Problem solving and quantum computation

Luís Tarrataca and Andreas Wichert
Department of Informatics
INESC-ID / IST - Technical University of Lisbon
Portugal
{luis.tarrataca,andreas.wichert}@ist.utl.pt

Abstract

Is quantum computing suitable for modeling problem solving, a domain which is traditionally reserved for the symbolic approach? We propose a hybrid quantum problem solving model. Our approach is motivated by several important theories from the fields of physics, computer science and psychology. We demonstrate our approach through a model for a quantum production system, based on the n-puzzle. The developed model can be extended in order to tackle any N-level depth search required by other problems. No preliminary knowledge concerning quantum computation is required.

Keywords: Inference, quantum computation, problem solving, production system, reversible computation

1 Introduction

Cognitive computation has focused for a long focused on concepts such as knowledge representation and the reasoning processes that allow knowledge to be applied. Traditionally, these concepts have been modeled with the assistance of classical probability theory which allows the representation of possible state transitions as a graph of probabilities. However, such an assumption may be inadequate depending on the physical laws in play during possible applications. This theory was exploited in [1, 2, 3, 4, 5] where quantum probability theory was explored in order to justify empirical variations that classical probability theory failed to explain. By employing quantum von Neumann probabilities the authors were able to obtain a closer fit to experimental data.

Additionally, knowledge can be employed by problem-solving agents trying to determine adequate actions when dealing with complex environments [6]. Illustrating examples include IBM's Deep Blue [7] and Watson initiatives which focus, respectively, on chess playing and the *Jeopardy!* quiz show. Both these

system employ knowledge alongside massive amounts of parallel computation. However, these tasks are routinely performed by human brains through neuronal processing on a time-scale of 10^{-3} seconds. Manin draws attention to these facts [8] and suggests quantum processing as a theoretical alternative. Additionally, some of the possible relationships between artificial intelligence and quantum computation have been established in [9]. In this work we extend these views by introducing a possible problem-solving technique from a quantum computation perspective. Our approach will be based on production system theory since it is well suited for problem solving scenarios. The following sections are organized as follows: Section 1.1 introduces some key concepts of production systems whilst Section 1.2 establishes the links between production system theory and classical search strategies.

1.1 Production System

The production system is a formalism proposed by Post [10] to describe computational procedures through problem-solving primitives. Production system theory describes how to form a sequence of actions leading to a goal state. A production system is composed of condition-action pairs, *i.e.* if-then rules, which are also called productions. The state description at any given time is also referred to as working memory. On each cycle of operation, the working memory is matched against the conditional part of all productions. A rule is applied when the conditional part is recognized to be part of the working memory. Applying a rule results in implementing the associated action which describes a form of problem-solving behaviour [11]. Applying an action results in the state of the problem instance changing accordingly. At any given point, more than one production might be deemed to be applicable. This subset of productions represents the conflict set. A conflict resolution strategy is then employed to this subset in order to determine an appropriate production. The operational cycle is brought to a close when a goal state is reached or when no more rules can be triggered [6]. A problem is described by the productions, the initial state, and the goal state. The overall problem-solving behaviour can be interpreted as a form of computation whose elementary steps are the individual productions applied. This general architecture is illustrated in Figure 1.

Some of the best known examples of human cognition-based production systems include the General Problem Solver [12] [13] [14] [15], ACT [16] and SOAR [17] [18]. Production systems are closely related to the approach taken by Markov algorithms [19]. Furthermore, both approaches can be shown to be equivalent in power to the Turing machine [20]. The production system is also a model of actual human problem-solving behavior [21, 22, 23, 24].

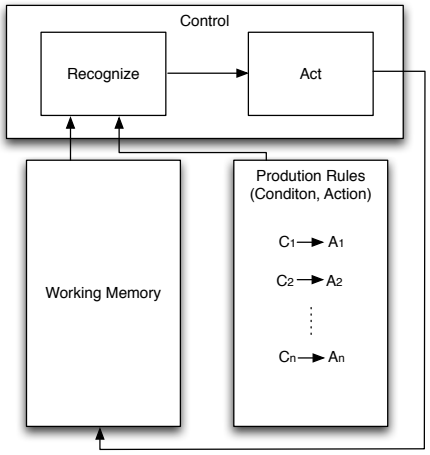


Figure 1: General architecture for a production system (adapted from [6]).

