TÉCNICO LISBOA

Quantum Computation for Artificial Intelligence

Pedro Araújo Rosa da Costa

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Andreas Miroslaus Wichert

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Andreas Miroslaus Wichert
Member of the Committee: Prof. João Carlos Serrenho Dias Pereira

March 2017

Acknowledgments

The author would like to thank Andreas Wichert for his openness, sense of humour, guidance and support over the previous year. Additionally, special thanks go to Luísa Coheur and Catarina Moreira for their invaluable suggestions and input on improving this work. Responsibility for any errors or mistakes remains solely with the author.

Resumo

Existem métodos da computação quântica capazes de obter uma aceleração quadrática na eficiência-temporal de uma pesquisa clássica (*e.g.*, o algoritmo de Grover [25]). Em Inteligência Artificial (IA), isto equivale à divisão por dois na profundidade de procura garantindo uma solução. Infelizmente, o estudo destes modelos encontra-se frequentemente circunscrito às comunidades da matemática e física. Uma vez o aparecimento do primeiro computador quântico ser apenas uma questão de tempo, propomo-nos investigar os benefícios dos seus métodos na resolução de problemas de IA, partindo da sua perspectiva simbólica. Neste documento, apresentamos uma introdução aos princípios da computação quântica. Seguidamente, descrevemos como um algoritmo quântico/clássico, Procura em Profundidade Iterativa Quântica [56], pode ser utilizado em tarefas de IA não-triviais. São propostas duas aplicações do mesmo, passando de um nível concreto de representação para um mais abstracto, nomeadamente: um sistema de solução para instâncias de n-blocos do Blocks World; e um sistema híbrido de inferência proposicional, que a partir de frases nesta linguagem, consegue provar outras expressões. Para o primeiro, exploramos também como melhorar os requisitos de espaço, e sugerimos uma nova perspectiva no uso de heurísticas para computação quântica. O sistema de resolução do Blocks World encontra soluções num espaço de estados de tamanho N em tempo $O(\sqrt{N})$. Em semelhança, para bases de dados proposicionais de grandes proporções, o sistema de inferência apresentado obtém provas em tempo $O(\sqrt{N})$, sendo N o número de sequências de dedução possíveis. Ambos exibem aceleração quadrática relativamente aos seus congéneres exaustivos clássicos, cujos requisitos temporais se estendem a $O(N)$ verificações.

Palavras-chave: computação quântica, inteligência artificial, resolução de problemas, blocks world, inferência automatizada

Abstract

Quantum computation methods (*e.g.*, Grover's algorithm [25]) are capable of quadratically improving the time-performance of classical search procedures. For Artificial Intelligence (AI) problems, this is roughly equivalent to halving the depth necessary for guaranteeing a solution. Unfortunately, study of these models is frequently left to those in the mathematics/physics communities, despite the aforementioned advantages. However, quantum computers will one day become a reality. Thus, we propose exploring how their methodologies may aid us in solving problems in AI, assuming the latter's symbolic outlook. In this document, we provide scientists interested in quantum computing models with a brief introduction to operating principles, upon which we describe how a hybrid quantum/classical algorithm, Quantum Iterative Deepening Search (QIDS) [56], can be employed for solving interesting AI tasks within the standard-circuit quantum computation paradigm. We present two different applications, one possessing a more concrete representation, another that is more abstract, namely: a solver for the n-blocks generalization of Blocks World; and a hybrid system of inference, able to prove statements in Propositional Calculus given a knowledge-base of assertions in that language. We also investigate how space-efficiency may be improved in the former, and suggest a new heuristic perspective for quantum computing frameworks. Our n-blocks solver can find solutions within an N-sized state-space in $O(\sqrt{N})$ time. Similarly, for large datasets, our inference system is able to prove a propositional statement in $O(\sqrt{N})$ time, with N the number of possible deduction sequences. Both exhibit a quadratic speed-up with regards to blind classical approaches, which require $O(N)$ evaluations.

Keywords: quantum computing, artificial intelligence, problem-solving, blocks world, automated inference

Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
BW	Blocks World
CNF	Conjunctive Normal Form
DNF	Disjunctive Normal Form
EBW	Elementary Blocks World
KB	Knowledge-base
LQIDS	Limited Quantum Iterative Deepening Search
MP	*Modus Ponens*
POP	Partial-order Planner
QFT	Quantum Fourier Transform
QIDS	Quantum Iterative Deepening Search
wff	well formed *formulae*

List of Algorithms

Nomenclature

Greek symbols

ν	Number of computation steps for Bennett's reversible machine.
π_{ϕ}	Toffoli's reversible analogue of an irreversible function ϕ.
ζ	Depth limit for the LQIDS procedure.
σ	Sub-knowledge-bases of knowledge-set S, each portraying a distinct alternative for every disjunction in *facts* or *consequents of rules* of S.
η	A given propositional query expression, which our approach attempts to prove.
$\alpha_{i,j}$	The j^{th} wff obtained via *Modus Ponens* from sub-knowledge-base i in a proof of η.
ω	The most complex sentence in $\mathcal{F}$ or $\mathcal{R}$, measured according to the operator depth of the propositional operation with the most precedence (excluding $\neg$).
μ_{ω}	Depth of the operation with the highest precedence in expression ω (excluding $\neg$).
ξ	Number of *facts in $\mathcal{F}$/consequents of $\mathcal{R}$* possessing a disjunction.

Roman symbols

N	Size of the state-space of a given problem.
I	Identity transform.
H	Hadamard transform.
W	Walsh-Hadamard transform.
G	Goal test matrix.
T	Transition matrix.
C_{not}	Controlled-not operator.
b	Uniform branching factor of a problem domain.
$\boldsymbol{s}_0$	Initial state vector description for a problem.

$\boldsymbol{g}$	Classical goal test function.
$\boldsymbol{f}$	Classical transition result function for choice $\boldsymbol{m}$ on state $\boldsymbol{x}$.
$\mathcal{M}$	Set of state transitions for a uniform tree search.
$\boldsymbol{m}$	Encoding for some transition within $\mathcal{M}$.
v	LQIDS verification procedure.
$U_{c,d}$	Operator corresponding to the circuitry of the d^{th} iteration of QIDS.
$n_{c,d}$	Total number of qubits required for a depth d circuit $U_{c,d}$.
U_c	Depth 1 counterpart of $U_{c,d}$, applied to 2-EBW problems and depicted in the circuit of figure 3.4.
$\|\boldsymbol{s}\rangle$	Initial state qubit register.
s	Number of qubits for recording a state, *i.e.*, length of $\|\boldsymbol{s}\rangle$
$\|\boldsymbol{m}_i\rangle$	Qubit register recording the i^{th} transition choice.
m	Number of qubits for keeping track of transition choices.
$\|\boldsymbol{w}\rangle$	Path qubit register.
w	Length of the path register.
$\|\boldsymbol{a}_i\rangle$	Auxiliary qubit register corresponding to the i^{th} transition gate.
$\|a_i\rangle$	The i^{th} ancillary qubit for QIDS circuits.
n	Number of blocks for an n-EBW problem. Also, the number of literals in a knowledge-base S.
$n^{(i)}$	Number of blocks in the i^{th} sub-problem found by our heuristic.
$\mathcal{G}_i$	Directed acyclic graph depicting state i.
V_i	Set of vertices (blocks) of $\mathcal{G}_i$.
E_i	Set of edges ("is on" relationships) of $\mathcal{G}_i$.
$\mathcal{G}$	The undirected graph extracted from the initial and final directed acyclic graphs of an EBW problem, and whose connected components indicate sets of blocks whose movements may be interdependent.
V	The set of vertices of $\mathcal{G}$.
E	The set of undirected edges of $\mathcal{G}$.
$V^{(i)}$	i^{th} connected component (*i.e.*, set of interdependent blocks) of $\mathcal{G}$.
$\mathcal{G}_j^{(i)}$	Partial state representation of the full state j, restricted to the i^{th} set of interdependent blocks.

$\boldsymbol{s}_i$ Full state encoding for state i.

$\boldsymbol{s}^{(i)}$ A partial state within the i^{th} sub-problem found by our heuristic.

$s^{(i)}$ Number of qubits necessary for recording states within the i^{th} sub-problem.

$\boldsymbol{s}_j^{(i)}$ Part-state representation of full state j within the i^{th} sub-problem.

$g^{(i)}$ Goal test function for sub-problem i.

$\boldsymbol{m}^{(i)}$ A movement decision encoding for sub-problem i.

$m^{(i)}$ Number of qubits needed to track movement choices in sub-problem i.

$\boldsymbol{f}^{(i)}$ The partial-state equivalent of the full transition function $\boldsymbol{f}$, under the i^{th} sub-problem.

$\boldsymbol{s}_j'^{(i)}$ Full state in which blocks of the i^{th} sub-problem are in positions defined by part-state $\boldsymbol{s}^{(i)}$, while the remaining ones conform with $\boldsymbol{s}_j$.

$\boldsymbol{m}'^{(i)}$ General encoding counterpart of the movement corresponding to i^{th} sub-problem choice $\boldsymbol{m}^{(i)}$.

$\boldsymbol{w}^{(i)}$ Path register for sub-problem i.

$\boldsymbol{p}^{(i)}$ Solution path for sub-problem i.

$l^{(i)}$ Shallowest solution path length for the i^{th} sub-problem.

$\mathcal{KB}$ A knowledge-base consisting of expressions in Propositional Calculus.

S The structured counterpart of $\mathcal{KB}$, required by our inference method.

$\mathcal{F}$ Set of simple sentences of S.

$\mathcal{R}$ Set of phrases in S of the form $\alpha \rightarrow \beta$.

l_i The length of a proof of η in knowledge subset i.

k Maximum number of unique operands of the full DNF form of any sentence in a knowledge-base, restricted to expansion over symbols it contains.

$|c\rangle$ The contradiction qubit, indicating whether a deductive state has become contradictory along an inference path.

$\mathcal{F}_i$ The i^{th} set of facts obtained from decomposition of S, and in which each sentence is a conjunction of literals, reflecting an individual case for a disjunction in S.

$\mathcal{R}_i$ i^{th} set of rules obtained from decomposition of S, in which each rule's consequent is a conjunction of literals, portraying an individual case for a consequent's disjunction in one of S's rules.

Superscripts

† Hermitian adjoint operator.

T	Transpose operator.

Other mathematical symbols

$O(\ )$	Assymptotic upper bound notation.
$\Omega(\ )$	Assymptotic lower bound notation.
$\oplus$	The bitwise exclusive-or operator.
$\otimes$	Tensor product operator.
$\mathbb{Z}_i$	The set of integers $0, \ldots, 2^i - 1$.
$\|-\rangle$	The ancillary superposition input $\frac{1}{2}(\|0\rangle - \|1\rangle)$ required by Grover's algorithm.
$\neg$	Unary negation operator.
$\wedge$	Conjunction operator.
$\vee$	Disjunction operator.
$\rightarrow$	Implication operator.
$\leftrightarrow$	Equivalence operator.
$\vdash$	Syntactic derivation relation.
$\models$	Semantic entailment relation.
$\|\boldsymbol{\kappa}\rangle$	Qubit register storing the truth/falsehood of each literal of knowledge-set S during deduction.
$E_\vee$	Disjunction elimination inference rule.

Chapter 1

Introduction

1.1 Motivation

Classical computation models are still at the heart most research techniques in the field of AI. However, some unconventional computation paradigms possess unique benefits. In particular, quantum computing algorithms which are able to quadratically improve the time performance of an equivalent classical procedure have been identified for two decades (*e.g.*, Grover's algorithm [25]).

Additionally, the inevitable advent of a universal quantum computer has already been generally accepted, with several private corporations and governments invested in realizing this vision. Thus, it is imperative that we in the AI community continue to accompany such efforts with our own research [65].

1.2 Objectives

In this work, we propose investigating the benefits inherent to the application of quantum methods, such as QIDS [56], to symbolic AI tasks. In particular, we present quantum/classical solving systems for two distinct problems, one more abstract than the other: the solution of generalized n-blocks Blocks World instances; and the proof of a given propositional expression from a program specified at the knowledge level, *i.e.*, a Knowledge-base (KB). For both systems, we wish to demonstrate that the use of quantum techniques can provide a quadratic time-performance advantage. Additionally, we will suggest how space requirements may be reduced when employing QIDS, and offer a new perspective on the utilisation of heuristics alongside quantum computing methods.

1.3 Thesis Outline

In chapter 2: Background we introduce the reader to quantum computation and the principles upon which it is founded, with a particular emphasis on its standard-circuit model. There we also describe the two problem domains on which this work is focused, namely those of Blocks World planning and theorem proving. A brief survey of past research efforts on each is conjointly provided.

Chapter 3: Blocks World Solver presents a quantum/classical approach to solving generalized Blocks World instances. In that chapter, we additionally propose a mechanism for reducing the solver's (and QIDS's) space requirements, as well as advancing an alternative viewpoint on the use of heuristics within quantum computation settings.

In chapter 4: Quantum Propositional Inference we show how QIDS may be applied to tackle a more abstract problem, that of proving entailed Propositional Calculus expressions from a knowledge-base specified in that language.

Lastly, we convey our conclusions in chapter 5: Conclusions, discuss the impact and future prospects of quantum computing methods in our field, and suggest a set of unsolved problems gleaned from our research on the latter domain.

Chapter 2

Background

In this chapter we provide context for our research, exploring major results from its related areas.

We begin with an Introduction to Quantum Computation and the main algorithms from that field which we will utilise in solving AI problems. This is followed by an overview of investigative work within the two specific problems we intend to tackle using symbolic strategies and the above computation models, in section Two Example Problems in Symbolic AI.

2.1 Introduction to Quantum Computation

This section first reviews those concepts from Reversible Computation which provide the basis for quantum computing in the standard circuit paradigm. We subsequently present a brief introduction to the foundations of Quantum Computation and describe the principal methods from this field which we will employ, such as Grover's algorithm, QIDS and the latter's limited-depth counterpart.

2.1.1 Reversible Computation

To be able to effect a quantum computation, we must perform some manner of transformation over a physical system. The transitions that these quantum systems are able to carry out are restricted by nature to those which adhere to certain properties.[1] The laws enforcing these properties reveal that reversible processes are precisely those which are permitted in nature [20].

However, standard Turing machines do not act in a reversible fashion. We cannot always obtain unique results from calculating the inverse of their transition functions, *i.e.*, the information which would allow us to retrace the process has been lost. Similarly, the commonplace computer also operates in an irreversible manner: its *NAND* and *NOR* gates, the cornerstones of digital circuitry, are irreversible.

To underscore the issue, we invite the reader to write down and consider the set of truth tables for the *AND*, *OR* and *NOT* operations. While the *NOT* operation is an obvious one-to-one, reversible mapping, we cannot find such bijections for the other two functions. If we are given an output value of **F** for an *AND*, or a **T** for an *OR* operator, where **F** stands for a false valuation and **T** for a true one,

[1] we describe these under heading Foundations, of Quantum Computation.

Stage	Input tape	History tape	Output tape
Computing	INPUT	-	-
Copying	OUTPUT	HISTORY	-
Reversing	OUTPUT	HISTORY	OUTPUT
	INPUT	-	OUTPUT

Table 2.1: Bennett's three stages of reversible computation

we do not know which input pair produced it. The information allowing us to trace back the function's procedure has actually been lost. In fact, for every bit of data lost in a computation, Landauer has proven [34] a minimum heat dissipation of $k_B T \ln 2$ Joules takes place[2]. Equivalent emissions are not found in reversible mechanisms.

Fortunately, in his study of *Logical Reversibility of Computation*, Bennett proved that every irreversible computation has a classical reversible analogue, which obtains identical calculations at similar cost [2].

The method he presents is able to simulate the operation of a single-tape Turing machine by way of a three-taped one, where each tape respectively records the *input*, the *history* of the computations, and the program *output*. Initially, the latter two are blank.

In such a machine, an irreversible computation is divided into three phases: the first stage executes the same calculation as its single-tape analogue, storing the result in the input tape. However, instead of losing intermediate calculations, these are recorded in the history tape; the second phase consists of copying the contents of the input to the output tape; finally, the inverse of the original computation in the input tape is calculated, using information from the history tape. When all three stages have been executed, the intermediate steps have been retraced, so as to obtain the program's input in the input tape, a blank history tape, and the output of the irreversible operation on the output tape. The entire process is depicted in table 2.1.

The method's main disadvantage lay in its high intermediate memory requirements whenever the number of computations ν was much larger than the amount of history storage bits. However, Bennett showed these could be improved, provided long calculations were divided into shorter segments, each behaving in the reversible, three-stage fashion detailed above. Thus, the history tape would be reused, reducing temporary space requirements to an $O(\log \nu)$ bound. The correction incurs a modest $O(\nu^2)$ increase in time-performance, due to the various retracing steps along the segments.

More importantly, Bennett's study implied that useful computations could be accomplished in a reversible way, at reasonable speeds, and without information loss.

The technique was subsequently expanded upon by Toffoli [58], who proposed an approach for calculating any finite function with a reversible logic network, which would make use of ancillary bits of data for storing the history of intermediate computations.

Toffoli demonstrated that a reversible analogue $\pi_\phi : \mathbb{Z}_l \to \mathbb{Z}_l$ of a finite function $\phi : \mathbb{Z}_n \to \mathbb{Z}_m$ could

[2] k_B is Boltzmann's constant and T the temperature of the computing environment.

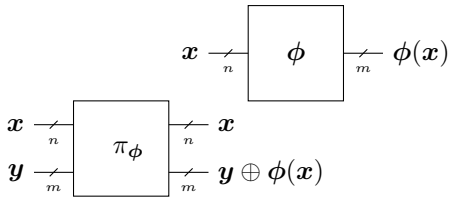

Figure 2.1: Similarly to how we interpret gates in digital networks, each box calculates a function based on its inputs (to the left of a gate, outputs are to the right). The rightmost gate calculates $\boldsymbol{\phi}(\boldsymbol{x})$ in an irreversible process. However, its reversible analogue, shown as the leftmost gate, computes $\boldsymbol{\phi}(\boldsymbol{x})$ in a reversible fashion, by using the ancillary inputs $\boldsymbol{y}$. Note: $\overset{\nearrow}{\underset{k}{}}$ represents a k bit wire. (Source: [58]).

be determined in the following manner:

$$\pi_{\boldsymbol{\phi}} : (\boldsymbol{x}, \boldsymbol{y}) \mapsto (\boldsymbol{x}, \boldsymbol{y} \oplus \boldsymbol{\phi}(\boldsymbol{x})) \tag{2.1}$$

where $\mathbb{Z}_l$ is the set of integers from 0 to $2^l - 1$, $l = n + m$, $\boldsymbol{y}$ are the ancillary bits and $\oplus$ is the bitwise *XOR* operation.

In figure 2.1 the reader may see both $\boldsymbol{\phi}$ and $\pi_{\boldsymbol{\phi}}$ depicted from a black box perspective, as if these were simple gates in a logic circuit. Note how passing $\boldsymbol{\phi}(\boldsymbol{x})$ as ancillary input to the reversible gate results in $\pi_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\phi}(\boldsymbol{x})) = (\boldsymbol{x}, \boldsymbol{\phi}(\boldsymbol{x}) \oplus \boldsymbol{\phi}(\boldsymbol{x})) = (\boldsymbol{x}, \boldsymbol{0})$.

The author also described how any function under composition could be computed by combining *AND/NAND* (or Toffoli) gates alone. Theoretically, such networks were shown to operate without loss of information, and with an efficiency comparable to the corresponding irreversible computation. Their storage requirements were also demonstrated to increase in proportion with the number of function arguments, instead of how many gates were employed.

The operations in Toffoli's reversible circuits may be mathematically expressed by *unitary* transforms of linear algebra.[3] These operators will reproduce the action of each gate in the circuitry, according to every input/output combination defined in equation (2.1). As we shall see in the next section, quantum mechanics is characterized by unitarity of its transformations.

Note that replacing each gate in Toffoli's classical reversible circuits with its quantum analogue, we obtain a quantum circuit that is equivalent in function, and that is comparably efficient to the irreversible calculation [42, p. 101].

2.1.2 Quantum Computation

Various models for quantum computation could be applied for problem solving in AI, although we will employ the *standard quantum-circuit model*. Nevertheless, we presently provide the reader with a concise introduction to some of the more popular alternative models.

