a computational basis $|mk\rangle$ is obtained. It is now possible to compute $\frac{mk}{mr} = \frac{k}{r}$ in lowest terms, since mr is known. If instead $\mathrm{QFT}_{2^n}^{-1}$ is applied to the stated depicted in Expression 9.13 then performing a measurement will yield a value x such that $\frac{x}{2^n}$ is close to $\frac{k}{r}$, for a random integer $k \in \{0, 1, 2, \cdots, r\}$, with probability $\frac{m_b}{2^n}\frac{4}{\pi^2}$ [118]. The probability of obtaining an incorrect estimate is at most ϵ, and can be made small with a marginal increase in the number of bits employed by the quantum Fourier circuit [164]. Additionally, it can be shown that $\left|\frac{x}{2^n} - \frac{k}{r}\right| \leq \frac{1}{2r^2}$ when the number of period repetitions is greater or equal than the period, *i.e.* $m \geq r$. This fact allows for the fraction $\frac{k}{r}$ to be subsequently determined through the continued fractions algorithm, which executes in time polynomial in n. Let $L = \lceil \log_2 M \rceil$, the computational resources required are [164]: (1) $O(L)$ Hadamard gates for the initial superposition; (2) $O(L^2)$ gates for the QFT^{-1}; (3) $O(L^3)$ gates for the modular exponentiation employed for phase estimation; (4) $O(L^3)$ gates for the continued fractions procedure involves. The overall procedure for determining periods through this manner executes in $O(L^3)$ and succeeds with $O(1)$ probability. In conclusion, given a periodic superposition state it is possible to efficiently determine the period of the respective sequence.

9.3.2 Quantum Periodic Search

How can the ideas supporting Shor's algorithm be adapted in order to perform search? As previously mentioned, two of the main concepts employed are: (1) the notion of periodicity

and (2) the use of the quantum Fourier transform to determine such a period. Accordingly, any Shor inspired approach to the search problem needs to somehow incorporate into its definition the concept of periodicity.

In the following sections a potential alternative search procedure is presented that explores these ideas, namely: Section 9.3.2 explains the general ideas supporting the search algorithm; Section 9.3.2 examines the running time of the proposal; Section 9.3.2 analyses the intricacies of the approach and elaborates on the specific details of how the search space should be composed in order for the algorithm to work.

General Procedure

One possible solution to the concept of search periodicity may reside in spanning the search space multiple times. This approach allows for any potential solution to occur multiple times and thus generate the equivalent of a periodic signal. More specifically, suppose a search space of dimension 2^n that is generated by n bits is employed. Traditional search procedures would need to evaluate such a space and determine which elements x are solution states, *i.e.* $x \in S_g$. Typically, this is done with the help of a boolean evaluation function f, as the one illustrated in Expression 9.16, which returns the symbolic truth-assignment value of 1 if a goal state is found, and 0, *i.e.* false, otherwise.

$$f(x) = \begin{cases} 1 & \text{if } x \in S_g \\ 0 & \text{otherwise} \end{cases} \tag{9.16}$$

In order to get a periodic distribution the original search space needs only to be extended with $m-1$ copies, for a total of m periodic repetitions. This procedure involves extending the original space in order to accommodate for such a periodicity. This can be done by building a superposition state $|\psi\rangle = \frac{1}{\sqrt{2^n m}} \sum_{x=0}^{2^n m-1} |x\rangle$. Accordingly, it is possible to build an extended search space where the original 2^n states need to be repeated m times, resulting in $2^n m$ states. Consequently, the verification function f employed needs to ensure that the space repetition is taken into account. This can be accomplished by redefining function f in order to accommodate for modular arithmetic, as described in Expression 9.17. Such a procedure guarantees that when function f is employed to analyse elements of the $2^n m$-dimensional space these will be decomposed into the smaller components of the original search space of size 2^n.

By employing such a binary formulation of the evaluation function it is possible to build the associated unitary operator required by quantum computation with relative ease. Specifically, for every irreversible function f a reversible mapping can be constructed with the form

$O_f|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$, where x is a set of bits, also known as a register, and y is an auxiliary bit. If register $|y\rangle$ is placed in the superposition state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, then performing the oracle evaluation function can be perceived as a phase-shift, which effectively flips the amplitude of the solution states, *i.e.* $|x\rangle \rightarrow (-1)^{f(x)}|x\rangle$. This type of quantum oracles employing irreversible functions can also be easily constructed with a linear growth in the amount of resources required (please refer to [201] and [198] for additional details).

$$f(x) = \begin{cases} 1 & \text{if } x \bmod 2^n \in S_g \\ 0 & \text{otherwise} \end{cases} \tag{9.17}$$

Once the notion of periodicity is clearly defined it becomes possible to develop a simple procedure to search a space. One possibility consists in evaluating the extended search space $|\psi\rangle$ and determining whether a period r exists. If the procedure to determine a period returns a valid period r that is different than zero (this subject is elaborated further in Section 9.3.2), then this is indicative that a solution exists. Conversely, an absence of a valid period indicates that no solution exists. In order to understand this notice that a period only occurs when certain states are marked as solution states, *i.e.* when an amplitude inversion occurs, otherwise every element x of the space will have the same amplitude, which will result in $\text{QFT}^{-1}|\psi\rangle$ failing to produce a valid period. Such a scheme provides the ability to efficiently determine whether a period exists. Accordingly, what is only required is the ability to somehow decompose the search space and systematically check if such decompositions produce a period. Failure to produce a period indicates that the procedure should focus on other decompositions. This can be done by recursively decomposing the extended search space and for each decomposition build a superposition state representing the range being currently considered.

The general idea is that ranges that are periodic should be further decomposed and examined until a solution state is found. These ideas are illustrated in Algorithm 3, which starts by determining the number of bits n required to encode a range $[a, b]$ into an adequate superposition state $|\psi\rangle$. The algorithm then proceeds to determine the period of $|\psi\rangle$ in Line 4. If a valid period exists, then the algorithm knows that the current range being considered contains a solution. Accordingly, the procedure tries to determine if the range in question cannot be further decomposed. This happens when the range only contains two elements, as tested in Line 6. If this is the case, the algorithm simply tests and outputs a solution state. Otherwise, the algorithm proceeds by decomposing the original range $[a, b]$ in half, as illustrated in Line 12 and Line 14. The algorithm then performs a recursive call to itself with the new sub-ranges (Line 12 and Line 14), which are then submitted again to the same evaluation process. The methods returns the first solution state it encounters, if one exists.

Otherwise, it simply outputs the symbolic constant NO_SOLUTION_FOUND.

Algorithm 3 Quantum Periodic Search

```
1:  function search(a, b)
2:      n = ⌈log_2(b − a)⌉
3:      |ψ⟩ = 1/√(2^n m) Σ_{x∈[a,m(b+1)−1]} O_f|x⟩
4:      r = determinePeriod(QFT^{−1}|ψ⟩)
5:      if r ≠ 0
6:          if b − a == 1
7:              if a ∈ S_g
8:                  return a
9:              else
10:                 return b
11:         else
12:             result = search(a, ⌊a + (b−a)/2⌋)
13:             if result == NO_SOLUTION_FOUND
14:                 search(⌈a + (b−a)/2⌉, b)
15:     else
16:         return NO_SOLUTION_FOUND
```

Performance Analysis

Analysing the performance of the algorithm requires determining an upper-bound on the maximum number of recursions performed. Each recursion splits the interval range evenly. Assuming that the dimension of the extended search space is a power of two, *i.e.* $2^n m = 2^k$, then after $d = k - 1$ levels of recursion the length of the ranges being considered will be 2. This length signals the terminating condition of the recursion. Each recursion level invokes at most two recursive calls. After d levels of recursion a maximum of $2d$ recursive calls will have been made since every recursion immediately returns once a solution is determined not to exist in a specific range. The overall complexity of the procedure grows linearly with k, *i.e.* $O(k)$.