1.2 Search as a decision problem

From a computer science perspective, the control structure of a production system can be defined in terms of classical tree-search procedures which is appropriate to artificial intelligence tasks trying to mimick problem-solving behaviour. This fact makes production system theory particularly well suited for problem-solving scenarios. In general, tree algorithms are applied when when we wish to perform an exhaustive examination of all possible combinations for the state space. For these cases, the search space can be viewed as forming a hierarchy reflecting the spectrum of combinations. All that is required to formulate a search problem is a set of states, a set of operators that map states into successor states, an initial state and a set of goal states [25]. The overall objective is to find a sequence of operators that map the initial state to a goal state.

The operational behaviour of a production system requires the ability to, given an initial state, determine which state is reached after applying a production rule. As previously mentioned these actions are also an integral part of the rules employed by production systems. This process is similar in function to classical tree-search strategies where actions are applied to states, yielding as a result new configurations, a process illustrated in Figure 2. The binary tree presented depicts the nodes reached from a root node A by applying one of two possible actions, respectively 0 or 1. The actions applied during the search are the production system equivalent of applying rules. The cardinality of the set of available actions is also referred to as the branching factor b. At a search depth level d there exist a total of b^d leaf nodes. Each leaf node translates into the state reached after having applied d actions, *e.g.* node I is reach after applying actions 0, 0 and 1. We will refer to a set of actions leading to a leaf node as the

path taken during the tree-search.

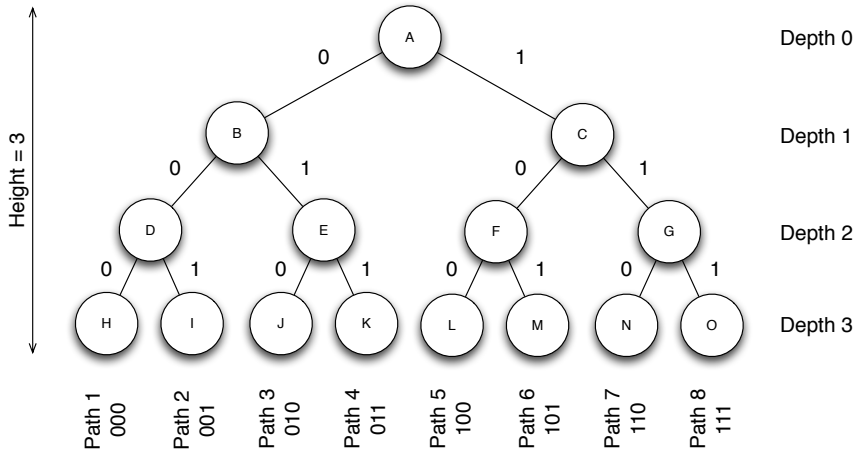


Figure 2: The possible paths for a binary search tree of depth 3.

1.3 Problem

Unfortunately, most classical tree-search procedures require exponential growth search space, *i.e.* $O(b^d)$ time. The advent of quantum computation promised sensational performance increases. Perhaps one of the most remarkable results is due to Peter Shor's algorithm for fast factorization [26] which delivered an exponential speedup when compared against the best performing classical algorithms. Later, Grover's algorithm [27] allowed for a quadratic speedup to be obtained when searching for a solution from amongst N elements. Although, not as dramatic as Shor's speedup, if we consider search spaces with b^d elements then the quantum search algorithm allows for an $O(b^{\frac{d}{2}})$ time, effectively cutting the search depth in half. Accordingly, it would be interesting to consider how to develop a mechanism incorporating classical tree-search concepts capable of being applied alongside Grover's algorithm in order to produce a hybrid quantum production system. Such a system would illustrate how to develop possible problem solving strategies from a quantum computation perspective. Our system will be built upon the concepts of the sliding block puzzle, which is a common example capable of illustrating problem-solving strategies. Others approaches to quantum search have been detailed in [28], [29], [30] and [31].