Quantum Walks [48], for instance, are structurally comparable to the use of Grover's algorithm. They are also appropriate for searching within a database, where items correspond to graph nodes and a quantum, discrete-time analogue to a classical random walk is effected within its topology, until a desired

[3] U is a unitary transform provided its conjugate transpose $U^\dagger$ is its inverse. *I.e.*, $U^{-1}U = U^\dagger U = I$ [35, p. 19].

element is found. The walk itself is controlled by a quantum coin, which is queried at every transition point within the network. An equivalent, but continuous-time approach, had already been advanced by Farhi and Gutmann [18].

Inspired in simulated annealing, another strategy, *Quantum Adiabatic* computation [19], is used in operating the D-WAVE computer. It is able to solve optimization problems, provided these can be characterized by an energy function complying with Schrödinger's equation. As with simulated annealing, such functions are crafted to generate a global minimum whenever their argument conforms with a certain goal representation. Perturbations to the energy function are then carried out, and after a prolonged period of time, a solution is extracted. Although adequate for dealing with this class of problem, estimating the time required to obtain an exact solution remains a difficulty.

Since we are focused on extracting exact solutions within time limits that are straightforward to establish, our method is grounded upon the standard-circuit model of quantum computation. Every operation that is carried out within this paradigm is reversible, and any randomness restricted to the act of measurement.

Quantum algorithms in this model typically apply one of two methodologies: using the *Quantum Fourier Transform (QFT)* to find the period of some problem describing function; or effecting *amplitude amplification* to discover a marked item in a database.

Peter Shor designed a QFT-centred strategy for performing fast integer factorization in his eponymous algorithm [49, 50], proving that quantum computation is capable of tractably breaking the classical cryptographic techniques we currently rely upon. Furthermore, it is widely considered that no efficient classical alternative to the technique exists.

However, such QFT-based strategies are effective when problems bear a unique structural regularity. Amplitude amplification methods, on the other hand, are applicable to a wider range of domains. Grover's algorithm [25] is paradigmatic of the latter methodology. It is able to ascertain a solution to a black box problem within an unstructured database of N elements in $O(\sqrt{N})$ time. Since Grover's method is central to our approach, we elaborate upon it in greater detail under its own heading of this sub-section.

In the next of these, Foundations, we present a minimum review of those concepts from quantum mechanics which are instrumental for realizing quantum computations. Grover's Algorithm, the fundamental technique behind the QIDS procedure that we make use of, is subsequently described. This is followed by a broad specification of the Quantum Iterative Deepening Search algorithm, as well as a discussion on how it can be applied as a problem-solving approach in AI. Finally, a restricted version of QIDS, Limited Quantum Iterative Deepening Search, which we will utilise in our inference method, is defined and analysed.

Foundations

The rules which govern quantum models of computation make these strikingly different paradigms from those of classical origin. The cornerstones of such models are the quantum mechanical laws which underlie any processing of data.

Operating with qubits In particular, we are no longer restricted to the classical notion of a *bit* of information. These can retain a value of either 1 or 0. However, in the realm of quantum mechanics, we may consider the *qubit*[4] as our primitive information storage unit. Unlike their classical peers, these units may hold both values simultaneously in a state of linear *superposition*. In Dirac vector notation [14], the state of a qubit $|\psi\rangle$ may be expressed as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where $|0\rangle$ and $|1\rangle$ are the two states in which the qubit may be measured as exclusively $|0\rangle$ or $|1\rangle$. These are also known as the qubit's *basis states*.

Typically, orthonormal basis states are assumed, in which case they generate a complex two-dimensional Hilbert space [35, p. 16] $\mathcal{H}^2$, and may be depicted as $|0\rangle = [1\ 0]^{\mathrm{T}}$ and $|1\rangle = [0\ 1]^{\mathrm{T}}$, with $^{\mathrm{T}}$ the transpose operator. The inner product over such a space is symbolically denoted $\langle\psi|\phi\rangle$,[5] for any vectors $|\psi\rangle$, $|\phi\rangle$. We use the traditional Euclidean, or L^2 norm, to measure vector length, by calculating $\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle}$.

The coefficients α, β are complex numbers such that $|\alpha|^2$ and $|\beta|^2$ represent the probability of obtaining $|0\rangle$ or $|1\rangle$ (respectively) upon measuring $|\psi\rangle$. It is thus natural that these respect the law of total probability $|\alpha|^2 + |\beta|^2 = 1$, *i.e.*, that the length of the qubit $\||\psi\rangle\| = 1$.

The adoption of these complex coefficients allows us to describe wave-like behaviour often observed in the destructive and constructive interference patterns of sound and light propagation, for example.

In this context, a *computation* is a series of linear operations over such qubits. An individual calculation can be understood as an operation over both bases of a qubit, described in the following manner: $|i\rangle \mapsto a_{i0}\,|0\rangle + a_{i1}\,|1\rangle$, where i is 0 or 1, the a_{ij} are also complex, and the qubit's length is preserved.

For a qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, its result $|\psi'\rangle$ under an operation U can be described in linear algebra terms by the following system of equations:

$$\begin{bmatrix}\alpha'\\ \beta'\end{bmatrix} = \begin{bmatrix}a_{00} & a_{10}\\ a_{01} & a_{11}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix} \quad \Leftrightarrow \quad |\psi'\rangle = U\,|\psi\rangle$$

Because nature itself is reversible, quantum mechanics prescribes that every operator U must be a reversible transform. That is, U's basis states must be orthonormal, which is equivalent to saying that U must be a *unitary*, length-preserving operation. Note that, under these conditions, for any two qubits $|\psi\rangle$, $|\phi\rangle$ and an operator U, their inner product is identical to the inner product of their images $U\,|\psi\rangle$, $U\,|\phi\rangle$:

$$(\langle\phi|\,U^\dagger)(U\,|\psi\rangle) = \langle\phi|\,U^\dagger U\,|\psi\rangle = \langle\phi|\,I\,|\psi\rangle = \langle\phi|\psi\rangle$$

Additionally, these principles hold for multiple-qubit systems. While a single qubit can encompass merely 2 basis states, for systems of n-qubits we may describe $N = 2^n$ dimensions. Each basis may then be labelled in some manner, and interpreted according to an encoding of our choice. For instance, we may label them like so $|\mathbf{0}\rangle, |\mathbf{1}\rangle, |\mathbf{2}\rangle, \ldots, |\mathbf{2^n - 2}\rangle, |\mathbf{2^n - 1}\rangle$ and decode the description as we see fit.

Typically, we find these depicted in the literature using consecutive n-length, binary strings. The above state sequence would thus correspond to $|0\ldots0\rangle, |0\ldots01\rangle, |0\ldots010\rangle, \ldots, |1\ldots10\rangle, |1\ldots11\rangle$.

[4] short for quantum bit.

[5] in Dirac notation, for any qubit $|\psi\rangle = [\,\alpha\ \beta\,]^{\mathrm{T}}$, its conjugate transpose peer is the vector $\langle\psi| = |\psi\rangle^\dagger = ([\,\alpha\ \beta\,]^{\mathrm{T}})^\dagger = [\,\overline{\alpha}\ \overline{\beta}\,]$, where $\overline{z}$ is the conjugate of complex number z, and the † symbol represents the conjugate transpose operator.

This is done with the understanding that there is an underlying operation implicitly acting upon the adjacent qubits, the *tensor product*. For instance, suppose we had measured a qubit register and obtained the state labelled 101. Thus, it would be in state $|101\rangle = |1\rangle \otimes |0\rangle \otimes |1\rangle$, where we have explicitly denoted the tensor product as $\otimes$.

Parallel Quantum Computation Combining the ability to place qubits in superposition, with the capacity to operate over these in order to produce quantum interference effects, is the principal methodology inherent to the field of quantum computation. Together, these faculties allow quantum models to execute parallel calculations in a single step.

Below, we describe how this is possible. We begin by specifying how to obtain the superpositions from a basis state, using the *Hadamard transform*. Depending on which basis a single qubit is in at a given moment, when acting upon it, the transform will yield one of two even distribution superpositions:

$$H : \begin{array}{l} |0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{array} \iff H = \tfrac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

The n-qubit generalization of H is known as the *Walsh-Hadamard transform*, and denoted W. Acting over a basis state, this operator is able to obtain an even superposition of all possible states. For instance:

$$\begin{aligned} W|00\ldots0\rangle &= (H \otimes H \otimes \ldots \otimes H)|00\ldots0\rangle \\ &= \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \ldots \otimes (|0\rangle + |1\rangle) \\ &= \frac{1}{\sqrt{2^n}}(|0\ldots0\rangle + |0\ldots01\rangle + \ldots + |1\ldots10\rangle + |1\ldots11\rangle) \\ &= \frac{1}{\sqrt{2^n}} \sum_{\boldsymbol{x}=\mathbf{0}}^{\mathbf{2^n-1}} |\boldsymbol{x}\rangle \end{aligned} \tag{2.2}$$

Let $\boldsymbol{f} : \boldsymbol{x} \mapsto \boldsymbol{f}(\boldsymbol{x})$ be an efficiently computable function of n arguments and m outputs. Using the result in equation (2.1), we can build a unitary operator $U_{\boldsymbol{f}}$ from its classical reversible counterpart $\pi_{\boldsymbol{f}}$. Application of $U_{\boldsymbol{f}}$ over a linear superposition of inputs, such as $\sum \alpha_{\boldsymbol{x}} |\boldsymbol{x}\rangle$, will derive the following:

$$U_{\boldsymbol{f}} : \sum_{\boldsymbol{x}} \alpha_{\boldsymbol{x}} |\boldsymbol{x}, \boldsymbol{a}\rangle \mapsto \sum_{\boldsymbol{x}} \alpha_{\boldsymbol{x}} |\boldsymbol{x}, \boldsymbol{a} \oplus \boldsymbol{f}(\boldsymbol{x})\rangle$$

where $\boldsymbol{a}$ is the vector of ancillary qubits and $\alpha_{\boldsymbol{x}}$ is the coefficient of ket $|\boldsymbol{x}\rangle$. When the $\boldsymbol{a}$ qubits are set to 0, we will obtain:

$$U_{\boldsymbol{f}} : \sum_{\boldsymbol{x}} \alpha_{\boldsymbol{x}} |\boldsymbol{x}, 0\ldots0\rangle \mapsto \sum_{\boldsymbol{x}} \alpha_{\boldsymbol{x}} |\boldsymbol{x}, \boldsymbol{f}(\boldsymbol{x})\rangle$$

Finally, applying $U_{\boldsymbol{f}}$ over the superposition of all states $\boldsymbol{x}$ in equation (2.2), $\boldsymbol{f}(\boldsymbol{x})$ is calculated for

each:

$$U_{\boldsymbol{f}} : (\, W \underbrace{|00\ldots0\rangle}_{n \text{ inputs}} \,) \otimes \underbrace{|\boldsymbol{a}\rangle}_{m \text{ outputs}} = \\ = \frac{1}{\sqrt{N}} \sum_{\boldsymbol{x}} |\boldsymbol{x}, \boldsymbol{a}\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{\boldsymbol{x}} |\boldsymbol{x}, \boldsymbol{a} \oplus \boldsymbol{f}(\boldsymbol{x})\rangle \tag{2.3}$$

where $N = 2^n$.

The above is how *quantum parallelism* can effect multiple computations expeditiously. Using quantum methods, we may evaluate these states in $O(\sqrt{2^n})$ steps. In a classical setting, we would need to calculate $\boldsymbol{f}$ $O(2^n)$ times in order to obtain the same amount of information. While it may seem revolutionary, there *is* a caveat to the result. The act of measuring a qubit will always *collapse*[6] it into a single basis state, limiting the amount of data we are able to acquire from equation (2.3).

The adequate description of experimental observations at the turn of the 19^{th} century [31] compelled the development of a quantum theory which would assume that, before observation, a qubit $|\psi\rangle$ could be in a state of linear superposition of its bases. Upon measurement, the collapse will always yield such a basis, *i.e.*, either a $|0\rangle$ or $|1\rangle$, and essentially, the act of observation will have changed $|\psi\rangle$'s state. Unless further observations (using another instrument) or calculations are performed, measuring $|\psi\rangle$ again will produce the same state.

Grover's Algorithm

Grover's search algorithm is able to discover a marked item from within an unsorted dataset bearing N constituents [25]. To do so, his procedure presumes access to an oracle function that can evaluate whether a given element is marked. In its initial formulation, the database held a single tagged element, which the method could find in $O(\sqrt{N})$ oracle queries. However, the algorithm can be generalized for multiple marked elements, without suffering an impact on its overall complexity.

Compared to classical procedures, and assuming the dataset is unstructured, Grover's method yields a quadratic speed-up, since the former techniques require $O(N)$ oracle interrogations. As we have stated in the previous sub-section, the advantage stems from the effects of quantum parallelism and interference.

Assuming an efficiently computable black-box predicate p exists, and is able to detect if an element $\boldsymbol{x}$ of the dataset is tagged, then we can construct a transform U_p from its reversible analogue π_p, as presented in the last section. Applying U_p to the dataset's equivalent to the superposition in equation (2.2), we can obtain a mapping akin to that of equation (2.3):

$$U_p : (\, W \,|00\ldots0\rangle \,) \otimes |a\rangle = \\ = \frac{1}{\sqrt{N}} \sum_{\boldsymbol{x}} |\boldsymbol{x}, a\rangle \mapsto \frac{1}{\sqrt{N}} \sum_{\boldsymbol{x}} |\boldsymbol{x}, a \oplus p(\boldsymbol{x})\rangle \tag{2.4}$$

where $p(\boldsymbol{x})$ is an n-ary boolean function, such that $p(\boldsymbol{x}) = 1$ for marked items and $p(\boldsymbol{x}) = 0$ otherwise,

[6] this is a traditional explanation of the effect, also known as the Copenhagen interpretation [28, 52] of quantum theory.

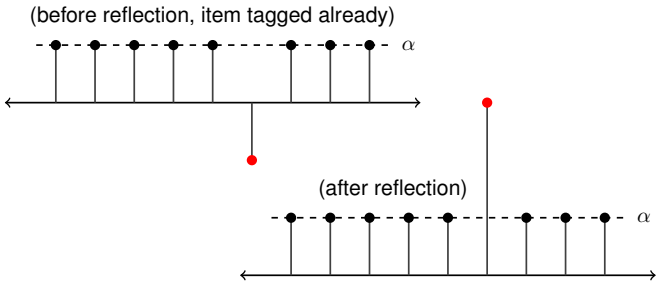

Figure 2.2: The diffusion step of the method reflects each item about the average α of a large enough number of superposition elements, N. Before the reflection, every superposition component has a coefficient with amplitude $1/\sqrt{N}$, except for one: the item we seek (marked red) has coefficient value $-1/\sqrt{N}$, *i.e.*, it is already tagged. Because the number of superposition components is large, $\alpha \approx 1/\sqrt{N}$, and the only element substantially affected by the procedure is the marked one, its magnitude having been boosted by 2α.

and a is an auxiliary qubit, prepared in the superposition $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \coloneqq |-\rangle$. The consequence is that each input is now intertwined (*i.e.*, *entangled*) with its corresponding predicate result.

At each step, the algorithm first inverts the sign of those elements which are marked. The remaining items are left unchanged. This is followed by a diffusion transformation,[7] which essentially reflects each element about the superposition's average.

The process progressively increases the probability of obtaining one of the marked elements upon measurement of the superimposed qubits (see figure 2.2). Successful computations are thus reinforced, and after $O(\sqrt{N})$ steps the marked item can be deterministically [6] retrieved on measurement. As mentioned, unstructured classical methods need $O(N)$ elements to be examined.

It was later shown [3] that any quantum algorithm making use of an oracle function cannot employ less than $\Omega(\sqrt{N})$ black box queries, thus demonstrating that Grover's method is optimal.

Quantum Iterative Deepening Search

This hybrid strategy uses quantum techniques, such as superpositioning and Grover's algorithm, along with classical iterative-deepening [32], to perform a quantum hierarchical search of a tree, *i.e.*, a *Quantum Iterative Deepening Search* [56] procedure.

The methodology inherent to the approach stipulates that we encode the state space of a problem into a qubit register (*i.e.*, a Hilbert space), which we manipulate using linear transformations. Ancillary qubits record each operator application, so that every transformation is reversible. For a particular goal or solution test $g(s)$ (with s a state encoding) of the problem, its respective solution test matrix U_g (we will refer to it as G) can be obtained according to equation (2.3), and implemented as a logic gate corresponding to figure 2.3, left. The gate will distinguish whether or not a given state is a solution by flipping its ancillary input qubit.

Transitions within the problem domain (*e.g.*, moving a disk in the Tower of Hanoi game, a robot's arm, etc) can be modelled according to equation (2.3) and a state transition function f, producing a unitary gate T, as in figure 2.3. As inputs, these transition operators receive both a state and some choice on

[7] that can be stated in terms of the Householder transformation [30].

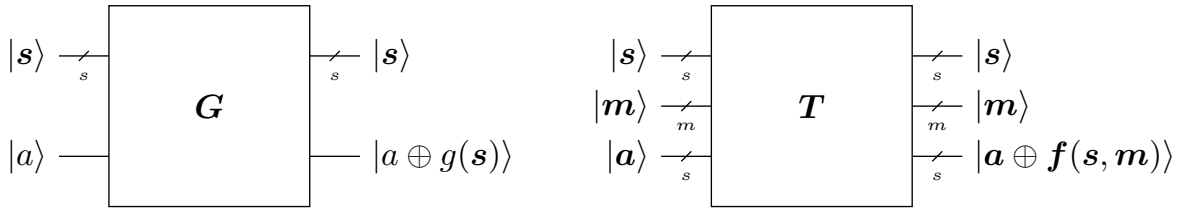

Figure 2.3: Unitary operator gates for general domains. The $|s\rangle$ represent states, the $|m\rangle$ are transition choices, and the qubits $|a\rangle$ and $|a\rangle$ are auxiliary inputs, for storing the output of computations. The left gate is the reversible analogue to the solution test function $g(s)$. The right gate computes the state resulting from a transition m over the state s, according to a transition function $f(s, m)$.

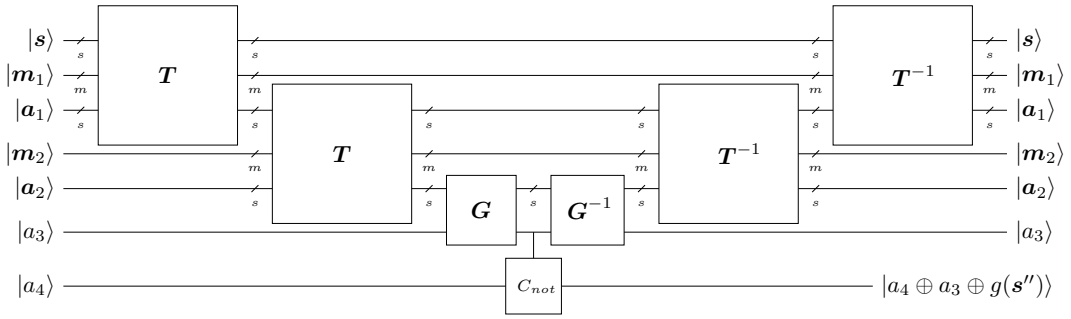

Figure 2.4: Reversible circuit for a tree search over two transition operations and a goal test, for a generic problem domain. We ensure that our input conforms with Grover's formalism by applying each gate's inverse transformation in opposite order, but "saving" the result of the circuit's operation using a C_{not}[8] gate. Note that $s'' = a_2 \oplus f(m_2, s')$, and $s' = a_1 \oplus f(m_1, s)$.

how to proceed, and are "chained" together, so that each following operator accepts the state resulting from the previous transition.

Because Grover's method requires that our input be entangled with the result of the predicate function, each gate's behaviour is "undone" by applying the corresponding operator inverses in opposite order (see figure 2.4 for a depth 2 chain, figure 2.5 for a depth d one). This would leave us with our input on the registers again, making the computation useless. However, the approach prescribes the use of the *controlled-not*, or C_{not}, gate[8] as a predicate for Grover's algorithm, thereby "preserving" the circuit's outcome without compromising permutability.