Probability Distribution Analysis

In this section the specific details of why the proposed method fails to deliver a solution state are examined. Given how crucial the QFT^{-1} is to the search procedure, namely when it comes to determining the period, it becomes important to try to precisely determine the consequences of evaluating such a periodic search space. Accordingly, is there something that can be said about the probability distribution associated with $\mathrm{QFT}^{-1}|\psi\rangle$? First, bear in mind that the extended search space consists of m periodic repetitions of a search space

of dimension 2^n. As a result, it is automatically known from the start that the period will be 2^n. In other words, $f(x) = f(y)$ if the period $r = 2^n$ divides $x-y$ evenly with zero remainder, respectively denoted as $r|x-y$. However, employing a binary function f does not guarantee that $f(x) = f(y)$ if and only if $r|x-y$. Consequently, evaluating superposition state $|\psi\rangle$ with such a scheme will potentially produce a binary superposition state, with one part containing the set of goal states and the other containing non-solution states. In practice, this means that the evaluated superposition state will have the form presented in Expression 9.18, where $X_\gamma = \{x|f(x) = \gamma\}$. Such a distribution is presented in Figure 9.2 where a search space of dimension 64 with a single randomly chosen solution state, respectively $|30\rangle$, was extended 64 times.

$$O_f|\psi\rangle = \sqrt{\frac{|X_0|}{2^n m}} \sum_{x \in X_0} |x\rangle|0\rangle - \sqrt{\frac{|X_1|}{2^n m}} \sum_{x \in X_1} |x\rangle|1\rangle \tag{9.18}$$

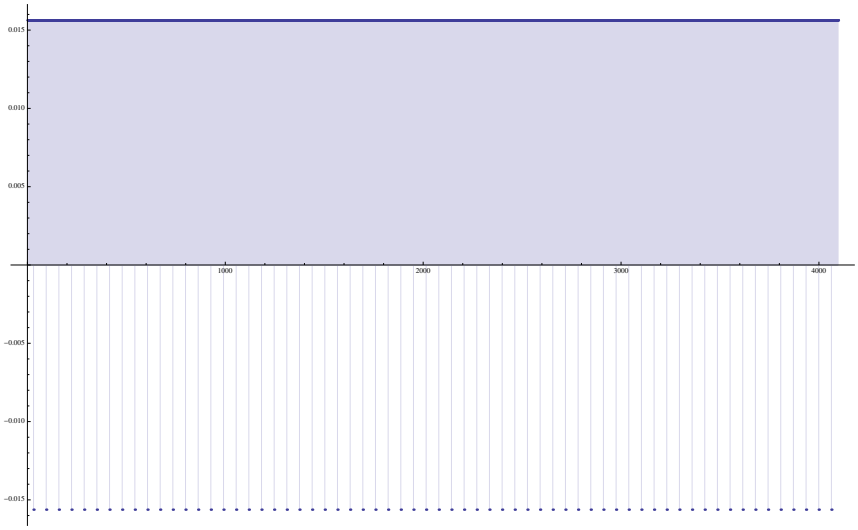


Figure 9.2: A search space of dimension 64 with a single randomly chosen solution state, respectively $|30\rangle$, replicated for 64 times.

Since the state represented by Expression 9.18 differs from Expression 9.13 it is important to understand how this fact impacts the application of the inverse quantum Fourier transform. The analysis is simplified if the QFT^{-1} is considered with a different form than that of Expression 9.14. Namely, in [182] the authors draw attention to the fact that the QFT^{-1} can be perceived as operating on a quantum state as described in Expression 9.19, where $a(x)$ denotes the amplitude of state $|x\rangle$. In addition, $A(x)$ represents the Fourier coefficients of the discrete Fourier transform of $a(x)$, respectively presented in Expression 9.20, where N

is chosen in order to process all the elements of the search space, respectively $2^n m$.

$$\mathrm{QFT}^{-1} \sum_x a(x)|x\rangle = \sum_x A(x)|x\rangle \tag{9.19}$$

$$A(x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a(k) e^{-2\pi i \frac{kx}{N}} \tag{9.20}$$

Such a formulation is important because it helps analysing two particular cases after applying QFT^{-1}, namely (1) what happens with state $|0\rangle$ and (2) what can be expected from the remaining states. Regarding the first question, notice that Expression 9.20 reduces to $A(x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a(k)$ for $x = 0$, which simply sums over all amplitudes. Now, consider a typical search space where the number of goal states is very small comparatively to the dimension of the space. As a result, performing the operation $\mathrm{QFT}^{-1}|\psi\rangle$ will produce a superposition where the amplitude associated with state $|0\rangle$ of the superposition will be the one described in Expression 9.21. Typically, $|X_1| << |X_0|$, as a result the number of amplitude flips occurring, respectively $|X_1|$, will contribute only very slightly to a general decrease in the DC component of the state $\mathrm{QFT}^{-1}|\psi\rangle$. Thus, performing any type of measurement on $|\psi\rangle$ will deliver $|0\rangle$ with a high probability. Also, if no solution exists, *i.e.* $|X_1| = 0$, then Expression 9.21 will reduce to $\frac{1}{\sqrt{2^n m}} \frac{2^n m}{\sqrt{2^n m}} = 1$, thus ensuring that state $|0\rangle$ will be the only one obtained upon measurement, thus justifying Line 5 of Algorithm 3.

$$\frac{1}{\sqrt{2^n m}} \left(\frac{|X_0|}{\sqrt{2^n m}} - \frac{|X_1|}{\sqrt{2^n m}} \right) = \frac{|X_0| - |X_1|}{2^n m} \tag{9.21}$$

Regarding the question of what happens with the remaining states, it is important to mention that if the period r divides N, *i.e.* $r|N$, the Fourier coefficients $A(x)$ will be non-zero only for those x that are multiples of N/r. If r does not divide N evenly, the result only approximates this behaviour, with the highest values at the integers closest to multiples of $u = N/r$ and low values at integers far from these multiples [182]. Given that it is known that $r = 2^n$ then choosing $N = 2^n m$ automatically guarantees that N will be divided evenly by r. In addition, it immediately follows that the u multiples will have the form $k\frac{N}{r} = k\frac{2^n m}{2^n} = km$ where $k \in [0, 1, 2, \cdots]$. Moreover, this means that for a search space of dimension $2^n m$ the Fourier coefficients will be different than zero for a total of 2^n times in the QFT^{-1} distribution since $\frac{2^n m}{m} = 2^n$.

It is now possible to determine the probability of the remaining states in $\mathrm{QFT}^{-1}|\psi\rangle$. Let $\mathcal{M}$ represent the set of states different than $|0\rangle$ and with non-zero probability. Set $\mathcal{M}$ corresponds to the complementary event of obtaining state $|0\rangle$. As a result, the probability

associated with set $\mathcal{M}$, respectively $P(\mathcal{M})$, can be easily obtained, as described by Expression 9.22. Accordingly, the individual probability of each state in $\mathcal{M}$ is $\frac{P(\mathcal{M})}{2^n-1}$. The amplitude distribution for the periodic search space described in Figure 9.2 is shown in Figure 9.3.

$$P(\mathcal{M}) = 1 - P(|0\rangle) = 1 - \left|\frac{|X_0| - |X_1|}{2^n m}\right|^2 \tag{9.22}$$

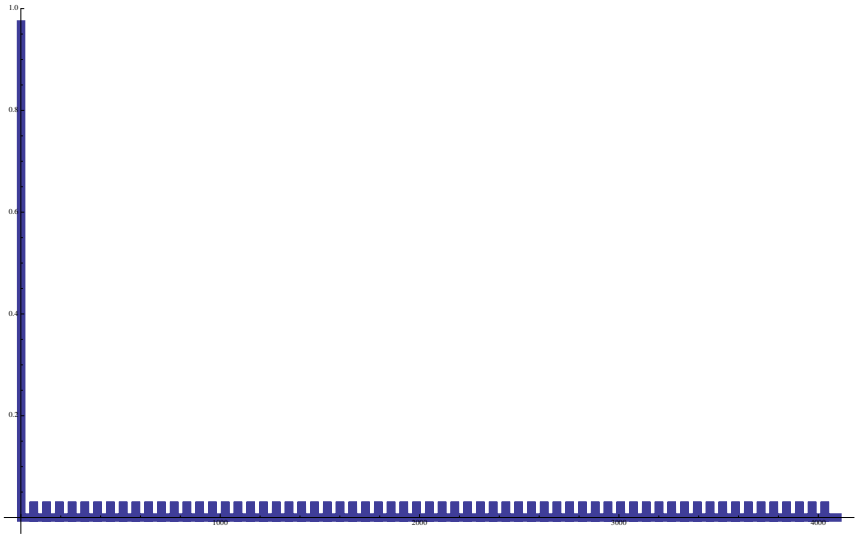


Figure 9.3: Fourier analysis of the search space illustrated in Figure 9.2.

Finally, $P(\mathcal{M}) \approx 0$ when $|X_0| >> |X_1|$, implying that in practice state $|0\rangle$ will always have a very high probability of being obtained upon measurement. This fact makes it very improbable that Line 5 of Algorithm 3 will be able to determine a period different than zero, causing the algorithm to fail. Accordingly, it is important to ask the question of when is it possible to efficiently determine r? For this to happen the probability of obtaining $\mathcal{M}$ should at least equal that of obtaining $|0\rangle$, *i.e.* $P(\mathcal{M}) = P(|0\rangle)$. This way it becomes possible to repeat the period finding process a constant number of times in order to deliver r on half of the attempts. Since $P(\mathcal{M}) = 1 - P(|0\rangle)$ this implies that $P(|0\rangle) = \frac{1}{2}$ for $P(\mathcal{M}) = P(|0\rangle)$. Determining the precise number of non-solution states that must exist in the space for such a probability distribution thus becomes a trivial task, as described by Expression 9.23. The term $2^{n-\frac{1}{2}}$ may generate non-integer numbers and as such requires the use of some type of rounding function.

$$\left|\frac{|X_0| - |X_1|}{2^n m}\right|^2 = \frac{1}{2} \Leftrightarrow |X_0| = 2^{n-\frac{1}{2}} + |X_1| \tag{9.23}$$

How does such a value compare against the overall length of the extended search space? A simple ratio, as illustrated by Expression 9.24, allows one to verify that approximately at most $\frac{1}{\sqrt{2}}$ of the search space should be composed of non-solution states. If this is the case then the period to be accurately determined. As a direct consequence this means that when roughly a third of the search space is composed of solution states then the algorithm is able to efficiently perform search, *i.e.* deliver a solution in polynomial time using Algorithm 3.

$$\frac{2^{n-\frac{1}{2}} + |X_1|}{2^n m} = 2^{-\frac{1}{2}} + \frac{|X_1|}{2^n m} \geq \frac{1}{\sqrt{2}} \tag{9.24}$$

The period finding algorithm can be generalized to determine the period r of a function $f : Z \to X$, for some finite set X. However, $f(x)$ must be different than $f(y)$ unless $r|x-y$, so that $f(x) = f(y)$ if and only if $r|x-y$ [118]. This means that for an extended search space of dimension $2^n m$ with period 2^n only m equal image values could be observed. This poses a problem when applied to decision problems that merely output *yes* or *no* answers, and are thus limited to codomains with binary cardinality. The proposed method allows for an alternative view of a periodic sequence, where elements need not obey the rule $f(x) = f(y)$ if and only if $r|x-y$, but are in fact restricted to non-solution states representing at most $\frac{1}{\sqrt{2}}$ of the search space. This means that it is still possible to discover the period of a sequence that is composed, in a worst-case scenario, of $\frac{1}{\sqrt{2}}$ 0's and $1 - \frac{1}{\sqrt{2}}$ 1's.

9.3.3 Conclusions

The quantum period finding algorithm is often described in general terms as being able to discover the period of a function. In this section the question of what would be the consequences for search if no conditions were attached to such an algorithm was examined. This process required redefining basic notions such as space as well as the verification function employed in order to accommodate for the concept of periodicity. This process required redefining basic notions such as space as well as the verification function employed in order to accommodate for the concept of periodicity. Although the proposed algorithmic has its drawbacks, it is still capable of performing search when roughly at most $\frac{1}{\sqrt{2}}$ of the search space represents non-solution states. Although such a behaviour can be perceived as a drawback, it can also be described as a characteristic that must be observed when trying to determine the period of a function. This fact is missing from current discussions of the period finding algorithm. Moreover, the algorithm is capable of executing in $O(k)$ time, where k represents the number of bits required to encode the extended search space. Even though such a search space composition is not a practical assumption, the overall procedure

still raises some important questions that deserve further investigation. Namely, it would be interesting to determine if the DC component of $\mathrm{QFT}^{-1}|\psi\rangle$ can somehow be removed. Or, if it is possible to develop methods of increasing the amplitude of the periodic peaks whilst diminishing the probability of $|0\rangle$.

It is also interesting to note that Grover's algorithm is able to deliver a solution with certainty and in $O(1)$ time if the number of solution states represents a quarter of the search space [99]. Such a search space composition is only marginally different than the one that is required by the proposition described in this section. As a further matter, the quantum search algorithm performance actually decreases when the number of solution states represents at least half of the search space. In this case there are two solutions available: (1) either simply pick a random state and verify if a solution is obtained with at least 50% probability; (2) or extend the search space with non-solution states thus diminishing the overall percentage of solution states. The use of entanglement detection schemes to perform search also has the potential to deliver efficient algorithms. However, for the time being, it appears that this task will also be difficult since it depends on the mathematical properties of linear positive maps λ, which have not been operationally characterized. This proposition shares with the former the use of quantum Fourier transform, although it does not rely on an amplitude amplification scheme, and with the latter the notion of space decomposition to perform search. Given the diversity of methods employed by the aforementioned search algorithms and the type of performance they deliver, it is not inconceivable to conjecture that search will remain a hard computational task in the foreseeable future.

Chapter 10

Conclusions and Reflections

Quantum computation can trace its origins to Feynman's seminal work [72] questioning the apparent difficulties probabilistic Turing machines had in simulating quantum physical systems. Since then an elaborated body of scientific work has extended our understanding of this computational paradigm. Up until now, the field of quantum computation has been largely dominated by the physics and mathematics communities. From the experience gathered during the last couple of years I believe this to be due to a combination of factors involving: (1) the physics background required; (2) the novelty of the theme; (3) an inherent different approach to computation that clashes with what is currently performed and (4) a largely theoretical framework, lacking any viable practical mechanism for implementing quantum computation. This last factor contributed for some members of the computer science community to view the field with a certain skepticism. My background on classical computer science enables a different perspective, which has the potential to help to bridge the gap between classical and quantum computation. In doing so, I hope to contribute in a meaningful way for a more complete understanding of some of the dynamics associated with quantum computation. However, the scientific community is still very far from having a detailed picture on the powers and limitations, of quantum computation devices.

The onset of this work started with a very simple question, namely, how to perform tree search, a task that is recurrent in artificial intelligence, in a quantum manner? I knew that quantum equivalents of the Turing machine existed. Accordingly, this seemed to provide an apparently good starting point to my research. The rationale was simple: present day computers are able to perform tree search and they essentially mimic the capabilities of Turing machines. Therefore it would be a reasonable assumption to perform the corresponding quantum mapping. However, further analysis revealed that such an assumption was in fact

erroneous. Multiple definitions of Turing machines existed, which were also not directly comparable. The Turing models considered included Deutsch's model [57], which was inefficient and later proven to be wrong in [158], to Bernsteins and Vazirani's improved model [27] to Perdrix measurement based devices (described in [171] and [170]). Besides the variety of exiting models, these were also inherently complex making it difficult to understand not only the associated dynamics but also any potential performance gains. As a result, although it was possible to express such computations by quantum Turing machines, it would be wise to consider other methods first.

Alternatively, another possibility consisted in exploring amplitude amplification schemes such as the ones described in Grover's algorithm [82] and in quantum random walks [190]. These approaches were specifically focused on search, which was a central part of the problem that was being tackled. However, by choosing such methods some of the computational character embedded in Turing machines would also be lost. Grover's algorithm in particular seemed ideally suited since it searched a space of possibilities. The only issue that remained was how exactly to express an abstract form of search where symbolical conditions need to be verified in order to trigger an adequate action.

It was also decided that it would be best to base any initial approach on a specific problem instance, namely the sliding block puzzle. Besides understanding the dynamics of the quantum search algorithm, such an approach would require developing a purposely built unitary operator with an oracle form. Given that unitary operators represent bijections and that such functions can be obtained by reversible computation, logic dictated that this would be an ideal place to start. By focusing on developing the required reversible circuitry it was possible to take a less theoretical and more practical approach to the problem. It would also enable for an understanding of the requirements associated with reversible computation and how such circuitry could be employed in order to obtain the equivalent unitary operator. Once the initial circuit operator had been developed it would be a relatively easy task to store the result and undo the computation in order to obtain the proper oracle. Such a practical approach was fundamental in gaining a firm understanding of how to represent unitary operators through operator composition and basic mathematical operators. In addition, it also made possible to understand how the dimensions and complexity of unitary operators were dependent on the corresponding reversible circuit. Namely, each additional bit would imply an exponential growth in the dimensions of the unitary operator.