Additionally, every possible sequence of choices (*i.e.*, a *path*) is explored by placing the qubit register recording these (the *path register*) in superposition, via the method in equation (2.2). In this manner, the technique effectively implements a uniform branching factor tree search over the problem space (see figure 2.6) with depth equivalent to the number of transitions in the chain.

Note that although most problems possess a non-constant branching factor domain, we can always map their non-uniform tree structure into a uniform analogue, such as in figure 2.6, by considering any excess transitions as having no effect over a state [55].

To execute search over all depths of the tree, the technique begins by building a depth 0 chain (0 transitions and a goal test). When a solution is found, the algorithm terminates, returning the path

[8] the gate flips an auxiliary qubit contingent on a control qubit's value. Its behaviour is defined by the mapping $C_{not} : (c, x) \mapsto (c, c \oplus x)$. In QIDS, the result of the goal test is taken as the control qubit, so that the C_{not}'s ancillary qubit may be used to predicate Grover's algorithm, as in equation (2.4).

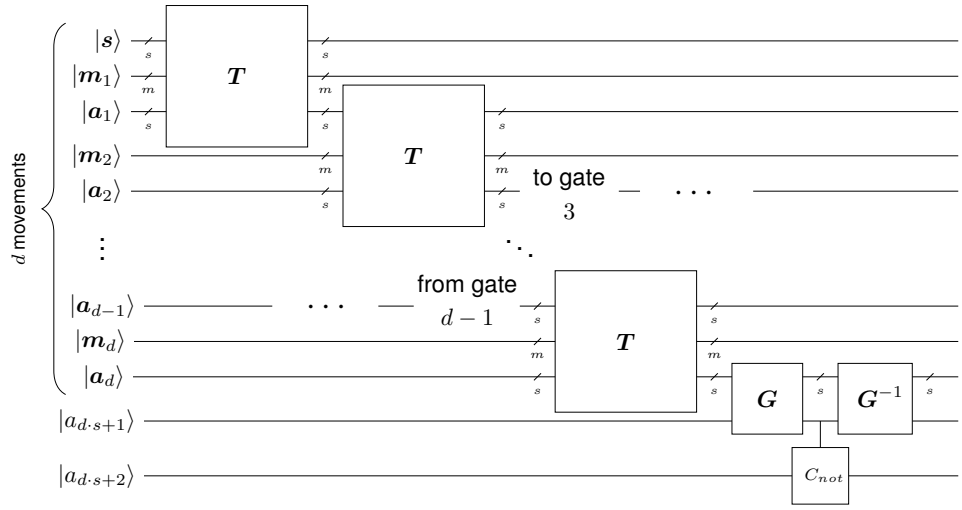

Figure 2.5: First section of the reversible chain for depth d. The remainder of the circuit is analogous with that of figure 2.4. Note that the auxiliary qubits differ in their indexing, according to whether or not they represent multiple-qubit systems. Multiple-qubit kets are highlighted in bold, indexed according to transition order. Single qubit kets are not shown in bold face, and are are indexed according to the number of auxiliary qubits for the entire circuit.

register. Otherwise, it increments depth, repeating the above. Naturally, if no solution exists, it will proceed indefinitely.

QIDS approaches have been successfully designed to solve such problems as the sliding block puzzle [54] or the simulation of classical Turing machines [56].

Limited Quantum Iterative Deepening Search

Limiting the depth at which QIDS should stop the search process allows us to define algorithm 1, aptly dubbed Limited QIDS or simply LQIDS.

The procedure accepts the following as input:

- A starting state s_0 for the search, which will be depicted in an encoding appropriate to the state space of the problem domain;
- A function $\boldsymbol{f}(s, \boldsymbol{m})$, which calculates the state resulting from transition $\boldsymbol{m} \in \mathcal{M}$ in state s, where $\mathcal{M}$ is a set of state transitions (valid in every state, due to the domain's uniform branching factor);
- A goal test function $g(s)$, evaluating whether its argument is a goal state;

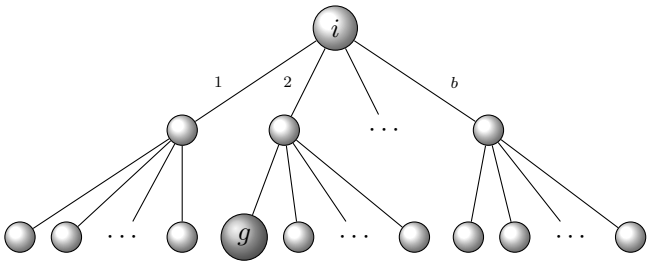

Figure 2.6: A depth 2 tree search. Circles represent states in the problem domain. Each edge corresponds to a transition, where b (the *branching factor*, *i.e.*, number different actions over each state) can be assumed uniform over the domain. g represents a goal and i the starting state.

Algorithm 1 The Limited Quantum Iterative Deepening Search algorithm. Receives an initial state $\boldsymbol{s}_0$, a transition result function $\boldsymbol{f}$, a solution test function g, and an exploration depth limit ζ.

procedure LQIDS($\boldsymbol{s}_0$, $\boldsymbol{f}$, g, ζ)
$d \leftarrow 0$

while $d \leq \zeta$ **do** ▷ verify depth limit
∘ Build the quantum circuit of depth d (see Figure 2.5) having transition operator T and goal operator G, s.t. T is the reversible gate constructed from transition result function $\boldsymbol{f}$ using equation (2.3) and G is the unitary solution test gate, obtained using the same method, from goal test function g
∘ Prepare the initial state register $|\boldsymbol{s}\rangle$ of our search in conformity with starting state vector $\boldsymbol{s}_0$
∘ Prepare the d transition choice registers in an even distribution superposition, *i.e.*, the path register $|\boldsymbol{w}\rangle = |\boldsymbol{m}_1\rangle \dots |\boldsymbol{m}_d\rangle = W\,|0 \dots 0\rangle$
∘ The ancillary input qubits are all set to $|0\rangle$. The exception being C_{not}'s auxiliary qubit, $|a_{d \cdot s+2}\rangle$, which is assigned the superposition $|-\rangle$ prescribed by Grover's method
∘ Run Grover's algorithm
∘ Measure the path register qubits $|\boldsymbol{w}\rangle$
if $\boldsymbol{w}$ leads to a goal according to verification process $v(\boldsymbol{s}, \boldsymbol{w}, \boldsymbol{f}, g, d)$ **then**
return $\langle True, \boldsymbol{w}\rangle$ ▷ solution found
else
$d \leftarrow d + 1$ ▷ search deeper
end if
end while

return $\langle False, \varnothing\rangle$ ▷ no solution found
end procedure

- A depth limit ζ for the search policy.

From depths 0 to ζ, LQIDS is identical to QIDS. Nonetheless, for clarity, we will describe it here in detail. Firstly, a quantum circuit akin to that of figure 2.5 for depth d is constructed, which entails using equation (2.3) and functions $\boldsymbol{f}$, g to obtain quantum gates for transitions and goal test (respectively T and G). Formally, the two gates may be portrayed as the two linear transforms:

$$\begin{aligned} T &: \alpha\,|\boldsymbol{s}\rangle\,|\boldsymbol{m}\rangle\,|\boldsymbol{a}\rangle \mapsto \alpha\,|\boldsymbol{s}\rangle\,|\boldsymbol{m}\rangle\,|\boldsymbol{a} \oplus \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{m})\rangle \\ G &: \alpha\,|\boldsymbol{s}\rangle\,|a\rangle \mapsto \alpha\,|\boldsymbol{s}\rangle\,|a \oplus g(\boldsymbol{s})\rangle \end{aligned} \tag{2.5}$$

where $\boldsymbol{a}$ is T's ancillary qubit vector (with length equal to that of $\boldsymbol{s}$), a is G's single auxiliary qubit (consult figure 2.3) and α are the coefficients for each ket.

Secondly, the starting state $\boldsymbol{s}_0$ is encoded into an initial state register $|\boldsymbol{s}\rangle$, and the superposition of every d-length path within the tree, $|\boldsymbol{w}\rangle = |\boldsymbol{m}_1\rangle \dots |\boldsymbol{m}_d\rangle$, obtained through equation (2.2).

For every depth d, an equivalent number of transitions T are effected, upon which a goal test gate and a C_{not} operator are applied, so our circuit's input is $|\boldsymbol{s}\rangle\,|\boldsymbol{m}_1\rangle\,|\boldsymbol{a}_1\rangle \dots |\boldsymbol{m}_d\rangle\,|\boldsymbol{a}_d\rangle\,|a_{d \cdot s+1}\rangle\,|a_{d \cdot s+2}\rangle$. Further, the entire circuit operation $U_{c,d}$, can be mathematically expressed over this input as follows:

$$\begin{aligned} U_{c,d} &= C_d^{-1}(I^{\otimes s+d(m+s)} \otimes C_{not})\,C_d \\ C_d &= (I^{\otimes d(m+s)} \otimes G) \prod_{i=1}^{d} (I^{\otimes (d-i)(m+s)} \otimes T) \end{aligned} \tag{2.6}$$

where $I^{\otimes n} = \bigotimes_{i=1}^{n} I$ does not alter the top n qubits. We also consider each qubit beyond G and T's

inputs in C_d's definition to merely suffer an identity I transform, otherwise the notation would become unnecessarily cumbersome.

Provided every auxiliary input qubit is set to $|0\rangle$ (save for $|a_{d\cdot s+2}\rangle$, which must be set to $|-\rangle$), running Grover's algorithm over the circuit will deterministically extract a solution (should one exist at that depth). Measuring the path register, we can readily verify if its choices lead us to a goal state via some verification function v, usually accepting $\boldsymbol{s}, \boldsymbol{w}, \boldsymbol{f}, g$ and the current depth as arguments. For our purposes, v involves applying the d transitions in $\boldsymbol{w}$ to $\boldsymbol{s}$ in conformity with $\boldsymbol{f}$, and testing the final state via g. Success should be reported accordingly, along with the problem's solution (*i.e.*, the sequence of transitions $\boldsymbol{w}$).

Otherwise, the algorithm should repeat the above for a higher depth, until such a solution is determined, or the depth limit surpassed. In the latter case, LQIDS will indicate that none was found.

Note that the gates T and G must be pre-calculated to implement the circuit, a one-off $O(2^{2s+m})$ time- and space-complexity process. Here, we establish a bound on the number of calculations by considering T as the largest of the gates.

A detailed runtime- and space-performance analysis of the LQIDS procedure is presented in Appendix A.

2.2 Two Example Problems in Symbolic AI

This section examines the two symbolic AI problems which we will attempt to solve with methods from quantum computation.

Its first sub-section, Blocks World, describes that popular toy-problem and imparts a comprehensive review of relevant previous work pertaining to it. Subsequently, we discuss research relating to our second problem, that of automated inference via Knowledge-based Systems. In that sub-section we also reacquaint the reader with the precepts of Propositional Logic, the formalism upon which our studies were focused.

2.2.1 Blocks World

We begin with an introduction to the specific task with which we will illustrate the QIDS technique, under the heading Problem Description. Finally, in the next of these, From Blocks World to General Planning, we discuss which features of this micro-world are often relevant to AI problems involving other domains, and how these have historically influenced the development of general-purpose planning methods.

Problem Description

Blocks World (BW) has been a popular planning domain in AI research [7, 45, 51, 53, 61], first appearing in Winograd's [63] work as a micro-world for the SHRDLU simulated robot. While this early version contemplated coloured blocks of varying size and shape arranged over a surface, a simpler variant assumes a 2D representation, where blocks are colourless squares of identical size.

A Blocks World problem is traditionally exemplified using three blocks, although an n-block world is analogous. In that micro-world, three blocks lie arranged in some initial configuration over a table (as in figure 2.7, left). Each may have at most one block over it, and each may be either over the table or another block.

Operators consist of moving one of the clear boxes[9] to the table, or over another block that is also clear. The table is considered infinite, *i.e.*, it is always said to be clear. Our goal is to find a sequence of operations enabling us to transform the initial block configuration into the final one (figure 2.7, right).

On problems of significant size, the optimal solution for a BW instance is generally hard to find. Additionally, similar problems can have drastically different solutions. These characteristics make BW a relevant planning toy-problem, particularly suited for showcasing search and planning efficiency over large instances.

Even when multiples blocks of the same label (*i.e.*, *interchangeable blocks*) can be present in the same state, finding a trivial solution is an easy task. For instance, we may simply place every block that is stacked over another in the start state on the table. The goal state can then be easily reached by moving each to its final position in succession. This approach yields a plan of length no worse than double that of an optimal solution.

However, Chenoweth [9] concluded that finding optimal solutions for a BW in which interchangeable blocks are permitted is an NP-hard problem, similar in complexity to the travelling salesman problem [23, 39], optimally solving sliding block puzzles [41] or the bin-packing problem [22]. These results apply to any BW version incorporating such blocks, including: Fahlman's [17] version, which considered boxes of different shapes and sizes; those where the table cannot be used to temporarily place blocks; where multiple goal or start states are assumed; multiple simultaneous actions are allowed on each step; and, where each block may play more than one role (such as being able to show a different face).

Chenoweth's earlier work [8] had already shown that if no interchangeable blocks were present, "good" quality solution plans could be found in time $O(n^2)$, where n is the number of blocks. We refer to such a version of BW as an Elementary Blocks World (EBW)[10]. In keeping with Chenoweth's results for BW with interchangeable blocks, Gupta and Nau [26, 27] demonstrated that while non-optimal solutions to EBW problems are similarly simple to obtain, optimal ones still involve NP-hard complexity (but no worse).

For years, the difficulty inherent to this task was ascribed to the existence of deleted-condition inter-

[9] that is, one without a block over it.
[10] its description is well-known, and can be found in several AI textbooks, such as [38, 44].

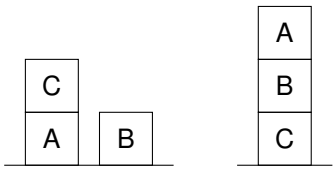

Figure 2.7: Example of a Blocks World problem: two block arrangements are depicted. The left corresponds to the initial state and the final state to the right. Solutions consist of sequences of single block movements capable of transforming the former configuration into the latter one.

actions, such as creative destruction [45] or Sussman's Anomaly[11] [53], where the side-effect of attaining one sub-goal deletes a condition needed for achieving the overall goal. Surprisingly though, Gupta and Nau found that optimal planning difficulty in EBW cannot be attributed to these interactions. Instead, they prove it to be the byproduct of a situation they term *deadlock*, wherein several goals have yet to be achieved, none of which can be accomplished directly.

Resolving deadlocks requires an *enabling-condition interaction* [27, 37]. These achieve several goals simultaneously and can make it easier to accomplish others. However, when multiple deadlocks exist, it is frequently unclear which course of action should be performed to obtain the best plan for the problem.

Problems in which no deadlocks are present can be optimally solved in time $O(n^3)$, using a hill-climbing strategy proposed by the authors, which moves blocks into positions consistent with the goal whenever doing so is possible. This is true regardless of the existence of deleted-condition interactions. In particular, traditional examples of situations which illustrate these interactions, such as creative destruction and Sussman's Anomaly, do not contain deadlocks, and are thus easily solved by such a method.

When deadlocks *are* present, a solution with length no more than double that of an optimal one can be obtained in $O(n^3)$ with the same approach. Furthermore, the authors demonstrate that deciding whether or not a given EBW problem is solvable can be done in $O(n \log n)$ time-complexity, and that provided the decision is affirmative, a solution which moves no block more than twice can be extracted in time $O(n)$.

Similar complexity results were found for other generalizations of EBW, where the above technique is still applicable. In worlds where the blocks may vary in size and shape, the problem complexity is equivalent. If the table capacity is limited, the results are comparable: querying for solution existance, and finding one of length $O(n \log n)$ if it exists, is a low-order polynomial time problem. However, when both features are present, blocks of variable shape/size and limited table capacity, planning is more difficult. Because the shortest plans for some instances of this problem have exponential length, no deterministic polynomial-time procedure exists for finding them.

Nevertheless, for all of these EBW worlds, producing an optimal plan remains an NP-hard problem.

From Blocks World to General Planning

Several features of the BW domain are present in more realistic planning environments. Those traits have influenced the strategies employed for solving the latter problems. For instance, deleted-condition interactions exhibit some of the limitations of linear (progression/regression) planning methodologies. Such totally-ordered approaches cannot take advantage of goal decomposition and often compromise solution quality.

This led to the development of Partial-order Planners (POPs), which allow interleaved actions to be included in a plan producing the overall objective. The distinction between the two strategies is pronounced. Whereas total-order planners are forced to make decisions earlier, POPs postpone these

[11] for the problem in figure 2.7, we invite the reader to attempt obtaining an optimal plan by sequentially achieving the two sub-goals: (A on B) and (B on C).

until the last possible moment, following a least-commitment heuristic. Plans generated from partially-ordered methods are often advantageous when adaptation to external factors is a necessity. Because some operators can be considered for execution in a non-deterministic fashion, more flexible scenarios may be contemplated.

On the other hand, the enabling-condition interactions observed in BW are particularly relevant when tractably obtaining an optimal plan is crucial. As observed in the previous heading, although creative destruction and Sussman's Anomaly do influence plan quality, enabling-condition interactions are actually responsible for the complexity of optimal planning in that space. These effects remain pertinent for several domains where such plans are required [37], including logistic and transportation problems [60], scheduling of tasks [16], assuring resource redistribution [57], etc.

The QIDS approach, along with the decomposition heuristic we will introduce in chapter 3, are particularly suited to these types of optimal, parallelization-susceptible planning problems.

2.2.2 Knowledge-based Systems

The traditional search algorithm's ability to make predictions and deductions from data they already possess is limited in scope by implementation-related details. Information-centric architectures arose from a need to endow our procedures with the capacity to operate over the knowledge level, reasoning about things that they know in a domain-independent fashion.

At the heart of these frameworks lies a *knowledge-base*, the collection of sentences currently held as being true. When combined with an *inference engine*, algorithms can deduce new facts from these sentences, and add them to the database. Typically, a particular form of logic (*e.g.*, Propositional, Predicate, First-order, etc) is enforced when representing and manipulating these to discover new information.

Under the first heading, we briefly characterize Propositional Logic, the particular form of calculus upon which our method is focused. This is followed by an exploration and description of the two principal approaches for fact deduction: Model Checking and Theorem Proving.

Propositional Logic

Although elementary, the propositional system of logic [4] is still capable of drawing far-reaching conclusions. The *well formed formulae (wff)*, phrases or sentences allowed in its context are defined by a familiar *syntax* [44, p. 244], which we describe here in prose.

Its sentences are either *atomic* or *compound*. The atomic variants consist of a constant such as *True* or *False*, or of a single *propositional symbol*.[12] Such symbols represent logically indivisible statements, so if we consider "Joseph is a shepherd" to be an atomic phrase, it will either be true or false, with none of its constituents having a bearing on inference. Typically, we will use alphabetical letters to represent these.

Compound phrases, on the other hand, can be built from any sentence (atomic or compound) by the application of logical connectives, or by placing the phrase in brackets (usually '(' and ')' are used).

[12] also known as a *positive literal* or *propositional variable*. *Literals*, however, include both positive and negative literals. The latter result simply from negation of a propositional variable, but are not atomic phrases.

The connectives are five: the unary negation $\neg$ operator, and the binary disjunction $\vee$, conjunction $\wedge$, implication $\rightarrow$[13] and equivalence $\leftrightarrow$[14] relations.

It is within this logic framework that we intend to show how a quantum/classical inference system can be designed in order to prove expressions within that language. Presently however, we will briefly survey the two main classical techniques used to perform propositional inference.

Model Checking

It is usual to consider the KB as a single sentence, constructed from its fact collection using some logical connective (*e.g.*, conjunction). We may then ask our knowledge-based system whether a particular sentence (a formula) logically follows from the dataset,[15] also known as the *propositional entailment problem*.

One way to go about answering such a query is dubbed *model checking*. It consists in enumerating every possible combination of true/false attributions to the atomic sentences in the database,[16] similarly to how we solve Boolean Constraint Satisfaction Problems. The combinations themselves are referred to as *models*, and we say that a model satisfies a formula α whenever α is true for that configuration of assignments. Provided our query sentence is satisfied for every model where the KB is also evaluated truthfully, we may say that it follows logically from it, or is entailed by it [44, p. 240].

Propositional entailment is a co-NP-complete problem, so all known inference procedures for it are exponential on input size. The above process, for example, is $O(2^n)$, where n is the number of atomic propositions in the KB. Of course, it quickly becomes intractable.

However, an alternative model checking approach consists in proving that there are no models which satisfy both the KB and our query's negation.[17] Assuming our inquiry phrase is α, this equates to solving the SAT problem [10, 59], *i.e.*, finding a model which satisfies the Boolean *formulae* in KB $\wedge \neg\alpha$.

SAT is an NP-complete problem and has been the subject of much research over the years. The design of procedures able to solve these problems is a vast area of investigation, so we do not explore it further. Suffice it to say that modern SAT algorithms are capable of dealing with tens of thousands of propositional variables in an efficient manner.

Theorem Proving

The model checking approach examined attribution possibilities in order to prove propositional entailment. *Theorem proving* methods, on the other hand, successively apply inference rules to formulas in a KB, until some given query sentence is obtained,[18] which may be advantageous for domains where the number of possible models for the KB is quite large, while the length of a proof is short. This is true in many task worlds, since a proof process can ignore irrelevant variables during search.

[13] where α, β are wff, we define $\alpha \rightarrow \beta = \neg\alpha \vee \beta$. α is known as the *antecedent* of the implication rule, while β is termed its *consequent*.
[14] $\alpha \leftrightarrow \beta$ iff $\alpha \rightarrow \beta \wedge \beta \rightarrow \alpha$, where α, β are wff.
[15] for a logical sentence α, this can be expressed as KB $\models \alpha$, *i.e.*, α *semantically* follows from fact set KB.
[16] the method was independently discovered by [40] and [64].
[17] also known as a proof by contradiction.
[18] if a sentence α can be derived from KB we write KB $\vdash \alpha$, *i.e.*, α is *syntactically* derivable from KB via rules of inference.

A variety of inference rules may be used in deduction, and as long as these are *sound*[19] they shall preserve entailment. One of the most familiar rules, *Modus Ponens* (MP),[20] is an example of a sound inference mechanism. The *resolution* rule[21] is another.

Theorem provers can typically be divided into two categories: those which use the general form of the resolution rule for inference; and forward or backward chaining methods, which restrict that mechanism to a subset of propositional *formulae*.

As long as the problem of finding a proof is adequately defined, any classical search algorithm which is complete can be used to solve it. However, without suitable inference rules, even these techniques cannot produce a semantically complete[22] deduction algorithm.

When used in conjunction with a complete search procedure, the resolution rule is sufficient for construction of a complete inference system. Because the resolution mechanism accepts disjunctions of literals (in this context, referred to as *clauses*), these processes must first convert the KB and query formula to a specific canonical format, Conjunctive Normal Form (CNF).[23]

A proof by contradiction strategy is then employed: the technique attempts to prove the query formula α by finding an inconsistency,[24] demonstrating that $KB \wedge \neg\alpha$ cannot be satisfied. The resolution rule is tried for each pair of clauses within $KB \wedge \neg\alpha$, until either all options have been exhausted (in which case α is false in KB), or an empty clause is obtained, and α has been proven.

Since any sentence in Propositional Calculus can be converted to its CNF analogue, the methodology is complete for that language.

In many situations, however, resolution performs less efficiently than simpler alternatives. Provided our KB is restricted to a certain subset of propositional expressions, forward or backward-chaining approaches may outperform more general resolution techniques. The language of Horn clauses[25] is one such subset, and allows chaining algorithms to construct reasoning processes which are typically easier for us to follow than the general mechanisms.

Chaining algorithms make use of a KB consisting of definite clauses[26] to prove a particular Horn formula α, using either MP or a restricted form of resolution as their inference rule. In forward variants, such as that of [15], deduction begins with the KB's set of facts, and every implication whose antecedent is satisfied will assert its consequent literal into the database. This is done (with or without heuristic assistance) until such time as α is obtained, and has thus been proved, or every implication rule has been evaluated. Backward chaining, on the other hand, starts from query formula α, which the method attempts to infer by recursively finding supporting evidence in the KB for every literal in α, essentially

[19] an inference rule is sound when $\alpha \vdash \beta$ implies $\alpha \models \beta$, where α, β are formulas of a particular logic.

[20] *Modus Ponens* is one of Aristotle's four syllogistic patterns. It is the process of concluding that β is true, provided α and $\alpha \rightarrow \beta$ are, too. [62, p. 94].

[21] the rule's general form asserts that provided l_i and m_j are complementary (*i.e.*, the negation of each other), s.t. either l_i or m_j is an atomic sentence, then if $\bigvee_{t=1}^{n} l_t$ and $\bigvee_{t=1}^{k} m_t$, we may conclude the *resolvent* $l_1 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_n \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_k$. Introduced in [43], along with a proof of its completeness.

[22] a system is semantically complete iff $\alpha \models \beta$ implies $\alpha \vdash \beta$.

[23] advanced by [46], the notation represents any expression in Propositional Logic as a conjunction of disjunctive clauses.

[24] a set of formulas Γ is inconsistent iff some formula γ can be found s.t. $\Gamma \vdash \gamma$ and $\Gamma \vdash \neg\gamma$.

[25] these are disjunctions of literals containing at most one positive literal [29], allowing us to represent implication relations in a more direct manner. For instance, the Horn clause $\neg A \vee \neg B \vee C$ is by definition (and De Morgan's laws)[34] equivalent to $A \wedge B \rightarrow C$, where only a positive literal is concluded. Goal clauses equate to those having no positive literals, while a fact is a Horn clause with a single positive literal and no negative ones.

[26] a subset of Horn clauses, these are disjunctions of literals with exactly one positive literal.

performing the dual procedure with respect to forward chaining approaches.

Both chaining strategies are complete for Horn clauses, and more importantly, run in time linear with the size of the Knowledge-base.

Chapter 3

Blocks World Solver

To illustrate the technique, we begin with its description and analysis for 2-blocks world problems in section 2-EBW Solver. In n-EBW Solver, the approach is extended for problems of n blocks, and the generalization is adapted to environments where qubit availability might be reduced, in Reducing Space Requirements. We conclude the chapter with Decomposition Heuristic, where we propose adopting an alternative heuristic perspective when solving problems using these computers.

3.1 2-EBW Solver

We assume the initial and final states of such problems are of a form similar to that portrayed in figure 3.1. This will also serve as our example problem instance.

Since each block may be either on the table or over the other cube, a single qubit is sufficient for keeping track of its position. Two boxes mean a total of two qubits, s_1 and s_2, can characterize a state. Similarly, one qubit is enough to reflect a single movement decision: we can either move block A or B. We will assume the encoding scheme in table 3.1 throughout our exposition.

Bit \ Value	0	1
s_1	A on table	A on B
s_2	B on table	B on A
m	move A	move B

Table 3.1: Encoding scheme for problems in 2-EBW used in our presentation.

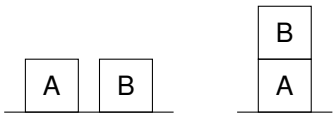

Figure 3.1: A 2-EBW problem: The block configuration on the left is the initial state, with the final state on the right.

Given a goal test of the form:

$$g(s_1, s_2) = \begin{cases} 1, & \text{if } (s_1, s_2) \text{ is a goal state} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

we are able to construct the goal test transform in accordance with equation (2.3), which we will designate as G in our calculations (respectively, **Goal Test** in our circuits). Assuming registers of the form $|s_1 s_2, a\rangle$, where a is the auxiliary qubit used to calculate the result of $g(s_1, s_2)$, then G maps every state s.t. $(s_1, s_2) \neq (0, 1)$ into itself. For $(s_1, s_2) = (0, 1)$, we have that $|s_1 s_2, a\rangle \mapsto |s_1 s_2, a \oplus g(s_1, s_2)\rangle$. The entire matrix can be consulted in appendix B.

We can extract a block movement operator similarly. Moving one of the boxes results in a new state description, so our output now involves two auxiliary qubits, as well as the input state (s_1, s_2) and a movement choice m. Thus, given a function $\boldsymbol{f}(s_1, s_2, m)$ which computes the new state:

$$\boldsymbol{f}(s_1, s_2, m) = \begin{cases} (\neg m, m), & \text{if } (s_1, s_2) = (0, 0) \\ (0, 0), & \text{if } (s_1, s_2, m) = (0, 1, 1) \\ (0, 0), & \text{if } (s_1, s_2, m) = (1, 0, 0) \\ (s_1, s_2), & \text{otherwise} \end{cases} \tag{3.2}$$

its unitary operator T (**Move Block** gates in our circuitry) can be determined. We leave its specification for appendix B.

The two operations can be portrayed as black box logic gates in a quantum circuit, as in figure 3.2. A single movement operation is presented in figure 3.3.

As we have stated in the previous chapter, circuits such as that of figure 3.3 do not correctly reflect equation (2.4). Notice that the input component of the equation, the $|\boldsymbol{x}\rangle$ register, is no longer present in our output. The qubits a_1 and a_2 have been altered, and as a consequence, Grover's algorithm will not guarantee correct results. We may undo these changes by applying the inverse of each operator, and preserving the circuit's outcome with a C_{not} gate, as in figure 3.4.

If we consider that the entire circuit is modelled by an operator U_c, its application to the problem depicted in figure 3.1 results in the following superposition:

$$U_c \, |00\rangle \otimes H \, |0\rangle \otimes |000\rangle \otimes |-\rangle = \frac{1}{\sqrt{2}}(|000000\rangle - |001000\rangle) \otimes |-\rangle \tag{3.3}$$

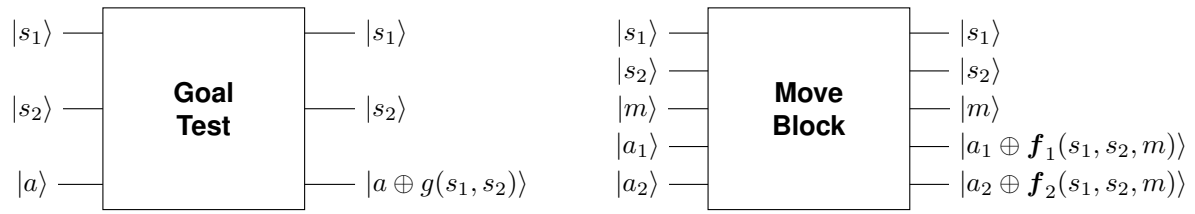

Figure 3.2: 2-EBW unitary operator gates: The left gate is the reversible analogue to the goal test function $g(s_1, s_2)$. The one on the right calculates the state resulting from the movement m on block configuration (s_1, s_2), according to transition function $\boldsymbol{f}$. Note that $\boldsymbol{f}_i$ are register $\boldsymbol{f}$'s components.

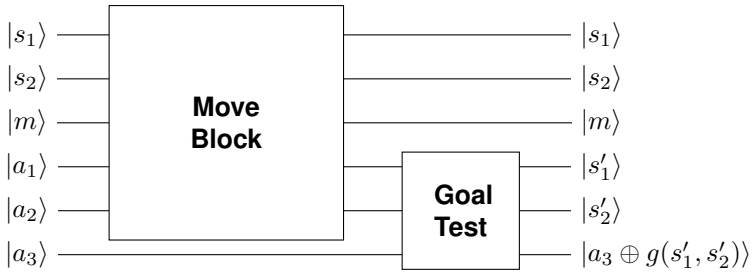

Figure 3.3: Circuit for a single movement operation and goal test in 2-EBW problems. Note that $s'_i = a_i \oplus \boldsymbol{f}_i(s_1, s_2, m)$.

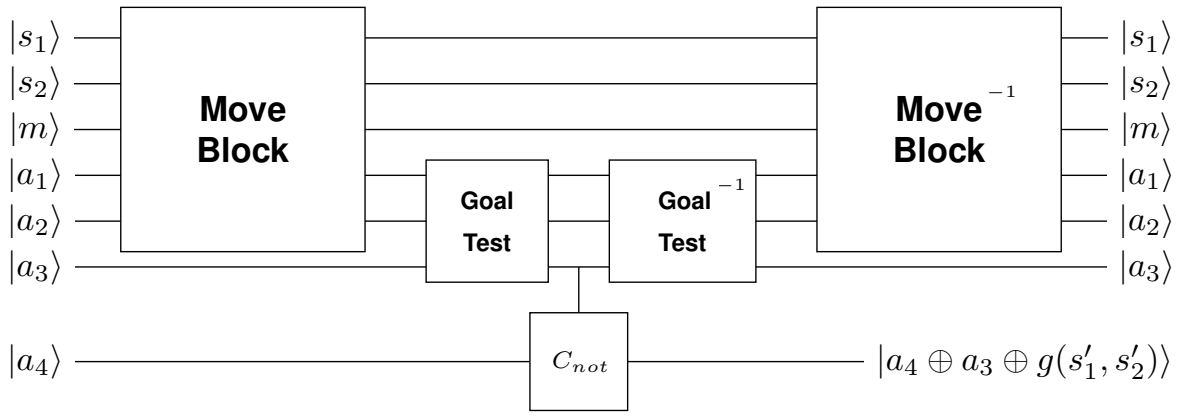

Figure 3.4: Circuit for a single movement operation and goal test in 2-EBW, complying with Grover's formalism. The final result will be stored in the bottom qubit, where the outcome of the goal test is used as the control for the C_{not} gate, and $s'_i = a_i \oplus \boldsymbol{f}_i(s_1, s_2, m)$.

where we assume the layout of the qubits follows the pattern $|s_1 s_2 m a_1 a_2 a_3 a_4\rangle$. Note the application of a Hadamard transform to those qubits representing the different path choices m, and the auxiliary superposition $|-\rangle$ prescribed by Grover's technique. The full computation is exhibited in appendix C.

In this case, the resulting superposition is composed of merely two basis elements, so the solution $|001000\rangle$ is easily discerned, where moving block B leads to the goal state. In superpositions consisting of a higher number of bases, running Grover's algorithm over them will deterministically recover the solution.

Iterative-deepening Although the circuit in figure 3.4 is able to solve any 2-EBW problem whose initial state is a single movement removed from its final one, it is incapable of finding solutions for other instances.

For example, consider the 2-EBW problem $\langle \begin{smallmatrix}\boxed{B}\\\boxed{A}\end{smallmatrix}, \begin{smallmatrix}\boxed{A}\\\boxed{B}\end{smallmatrix} \rangle$ where we regard the left-hand side of the tuple to be the initial configuration and the right-hand side the goal state. In this case, we can only achieve our goal after executing at least two movements.

Because we do not always know how many actions must be performed in order to obtain a solution, we adopt the QIDS approach when designing these quantum circuits, in keeping with their portrayal of figure 2.5.

3.2 n-EBW Solver

For an arbitrary n-block EBW problem, we can follow a similar encoding strategy. We may represent a state in such a world by registering the position of each block, *i.e.*, which object it is on. Each of these may be over one of $n-1$ blocks, or the table (totalling n possibilities). Thus, n^n different state configurations exist, for which $s = \lceil \log_2 n^n \rceil$ qubits are required.

In addition, a single block movement can be defined as a (*source*, *destination*) pair, in which case n blocks may be chosen as a *source*, with $n-1$ other blocks[27] as a *destination*. This amounts to $n(n-1)$ eventualities, and $m = \lceil \log_2 n(n-1) \rceil$ qubits for each action.

Assuming we have access to goal test $g(\boldsymbol{s})$ and transition result $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{m})$ functions analogous to those in equations (3.1) and (3.2), each of our operators can be reversibly defined as per equation (2.3), where $\boldsymbol{s}$ is a state description vector in the complex Hilbert space $\mathcal{H}^{2^s}$ and $\boldsymbol{m}$ encodes our movement in a space $\mathcal{H}^{2^m}$. Consequently, our goal test gate G now accepts a state representation consisting of s qubits, and a single auxiliary qubit in which to calculate the test. Similarly, the movement gate T takes s qubits describing the current state, m qubits defining the movement decision, and a further s auxiliary qubits in which to compute the resulting state. Both gates are depicted in figure 3.5. Notice that the largest of these is the movement transition transform, requiring $O(2^{2s+m})$ pre-calculations.

Using the QIDS methodology reflected in figure 2.5's design for n-EBW problems, we see that a total of $n_{c,d} = s + d(m+s) + 2$ qubits are needed for construction of a depth d search (*i.e.*, performing d movements), $n_{c,d} - (s + dm) = ds + 2$ of which are auxiliary qubits. Applying Grover's algorithm means that each iteration requires $O(\sqrt{N})$ steps, with $N = 2^{dm}$, *i.e.*, the number of different paths. Naturally, if the shallowest depth for a solution is l, the entire process takes $O(\sqrt{2^{lm}})$ steps, with $O(l(m+s))$ space-complexity.

We may structure the d^{th} such circuit's input register $|\mathbf{x}\rangle$ into several functionally distinct segments, like so $|\mathbf{x}\rangle = |\boldsymbol{s}\rangle \left(\bigotimes_{j=1}^{d} |\boldsymbol{m}_j\rangle |\boldsymbol{a}_j\rangle\right) |a_{d\cdot s+1}\rangle |a_{d\cdot s+2}\rangle$, where $|\boldsymbol{s}\rangle$ contains the initial state description, $|\boldsymbol{m}_j\rangle$ and $|\boldsymbol{a}_j\rangle$ correspond (respectively) to the action superposition and ancillary qubits for the j^{th} application of the movement operator, $|a_{d\cdot s+1}\rangle$ is the auxiliary qubit for the goal test, and $|a_{d\cdot s+2}\rangle$ for the C_{not}.

Accordingly, for circuit d, we can define the unitary operator $U_{c,d}$, as per equation (2.6). This would act over members of the above register space, performing the n-block equivalent of the computation

[27] when the source is over another block, this includes the table, but not the block it is already on.

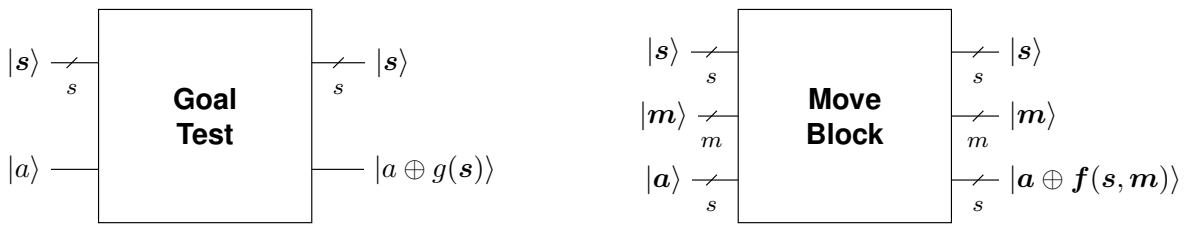

Figure 3.5: Reversible operator gates for an n-blocks EBW problem: The left gate is the reversible analogue to the problem's goal test function $g(\boldsymbol{s})$, assuming $\boldsymbol{s} \in \mathcal{H}^{2^s}$ describes a particular state, with $s = \lceil \log_2 n^n \rceil$. The right gate computes the state resulting from the movement $\boldsymbol{m} \in \mathcal{H}^{2^m}$ on block configuration $\boldsymbol{s}$, where $m = \lceil \log_2 n(n-1) \rceil$.

suggested by figure 2.5.

Note that, EBW problems do not have a constant branching factor. For example, we may act in two distinct ways when on the initial state of figure 3.1, whereas on its final one a single action is available to us. However, for the sake of practicality, our encoding scheme assumed that a problem's maximum branching factor $b_{max} = n(n-1)$, would determine the number of actions in every state. Furthermore, because that scheme is binary, and b_{max} is not always a power of two, certain ranges of values for b_{max} will require the same number of qubits. This is reflected in the ceiling function for $\lceil \log_2 n(n-1) \rceil$. The excess bit strings, *i.e.*, those corresponding to an invalid action for a specific state, are suitably understood as effecting no operation under T, as we have mentioned in chapter 2.

3.3 Reducing Space Requirements

The reader will note that the n-EBW Solver we have presented requires a number of qubits that is linearly depth-dependent. However, in all likelihood, employing qubits will continue to be an expensive affair for a long time to come. Therefore, we may wish to decrease the amount of qubits the solver requires, at the expense of further one-off, pre-runtime calculation steps.