Furthermore, a comparison of how the number of quantum iterations fared against the classical one was also performed. The overall conclusion reached was that both approaches differ by a quadratic factor that is dependent on the number of initial states, respectively, $\sqrt{|S_i|}$. Each additional bit that the system requires implies a $\frac{1}{\sqrt{2}}$ performance penalty in the worst

case complexity. By studying the dynamics of such a ratio it was possible to determine general boundary conditions, which were expressed as dimensions of the lengths of the registers employed. Although this initial analysis hinted at some of the variables influencing overall performance it did not take into account concepts that are relevant to tree search. Accordingly, a more comprehensive examination was required, some of the crucial points examined included: (1) the reverberations of contemplating the use of non-constant branching factors and (2) determining the consequences of incorporating a heuristic perspective into a quantum tree search model. As a result, two variables were determined that influence the performance of performing tree search in such a fashion, namely, the average and maximum branching factor, respectively b_{avg} and b_{max}. It was also determined that such a strategy generally works best with balanced trees. This result is due to $b_{max} \approx b_{avg}$, which allows for a more complete use of all possible combinations represented in the quantum superposition, thus minimizing the number of times Grover's iterate needs to be applied.

In addition, the system performance also exhibits a ladder effect that reflects the overall number of bits required for the average branching factor. Regarding the use of heuristic functions, it was shown how to fine tune a unitary operator in order to produce a set of states that fall below a certain evaluation level T. Such a threshold represents an estimate of the cost to reach a goal state from the last node obtained after having applied a sequence of productions. The concept of the heuristic function can also be used in order to mark a quarter of the search space as goal states. This procedure enables one to obtain with a single application of Grover's iterate one of those states, whilst classically each state evaluated would have a $\frac{3}{4}$ probability of not being a marked state, which for a d-level depth search would mean a total probability of $\left(\frac{3}{4}\right)^d$ chance of not obtaining a marked state.

Starting with a concrete example allowed for a foundation to be established, which would later be used to develop a more stable set of theoretical definitions. The initial approach lacked the mathematical formalism that is characteristic when defining models of computation. As a result, a re-examination of the proposed mechanisms from a theoretical perspective was required. Initially, a choice was made to focus on presenting a formal set of definitions specifying the classical behaviour of production systems. Once the classical behaviour had been defined it was possible to re-address the issue of reversibility, but this time the focus was placed on developing a model for a reversible production system, which was a simple mapping of Bennett's reversible Turing machine. This procedure allowed for a set of definitions to be obtained specifying the requirements and behaviour associated with a reversible production system. This allowed for the presentation of a probabilistic control strategy, which enabled for a mapping from real-valued probabilities to complex quantum amplitudes. Solving the issues of reversibility was important before defining any type of probabilistic control strategies for production systems. This process guaranteed that the conditions required further

along by unitary operators would be ensured.

I also realized that a specific field of quantum computation, namely, quantum random walks had interesting parallels with the problem being tackled. A significant portion of the literature focused on analysing the dynamics of quantum random walks applied to graphs alongside the corresponding hitting and spreading times. However, only a single reference, respectively [190], was found that focused on obtaining a marked stated. This approach also possessed some similarities with Grover's search, namely, the quantum coin employed was in fact Grover's iterate. Logically, I considered if such an approach could be adapted in order to: (1) perform tree search and (2) store the sequence of transitions performed that led to goal states. This process would simply require storing the corresponding edge choses at a particular time t during the walk in an auxiliary register. Given that at each time step the walk is allowed to choose from a multitude of directions, and that the complexity was still $O(\sqrt{N})$ this would imply a register whose length grew in accordance. Furthermore, given the randomness nature of the walk it would also not be possible to guarantee that the path obtained was the shortest or even free of cycles. However, it was possible to perform a comparison of the total states generated by both approaches. This allowed me to conclude that the randoms walks approach could be perceived as searching an extra level of depth comparatively with the model proposed. However, both approaches are not without their merits: quantum random walks are ideally suited to evaluate parent nodes that are calculated as a function of children, whilst the approach inspired by production system theory delivers an increase in performance when the children are calculated solely as a function of the parent nodes involved.

More recently, quantum random walks were applied to calculate the value of balanced NAND trees in [67]. In such a structure, each node of the tree corresponds to a binary NAND gate that is recursively calculated as a function of its children up to the leafs that are assigned a specific bit value. The overall objective is to calculate the value of the root node. The algorithm devised by the authors makes use of a continuous time quantum walk with time $O(N^{\frac{1}{2}+o(1)})$. The optimal classical randomized algorithm for NAND tree evaluation executes in $\Theta(N^{0.7537})$ time, where N is the number of leafs (please refer to [194], [184] and [185]). The quantum NAND algorithm inspired an approach to evaluate AND-OR trees that depends on the equivalent NAND tree structure [9]. The authors prove that there exists a bounded-error $N^{\frac{1}{2}+o(1)}$-time quantum algorithm based on a discrete-time quantum random walk that is close to optimal. The authors also prove that for balanced or approximately balanced trees the algorithm executes in time $O(N^{\frac{1}{2}})$, which is optimal. The algorithm first starts by performing a random walk on the tree in time $O(\sqrt{N})$, which is the number of steps required to evaluate the root node. During this time, the algorithm also performs a dynamic change of the adjacency operator H associated with the underlying NAND tree. The authors

then proceed to demonstrate that it is possible to detect these changes in the spectrum of the operator based on the underlying value of the root. Namely, if the root evaluates to 0 then the associated phase will be 0 with a certain probability. However, if the root evaluates to 1 then the phase will be different than zero. All that is required is that the phase estimation procedure has sufficient precision (namely $O(\frac{1}{\sqrt{N}})$ bits) in order to allow for this distinction.

This algorithm led to the development of a quantum procedure for evaluating Min-Max trees [50]. The authors devise a method for evaluating Min-Max trees based on examining the entire tree as an AND-OR tree with different thresholds. They are able to achieve a $O(\sqrt{N}\log N)$ time, which is the result of a logarithmic slowdown of the original AND-OR quantum tree evaluation. Although it is not explicitly mentioned most of these approaches only focus on evaluating a tree, and do not take into account the time or space required to generate the respective structure. Clearly, generating such an input through traditional mechanisms will still require an exponential amount of time. This is a significant difference from the approach discussed in this thesis, since the superposition principle employed enables the construction of an exponential-growth input using polynomial resources without hindering the original speedups. In terms of space required, the adjacency matrix employed will require $|V|^2 \approx b^{2d}$ elements. By comparison, the C operator described in Chapter 6 requires $2^{2(\alpha+\beta+\gamma)}$ space. If operator C is perceived as acting upon a branching factor of dimension 2^{α} then both operators will have similar sizes when $d = \frac{\beta+\gamma}{2}$.

Parallel to these initiatives a simple extension to the initial quantum production system proposal was also envisaged in order to yield an iterative version of the production system, which would still deliver the original quadratic speedup. Also, such a model of computation would be ideal to tackle the quantum halting issue that was highlighted in [158]. In the halting issue each time an observer wishes to verify if the computation has halted he needs to measure a halt qubit. As a result the interference patterns are irremediably changed. Therefore the original interference pattern cannot be employed if the computation is to proceed from that point onwards. The only way to obtain the original pattern is to restart the computation, but the problem is not solved since a measurement will still eventually need to be made. Developing such an extension required incorporating into the model the capabilities that are characteristic of μ-recursive functions, a class of functions, which allows for the possibility of non-termination. The extensions that were incorporated allowed for the model to be endowed with an iterative character. In the end it was possible to obtain a model of quantum computation that: (1) allows for a detection of halt states without interfering with the final result of a computation; (2) contemplates the possibility of non-termination and (3) allows for an inherent speedup to occur during computations susceptible of parallelization. Additionally, the model can also be employed in order to simulate classical

Turing machines.

Given that search is such a crucial part of this work it was also important to consider other methods besides the one described by Grover. One such approach was to explore forms of decomposing a quantum search space. Ideally, some form of space analysis would need to be performed alongside the ability to determine if a solution was present or not. Two possibilities existed, one exploiting quantum entanglement detection schemes and the other utilising the notion of periodicity. From a computational perspective each time an oracle evaluates a quantum superposition and marks a state as being a solution it also entangles the superposition. Consequently, it would be opportune to determine if quantum entanglement detection schemes existed. Several methods were found to exist, of which [108] seemed particularly well suited since it offered a method for direct detection of quantum entanglement. Curiously, the method's original motivation placed a strong emphasis on an experimentally viable direct detection of quantum entanglement. As a result, such an application would be considered novelty in the context of quantum computation. Subsequently, a recursive definition of a unitary operator was developed that was capable of dividing the search space into equal-length intervals.