Each iteration d of the method produces a circuit requiring $n_{c,d} = s + d(m+s) + 2$ qubits. Of these, $n_{c,d} - (s + dm) = ds + 2$ are auxiliary, *i.e.*, correspond to intermediate calculations. Because $s \gg m$, eliminating these will conserve more memory than any alternate encoding for our actions. This can be done, for instance, by calculating the results from performing the d actions in iteration d at once, instead of saving the intermediate states.

Assuming we have the function $\boldsymbol{f}$ of n-EBW Solver, which outputs the state produced when executing an individual action $\boldsymbol{m}$ in a given state $\boldsymbol{s}$, then we may define the construct described in the previous paragraph:

$$\boldsymbol{f}^{\circ d}(\boldsymbol{s}, \boldsymbol{w}) = \begin{cases} \underbrace{\boldsymbol{f}(\ldots \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{m}_1) \ldots, \boldsymbol{m}_d)}_{\text{d times}}, & \text{if action sequence} \\ & \boldsymbol{w} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_d) \\ & \text{is valid in state } \boldsymbol{s} \\ \boldsymbol{s}, & \text{otherwise} \end{cases}$$

where our d actions are sequentially ordered into the vector $\boldsymbol{w} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_d)$, with $\boldsymbol{m}_d$ corresponding to the last chronological movement, $\boldsymbol{m}_1$ equating to the first, and each $\boldsymbol{m}_{i+1}$ immediately following $\boldsymbol{m}_i$ for all $i \in \mathbb{N}$ s.t. $i < d$. Note that an action sequence $\boldsymbol{w}$ is said to be valid when each of its individual actions are applicable in the state on which they are to be performed. Given these definitions, $\boldsymbol{f}^{\circ d}(\boldsymbol{s}, \boldsymbol{w})$'s[28] reversible counterpart, which we refer to as T_d, can be extracted using equation (2.3), producing the circuit in figure 3.6.

While we may still use the state encoding described in n-EBW Solver, our actions must be mapped differently. Because we wish to perform the d actions in a single operation and each requires $m = \lceil \log_2 n(n-1) \rceil$ qubits, a total of $w = dm$ qubits can be used for path specification. Consequently,

[28] we apologise for the slight abuse of compositional notation in our reference to $\boldsymbol{f}^{\circ d}$.

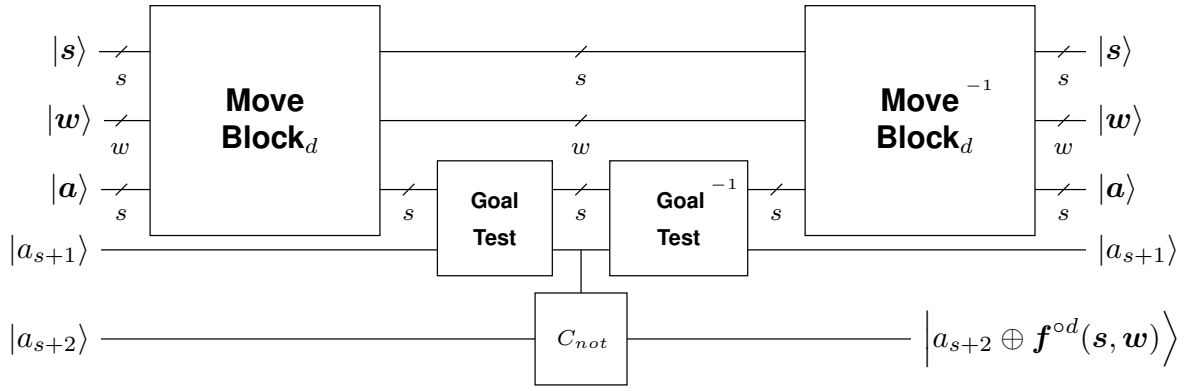

Figure 3.6: Circuit using the **Move Block**$_d$ operation (*i.e.*, T_d), computing the result of a sequence of d movements and a goal test in n-EBW in a single step, and preserving the input required by Grover's Amplification. $\boldsymbol{s}, s$ and m are as in figure 3.5. $w = dm$, and $\boldsymbol{w} = (\boldsymbol{m}_1, \ldots, \boldsymbol{m}_d)$ is the path register, where each $\boldsymbol{m}_i \in \mathcal{H}^{2^m}$ corresponds to an action over the i^{th} state obtained from executing actions $\boldsymbol{m}_1$ to $\boldsymbol{m}_{i-1}$ sequentially, beginning at the initial state $\boldsymbol{s}$.

an action sequence vector $\boldsymbol{w}$ lies in a $\mathcal{H}^{2^{dm}}$ Hilbert space, with separate qubits for different actions, organized hierarchically so that those referring to $\boldsymbol{m}_{i+1}$ are contiguous to those of $\boldsymbol{m}_i$, and possess a lower order of magnitude (for instance).

The T_d operation involves a total cost of $2s + w = 2s + dm$ qubits. Whenever $d \geq 1, n \geq 2$, this is less than or equal to the $s + d(m + s)$ qubits needed solely for movement computation in the n-EBW extension to figure 2.5's circuit. For the entire circuit of figure 3.6, the tally is $n_{c,d} = 2s + dm + 2$ qubits, and the amount of auxiliary space, $n_{c,d} - (s + dm) = s + 2$, is no longer depth-dependent. Assuming the shallowest solution has depth l, the solver will still extract a solution in $O(\sqrt{2^{lm}})$ steps. However, its circuits involve merely $O(s + lm)$ space-complexity. The absence of a depth coefficient over s is of note, considering its higher degree compared to m.

Because the goal test and C_{not} operators add a constant number of qubits to our circuits, the qubit reduction achieved from pre-calculating their operations within such a construct would be negligible. Thus, we assume their operation remains reflected within separate gates, and is identical to that observed in the previous sections.

While the reduction in space is significant, the new movement operator has a drawback: for every iteration d, the T_d transform must have been pre-calculated. The cost of this computation is bounded from above by that of determining the largest such matrix, equivalent to $O(2^{2s+lm})$ steps. Although this is a one-off calculation, it rapidly becomes intractable, so would only be practical for less complicated instances.

An additional consequence is that the technique is only applicable up to a certain depth-limit, unlike our n-EBW Solver. Fortunately, in any n-EBW problem, the worst case length of an optimal solution is bounded from above by $2n$, equivalent to placing every block over the table, and then stacking each into their final positions in a bottom-up fashion [27]. Thus, we need only determine the T_d transforms up to that depth.

3.4 Decomposition Heuristic

While the quadratic speedup obtained from employing Grover's algorithm is desirable, one may naturally consider if it would be possible to yield further gains using quantum analogues to classical heuristics. The latter usually provide a way to compare between different states, so as to select the best option among them.

In order to simulate such behaviour within a quantum framework, we would need the ability to exchange information between the various states of our superposition. Unfortunately, quantum mechanical laws such as the No-cloning theorem impose severe limitations on the communication mechanisms that may be applied.

While no general information transfer technique exists for such purposes, global state properties *can* be used to incorporate heuristic methodologies into quantum frameworks. In particular, Tarrataca and Wichert [55] were able to integrate heuristic concepts into their QIDS approaches by estimating the cost of executing an action sequence from a given state, and limiting solutions to those for which this appraisal laid below a specified threshold.

They assumed heuristic functions reflecting some form of probabilistic distribution, and studied these for both 8-puzzle and n-puzzle instances. For instance, their 8-puzzle experiments considered the discrete distribution heuristic counting the number of misplaced tiles of a state in relation to the goal, whereas for n-puzzles, the sum of Euclidean distances between tiles of both states was adopted.

By assuming states closest to the goal were less common than others, and setting the aforementioned threshold to the value of a quantile function for a cumulative distribution of the corresponding heuristic, the authors developed a mechanism for selecting a state within a particular goal-distance.

The process involves a single Grover iterate and, in a sense, can be used to cull every state not within the best one-fourth of the entire state-space, before executing a search. However, Grover's algorithm must still perform the same amount of steps in order to obtain a single state within this range.

In this section, we argue that a different heuristic perspective must be adopted within quantum computation, to circumvent the communication restrictions preventing path comparison. In particular, we propose that divide-and-conquer heuristics are more useful in such a scenario. These methods are capable of simplifying search processes whenever certain problem regularities are found, decomposing them into less complicated sub-problems.

We demonstrate an example of such a heuristic for an n-EBW problem. Suppose we can extract the directed acyclic graphs $\mathcal{G}_I = (V_I, E_I)$ and $\mathcal{G}_F = (V_F, E_F)$ from whichever representation we hold of the initial and final states of the problem, where V_i is graph $\mathcal{G}_i$'s set of *vertices*, containing a labelled vertex for each block in state i, and E_i is $\mathcal{G}_i$'s set of *edges*. For every block x over another y in state i s.t. y is not the table, a directed edge (x, y) is included in E_i. Figure 3.7 gives examples of these graphs for the 6-EBW problem in figure 3.8.

From these structures, we are able to devise an undirected graph $\mathcal{G} = (V, E)$, where $V = V_I = V_F$ and every edge in $E_I \cup E_F$ is present in E in an undirected manner. Figure 3.9 shows $\mathcal{G}$ for the cited 6-EBW instance. A trivial, uninformed search procedure over it can obtain the graph's *connected*

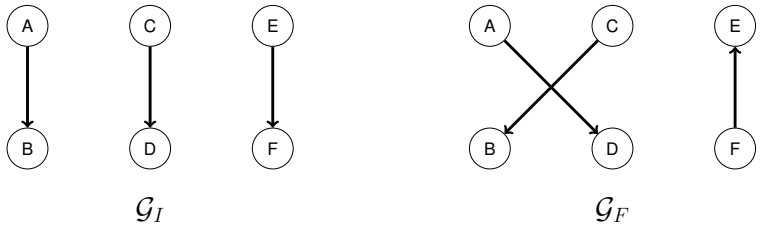

Figure 3.7: Directed acyclic graphs for representing initial ($\mathcal{G}_I$, left) and final ($\mathcal{G}_F$, right) states of the 6-EBW problem portrayed in figure 3.8.

A	C	E
B	D	F

C	A	F
B	D	E

Figure 3.8: 6-EBW instance used to describe the decomposition approach. As usual, to the left lies the initial state and to the right our goal (these correspond respectively to the graphs $\mathcal{G}_I$ and $\mathcal{G}_F$ of figure 3.7).

components,[29] *i.e.*, sets of vertices reachable to and from each other along a sequence of edges. These components capture sets of blocks whose movements from the initial to the final state *may* depend on each other's interim positions.

For each such component $V^{(i)}$, we then extract from $\mathcal{G}_I$ and $\mathcal{G}_F$ the sub-graphs $\mathcal{G}_I^{(i)}$ and $\mathcal{G}_F^{(i)}$ induced by $V^{(i)}$.[30] These may be understood as representing partial-states, *i.e.*, reduced sections of the more general states described by $\mathcal{G}_I$ and $\mathcal{G}_F$. In practice, we have decomposed our problem $\langle \mathcal{G}_I, \mathcal{G}_F \rangle$ into (hopefully) smaller, $|V^{(i)}|$-block EBW sub-problems $\langle \mathcal{G}_I^{(i)}, \mathcal{G}_F^{(i)} \rangle$, where each part-state of component i can be encoded by our solver approach using $s^{(i)} = \lceil \log_2 |V^{(i)}|^{|V^{(i)}|} \rceil$ qubits.

Assume the initial and final part-state configuration encodings are $\boldsymbol{s}_I^{(i)}$ and $\boldsymbol{s}_F^{(i)}$. Then we may define the following goal test function, where $\boldsymbol{s}^{(i)}$ represents a part-state within component i's state-space:

$$g^{(i)}(\boldsymbol{s}^{(i)}) = \begin{cases} 1, & \text{if } \boldsymbol{s}^{(i)} = \boldsymbol{s}_F^{(i)} \\ 0, & \text{otherwise} \end{cases}$$

and craft its respective goal test operator as we have previously described.

[29] the formal definition prescribes these to be vertex equivalence classes, under an "is reachable from" relation [11, p. 88].

[30] given a set $V' \subseteq V$, the sub-graph of $\mathcal{G} = (V, E)$ induced by V' is $\mathcal{G}' = (V', E')$ s.t. $E' = \{(u, v) \in E : u, v \in V'\}$ [11, p. 88].

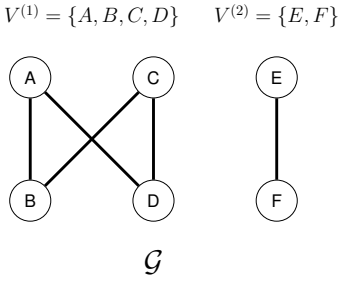

Figure 3.9: The undirected graph $\mathcal{G}$, crafted from $\mathcal{G}_I$ and $\mathcal{G}_F$ of figure 3.7. Two connected components $V^{(1)}$ and $V^{(2)}$ can be extracted from it, which reflect sets of blocks whose movements from initial to final configuration may be interdependent.

If individual movement operations are considered, we know that $m^{(i)} = \lceil \log_2 |V^{(i)}|(|V^{(i)}| - 1) \rceil$ qubits are sufficient for capturing a decision. Thus, given the transition function $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{m})$ for the generalized states, we can obtain an $\boldsymbol{f}^{(i)}(\boldsymbol{s}^{(i)}, \boldsymbol{m}^{(i)})$ for sub-problem i. To do so, we must define a relevant general state from which the action is to be performed. Let $\boldsymbol{s}_I'^{(i)}$ be the generalized state in which every block in $V^{(i)}$ is over the same block specified in part-state encoding $\boldsymbol{s}^{(i)}$, while every other block is on its position in $\boldsymbol{s}_I$.[31] Likewise, let $\boldsymbol{m}'^{(i)}$ define the generalized encoding of the action equivalent to $\boldsymbol{m}^{(i)}$. We may then extract those values of $\boldsymbol{f}(\boldsymbol{s}_I'^{(i)}, \boldsymbol{m}'^{(i)})$ which are relevant to our $V^{(i)}$ blocks, and map them from the generalized encoding to our partial state representation. This gives us $\boldsymbol{f}^{(i)}(\boldsymbol{s}^{(i)}, \boldsymbol{m}^{(i)})$, which we may use to compute the unitary movement operator for sub-problem i.

Finally, for each $V^{(i)}$ with more than one block, we prepare a superposition of actions $\boldsymbol{w}^{(i)}$ and a quantum circuit according to the method shown in section n-EBW Solver, where the initial state qubits should correspond to part-state $\boldsymbol{s}_I^{(i)}$. Running the described iterative-deepening approach alongside Grover's algorithm over these will recover the solution.

If there are k such connected components with $|V^{(i)}| > 1$, we obtain as many movement sequences, *i.e.*, solution paths $\boldsymbol{p}^{(i)}$. Since blocks in $V^{(i)}$ are never affected by those in $V^{(j)}$, where $i \neq j$, (*i.e.*, there are no deleted or enabling-condition interactions between the components) these can be independently ordered or interleaved in any plan taking us from the initial state to the final one. In fact, the same holds for any of the individual actions $\boldsymbol{m}^{(i)}, \boldsymbol{m}^{(j)}$ of different paths, as long as their relative order within their own $\boldsymbol{p}$ is preserved. In a sense, this result is similar to what a POP framework would obtain.

The entire approach has total time-complexity of $O(\sum_{i=1}^{k} \sqrt{2^{l^{(i)} m^{(i)}}})$ oracle queries, where $l^{(i)}$ is the shallowest solution depth of the i^{th} connected component, *i.e.*, the length of $\boldsymbol{p}^{(i)}$. Naturally, the technique is dependent on how difficult the most complex of the sub-problems is. However, a less restricted but easier to determine upper bound can be defined. Since the worst case optimal plan length for every $|V^{(i)}|$-block world is double the number of blocks, we know the process is $O(\sum_{i=1}^{k} \sqrt{2^{2|V^{(i)}| m^{(i)}}}) = O(\sum_{i=1}^{k} 2^{|V^{(i)}| m^{(i)}})$. This is dominated by the component with the most blocks, and so is $O(2^{|V^{(j)}| m^{(j)}})$, assuming $\forall_{i, i \neq j} |V^{(i)}| \leq |V^{(j)}|$.

The process is able to reduce time-complexity whenever a problem is decomposable. We posit that this is true of many problem instances, where certain parts of the problem are not interdependent.

In remaining situations, the method does not increase time-complexity or compromise answer optimality. If every block is interdependent, a single connected component is extracted with $|V^{(1)}| = |V|$, yielding time-complexity $O(\sqrt{2^{lm}})$, equivalent to solving the more general problem without using the heuristic, where l is the shallowest solution depth and m is as defined in section n-EBW Solver.

On the other hand, if we assume more than one component is found, we admit that the largest exponent in the terms of sum $\sum_{i=1}^{k} \sqrt{2^{l^{(i)} m^{(i)}}}$ occurs for some connected component j. Then decomposition allows us to solve the instance in $O(\sqrt{2^{l^{(j)} m^{(j)}}})$ steps, with $l^{(j)}$ the shallowest solution depth for the j^{th} sub-problem. The method we have described always finds the shortest paths for each of the sub-problems, and resolving the problem in this manner will never conceive plans which include movements between blocks in disconnected components. Consequently, we know $l^{(j)} \leq l$. Since $s^{(j)} < s$ and

[31] it is not particularly relevant whether $\boldsymbol{s}_I$ or $\boldsymbol{s}_F$ is actually considered for blocks not in $V^{(i)}$. Because our part-state transition $\boldsymbol{m}^{(i)}$ merely refers to blocks within $V^{(i)}$, the rest will not be affected.

$m^{(j)} < m$, time and space-complexity are improved.

However, such a technique is not always applicable. In the n-puzzle case [54], our divide-and-conquer approach is not so easily achieved, particularly when optimal solutions are desired. Some parts of the puzzle may not be independently solvable, and in many situations, a player must move tiles already placed in their final position to complete other regions of the board.

Chapter 4

Quantum Propositional Inference

By accomplishing propositional inference using a combination of quantum and classical processes, we propose a knowledge-based mechanism which is able to prove Propositional Calculus expressions. Our strategy is detailed in this chapter.

We begin by presenting the Inference Principles upon which our system is grounded, along with a proof by cases approach to deducing expressions. Here, we also introduce the principal algorithm achieving that purpose. We then characterize the process of transforming each fact of a KB consisting of sentences from Propositional Logic into a structured dataset S suitable for our goals, in section Preparing the Knowledge-base. In Fact Representation we discuss how we have chosen to portray the state of inferences within our framework, as well as the alternatives considered in that regard. Section Dividing the Knowledge-base contains a detailed exposition of how we decompose KB S into several sub-KBs σ of S, where proofs may be crafted in a more direct fashion. The use of quantum search techniques in our Deductive Process is subsequently elaborated upon. Finally, our Analysis of the Method is conveyed.

4.1 Inference Principles

Suppose we are given a certain knowledge-base $\mathcal{KB}$, a collection of assertions in Propositional Logic.

We designate $S = \langle \mathcal{F}, \mathcal{R} \rangle$ as the structured fact database resulting from the particular set of transformations we describe in the next section. These will partition the original KB into two sets: whereas the *set of facts* $\mathcal{F}$ will contain those expressions in $\mathcal{KB}$ that are classified as *simple sentences* (*i.e.*, they do not explicitly follow a pattern $\alpha \rightarrow \beta$, where an implication is the operator with the *lowest precedence*),[32] the *set of rules* $\mathcal{R}$ shall encompass the remaining *formulae*.

Elements of $\mathcal{F}$ will then correspond to a collection of states of inference (with each state consisting of a semantic valuation of every literal), and rules in $\mathcal{R}$ to propositions upon which MP may be applied to produce new assessments. The modification procedure induces a simpler characterization of a deductive state, as well as of the inference process itself, while still affording the knowledge-base designer with specification flexibility.

[32] some authors also describe such connectives as having *greater force* or *scope* [62, p. 9].