This type of operators was labeled as entanglement inducing operators, which were to be applied to a superposition state followed by the entanglement detection method. The method was developed in order to produce a solution state with certainty. This proposal sacrificed space for speed, since the number of state copies L grew as a function of the original dimension of the search space. Such an entanglement based search method was also dependent on the mathematical properties of linear positive maps Λ, which have not been operationally characterized. Whether such a Λ can be easily determined remains an open question and would have far-reaching consequences for quantum computational theory. However, if this were to be the case then the entanglement search method would search an exponential-growth search space in polynomial time.

Besides trying to perform search by entanglement detection the question of how periodicity could affect quantum search was also posed. The original idea supporting this question was based on Shor's factoring algorithm that was able to deliver a superpolynomial speedup by employing the quantum Fourier transform to analyse the period of a signal. This behaviour led to the question if it would be possible to employ the same principles but this time applied to the specific constraints of search. By extending the search space in order to accommodate multiple repetitions of the original one it was possible to develop a recursive algorithm capable of running in time $O(k)$, where k represents the number of bits employed by the extended space. In addition, it was also showed that the algorithm is only able to work correctly when at most $\frac{1}{\sqrt{2}}$ of the search space is composed of non-solution states.

These statements form important contributions proving that the proposition for an alternative model of quantum computation based on production system theory has merit. The quantum production system employs the concept of tree search to describe computation in a quantum fashion. I believe the solution is simple and elegant in contrast with the complexity of the existing proposals for quantum Turing machines. Furthermore, these models are also not without some issues, the most prominent of which is the quantum halting matter. In comparison, the model is capable of dealing with the issue of quantum halting without interfering with the overall interference pattern. Such behaviour is ideal in the context of non-terminating computation. In addition, the proposed approach is also able to benefit from an explicit quadratic speedup. The only condition required for this speedup is that a non-unary branching factor is employed. Accordingly, it is my believe that the results presented form important contributions that lend support to this thesis.

References

[1] Samson Abramsky. A structural approach to reversible computation. *Theoretical Computer Science*, 347(3):441 – 464, 2005.

[2] Samson Abramsky, Artemov S., R.A. Shore, and A.S. Troelstra. *Handbook of computability theory.* Elsevier, Amsterdam, Netherlands, 1999.

[3] Dorit Aharonov. *Noisy Quantum Computation.* PhD thesis, Hebrew University, July 1999.

[4] Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. Quantum walks on graphs. In *Proceedings of ACM Symposium on Theory of Computation (STOC'01)*, pages 50–59, July 2001.

[5] Y. Aharonov, L. Davidovich, and N. Zagury. Quantum random walks. *Phys. Rev. A*, 48(2):1687–1690, Aug 1993.

[6] A. Ambainis. Quantum lower bounds by quantum arguments. *eprint arXiv:quant-ph/0002066*, February 2000.

[7] A. Ambainis. Quantum search algorithms. *SIGACT News*, 35(2):22–35, 2004.

[8] A. Ambainis. A nearly optimal discrete query quantum algorithm for evaluating NAND formulas. *ArXiv e-prints*, April 2007.

[9] A. Ambainis, A.M. Childs, and B.W. Reichardt. 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*, pages 363 –372, oct. 2007.

[10] Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1:507, 2003.

[11] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37:210, 2007.

[12] Andris Ambainis, Eric Bach, Ashwin Nayak, Ashvin Vishwanath, and John Watrous. One-dimensional quantum walks. In *ACM Symposium on Theory of Computing*, pages 37–49, 2001.

[13] Andris Ambainis and Rusins Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, pages 332–, Washington, DC, USA, 1998. IEEE Computer Society.

[14] Andris Ambainis, Julia Kempe, and Alexander Rivosh. Coins make quantum walks faster, 2005.

[15] Josephine M. Ammer, Michael Frank, Tom Knight, Nicole Love, Norman Margolus, and Carlin Vieri. A scalable reversible computer in silicon. In C. S. Calude, John L. Casti, and M. J. Dinneen, editors, *Unconventional Models of Computation, 1st edition*, Cambridge, Massachusetts, USA, 1998. Springer-Verlag Singapore Pte. Limited.

[16] John R. Anderson. *The Architecture of Cognition.* Harvard University Press, Cambridge, Massachusetts, USA, 1983.

[17] Howard Anton and Chris Rorres. *Elementary Linear Algebra: Applications Version Eighth Edition.* John Wiley and Sons, Inc., 2000.

[18] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–67, 1995.

[19] Adriano Barenco, Andre Berthiaume, David Deutsch, Artur Ekert, Richard Jozsa, and Chiara Macchiavello. Stabilization of quantum computations by symmetrization. *SIAM Journal on Computing*, 26(5):1541–1557, 1997.

[20] G.M. Baudet. *The Design and Analysis of Algorithms for Asynchronous Multiprocessors.* PhD thesis, Carnegie Mellon University, Pittsburgh, Pensilvania USA, 1978.

[21] R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum Lower Bounds by Polynomials. *ArXiv Quantum Physics e-prints*, February 1998.

[22] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48:778–797, July 2001.

[23] C.H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, November 1973.

[24] Charles H. Bennett. Notes on the history of reversible computation. *IBM J. Res. Dev.*, 32(1):16–23, 1988.

[25] Charles H. Bennett. Notes on landauer's principle, reversible computation, and maxwell's demon. *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, 34(3):501 – 510, 2003. Quantum Information and Computation.

[26] Charles H. Bennett, Ethan Bernstein, Gilles Brassard, and Umesh Vazirani. Strengths and weaknesses of quantum computing, 1997.

[27] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, New York, NY, USA, 1993. ACM.

[28] N. Bourbaki. *Elements of mathematics: theory of sets*. Number vol. 1 in Elements of mathematics. Springer, Berlin, Germany, 2004.

[29] Michel Boyer, Gilles Brassard, Peter Hoeyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46:493, 1998.

[30] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. Quantum Amplitude Amplification and Estimation. *eprint arXiv:quant-ph/0005055*, May 2000.

[31] G. Brassard, P. Hoyer, and A. Tapp. Quantum Algorithm for the Collision Problem. *eprint arXiv:quant-ph/9705002*, May 1997.

[32] Mark G. Brockington. A taxonomy of parallel game-tree search algorithms. *ICCA Journal*, 19(3):162–175, September 1996.

[33] Mark G. Brockington and Jonathan Schaeffer. Aphid game-tree search. *Journal of Parallel and Distributed Computing*, 6:90–114, 1997.

[34] Dan Browne, Elham Kashefi, and Simon Perdrix. Computational depth complexity of measurement-based quantum computation. In Wim Dam, VivienM. Kendon, and Simone Severini, editors, *Theory of Quantum Computation, Communication, and Cryptography*, volume 6519 of *Lecture Notes in Computer Science*, pages 35–46. Springer Berlin Heidelberg, 2011.

[35] Bruce G. Buchanan and Edward H. Shortliffe. *Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[36] H. Buhrman, C. Durr, M. Heiligman, P. Hoyer, F. Magniez, M. Santha, and R. de Wolf. Quantum algorithms for element distinctness. *Computational Complexity, Annual IEEE Conference on*, 0:0131, 2001.

[37] Harry Buhrman, Richard Cleve, and Avi Wigderson. Quantum vs. classical communication and computation. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 63–68, New York, NY, USA, 1998. ACM.

[38] Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu. Deep blue. *Artificial Intelligence*, 134:57–83, 2002.

[39] A. Chi-Chih Yao. Quantum circuit complexity. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 352 –361, nov 1993.

[40] A. M. Childs, B. W. Reichardt, R. Spalek, and S. Zhang. 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*, March 2007.

[41] Andrew M. Childs. *LECTURE 14: Discrete-time quantum walk*. University of Waterloo, 2011.

[42] Andrew M. Childs, Richard Cleve, E. Deotto, Edward Farhi, Sam Gutmann, and D.A. Spielman. Exponential algorithmic speedup by quantum walk. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003)*, pages 59–68, September 2003.

[43] Andrew M. Childs, Richard Cleve, Stephen P. Jordan, and David Yonge-Mallo. Discrete-query quantum algorithm for nand trees. *Theory of Computing*, 5(1):119–123, 2009.

[44] Andrew M. Childs, Edward Farhi, and Sam Gutmann. An example of the difference between quantum and classical random walks. *Quantum Information Processing*, 1(1):35–43, 2002.

[45] B.-S. Choi and V. Korepin. Quantum Partial Search of a Database with Several Target Items. *ArXiv Quantum Physics e-prints*, August 2006.

[46] Isaac L. Chuang, Neil Gershenfeld, and Mark Kubinec. Experimental implementation of fast quantum searching. *Phys. Rev. Lett.*, 80(15):3408–3411, Apr 1998.

[47] Alonzo Church. A note on the entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, March 1936.

[48] Alonzo Church. *The Calculi of Lambda-Conversion*. Annals of Mathematics Studies. Princeton University Press, Princeton, New Jersey, USA, 1941.

[49] Richard Cleve, A Ekert, C. Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society A*, 454(1969):339–354, January 1998.

[50] Richard Cleve, Dmitry Gavinsky, and David Yonge-Mallo. Quantum algorithms for evaluating min-max trees. In Y. Kawano and M. Mosca, editors, *Proceedings of Theory of Quantum Computation, Communication, and Cryptography (TQC 2008)*, volume 5106, pages 11–15. Springer Berlin / Heidelberg, 2008.

[51] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[52] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 2/e.* MIT Press, 2001.

[53] Martin Davis. *The Universal Computer: The Road from Leibniz to Turing.* Norton, New York, NY, USA, 2000.

[54] Martin Davis. *Engines of logic: mathematicians and the origin of the computer.* Norton, New York, NY, USA, 2001.

[55] Morris H. DeGroot and Mark J. Schervish. *Probability and statistics.* Addison-Wesley, 3 edition, 2002.

[56] D. Deutsch and R. Jozsa. Rapid Solution of Problems by Quantum Computation. *Royal Society of London Proceedings Series A*, 439:553–558, December 1992.

[57] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. In *Proceedings of the Royal Society of London- Series A, Mathematical and Physical Sciences*, volume 400, pages 97–117, 1985.

[58] David Deutsch. Quantum computational networks. In *Proceedings of the Royal Society of London A*, volume 425, pages 73–90, 1989.

[59] Paul Adrien Maurice Dirac. A new notation for quantum mechanics. In *Proceedings of the Cambridge Philosophical Society*, volume 35, pages 416–418, 1939.

[60] Paul Adrien Maurice Dirac. *The Principles of Quantum Mechanics - Volume 27 of International series of monographs on physics (Oxford, England) Oxford science publications.* Oxford University Press, 1981.

[61] Ross Duncan and Simon Perdrix. Rewriting measurement-based quantum computations with generalised flow. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and PaulG. Spirakis, editors, *Automata, Languages and Programming*, volume 6199 of *Lecture Notes in Computer Science*, pages 285–296. Springer Berlin Heidelberg, 2010.

[62] Carl Ebeling. *All the Right Moves: A VLSI Architecture for Chess.* PhD thesis, Carnegie Mellon University, Pittsburgh, Pensilvania USA, 1986.

[63] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[64] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47(10):777–780, May 1935.

[65] Artur Ekert and Richard Jozsa. Quantum computation and shor's factoring algorithm. *Rev. Mod. Phys.*, 68(3):733–753, Jul 1996.

[66] G.W. Ernst and Allen Newell. *GPS: a case study in generality and problem solving.* Academic Press, New York, NY, USA, 1969.

[67] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum algorithm for the hamiltonian nand tree. *Theory of Computing*, 4(1):169–190, 2008.

[68] Edward Farhi and Sam Gutmann. Quantum computation and decision trees. *Phys. Rev. A*, 58(2):915–928, Aug 1998.

[69] Rainer Feldmann. *Game Tree Search on Massively Parallel Systems.* PhD thesis, University of Paderborn, 1993.

[70] E. W. Felten and S. W. Otto. Chess on a hypercube. In *Proceedings of the third conference on Hypercube concurrent computers and applications*, pages 1329–1341, New York, NY, USA, 1988. ACM.

[71] Chris Ferguson and Richard E. Korf. Distributed tree search and its application to alpha-beta pruning. In *AAAI-88 Proceedings*, 1988.

[72] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.

[73] Jaromír Fiurášek. Structural physical approximations of unphysical maps and generalized quantum measurements. *Phys. Rev. A*, 66(5):052315, Nov 2002.

[74] C. Forgy. Ops5 user's manual cmu-cs-81-135. Technical report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pensilvania USA, 1981.

[75] Stan Franklin. *Artificial Minds.* MIT Press, 1997.

[76] Martin Gardner. The hypnotic fascination of sliding-block puzzles. *Scientific American*, 210:122–130, 1964.

[77] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman, 1979.

[78] S. Gharibian. Strong NP-Hardness of the Quantum Separability Problem. *ArXiv e-prints*, October 2008.

[79] Warren Gilchrist. *Statistical Modelling with Quantile Functions*. Chapman and Hall/CRC, 2000.

[80] L. K. Grover and J. Radhakrishnan. Is partial quantum search of a database any easier? *eprint arXiv:quant-ph/0407122*, July 2004.

[81] Lov Grover and Terry Rudolph. How significant are the known collision and element distinctness quantum algorithms? *Quantum Information and Computation*, 4:201, 2004.

[82] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, New York, NY, USA, 1996. ACM.

[83] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79:325, 1997.

[84] Lov K. Grover. A framework for fast quantum mechanical algorithms. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 53–62, New York, NY, USA, 1998. ACM.

[85] Lov K. Grover. Quantum computers can search rapidly by using almost any transformation. *Phys. Rev. Lett.*, 80(19):4329–4332, May 1998.

[86] Lov K. Grover. Quantum search on structured problems. *Chaos, Solitons & Fractals*, 10(10):1695 – 1705, 1999.

[87] Lov K. Grover. Trade-offs in the quantum search algorithm. *Phys. Rev. A*, 66(5):052314, Nov 2002.

[88] Lov K. Grover. Fixed-point quantum search. *Phys. Rev. Lett.*, 95(15):150501, Oct 2005.

[89] O. Gühne and G. Tóth. Entanglement detection. *Physics Reports*, 474:1–75, April 2009.

[90] Leonid Gurvits. Classical deterministic complexity of edmonds' problem and quantum entanglement. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 10–19, New York, NY, USA, 2003. ACM.

[91] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, Cambridge, MA, USA, 1997.

[92] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[93] Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, July 1983.

[94] T.P. Hart and D. J. Edwards. The tree prune (tp) algorithm. artificial intelligence project memo 30. Technical report, Massschusetts Institute of Technology, Cambridge, Massachusetts, 1961.

[95] Robert A. Hearn. The complexity of sliding-block puzzles and plank puzzles. In *Tribute to a mathemagician*, pages 1–11. A K Peters, 2005.

[96] Robert A. Hearn and Erik D. Demaine. Pspace-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72 – 96, 2005. Game Theory Meets Theoretical Computer Science.

[97] David Hilbert. Mathematische probleme. In Göttingen, editor, *Proceedings of the International Congress of Mathematicians in Paris 1900*, pages 253–297, 1900.

[98] Mika Hirvensalo. On quantum computation. Technical report, Turku Centre for Computer Science, 1997.

[99] Mika Hirvensalo. *Quantum Computing.* Springer-Verlag, Berlin Heidelberg, 2004.

[100] Tad Hogg. Quantum computing and phase transitions in combinatorial search. *J. Artif. Int. Res.*, 4(1):91–128, 1996.

[101] Tad Hogg. A framework for structured quantum search. *PHYSICA D*, 120:102, 1998.

[102] Tad Hogg. Quantum search heuristics. *Phys. Rev. A*, 61(5):052311, Apr 2000.

[103] Tad Hogg, Bernardo A. Huberman, and Colin P. Williams. Phase transitions and the search problem. *Artif. Intell.*, 81(1-2):1–15, 1996.

[104] John Hopcroft and Robert Tarjan. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, 1973.

[105] Edward Hordern. *Sliding Piece Puzzles*. Recreations in Mathematics, No 4. Oxford University Press, USA, 1987.

[106] Michal Horodecki, Pawel Horodecki, and Ryszard Horodecki. Separability of mixed states: necessary and sufficient conditions. *Physics Letters A*, 223(1-2):1 – 8, 1996.

[107] Michal Horodecki, Pawel Horodecki, and Ryszard Horodecki. Separability of n-particle mixed states: necessary and sufficient conditions in terms of linear maps. *Physics Letters A*, 283(1-2):1 – 7, 2001.

[108] Paweł Horodecki and Artur Ekert. Method for direct detection of quantum entanglement. *Phys. Rev. Lett.*, 89(12):127902, Aug 2002.