Algorithm 2 Top level routine of our method, which attempts to prove the query η from the assumptions in S. Returns a double in which the first element is $True$, $False$ or $Impossible$. When $True$, the second element of the tuple contains the proof. $False$ indicates $S \nvdash \eta$, and $Impossible$ that S is a contradictory set of facts.

```
procedure PROVE(S, η)
    ALLσCONTRADICTORY ← True
    PROOF ← ∅
    σPROOF ← ∅

    SYSLIST ← DIVIDE-KB(S)

    for each σ ∈ SYSLIST do
        if CONTRADICTORY?(σ) = False then
            ALLσCONTRADICTORY ← False
            σPROOF ← PROVE-QUERY(σ,η)
            if FIRST(σPROOF) = False then
                return ⟨False, ∅⟩
            else
                PROOF ← PROOF ∪ {⟨σ, SECOND(σPROOF)⟩}
            end if
        end if
    end for each

    if ALLσCONTRADICTORY = True then        ▷ all contradictory ↔ no model satisfies S
        return ⟨Impossible, ∅⟩
    else
        return ⟨True,PROOF⟩
    end if
end procedure
```

Often, we will also refer to S as $\mathcal{F} \cup \mathcal{R}$. The two definitions are analogous, and represent the same KB in different ways, relying on context to disambiguate.

Our inference engine is predicated upon the use of two traditional inference rules, namely: *Modus Ponens* and *disjunction elimination*. As stated, we wish to determine if a given query expression η,[33] given in propositional form, follows from S. Algorithm 2 describes our approach, though we also provide an overview of our strategy here.

We first transform the facts in $\mathcal{F}$ so that they posses only expressions containing $\neg, \vee, \wedge$. This is done in accordance with definitions for the remaining operators of Propositional Calculus. In addition, rules in $\mathcal{R}$ are modified similarly, so that their antecedents and consequents are expressed merely with the connectives above.

Because it is more direct to carry out deduction in a KB σ whose facts in $\mathcal{F}$ and consequents of $\mathcal{R}$ are sentences composed only with literals and $\wedge$, a proof by cases strategy is employed in deriving query η. This requires that we first eliminate the disjunctions present in these expressions. We do so by dividing S into the unique sub-cases σ of the KB, each portraying the disjunction alternatives, and such that $\bigvee_i \sigma_i = S$.

In essence, each of the resulting independent knowledge-bases σ consist of a collection of working hypotheses, the cases over which disjunction elimination (or *proof by cases*) may be performed. Obtain-

[33] where it is assumed that the propositional symbols in η are present in $\mathcal{F} \cup \mathcal{R}$.

ing a proof of η in S can be accomplished by deducing a set of proofs for all of the σ. If two systems σ_1 and σ_2 were generated from the process of KB division, we would extract a proof using this technique. Frequently, its application is depicted as follows:

$$\dfrac{\begin{array}{ccc} & [\sigma_1] & [\sigma_2] \\ & \vdots & \vdots \\ (\sigma_1 \vee \sigma_2) & \eta & \eta \end{array}}{\eta}\, E_\vee$$

where the vertically bracketed σ_i are hypotheses, and the vertical dots indicate intermediate deductions via MP from which η can be inferred.

Thus, in addition to indicating whether or not a proof attempt has succeeded, Algorithm 2 returns the set PROOF, a collection of $\langle \sigma_i, \alpha_{i,1} \ldots \alpha_{i,l_i} = \eta \rangle$, where $\alpha_{i,1} \ldots \alpha_{i,l_i} = \eta$ is the query's proof in each σ_i, with the $\alpha_{i,j}$ wff, and l_i the length of such a sequence in σ_i.

4.2 Preparing the Knowledge-base

Before inclusion into $\mathcal{F}$, simple statements must undergo the transformation process described below. Every $\leftrightarrow$ and $\rightarrow$ operator in the expression should be expanded in accordance with the definitions for implication[13] and equivalence.[14] Subsequently, every negation is successively administered using De Morgan's laws[34] until these all occur in immediate precedence to a positive literal. Thus, a formula $\neg\big[A \vee (B \leftrightarrow C)\big]$ would first suffer expansion into $\neg\big[A \vee \{(\neg B \vee C) \wedge (\neg C \vee B)\}\big]$, followed by a deepening of negation. This would result in the expression $\neg A \wedge \big[(B \wedge \neg C) \vee (C \wedge \neg B)\big]$, added to the fact set $\mathcal{F}$.

Likewise, those phrases in $\mathcal{KB}$ not classified as simple sentences, thus being in the form $\alpha \rightarrow \beta$, will suffer the same transformations, but only for α and β. This leaves us with an expression retaining a single implication, where α and β contain no $\leftrightarrow$ or $\rightarrow$ operators. For instance, the syllogism $(B \rightarrow C) \rightarrow \big[(A \rightarrow B) \rightarrow (A \rightarrow C)\big]$ would initially become the phrase $(\neg B \vee C) \rightarrow \big[\neg(\neg A \vee B) \vee (\neg A \vee C)\big]$, and finally $(\neg B \vee C) \rightarrow \big[(A \wedge \neg B) \vee \neg A \vee C\big]$ would be added to $\mathcal{R}$.

Finally, an index over each sentence in the KB should be constructed, recording whether the phrase exhibits a disjunctive connective and if so, cataloguing the set of its disjunctive cases. For example, formula $A \wedge (B \vee C)$ would be stored as bearing a disjunction, along with its set of individual cases $\{A \wedge B \wedge C\}$, $\{A \wedge \neg B \wedge C\}$ and $\{A \wedge B \wedge \neg C\}$. Although the expansion is exponential, it is limited to the propositional variables within the sentence, and need only be done once, upon which multiple queries to the KB may be performed.

We assume Algorithm 2 to accept a KB $S = \langle \mathcal{F}, \mathcal{R} \rangle$ that has already been processed in this manner, and can access the above index in $O(1)$ time. Thus, when S is divided into its disjunctive cases $\sigma = \langle \mathcal{F}_i, \mathcal{R}_j \rangle$, definite semantic values for each disjunction in $\mathcal{F}$ will be obtained from σ's fact set $\mathcal{F}_i$. Similarly, because for each rule $\alpha \rightarrow \beta \in \mathcal{R}_j$, its consequent is now a conjunction of literals, it is easy to evaluate whether α is satisfied, and to modify literal values to conclude β when this is so.

[34] the two laws state that if α, β are wff, then we have: $\neg(\alpha \vee \beta) \leftrightarrow \neg\alpha \wedge \neg\beta$ and $\neg(\alpha \wedge \beta) \leftrightarrow \neg\alpha \vee \neg\beta$ [62, p. 120].

Note that although some facts of $\mathcal{F}$ could have been transformed in a different manner and included in $\mathcal{R}$ as rules (*i.e.*, those with an $\vee$ connective as the least precedent operator), we have chosen to provide the KB designer with some flexibility in defining assertions. Such a choice essentially translates into his/hers ability to prescribe a set of models in a more straightforward fashion.

4.3 Fact Representation

Our first task lies in assuring that the state of inferences have a manageable, but faithful representation.

If we were to consider capturing the truth/falsehood of every possible statement resulting from a combination of literals under the $\vee$ and $\wedge$ connectives, our memory requirements would sky-rocket.

As an example, consider the set of positive literals $\Sigma = \{A, B\}$. If we chose the above exhaustive approach, we would have to verify whether any obtainable propositional expression had been asserted after an inference. These are infinite, and each would require a qubit to record its truth or falsehood. However, there are only a total of $2^{2^2} = 16$ [12, p. 104] unique Boolean functions (*i.e.*, truth tables) $H : \mathbb{B}^2 \to \mathbb{B}$ under Σ, where $\mathbb{B} = \{\mathsf{F}, \mathsf{T}\}$, F represents a false value and T a true one. These tables uniquely determine (and are in turn singularly determined by) a sentence in a canonical format, namely *full Disjunctive Normal Form (DNF)*.[35] It is not surprising then that any propositional phrase can be portrayed by a unique one in full DNF [5, p. 73].

Since 16 unique functions are possible, the same number of qubits would be required to track these essential statements during inference with two symbols. For a set with 10 propositional variables, however, $2^{2^{10}} = 1024$ qubits would be needed. We would like to have a more condensed account of truthfulness between deductions, without sacrificing propositional expressiveness.

Notice that sentences in $\mathcal{F} \cup \mathcal{R}$ may contain disjunctive connectives. Suppose this were not the case for facts in $\mathcal{F}$, as well as consequents of rules in $\mathcal{R}$. These would then be conjunctions of literals. $\mathcal{F}$ would be a single, large, disjunction-free conjunction, the *fact conjunct*. In that case, the deductive process would be simple: for each rule $\alpha \to \beta \in \mathcal{R}$ check that α's propositional symbols are appropriately asserted in the fact conjunct so as to make α true. If so, and upon inference with *Modus Ponens*, every factor of the conjunction β can be independently asserted.[36] Thus, we need only keep a record of each literal's semantic value. For n propositional symbols, this would involve linear memory requirements, namely a $2n$ qubit register would suffice.

We thus define our KB state in the following manner: if $\{a_1, a_2, \ldots, a_n\}$ is the set of propositional symbols appearing in $\mathcal{F} \cup \mathcal{R}$, then qubit register $|\boldsymbol{\kappa}\rangle = |a_1 a_2 \ldots a_n \bar{a}_1 \bar{a}_2 \ldots \bar{a}_n\rangle$ will portray the state of our fact database at each step. Initially, each a_i in the register will have its value set to 1 for truth, provided a_i is affirmed in the fact conjunct $\mathcal{F}$. Otherwise, its value will be 0. The same shall apply to the $\bar{a}_i$ qubits, each reflecting whether or not the negative literal $\neg a_i$ is asserted in $\mathcal{F}$. This entails an initial stage of

[35] that is, *formulae* which are disjunctions of conjunctive clauses, where the latter contain exactly one of the two literals corresponding to each propositional variable [12, p. 103]. Originally proposed by Boole in [5].

[36] if $\alpha \wedge \beta$, then we may assert α and β individually, with α, β wff. These are theorems in Propositional Calculus (results $*3.26$ and $*3.27$ of Whitehead and Russell's *Principia Mathematica*, [62, p. 110]), although we strike such minute details from our proofs.

$O(2^{\mu_\omega} \cdot |\mathcal{F}|)$ time-complexity, where $|\mathcal{F}|$ is the fact set's cardinality, $\omega \in \mathcal{F}$ is its most complex formula[37] and μ_ω the depth within the phrase of the conjunction with the highest precedence.

At this point, the reader may make the following objection: why should we keep both positive and negative literal values? Will not the information captured in the former be mirrored in the latter? We justify such a choice with our intention of maintaining the inference system *monotonic*,[38] and as simple as possible, without sacrificing recognition of propositional *formulae*. If we only tracked variables a_i, a *non-monotonic* logic would result.

For instance, consider the following KB $S = \mathcal{F} \cup \mathcal{R}$, where a is a propositional symbol, and each element of the rule set lies on its own left braced row:[39]

$$\begin{aligned} \mathcal{F} &= \{a\} \\ \mathcal{R} &= \begin{cases} a \rightarrow \neg a \\ \neg a \rightarrow a \end{cases} \end{aligned}$$

Notice that $S \vdash \neg a$, and if such an inference were to be carried out, qubit a of our register would be flipped. We could then assert $\{\neg a\} \vdash a$, which would again reconsider our record of a. The case in point is a non-monotonic system.

While such mechanisms can be appropriate for situations in which complete knowledge is impossible, or beliefs may change over time, in general they are also more complex and computationally demanding, both in time and space terms [21, p. 220]. For this reason, we opt to model the monotonic variety, maintaining the record described by $|\boldsymbol{\kappa}\rangle$.

4.4 Dividing the Knowledge-base

We now turn our attention to the obvious: the hypothesis laid out in the previous section, that every fact in $\mathcal{F}$ and the consequents of rules in $\mathcal{R}$ are all conjuncts of literals, does not hold for every KB expressed in Propositional Logic.

However, any knowledge-base S for which the above conjecture is unsupported can be divided into several modules in which it *is* satisfied. This is what is carried out at the beginning of Algorithm 2, when the procedure DIVIDE-KB is executed. Its operation is depicted in Algorithm 3.

DIVIDE-KB relies on two other procedures, DIVIDE-FACTS and DIVIDE-RULES, which we will describe next. For now, suffice it to say that these respectively determine from $\mathcal{F}$ and $\mathcal{R}$ the collections of fact sets $\{\mathcal{F}_i\}$ and rule sets $\{\mathcal{R}_i\}$ which no longer possess any disjunction in members of $\mathcal{F}_i$ or in consequents of rules in $\mathcal{R}_i$. Instead, each of the aforementioned expressions has been replaced by one of its unique cases. For instance, consider the single fact set $\{A \wedge (B \vee C)\}$, which would be sub-divided into the three sets $\{A \wedge B \wedge C\}$, $\{A \wedge \neg B \wedge C\}$ and $\{A \wedge B \wedge \neg C\}$, each stored in the fact set list.[40] Similarly, for the

[37] this is measured according to how deep in the formula the conjunction with the highest precedence is. *E.g.*, for a phrase $\omega = a \wedge ((b \wedge (c \wedge d)) \wedge e)$, $\mu_\omega = 4$ and we may establish an $O(2^{\mu\omega})$ upper bound on the number of literals to set.

[38] in a monotonic logical system, an inference will never remove a literal from the set of true statements. Non-monotonic mechanisms, however, allow this to occur [36, p. 335].

[39] note that the example KBs we present are for demonstrative purposes. In reality, Grover's algorithm would require a larger rule set to obtain the intended results.

[40] while we use the terminology of lists, this is only to avoid speaking in terms of sets of fact sets or sets of rule sets. Obviously, any appropriate data structure can be used.

Algorithm 3 Divides a knowledge-base $\langle \mathcal{F}, \mathcal{R} \rangle$, possibly possessing disjunctions in facts of $\mathcal{F}$ or in consequents of rules in $\mathcal{R}$, into a set of KBs whose elements reflect the individual cases of each disjunction.

```
procedure DIVIDE-KB(⟨F, R⟩)
    FACTSETLIST ← DIVIDE-FACTS(F)
    RULESETLIST ← DIVIDE-RULES(R)
    SYSLIST ← ∅

    for each F′ ∈ FACTSETLIST do
        for each R′ ∈ RULESETLIST do
            SYSLIST ← SYSLIST ∪ {⟨F′, R′⟩}
        end for each
    end for each

    return SYSLIST
end procedure
```

single element rule set $\{A \to B \vee C\}$, we would obtain the three rule sets $\{A \to B \wedge C\}$, $\{A \to \neg B \wedge C\}$ and $\{A \to B \wedge \neg C\}$.

The two subsequent **for** loops in the procedure perform a Cartesian product of the fact and rule set lists, from which we obtain each individual combination of fact/rule sets $\sigma_k = \langle \mathcal{F}_i, \mathcal{R}_j \rangle$ (for appropriate i, j, k) consisting of elements which satisfy the respective disjunctives present in $S = \langle \mathcal{F}, \mathcal{R} \rangle$.

Later on, we will demonstrate that DIVIDE-FACTS and DIVIDE-RULES are $O(|\mathcal{B}| + k^{\xi})$ time processes, where $\mathcal{B} = \mathcal{F}$ or $\mathcal{B} = \mathcal{R}$, ξ is (respectively) the number of *facts in $\mathcal{F}$/consequents of $\mathcal{R}$* possessing disjunctions, k the maximum number of unique operands of the full DNF form of any of the above expressions, restricted to the symbols these contain,[41] and $|\mathcal{B}|$ represents $\mathcal{B}$'s cardinality. Naturally, these values are different between $\mathcal{F}$ and $\mathcal{R}$, so we take $\mathcal{B}, k, \xi$ so as to make expression $|\mathcal{B}| + k^{\xi}$ the largest, as a worse case analysis. Furthermore, the two procedures each return a set of size $O(k^{\xi})$, where we make the same assumptions on k, ξ as above.

Thus, DIVIDE-KB returns a list of $O(k^{2\xi})$ independent knowledge bases, where disjunctions can only occur in the antecedents of elements of their rule sets. Space-performance is also $O(k^{2\xi})$, whereas with respect to time, the entire method runs in $O(2(|\mathcal{B}| + k^{\xi}) + k^{2\xi}) = O(|\mathcal{B}| + k^{2\xi})$ operations.

Now that we have these independent knowledge-bases, how can we prove an expression η for the original fact set S? We know how we would proceed in each separate sub-KB, but our desire is a proof in S. Although we have already stated our strategy, at the risk of verbosity, the answer to this is best exemplified.

Suppose S consists of the following system:

$$\mathcal{F} = \{a \vee b\}$$

$$\mathcal{R} = \begin{cases} a \to c \\ b \to c \end{cases}$$

where we would like to prove that c is true. The reader will immediately recognize this as fact, of course, because for every hypothetical situation, the expression is obtained. So if it is true that a, the first rule shall be used for the proof; and in case b is true, the second will establish c.

[41] for instance, if our KB contains phrases composed merely of propositional symbols A, B, C, then expressions $A \vee A$ and A both have size 1, while sentence $A \wedge (B \vee C)$ has size 3.

DIVIDE-FACTS and DIVIDE-RULES themselves are presented in Algorithms 4 and 5. We merely analyse the first of these, since they are similar in nature.

Algorithm 4 Divides a set of facts $\mathcal{F}$, which may comprise sentences with disjunctive connectives, and returns a collection of fact sets reflecting the individual cases inherent to those disjunctions. Each of these sets contains assertions crafted merely from literals or $\wedge$.

```
procedure DIVIDE-FACTS(F)
    FACTSETLIST ← {F}
    AUXLIST ← ∅
    CASES ← ∅

    for each a ∈ F do
        if HAS-OR?(a) = True then
            CASES ← EXTRACT-CASES(a)
            AUXLIST ← FACTSETLIST

            for each F' ∈ FACTSETLIST do
                AUXLIST ← AUXLIST − {F'}
                for each c ∈ CASES do
                    AUXLIST ← AUXLIST ∪ {(F' − {a}) ∪ {c}}
                end for each
            end for each

            FACTSETLIST ← AUXLIST
        end if
    end for each

    return FACTSETLIST
end procedure
```

Naturally, the method begins by examining each of the sentences in $\mathcal{F}$ for the presence of disjunctions. This is embodied by the HAS-OR? sub-routine. It is a simple test, so we will not describe it in detail. We do assume that the index over statements in S, characterized in section 5.2, allows it to run in $O(1)$ time.

No actions are necessary when fact a involves no disjunctions. Otherwise, the collection of individual cases of a is retrieved by calling EXTRACT-CASES. Since these are stored in our index, EXTRACT-CASES can also be designed so that it runs in constant time.

DIVIDE-FACTS will obtain at most $O(k^{\xi})$ sets, where k, ξ possess the meanings already mentioned. This is also its space-complexity. In terms of time-performance, it runs in $O(|\mathcal{F}| + k^{\xi})$, with $|\mathcal{F}|$ the size of the fact set.

Analysis of DIVIDE-RULES would yield the same conclusions, with suitable substitutions for $k, \xi, |\mathcal{F}|$.

4.5 Deductive Process

Having divided our original S, we now describe our approach for proof using quantum techniques. We ask the reader to refer back to Algorithm 2 whenever necessary.

More formally, a proof of query formula η from $S = \langle \mathcal{F}, \mathcal{R} \rangle$ is the sequence of wff s.t.:

Algorithm 5 Divides a set of rules $\mathcal{R}$, which may contain rules with disjunctions in their consequents, and returns a collection of rule sets reflecting each of those disjunction's individual cases. The resulting sets in that collection possess only rules whose consequents are formulated with literals or $\wedge$.

```
procedure DIVIDE-RULES(R)
    RULESETLIST ← {R}
    AUXLIST ← ∅
    CASES ← ∅

    for each α → β ∈ R do
        if HAS-OR?(β) = True then
            CASES ← EXTRACT-CASES(β)
            AUXLIST ← RULESETLIST

            for each R' ∈ RULESETLIST do
                AUXLIST ← AUXLIST − {R'}
                for each c ∈ CASES do
                    AUXLIST ← AUXLIST ∪ {(R' − {α → β}) ∪ {α → c}}
                end for each
            end for each

            RULESETLIST ← AUXLIST
        end if
    end for each

    return RULESETLIST
end procedure
```

$S = \mathcal{F} \cup \mathcal{R} = \bigvee_{i,j} \mathcal{F}_i \cup \mathcal{R}_j$	Premiss
$\sigma_1 = \mathcal{F}_1 \cup \mathcal{R}_1$	Hypothesis
$\alpha_{1,1}$	*Modus Ponens*
$\vdots$	$\vdots$
$\alpha_{1,l_1} = \eta$	*Modus Ponens*
$\vdots$	$\vdots$
$\sigma_{u \cdot v} = \mathcal{F}_u \cup \mathcal{R}_v$	Hypothesis
$\alpha_{u \cdot v,1}$	*Modus Ponens*
$\vdots$	$\vdots$
$\alpha_{u \cdot v, l_{u \cdot v}} = \eta$	*Modus Ponens*
η	$E_\vee \ (S = \bigvee_i \sigma_i)$

where we consider $\mathcal{F} \cup \mathcal{R}$ as the conjunction of all expressions in that set. We also admit that u, v are bounded by $O(k^\xi)$ sets of $\mathcal{F}_i, \mathcal{R}_i$. In addition, an ordering of the $\mathcal{F}_i \cup \mathcal{R}_j$ pairs is assumed, so that these are indexable by each σ's subscript. Thus, we can establish that there are $u \cdot v = O(k^{2\xi})$ independent sets σ_i.