[109] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki. Quantum entanglement. *ArXiv Quantum Physics e-prints*, February 2007.

[110] Feng-hsiung Hsu. *Large Scale Parallilization of Alpha-Beta Search: An Algorithmic and Architectural Study*. PhD thesis, Carnegie Mellon University, Pittsburgh, USA, 1990.

[111] Feng-hsiung Hsu. Ibm's deep blue chess grandmaster chips. *Micro, IEEE*, 19(2):70–82, March-April 1999.

[112] Feng-hsiung Hsu. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton University Press, 2002.

[113] Chia-Ren Hu. A family of sure-success quantum algorithms for solving a generalized grover search problem, 2002.

[114] B.D. Hughes. *Random Walks and Random Environments: Volume 1: Random Walks*. Random Walks and Random Environments. Oxford University Press, USA, 1995.

[115] L. M. Ioannou. Computational complexity of the quantum separability problem. *ArXiv Quantum Physics e-prints*, March 2006.

[116] Lawrence M. Ioannou and Benjamin C. Travaglione. Quantum separability and entanglement detection via entanglement-witness search and global optimization. *Phys. Rev. A*, 73(5):052314, May 2006.

[117] S. Iriyama, M. Ohya, and I. Volovich. Generalized Quantum Turing Machine and its Application to the SAT Chaos Algorithm. *eprint arXiv:quant-ph/0405191*, May 2004.

[118] Philip R. Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford University Press, USA, 2007.

[119] Julia Kempe. Quantum random walks - an introductory overview. *Contemporary Physics*, 44:307, 2003.

[120] M. Keyl and R. F. Werner. Estimating the spectrum of a density operator. *Phys. Rev. A*, 64(5):052311, Oct 2001.

[121] Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *Proceedings of the 38th IEEE Conference on Foundations of Computer Science*, pages 66–75, 1997.

[122] V. E. Korepin and Y. Xu. Hierarchical Quantum Search. *International Journal of Modern Physics B*, 21:5187–5205, 2007.

[123] Vladimir Korepin and Lov Grover. Simple algorithm for partial quantum search. *Quantum Information Processing*, 5:5–10, 2006. 10.1007/s11128-005-0004-z.

[124] Richard E. Korf. Depth-first iterative-deepening : An optimal admissible tree search. *Artificial Intelligence*, 27(1):97 – 109, 1985.

[125] Richard E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189 – 211, 1990.

[126] Richard E. Korf. Best-first search with limited memory. *UCLA Computer Science Annual*, 1991.

[127] Richard E. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[128] Philipp Krammer. Quantum entanglement - detection, classification, and quantification. Master's thesis, University of Vienna, October 2005.

[129] B.C. Kuzmaul. *Synchronized MIMD Computing*. PhD thesis, Masschussets Institute of Technology, Cambridge, Massachusetts, USA, 1994.

[130] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[131] John E. Laird, Paul S. Rosenbloom, and Allen Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46, 03 1986.

[132] R. Landauer. Information is physical. In *Physics and Computation, 1992. PhysComp '92., Workshop on*, pages 1–4, Oct 1992.

[133] Rolph Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[134] A. Lenstra, H. Lenstra, M. Manasse, and J. Pollard. The number field sieve. In Arjen Lenstra and Hendrik Lenstra, editors, *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 11–42. Springer Berlin / Heidelberg, 1993. 10.1007/BFb0091537.

[135] Arjen Lenstra, Hendrik Lenstra, M.S. Manasse, and J. Pollard. The factorisation of the ninth fermat number. *Math. Comp.*, 61:319–349, 1993.

[136] M. Lewenstein, B. Kraus, J. I. Cirac, and P. Horodecki. Optimization of entanglement witnesses. *Phys. Rev. A*, 62(5):052310, Oct 2000.

[137] M. Lewenstein, B. Kraus, P. Horodecki, and J. I. Cirac. Characterization of separable states and entanglement witnesses. *Physical Review A*, 63(4):044304, March 2001.

[138] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[139] N. Linden and S. Popescu. The Halting Problem for Quantum Computers. *eprint arXiv:quant-ph/9806054*, June 1998.

[140] Seth Lloyd. Quantum-mechanical computers. *Scientific American: The Solid-State Century*, 1997.

[141] G. L. Long. Grover algorithm with zero theoretical failure rate. *Phys. Rev. A*, 64(2):022307, Jul 2001.

[142] Erik Lucero, R. Barends, Y. Chen, J. Kelly, M. Mariantoni, A. Megrant, P. O'Malley, D. Sank, A. Vainsencher, J. Wenner, T. White, Y. Yin, A. N. Cleland, and John M. Martinis. Computing prime factors with a josephson phase qubit quantum processor. *Nat Phys*, advance online publication:–, 08 2012.

[143] George F. Luger and William A. Stubblefield. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving: Second Edition*. The Benjamin/Cummings Publishing Company, Inc, Menlo Park, CA, USA, 1993.

[144] M.Morris Mano and Charles R. Kime. *Logic and Computer Design Fundamentals: 2nd Edition*. Prentice Hall, Englewood Cliffs, NJ, USA, 2002.

[145] Andrey Markov. *The theory of algorithms*. National Academy of Sciences, USSR, 1954.

[146] T. A. Marsland and M. Campbell. Parallel search of strongly ordered game trees. *ACM Comput. Surv.*, 14(4):533–551, 1982.

[147] T. A. Marsland and Y. Gao. Speculative parallelism improves search? Technical Report 95-05, Department of Cumputing Science, University of Alberta, Edmonton, Alberta, Canada, 1995.

[148] T. A. Marsland, M Olafsson, and J Schaeffer. Multiprocessor tree-search experiments. *Pergamon Press, Inc. Chess Series*, pages 37–51, 1986.

[149] C Mereghetti and B. Palano. On the size of one-way quantum finite automata with periodic behaviors. *Theoretical Informatics and Applications*, 36(3):277–291, 2002.

[150] David Meyer. From quantum cellular automata to quantum lattice gases. *Journal of Statistical Physics*, 85(5):551–574, 12 1996.

[151] M. Mhalla and S. Perdrix. Graph States, Pivot Minor, and Universality of (X,Z)-measurements. *ArXiv e-prints*, February 2012.

[152] T. Miyadera and M. Ohya. On Halting Process of Quantum Turing Machine. *eprint arXiv:quant-ph/0302051*, February 2003.

[153] C. Moore and A. Russell. Quantum Walks on the Hypercube. *eprint arXiv:quant-ph/0104137*, April 2001.

[154] Cristopher Moore and James P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 237, 2000.

[155] E.F. Moore. The shortest path through a maze. In *Proceeding of an International Symposium on the Theory of Switching, Part II*, pages 285–292, Cambridge, Massachusetts, 1959. Harvard University Press.

[156] M. Muller. *Quantum Kolmogorov Complexity and the Quantum Turing Machine*. PhD thesis, Technical University of Berlin, 2007.

[157] M. Muller. Strongly universal quantum turing machines and invariance of kolmogorov complexity. *Information Theory, IEEE Transactions on*, 54(2):763 –780, feb. 2008.

[158] John M. Myers. Can a universal quantum computer be fully quantum? *Phys. Rev. Lett.*, 78(9):1823–1824, Mar 1997.

[159] Ashwin Nayak and Ashvin Vishwanath. Quantum walk on the line. Technical report, DIMACS Technical Report, 2000.

[160] Allen Newell. A guide to the general problem-solver program gps-2-2. Technical Report RM-3337-PR, RAND Corporation, Santa Monica, CA, USA, 1963.

[161] Allen Newell and Ernst. G. The search for generality. In *Information Processing 1965: Proceeding of IFIP Congress*, volume 1, pages 17–24, Chicago, 1965. Spartan.

[162] Allen Newell, J.C. Shaw, and Herbert Alexander Simon. Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264, 1959.

[163] Allen Newell and Herbert Alexander Simon. *Human problem solving*. Prentice Hall, Englewood Cliffs, NJ, USA, 1 edition, 1972.

[164] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, MA, USA, 2000.

[165] Nils J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc, 1982.

[166] Masanao Ozawa. On the halting problem for quantum turing machines. Technical report, Kyoto University, Japan, 1998.

[167] Masanao Ozawa. Quantum nondemolition monitoring of universal quantum computers. *Phys. Rev. Lett.*, 80:631–634, Jan 1998.