The $\alpha_{i,j}$ are the wff obtained by inference from fact set σ_i using MP, and which lead up to the proof of η in that knowledge set. Their index j denotes the j^{th} wff in that proof, and l_i its length in σ_i. These $\alpha_{i,j}$ are deduced exclusively using MP, a fact which we will expound upon in concluding this section.

At the end of the proof structure, and upon obtaining η from each σ, we may effect disjunction elimination over the sub-KBs to derive η in S.

Returning to our analysis of Algorithm 2, having our list of σ_i we must now show η will be deduced from each. However, in proceeding, we require the notion of a *contradiction* within a set of facts σ: let a be a propositional symbol, should both a and $\neg a$ follow by deduction from σ when applying any combination of its rules and MP in sequence, then we may classify σ as contradictory, *i.e.*, there are no models satisfying σ's collection of facts and σ is inconsistent. In particular, no model of σ will satisfy S, and in that regard, a proof of η from S ignores that σ in the proof structure above. This is what is first ascertained by the CONTRADICTORY? procedure, presented in Algorithm 6.

Algorithm 6 Detects whether a given knowledge-base $\sigma = \mathcal{F}_i \cup \mathcal{R}_j$ is contradictory.

procedure CONTRADICTORY?($\sigma = \mathcal{F}_i \cup \mathcal{R}_j$)
- Prepare the initial KB state vector $\boldsymbol{s}_0$ according to the facts in $\mathcal{F}_i$
- Let g be the goal test function and $\boldsymbol{x}$ a KB state, such that

$$g(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x} \text{ has its contradiction bit set} \\ 0, & \text{otherwise} \end{cases}$$

return FIRST(LQIDS($\boldsymbol{s}_0$, $\boldsymbol{f}$, g, $|\mathcal{R}_j|$))
end procedure

The CONTRADICTORY? function will accept a set of assertions $\sigma = \mathcal{F}_i \cup \mathcal{R}_j$, and make use of the limited form of the Quantum Iterative Deepening Search methodology (Algorithm 1) to discern if these derive a contradiction. This requires that we redefine our notion of a knowledge-base state to track such a situation.

Henceforth, our KB state will be $|\boldsymbol{s}\rangle = |\boldsymbol{\kappa}\rangle\,|c\rangle = |a_1 a_2 \ldots a_n \bar{a}_1 \bar{a}_2 \ldots \bar{a}_n\rangle\,|c\rangle$, where the $|c\rangle$ qubit will record whether or not a state has become contradictory. Although such information is already contained within $|\boldsymbol{\kappa}\rangle$, we would like to test and record such an occurrence in $O(1)$ time after its starting value is set. Having $|c\rangle$ included in the state description provides a simple means of achieving this.

As we discussed in chapter 2, when we have defined a domain (state space, transition function, goal test), an iterative tree search procedure can be performed over it using QIDS. In this case, our domain consists of a state space given by every possible $|\boldsymbol{s}\rangle$, where we may trigger the rules in $\mathcal{R}_j$ to transition between these. The depth of the search can be limited to the number of rules $\zeta = b = |\mathcal{R}_j|$ (*i.e.*, the branching factor), since repetition of a rule will never infer additional facts in a monotonic system of inference. The solution/goal test g will control the termination behaviour of CONTRADICTORY?'s call to LQIDS. For any KB state $\boldsymbol{x}$, it is able to test the $|c\rangle$ qubit and assume a value 0 or 1 according to its definition in Algorithm 6.

In this case, LQIDS takes $\boldsymbol{s}_0$ for its initial state (ascertained from $\mathcal{F}_i$, according to our description in section 4.3), a transition function $\boldsymbol{f}$ (defined below) and the solution test g. Its depth limit is determined from $\mathcal{R}_j$, as $|\mathcal{R}_j|$. We describe the algorithm's execution for each step, from $d = 0$ to $|\mathcal{R}_j|$.

To begin with, the quantum circuit for the current depth is realized. The transition operator T will behave according to the rules in $\mathcal{R}_j$. This requires we formalize its classical counterpart $\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{m})$, which will

output the state resulting from transition $\boldsymbol{m}$ when the KB state is a vector $\boldsymbol{s} = [a_1 a_2 \dots a_n \bar{a}_1 \bar{a}_2 \dots \bar{a}_n c]^{\mathrm{T}}$:

$$\boldsymbol{f}(\boldsymbol{s}, \boldsymbol{m}) = \begin{cases} \boldsymbol{s}, & \text{if } \boldsymbol{m}\text{'s antecedent is not satisfied in } \boldsymbol{s} \\ \boldsymbol{s}', & \text{if } \boldsymbol{m}\text{'s antecedent is satisfied in } \boldsymbol{s} \text{ but} \\ & \text{a contradiction ensues}^{42} \text{ from it} \\ \boldsymbol{s}'', & \text{otherwise} \end{cases}$$

where $\boldsymbol{s}''$ has bits corresponding to literals asserted in $\boldsymbol{m}$'s consequent conjunct set to 1, while every other bit is inherited from $\boldsymbol{s}$. $\boldsymbol{s}'$ bears the same properties as $\boldsymbol{s}''$, except that its contradiction bit is set. Note that we refer to rules in $\mathcal{R}_j$ and their encodings $\boldsymbol{m}$ interchangeably, since these uniquely determine each other.

In this way, using equation (2.5) and $\boldsymbol{f}, g$, the gates T and G are extracted and the circuit constructed.

The reader will notice $\boldsymbol{f}$ is not defined within Algorithm 6. We consider the function a constant, since it will be reused in a further procedure, preferring to characterize it here with the description above.

$\boldsymbol{f}$'s evaluation can be made $O(1)$, provided we pre-calculate and index its output for every situation. Assuming $s = 2n + 1$ is the number of bits required for state data, and $m = \lceil \log_2 b \rceil$ the amount of bits needed to encode transition decisions, this entails an $O(2^{4n + \log_2 b})$ calculation.[43]

Evaluation of g can also be carried out in $O(1)$ time by adopting a similar indexing strategy, which requires a one-off $O(2^{2n})$ pre-computation. Multiple queries can then be performed without recalculation of these gates.

We now delineate the circuit's initial input values. The state register $|\boldsymbol{s}\rangle = |\boldsymbol{\kappa}\rangle\,|c\rangle$ is defined by $\boldsymbol{s}_0$ (and thus, by $\mathcal{F}_i$), as per our previous instructions. $|c\rangle$'s starting value can be set in $O(n)$ time, by checking whether $\forall_i\, a_i$ and $\bar{a}_i$ are simultaneously asserted in $\boldsymbol{\kappa}$. Every possible d-length rule application sequence in $\mathcal{R}_j$ is constructed by placing the path qubits $|\boldsymbol{w}\rangle$ in superposition using equation (2.2). Finally, each ancillary qubit is set according to the description laid out in chapter 2.

We subsequently run Grover's procedure over the circuit's superposition. Should there be a solution at depth d, then Grover's algorithm will deterministically find it, *i.e.*, measuring the path register $|\boldsymbol{w}\rangle$ and verifying the sequence of rules from $\mathcal{R}_j$ as leading to a solution will uncover whether the search succeeded. Should this be the case, LQIDS returns $\langle \textit{True}, \boldsymbol{w} \rangle$. Otherwise, it repeats the above for the next depth, until a solution is found or depth $|\mathcal{R}_j|$ is surpassed. If g is unsatisfied past this point, *False* is reported.

CONTRADICTORY? will use the first element of the LQIDS algorithm's result to determine if indeed some contradiction has been found in the facts of σ. This would preclude its inclusion in a proof of η, since as we have stated, no model of σ is a model of S. Note however, that should every σ be contradictory, then of course, there are no models for S. In such a context, anything and everything is provable,[44] so we must detect this situation, and classify the proof attempt as *Impossible* within our

[42] *i.e.*, at least one of the a_i and $\bar{a}_i$ in $\boldsymbol{s}'$ are both set.

[43] $\lceil \log_2 b \rceil \leq \log_2 b + 1$.

[44] originally attributed to the medieval philosopher and theologian John Duns Scotus in his *Opera Omnia* tomes [47], this principle is now conjectured to have been presented by someone else. Consult Lagerlund's work [33, p. 165] for historical notes. Because the author is unknown, the philosophical community refers to both the principle and the author as Pseudo-Scotus, although it is

knowledge-base.

On the other hand, if at least one of the σ is not contradictory, then perhaps a proof for η can be found. Algorithm 2 attempts to extract one for each of these, via PROVE-QUERY, defined in Algorithm 7.

Algorithm 7 Finds a sequence of inferences from a knowledge-base $\sigma = \mathcal{F}_i \cup \mathcal{R}_j$, which when applied in succession will satisfy the query expression η.

procedure PROVE-QUERY($\sigma = \mathcal{F}_i \cup \mathcal{R}_j$,$\eta$)
- Prepare the initial KB state vector $\boldsymbol{s}_0$ according to the facts in $\mathcal{F}_i$
- Let g be the goal test function and $\boldsymbol{x}$ a KB state, such that

$$g(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x}\text{'s literal values satisfy expression } \eta \\ 0, & \text{otherwise} \end{cases}$$

return LQIDS($\boldsymbol{s}_0$, $\boldsymbol{f}$, g, $|\mathcal{R}_j|$)
end procedure

The procedure's structure is almost identical to that of CONTRADICTORY?'s. The differences lie in the goal test used to invoke LQIDS and built into the circuit, as well as in how it returns its result. While in CONTRADICTORY? we tested the contradiction qubit, in this case we look for satisfiability of η. By proceeding in the manner prescribed by LQIDS, we will obtain the shortest proof of η under σ, should one exist. If it does not, PROVE-QUERY shall report η to be unprovable in σ, which is enough to signal to PROVE that it was unable to derive η from S.

Otherwise, Algorithm 7 returns the tuple $\langle \textit{True}, \boldsymbol{w} \rangle$ from LQIDS. Algorithm 2 verifies that the deduction is successful and adds the pair $\langle \sigma, \boldsymbol{w} \rangle$ to the set PROOF. Should η be inferred for every non-contradictory σ, then a tuple $\langle True, \text{PROOF} \rangle$ is returned from PROVE, where set PROOF reflects the proof by cases strategy delineated at the beginning of this chapter.

Notice how LQIDS's use of our $\boldsymbol{f}$ effectively models MP. This is why for each σ_i, the $\alpha_{i,j}$ of the aforementioned strategy are all deduced merely with this inference rule.

4.6 Analysis of the Method

Since LQIDS is central to our approach, we remind the reader that its time- and space-performance may be consulted in appendix A.

The entire PROVE method has $O(|\mathcal{B}| + k^{2\xi} + k^{2\xi} \cdot 2 \cdot (2^{b\lceil \log_2 b \rceil/2} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b))) = O(|\mathcal{B}| + k^{2\xi} \cdot (2^{b\lceil \log_2 b \rceil/2} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b)))$ time-performance, where $\mathcal{B}$, k, ξ, b, n and μ_ω retain their previously defined meanings. Term $2 \cdot (2^{b\lceil \log_2 b \rceil/2} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b))$ arises from running LQIDS twice for each σ: once in CONTRADICTORY? and again in PROVE-QUERY. Within these procedures, $2^{\mu_\omega} \cdot |\mathcal{F}|$ operations are needed to initialize the literal qubits of the starting state of the KB, n are required to set the first contradiction qubit and our verification process v is $O(b)$. Each of these steps is executed up to a maximum depth of b. From footnote [43] and provided $b \geq 2$, we can also assert the bound to be $O(|\mathcal{B}| + k^{2\xi} \cdot (2^{(b \log_2 b)/2} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b)))$.

The equivalent classical technique performs in $O(|\mathcal{B}| + k^{2\xi} \cdot (2^{b \log_2 b} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b)))$ time. Since $|\mathcal{B}| + k^{2\xi} \cdot (2^{(b \log_2 b)/2} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b)) \leq |\mathcal{B}| + k^{2\xi} \cdot (2^{b \log_2 b} + b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b)) \leftrightarrow k^{2\xi} \cdot 2^{(b \log_2 b)/2} \leq$

also often invoked as the principle of explosion.

$k^{2\xi} \cdot 2^{b \log_2 b} \leftrightarrow 2^{2\xi \cdot \log_2 k + (b \log_2 b)/2} \leq 2^{2\xi \cdot \log_2 k + b \log_2 b}$, we know that there is an upper bound improvement.

As b grows quite large, $|\mathcal{B}| = |\mathcal{R}| = b$. Terms $|\mathcal{B}|$ and $b \cdot (n + 2^{\mu_\omega} \cdot |\mathcal{F}| + b)$ may be disregarded, as they are executed in both procedures. In that case, we let $N \simeq k^{2\xi} \cdot 2^{b \log_2 b}$ represent the number of deduction paths which are classically examined. When there are $\xi = b$ rules of $\mathcal{R}$ with consequents possessing a disjunction and $k \ll b$, if we compare the ratios of the two exponents at the end of the last paragraph, we conclude that the speed-up approaches a quadratic order:

$$\lim_{\substack{b\to\infty \\ k\ll b \\ \xi = b}} \frac{2\xi \cdot \log_2 k + {}^{(b \log_2 b)}/_2}{2\xi \cdot \log_2 k + b \log_2 b} = \lim_{\substack{b\to\infty \\ k\ll b \\ \xi = b}} \frac{1 + \frac{b \log_2 b}{2 \cdot 2\xi \cdot \log_2 k}}{1 + \frac{b \log_2 b}{2\xi \cdot \log_2 k}}$$

$$= \lim_{\substack{b\to\infty \\ k\ll b \\ \xi = b}} \frac{1}{2}\left[\frac{2 + \frac{b \log_2 b}{2\xi \cdot \log_2 k}}{1 + \frac{b \log_2 b}{2\xi \cdot \log_2 k}}\right] = \lim_{\substack{b\to\infty \\ k\ll b \\ \xi = b}} \frac{1}{2}\left[\frac{1}{1 + \frac{b \log_2 b}{2\xi \cdot \log_2 k}} + 1\right]$$

$$= \lim_{\substack{b\to\infty \\ k\ll b}} \frac{1}{2}\left[\frac{1}{1 + \frac{b \log_2 b}{2b \cdot \log_2 k}} + 1\right] = \frac{1}{2}(0 + 1) = \frac{1}{2}$$

Thus, for knowledge-bases with a large enough rule set, the algorithm runs at $O(\sqrt{N})$ time, a quadratic speed-up in comparison with classical $O(N)$ exhaustive search approaches.

In space-complexity terms, the technique requires that we store $O(k^{2\xi})$ set elements. This bound includes pairs of $\langle \mathcal{F}_i, \mathcal{R}_j \rangle$, as well as $\langle \sigma, \text{PROOF} \rangle$ tuples. Additionally, since $s = 2n+1$ qubits are required for state characterization, the quantum circuits need up to $O(n(b+1) + b \log_2 b)$ qubits, which can be reused for inference in each of the σ. These two bounds constitute the method's space-performance.

The methodology is semantically incomplete for wff of Propositional Calculus built using its logical connectives and positive literals present in $S = \mathcal{F} \cup \mathcal{R}$.[45] On the other hand it does produce sound deductions. To see why this is so, note that both MP and $E_\vee$ are sound derivation mechanisms.

PROVE is a complete search algorithm. For any input query η and KB S, it returns a solution, should one exist according to g. However, although the procedure LQIDS provides an optimal deduction of η (if it exists) for each σ, PROVE itself cannot ensure optimal proofs. Indeed, one merely need consider the following KB for query $\eta = b$:

$$\mathcal{F} = \{a\}$$

$$\mathcal{R} = \begin{cases} a \to b \\ c \to d \vee e \end{cases}$$

The technique would provide a proof by cases due to the 2^{nd} rule having a disjunction, whereas the optimal proof would assert a, the 1^{st} rule and b by MP.

[45] for instance, the KB $\{a, b \to c\} \models c \vee \neg c$, yet $c \vee \neg c$ is never derived. This occurs when, along every deduction path of some sub-KB σ, the query always contains some propositional symbol essential to its satisfaction in an undefined state, *i.e.*, neither affirmed nor contested in $|s\rangle$.

Chapter 5

Conclusions

Here we convey our major Results for each of the proposed methodologies in the corresponding section. A brief Discussion of the impact and future prospects of quantum computation within the field of AI is provided next. We conclude the chapter by emphasizing a set of Open Problems suggested by our work on quantum propositional inference.

5.1 Results

Our introduction of hybrid quantum/classical mechanisms for solving BW instances and performing knowledge-based propositional inference shows how quantum computation models may be utilised for solving non-trivial symbolic AI problems.

In particular, our BW Solver can find a goal state from an N-sized state-space in $O(\sqrt{N})$ steps, a quadratic improvement over blind classical methods, which require $O(N)$ operations. For n-block problems, the procedure has a complexity of $O(\sqrt{2^{lm}})$ steps, with $O(l(m+s))$ space-complexity, where l is the shallowest solution depth, and $m = \lceil \log_2 n(n-1) \rceil$, $s = \lceil \log_2 n^n \rceil$ the number of qubits needed for action and state encoding (respectively).

Additionally, we have suggested a modification that is able to reduce the solver's qubit requirements, so that it may be applied in more conservative memory environments. Because most of the necessary space emanates from auxiliary computations, omitting these was our primary concern. Consequently, instead of storing the results of d movement operations, the d^{th} iteration of the procedure prescribes an operation T_d, which calculates the outcome of the d movements at once, without keeping intermediate states within memory.

While the technique's time-complexity is identical to that of the conventional solver's, qubit requirements are now $O(s+lm)$, where the depth-dependent amount of auxiliary state storage has been eliminated. However, the process entails pre-calculating each of the T_d operators, *i.e.*, a one-off computation of at least $O(2^{2s+lm})$ steps, corresponding to the largest such transform. Since this quickly becomes intractable, application would be limited to less complicated instances, although the strategy is similarly amenable to more general tasks.

Within this domain, a novel heuristic perspective has also been proposed for use in quantum computation settings, as a contrast to the informed, comparative-appraisal heuristics often seen in traditional classical approaches. Because of the no-communication constraints intrinsic to quantum mechanical theory, the latter provide limited benefits in this context. Alternatively, we show how a divide-and-conquer heuristic can produce better results than our conventional QIDS solver for the same problem, provided it is decomposable.

If such is the case, then a classical method determines sets of blocks which are interdependent based on the initial and final configurations. These are separately solved using our n-EBW approach, resulting in $O(\sqrt{2^{l^{(j)}m^{(j)}}})$[46] time-complexity, where j is the most complex of the sub-problems found, and $l^{(j)}$, $m^{(j)}$ have the same meaning in j as l, m had for the non-decomposed instance. Since the general problem is divisible, we have that $l^{(j)} \leq l$ and $m^{(j)} < m$. In particular, the speed-up is significant for situations where $l^{(j)} \ll l$ and $m^{(j)} \ll m$, *i.e.*, when the largest set of interdependent blocks is much smaller than considering all n blocks to be coupled. In addition, because $s^{(j)} < s$, space-complexity is also lowered.

On the other hand, when the problem instance is not decomposable, we have that $l^{(j)} = l$, $m^{(j)} = m$ and $s^{(j)} = s$, so the approach has complexity identical to that of solving without the heuristic.