[168] Masanao Ozawa. Halting of quantum turing machines. In *Unconventional Models of Computation*, volume 2509 of *Lecture Notes in Computer Science*, pages 58–65. Springer Berlin Heidelberg, 2002.

[169] C.H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[170] S. Perdrix and P. Jorrand. Measurement-Based Quantum Turing Machines and Questions of Universalities. *eprint arXiv:quant-ph/0402156*, February 2004.

[171] S. Perdrix and P. Jorrand. Measurement-Based Quantum Turing Machines and their Universality. *eprint arXiv:quant-ph/0404146*, April 2004.

[172] Simon Perdrix. Partial observation of quantum turing machines and a weaker well-formedness condition. *Electronic Notes in Theoretical Computer Science*, 270(1):99 – 111, 2011. ¡ce:title¿Proceedings of the Joint 5th International Workshop on Quantum Physics and Logic and 4th Workshop on Developments in Computational Models (QPL/DCM 2008)¡/ce:title¿.

[173] Simon Perdrix and Philippe Jorrand. Classically-controlled quantum computation. *Electronic Notes in Theoretical Computer Science*, 135(3):119 – 128, 2006. ¡ce:title¿Proceedings of the First International Workshop on Developments in Computational Models (DCM 2005)¡/ce:title¿ ¡xocs:full-name¿Developments in Computational Models 2005¡/xocs:full-name¿.

[174] Asher Peres. Separability criterion for density matrices. *Phys. Rev. Lett.*, 77(8):1413–1415, Aug 1996.

[175] Alberto Politi, Jonathan C.F. Matthews, and Jeremy L. O'Brien. Shor's quantum factoring algorithm on a photonic chip. *Science*, 325(5945):1221, 2009.

[176] J. Pollard. Factoring with cubic integers. In Arjen Lenstra and Hendrik Lenstra, editors, *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 4–10. Springer Berlin / Heidelberg, 1993. 10.1007/BFb0091536.

[177] J. Pollard. The lattice sieve. In Arjen Lenstra and Hendrik Lenstra, editors, *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 43–49. Springer Berlin / Heidelberg, 1993. 10.1007/BFb0091538.

[178] Emil Post. Formal reductions of the general combinatorial problem. *American Journal of Mathematics*, 65:197–268, 1943.

[179] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.

[180] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, May 2001.

[181] Robert Raussendorf, Daniel E. Browne, and Hans J. Briegel. Measurement-based quantum computation on cluster states. *Phys. Rev. A*, 68:022312, Aug 2003.

[182] Eleanor Rieffel and Wolfgang Polak. *Quantum Computing - A Gentle Introduction.* The MIT Press, 2011.

[183] Stuart J. Russell, Peter Norvig, John F. Canny, Douglas D. Edwards, Jitendra M. Malik, and Sebastian Thrun. *Artificial Intelligence: A Modern Approach (Second Edition).* Prentice Hall, 2003.

[184] Michael Saks and Avi Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 29–38, Washington, DC, USA, 1986. IEEE Computer Society.

[185] Miklos Santha. On the monte carlo boolean decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 6(1):75–87, 1995.

[186] André Coelho Santos, Luís Tarrataca, and Cardoso João. An analysis of navigation algorithms for smartphones using j2me. In *In Proceedings of the Second International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (Mobilware'09, Berlin-Germany, April 28-29), LNICST*, volume 7, pages 266–279. Springer, 2009.

[187] André Coelho Santos, Luís Tarrataca, and Cardoso João. Context inference for mobile applications in the upcase project. In *In Proceedings of the Second International ICST Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications (Mobilware'09, Berlin-Germany, April 28-29), LNICST*, volume 7, pages 352–365. Springer, 2009.

[188] Erwin Schrödinger. Die gegenwärtige situation in der quantenmechanik. *Naturwissenschaften*, 23(807), 1935.

[189] A. Sharma. *Theory of Automata and Formal Languages*. Laxmi Publications (P) Limited, May 2006.

[190] Neil Shenvi, Julia Kempe, and K. Birgitta Whaley. Quantum random-walk search algorithm. *Phys. Rev. A*, 67(5):052307, May 2003.

[191] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Nov 1994.

[192] M. Sipser. *Introduction to the theory of computation*. Computer Science Series. Thomson Course Technology, 2006.

[193] D.J. Slate and L. R. Atkin. Chess 4.5 - northwestern university chess program. In *Chess Skill in Man and Machine*, pages 82–118, Berlin, 1977. Springer-Verlag.

[194] Marc Snir. Lower bounds on probabilistic linear decision trees. *Theoretical Computer Science*, 38(0):69 – 82, 1985.

[195] Tom Stuart. Partial recursive functions. Technical report, University of Cambridge: Computer Laboratory: Faculty of Computer Science and Technology, 2004.

[196] Luís Tarrataca and Wicher. Quantum iterative deepening with an application to the halting problem. *PLOS One*, 2013.

[197] Luís Tarrataca and Andreas Wichert. A hierarchical sorting oracle. In Massimo Melucci, Dawei Song, and Ingo Frommholz, editors, *Proceedings of the Fifth International Quantum Interaction Symposium*, 2011.

[198] Luís Tarrataca and Andreas Wichert. Problem-solving and quantum computation. *Cognitive Computation*, 3:510–524, 2011.

[199] Luís Tarrataca and Andreas Wichert. Tree search and quantum computation. *Quantum Information Processing*, 10(4):475–500, 2011. 10.1007/s11128-010-0212-z.

[200] Luís Tarrataca and Andreas Wichert. Can quantum entanglement detection schemes improve search? *Quantum Information Processing*, 11(1):55–66, 2012. 10.1007/s11128-011-0231-4.

[201] Luís Tarrataca and Andreas Wichert. A quantum production model. *Quantum Information Processing*, 11(1):189–209, 2012. 10.1007/s11128-011-0231-4.

[202] Luís Tarrataca and Andreas Wichert. Intricacies of quantum computational paths. *Quantum Information Processing*, 12:1365–1378, 2013.

[203] Tommaso Toffoli. Reversible computing. Technical report, Massschusetts Institute of Technology, Laboratory for Computer Science, Massachusetts, MA, USA, 1980.

[204] Tommaso Toffoli. Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 632–644, London, UK, 1980. Springer-Verlag.

[205] A.M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 2, pages 260–265, 1936.

[206] Salvador Elías Venegas-Andraca. *Discrete Quantum Walks and Quantum Image Processing*. PhD thesis, University of Oxford, 2009.

[207] Salvador Elías Venegas-Andraca and S. Bose. Quantum computation and image processing: New trends in artificial intelligence. In *Proceedings of the International Conference on Artificial Intelligence IJCAI-03*, pages 1563–1564, 2003.

[208] Salvador Elías Venegas-Andraca and S. Bose. Storing, processing and retrieving an image using quantum mechanics. In *Proceedings of the 2003 SPIE Conference on Quantum Information and Quantum Computation.*, pages 1563–1564, 2003.

[209] John von Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[210] John von Neumann. *Mathematische Grund lagen der Quantenmechanic*. Springer, Berlin, 1932.

[211] John Watrous. Quantum simulations of classical random walks and undirected graph connectivity. *CoRR*, cs.CC/9812012, 1998.

[212] Andrew Whitaker. *Einstein, Bohr and the Quantum Dilemma: From Quantum Theory to Quantum Information 2nd Edition*. Cambridge University Press, 2006.

[213] Patrick Henry Winston. *Artificial Intelligence (Third Edition)*. Addison-Wesley, Reading, MA, USA, 1992.

[214] Wolfgang Woess. *Random Walks on Infinite Graphs and Groups*. Number 138 in Cambridge Tracts in Mathematics. Cambridge University Press, 2000.

[215] Mingsheng Ying. Quantum computation, quantum theory and ai. *Artificial Intelligence*, 174:162–176, 2010.

[216] S. Ying, Y. Feng, N. Yu, and M. Ying. Reachability Probabilities of Quantum Markov Chains. *ArXiv e-prints*, March 2013.

[217] Shenggang Ying and Mingsheng Ying. Removing measurements from quantum walks. *Phys. Rev. A*, 87:012337, Jan 2013.

[218] Nengkun Yu and Mingsheng Ying. Reachability and termination analysis of concurrent quantum programs. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 – Concurrency Theory*, volume 7454 of *Lecture Notes in Computer Science*, pages 69–83. Springer Berlin Heidelberg, 2012.

[219] Christof Zalka. A grover-based quantum search of optimal order for an unknown number of marked elements, 1999.

[220] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60:2746, 1999.