Regarding the problem of propositional inference, our technique can provide a quadratic time-performance benefit with respect to its classical analogue, given a large KB of propositions. For an N sized state space (N different deduction paths), the speed-up obtained is equivalent to executing an $O(\sqrt{N})$ search, instead of an exhaustive $O(N)$ procedure, halving necessary verification depth within the domain.

The approach is a first attempt at employing the QIDS framework to address the problem of proving Propositional Logic *formulae*. Its characteristics make it a middle ground alternative to classical model checkers and theorem provers, lying within the complexity interval defined by the co-NP-complete propositional entailment problem and the linear time bounds encountered in language restricted, resolution-based procedures.

5.2 Discussion

Although efforts to implement qubit processors have been steadily producing results, their short to medium term costs are likely to remain high. This begs the question, how soon could small-scale quantum computation units outperform existing parallel computing solutions such as *Amazon Elastic MapReduce* or *Google Cloud DataProc*. While this is still an open question, several AI problems are inherently autonomous in nature. Space exploration probes or rovers [24], for instance, do not have access to such distributed processing resources, and must often plan entirely on their own.

Nevertheless, access to quantum hardware is already available online, although at a small scale. In 2013, researchers from the University of Bristol, UK, introduced the *Quantum in the Cloud* internet

[46] j is the most complex of the decomposed problems, $l^{(j)}$ its shallowest solution depth, $m^{(j)} = \lceil \log_2 n^{(j)}(n^{(j)} - 1) \rceil$ the number of qubits coding j's actions, and $n^{(j)}$ the j^{th} sub-problem's number of blocks.

service, through which anyone could program and run experiments on a 2-qubit optical processing unit. Shortly thereafter, IBM launched a similar enterprise. Its *Quantum Experience* program allows us to define and execute instructions on a 5-qubit processor. Such initiatives have already led to new research [1, 13], and both institutions have stated they mean to expand the number of qubits involved. Hopefully, this escalation will continue, and invigorate investigative efforts within our particular field as well.

As further work is produced within this frontier, we will most likely bear witness to cross-fertilization between methods in quantum computation and AI [65], and the divide-and-conquer heuristic we have proposed for these settings is merely one example of such an occurrence.

5.3 Open Problems

Although sound inference is achieved by our theorem proving methodology, the mechanism is still semantically incomplete. However, for large KBs, we posit that an additional restricted model checking procedure, branching on the possible interpretations over propositional symbols in the query that are never asserted or negated along a deduction path, may deliver this property with time-performance bounds identical to those we have presented.

Furthermore, since traditional classical heuristics cannot be used in quantum computation, it would be interesting to explore the approach's behaviour when divide-and-conquer strategies are applied to the problem within these settings.

We have also yet to explore the possibility (and ramifications) of adding proven expressions to the KB during deduction of a more complex phrase, in a manner akin to that of means-end methods.

Bibliography

[1] D. Alsina and J. I. Latorre. Experimental test of mermin inequalities on a five-qubit quantum computer. *Physical Review A*, 94:012314, July 2016. doi: 10.1103/PhysRevA.94.012314. 45

[2] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17 (6):525–532, Nov. 1973. ISSN 0018-8646. doi: 10.1147/rd.176.0525. 4

[3] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM Journal on Computing*, 26(5):1510–1523, Oct. 1997. doi: 10.1137/S0097539796300933. 10

[4] G. Boole. *The Mathematical Analysis of Logic*. Philosophical Library, New York, NY, USA, 1847. 17

[5] G. Boole. *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*. George Boole's collected logical works. Walton and Maberly, London, UK, 1854. doi: 10.5962/bhl.title.29413. 34

[6] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. *arXiv eprint: quant-ph/0005055*, May 2000. doi: quant-ph/0005055. 10

[7] D. Chapman. Planning for conjunctive goals. Technical Report 802, AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA, May 1985. 14

[8] S. V. Chenoweth. Mathematical foundations for blocks world including a proof of np-completeness. Master's thesis, Wright State University, Dayton, OH, USA, 1986. 15

[9] S. V. Chenoweth. On the np-hardness of blocks world. In T. L. Dean and K. McKeown, editors, *Proceedings of the Ninth National Conference on Artificial Intelligence, Volume 2*, AAAI '91, pages 623–627, Menlo Park, CA, USA, July 1991. AAAI Press / MIT Press. ISBN 0-262-51059-6. 15

[10] S. A. Cook. The complexity of theorem-proving procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, May 1971. ACM. doi: 10.1145/800157.805047. 18

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, USA, 1st edition, 1990. ISBN 0-262-03141-8. 28

[12] B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge mathematical text books. Cambridge University Press, Cambridge, UK, 2nd edition, June 2002. ISBN 9780511809088. doi: 10.1017/CBO9780511809088. 34

[13] S. J. Devitt. Performing quantum computing experiments in the cloud. *Physical Review A*, 94: 032329, Sept. 2016. doi: 10.1103/PhysRevA.94.032329. 45

[14] P. A. M. Dirac. A new notation for quantum mechanics. *Mathematical Proceedings of the Cambridge Philosophical Society*, 35(3):416–418, July 1939. ISSN 1469-8064. doi: 10.1017/S0305004100021162. 7

[15] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267–284, Oct. 1984. doi: 10.1016/0743-1066(84)90014-1. 19

[16] R. Englert. Planning to optimize the umts call setup for the execution of mobile applications. *Applied Artificial Intelligence*, 19(2):99–117, Sept. 2005. 17

[17] S. E. Fahlman. A planning system for robot construction tasks. *Artificial Intelligence*, 5(1):1–49, Jan. 1974. ISSN 0004-3702. doi: 10.1016/0004-3702(74)90008-3. 15

[18] E. Farhi and S. Gutmann. Quantum computation and decision trees. *Physical Review A*, 58(2): 915–928, Aug. 1998. ISSN 1050-2947. doi: 10.1103/physreva.58.915. 6

[19] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. *arXiv eprint: quant-ph/0001106*, Jan. 2000. doi: quant-ph/0001106. 6

[20] R. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21 (6-7):467–488, June 1982. ISSN 0020-7748. doi: 10.1007/BF02650179. 3

[21] D. Gabbay and K. Schlechta. *Conditionals and Modularity in General Logics*. Cognitive Technologies. Springer, Berlin, Deutschland, 2011. ISBN 9783642190681. doi: 10.1007/978-3-642-19068-1. 35

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447. 15

[23] M. R. Garey, R. L. Graham, and D. S. Johnson. Some np-complete geometric problems. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, STOC '76, pages 10–22, New York, NY, USA, 1976. ACM. doi: 10.1145/800113.803626. 15

[24] R. Godwin and S. Whitfield. *Deep Space: The NASA Mission Reports*. Collector's Guide Publishing, Burlington, ON, Canada, 2005. ISBN 9781894959155. 44

[25] L. K. Grover. A fast quantum mechanical algorithm for database search. In G. L. Miller, editor, *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM. doi: 10.1145/237814.237866. v, vii, 1, 6, 9

[26] N. Gupta and D. S. Nau. Complexity results for blocks-world planning. In T. L. Dean and K. McKeown, editors, *Proceedings of the Ninth National Conference on Artificial Intelligence, Volume 2*, AAAI '91, pages 629–633, Menlo Park, CA, USA, July 1991. AAAI Press / The MIT Press. ISBN 0-262-51059-6. 15

[27] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3): 223–254, Aug. 1992. ISSN 0004-3702. doi: 10.1016/0004-3702(92)90028-V. 15, 16, 26

[28] W. Heisenberg. *The Physical Principles of the Quantum Theory*. University of Chicago Press, Chicago, IL, USA, 1930. ISBN 9780486601137. 9

[29] A. Horn. On sentences which are true of direct unions of algebras. *The Journal of Symbolic Logic*, 16(1):14–21, Mar. 1951. doi: 10.2307/2268661. 19

[30] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM*, 5(4): 339–342, Oct. 1958. doi: 10.1145/320941.320947. 10

[31] G. Kirchhoff. Ueber das verhältniss zwischen dem emissionsvermögen und dem absorptionsvermögen der körper für wärme und licht. *Annalen der Physik*, 185(2):275–301, 1860. doi: 10.1002/andp.18601850205. 9

[32] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, Sept. 1985. ISSN 0004-3702. doi: 10.1016/0004-3702(85)90084-0. 10

[33] H. Lagerlund. *Modal Syllogistics in the Middle Ages*, volume 70 of *Studien und Texte zur Geistesgeschichte des Mittelalters*. Brill, Leiden, Nederland, 2000. ISBN 9789004116269. doi: 10.1604/9789004116269. 40

[34] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961. ISSN 0018-8646. doi: 10.1147/rd.53.0183. 4

[35] R. J. Lipton and K. W. Regan. *Quantum Algorithms via Linear Algebra: A Primer*. MIT Press, Cambridge, MA, USA, 2014. ISBN 9780262028394. 5, 7

[36] G. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson Addison Wesley, Boston, MA, USA, 5th edition, Oct. 2005. ISBN 9780321263186. 35

[37] D. S. Nau. Enabling-condition interactions and finding good plans. In A. Lansky, editor, *Proceedings of the 1993 AAAI Spring Symposium on Foundations of Automatic Planning: The Classical Approach & Beyond*, pages 93–97, Menlo Park, CA, USA, Apr. 1993. AAAI Press. ISBN 978-0-929280-40-0. 16, 17

[38] N. J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Apr. 1998. ISBN 1-55860-467-7. 15

[39] C. H. Papadimitriou. The euclidean traveling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, June 1977. ISSN 0304-3975. doi: 10.1016/0304-3975(77)90012-3. 15

[40] E. L. Post. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43(3):163–185, July 1921. doi: 10.2307/2370324. 18

[41] D. Ratner and M. Warmuth. Finding a shortest solution for the n x n extension of the 15-puzzle is intractable. In T. Kehler and S. J. Rosenschein, editors, *Proceedings of the Fifth National Conference on Artificial Intelligence*, AAAI '86, pages 168–172, Menlo Park, CA, USA, Aug. 1986. AAAI Press. ISBN 978-0-262-51054-7. 15

[42] E. Rieffel and W. Polak. *Quantum Computing: A Gentle Introduction*. Scientific and Engineering Computation. MIT Press, Cambridge, MA, USA, 2011. ISBN 9780262015066. 5

[43] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12 (1):23–41, Jan. 1965. doi: 10.1145/321250.321253. 19

[44] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594. 15, 17, 18

[45] E. D. Sacerdoti. *A Structure for Plans and Behavior*. PhD thesis, Stanford University, Stanford, CA, USA, Aug. 1975. 14, 16

[46] E. Schröder. *Der Operationskreis des Logikkalkuls*. B. G. Teubner, Leipzig, Deutschland, 1877. 19

[47] J. Scotus and M. O'Fihely. *Johannes Duns Scotus: Opera omnia*, volume 1. Georg Olms, Hildesheim, Deutschland, 1968. doi: (OCoLC)961799163. 40

[48] N. Shenvi, J. Kempe, and K. B. Whaley. Quantum random-walk search algorithm. *Physical Review A*, 67(5):052307, May 2003. doi: 10.1103/PhysRevA.67.052307. 5

[49] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In S. Goldwasser, editor, *Proceedings of the Thirty-fifth Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 124–134, Washington, DC, USA, Nov. 1994. IEEE Computer Society. doi: 10.1109/SFCS.1994.365700. 6

[50] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct. 1997. ISSN 0097-5397. doi: 10.1137/S0097539795293172. 6

[51] E. M. Soloway and E. M. Riseman. Levels of pattern description in learning. In R. Reddy, editor, *Proceedings of the Fifth International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI '77, pages 801–811, San Francisco, CA, USA, Aug. 1977. Morgan Kaufmann Publishers Inc. 14

[52] H. P. Stapp. The copenhagen interpretation. *American Journal of Physics*, 40(8):1098–1116, 1972. doi: 10.1119/1.1986768. 9

[53] G. J. Sussman. A computational model of skill acquisition. Technical Report 297, AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA, Aug. 1973. 14, 16

[54] L. Tarrataca and A. Wichert. Problem-solving and quantum computation. *Cognitive Computation*, 3(3):510–524, Dec. 2011. doi: 10.1007/s12559-011-9103-6. 12, 30

[55] L. Tarrataca and A. Wichert. Tree search and quantum computation. *Quantum Information Processing*, 10(4):475–500, Nov. 2011. doi: 10.1007/s11128-010-0212-z. 11, 27

[56] L. Tarrataca and A. Wichert. Quantum iterative deepening with an application to the halting problem. *PLoS ONE*, 8(3):1–9, Mar. 2013. doi: 10.1371/journal.pone.0057309. v, vii, 1, 10, 12

[57] S. Thiébaux, M. Cordier, O. Jehl, and J. Krivine. Supply restoration in power distribution systems: A case study in integrating model-based diagnosis and repair planning. In E. Horvitz and F. Jensen, editors, *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, UAI '96, pages 525–532, San Francisco, CA, USA, Aug. 1996. Morgan Kaufmann Publishers Inc. ISBN 1-55860-412-X. 17

[58] T. Toffoli. Reversible computing. In J. de Bakker and J. van Leeuwen, editors, *Proceedings of the Seventh Colloquium on Automata, Languages and Programming*, volume 85, pages 632–644, Berlin, Deutschland, July 1980. Springer. ISBN 3-540-10003-2. doi: 10.1007/3-540-10003-2. 4, 5

[59] B. A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, Oct. 1984. ISSN 1058-6180. doi: 10.1109/MAHC.1984.10036. 18

[60] M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, Aug. 1992. 17

[61] R. Waldinger. Achieving several goals simultaneously. Technical Report 107, AI Center, SRI International, Stanford, CA, USA, July 1975. 14

[62] A. N. Whitehead and B. Russell. *Principia Mathematica*, volume 1. Cambridge University Press, Cambridge, UK, 1910. 19, 31, 33, 34

[63] T. Winograd. *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, Feb. 1971. 14

[64] L. Wittgenstein. *Tractatus Logico-philosophicus*. International Library of Psychology, Philosophy, and Scientific Method. Harcourt, Brace and Company, New York, NY, USA, 1922. 18

[65] M. Ying. Quantum computation, quantum theory and ai. *Artificial Intelligence*, 174(2):162–176, 2010. ISSN 0004-3702. doi: 10.1016/j.artint.2009.11.009. 1, 45

Appendix A

LQIDS Performance Analysis

If we assume b is the problem domain's branching factor, $m = \lceil \log_2 b \rceil$ the number of qubits needed to encode a transition choice, and $\zeta = b$ is our exploration limit (*i.e.*, beyond it we are *necessarily* repeating a transition, which is uninteresting for our purposes), then the entire algorithm is an $O(\sum_{d=0}^{b} 2^{dm/2}) = O(2^{b\lceil \log_2 b \rceil/2})$ time technique. Furthermore, we can conclude[43] that it executes in $O(2^{b(\log_2 b+1)/2})$ queries to the oracle. A classical exhaustive approach would need to evaluate $O(b^b) = O(2^{b \log_2 b})$ items. For $b \geq 2$, it is easy to show by induction that $2^{b \log_2 b} \geq 2^{b(\log_2 b+1)/2}$, thus proving it as an improvement. Note that both methods must perform the same number of initialization and verification steps, so we have factored these costs out of this analysis.

The extent of the speed-up approaches a quadratic order as the branching factor increases, *i.e.*, analysing the evolution of the exponents' ratio, we have that:

$$\lim_{b\to\infty} \frac{b(\log_2 b+1)/2}{b \log_2 b} = \lim_{b\to\infty} \frac{1}{2}\left(1 + \frac{1}{\log_2 b}\right) = \frac{1}{2}(1+0) = \frac{1}{2}$$

Finally, in terms of space-complexity, the methodology requires $O(\sum_{d=0}^{b} s + d(m+s) + 2) = O(s + b(m+s))$ qubits be used for the circuit, where s is the amount of qubits reserved for recording a state. By footnote [43], a worst case bound of $O(s(b+1) + b \log_2 b)$ qubits may be asserted.

Appendix B

2-EBW Unitary Transforms

Table B.1: The goal test unitary operator G. It can be inferred from $g : \{0,1\}^2 \to \{0,1\}$ in equation (3.1) over its $2^{(2+1)} \times 2^{(2+1)}$ tuples.

$$G = \begin{array}{c|cccccccc} |s_1 s_2 a\rangle & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ \hline 000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 001 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 010 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 011 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 100 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 101 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 110 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 111 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

Table B.2: The movement unitary operator T can be determined from $\boldsymbol{f} : \{0,1\}^3 \to \{0,1\}^2$ in equation (3.2). Each of its $2^{(3+2)} \times 2^{(3+2)}$ cells has a 1 if its row is (s_1, s_2, m, a_1, a_2) and its column $(s_1, s_2, m, a_1 \oplus \boldsymbol{f}_1(s_1, s_2, m), a_2 \oplus \boldsymbol{f}_2(s_1, s_2, m))$. Otherwise, the cell is valued as 0. Its truth table is partially represented here, where we assume $y_i = a_i \oplus \boldsymbol{f}_i(s_1, s_2, m)$ and only non-zero valued row/column pairs are depicted.

Input					**Output**				
s_1	s_2	m	a_1	a_2	s_1	s_2	m	y_1	y_2
0	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	0	1	1	0	1	0	1	0
0	1	1	0	0	0	1	1	0	0
0	1	1	0	1	0	1	1	0	1
0	1	1	1	0	0	1	1	1	0
0	1	1	1	1	0	1	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0	1	0	0	1	0	1	1	0
1	0	1	0	1	1	0	1	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1	0	0

Appendix C

2-EBW Calculations

We detail the steps involved in computing the result of equation (3.3) here. Recall that U_c reflects the circuitry portrayed in figure 3.4. Throughout the exposition we assume that the layout of the qubits corresponds to $|s_1 s_2 m a_1 a_2 a_3 a_4\rangle$. Registers presenting merely n qubits, with $n < 7$, refer to the first n left qubits of this arrangement.

The first gate in the circuit acts upon the encoding of the initial state in the following manner:

$$\begin{aligned} T\,|00\rangle \otimes H\,|0\rangle \otimes |00\rangle &= T\,|00\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |00\rangle \\ &= \frac{1}{\sqrt{2}}(|00010\rangle + |00101\rangle) \end{aligned}$$

where the Hadamard transformation has been applied to qubit m so that we may explore all avenues. The remaining auxiliary qubits are appropriately assumed to be 0.

Appending the auxiliary a_3 qubit to this result and executing the goal test, we obtain:

$$I^{\otimes 3} \otimes G\left(\frac{1}{\sqrt{2}}(|00010\rangle + |00101\rangle) \otimes |0\rangle \right) = \frac{1}{\sqrt{2}}(|000100\rangle + |001011\rangle)$$

where $I^{\otimes n} = \underbrace{I \otimes ... \otimes I}_{\text{n times}}$ leaves the top n qubits unchanged.

This is subjected to the C_{not} gate, after concatenating the results with the a_4 auxiliary qubit, suitably prepared in the superposition $|-\rangle$, as disclosed in Grover's method:

$$I^{\otimes 5} \otimes C_{not}\left(\frac{1}{\sqrt{2}}(|000100\rangle + |001011\rangle) \otimes |-\rangle \right) = \frac{1}{\sqrt{2}}(|000100\rangle - |001011\rangle) \otimes |-\rangle$$

Finally, we wish to recover the auxiliary values present at the outset of the computation, in keeping with Bennett's stages of reversible computing. Thus, the inverse of the goal test is first applied over the superposition above:

$$I^{\otimes 3} \otimes G^{-1} \otimes I\left(\frac{1}{\sqrt{2}}(|000100\rangle - |001011\rangle) \otimes |-\rangle \right) = \frac{1}{\sqrt{2}}(|000100\rangle - |001010\rangle) \otimes |-\rangle$$

followed by the inverse of the movement operator:

$$T^{-1} \otimes I^{\otimes 2} \left(\frac{1}{\sqrt{2}} (|000100\rangle - |001010\rangle) \otimes |-\rangle \right) = \frac{1}{\sqrt{2}} (|000000\rangle - |001000\rangle) \otimes |-\rangle$$

Since in this case our superposition consists of two basis elements, the solution $|001000\rangle$ is easy to perceive, so we know that moving block B will lead to our goal. In more complex superpositions, running Grover's amplification over these will recover the solution with certainty.