All of the above issues will be key features of our quantum production system which will be explored in the remainder of this work. We will start by presenting a brief introduction in Section 2 to reversible circuitry which is an integral part of quantum computation. Section 3 discusses a possible approach to a reversible production system capable of solving instances of the 3-puzzle. Section

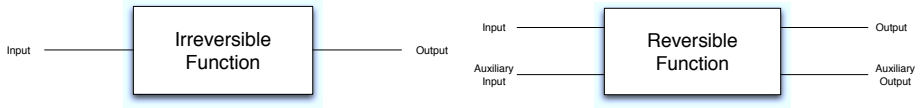
4 proceeds by providing the necessary background for a quantum extension of our propositions. Section 5 presents the conclusions of our work.

2 Insights into reversible computation

Reversibility is a key feature of quantum physics [32] [33] [34]. Changes occurring to a quantum state can be described using the language of quantum computation. A quantum computer is built from a quantum circuit containing wires and elementary quantum gates to carry around and manipulate quantum information [35]. This operational behaviour is similar to the one found in classical computer logic employing circuits constituted by gates. In quantum computation, it is possible to employ equivalent logical operations with a specific caveat, namely all operations must be performed in a reversible manner. The origins of reversible computation can be traced back to [36], [37], [38] and [39].

Mathematically, reversible circuits, and respective gate components, can also be represented in linear algebra terms as matrices known as unitary operators. A matrix A is said to be unitary if A's transpose complex conjugate, denoted by A^{*^T}, or simply by $A^\dagger$, is also the inverse matrix of A [40]. Notice that this is equivalent to $A^{-1} = A^\dagger$ and consequently $A^{-1}A = A^\dagger A = I$. In this notation, each matrix column describes the transformation suffered at a specific column index. Additionally, unitary operators preserve the norm of the vectors. These concepts correspond to those of bijective functions. In the remainder of this document we will use the terms reversible circuit, reversible gate and unitary operator in an equivalent manner.

Traditionally, classical computation is seen as an irreversible process, a direct consequence from the use of many-to-one binary gates. A logical gate is a function $f : \{0,1\}^k \rightarrow \{0,1\}^l$ from some fixed number k of input bits to some fixed number l of outputs bits [41]. A computation is said to be reversible if given the outputs we can uniquely recover the originating inputs [38] [42]. As an example and counterexample consider, respectively, the logical gates NOT and AND. The former gate is reversible because if the output is 1 we know that the input must have been 0, and the same is also true the other way around. However, the same cannot be said for the AND gate. More precisely, given output 0 there are a total of three combinations which might have yielded the stated result. So, the question naturally arises: Is there a general mechanism for converting irreversible computations into reversible ones? It turns out that there is such a mechanism. Each irreversible gate can be made reversible by adding some additional input and output wires [43]. This conversion introduces a certain number of inputs and outputs to each irreversible gate. It is this additional information that provides for reversible computation. This process is illustrated in Figure 3.



(a) (b)

Figure 3: An irreversible function 3a can be mapped into a reversible function 3b through the introduction of a number of constants and auxiliary input and output bits. (Source: [42])

Accordingly, given an irreversible function f, a reversible mapping can be constructed with the form illustrated in Expression 1 [43], where x is the input register, and c is an auxiliary control bit. The reversible gate will require $\lceil \log_2 x \rceil + 1$ input bits and an additional $\lceil \log_2 x \rceil + 1$ output bits, *i.e.* a combined total of $2(\lceil \log_2 x \rceil + 1)$ bits. Such behaviour contrasts with the irreversible counterpart which would require $\lceil \log_2 x \rceil + 1$ bits. Accordingly, the number of bits employed by both versions differs by a constant factor $k = 2$.

$$(x, c) \mapsto (x, c \oplus f(x)) \tag{1}$$

The conversion process of the irreversible AND gate to a reversible form can be visualized as adding an additional input wire (for the control bit) and two additional output wires (in order for the inputs to be part of the outputs). The reversible AND gate and its truth table are presented, respectively, in Figure 4 and in Table 1. Notice that the AND operation of bits a and b, *i.e.* ab, is effectively stored in the third output bit. Given that bit c is known from the start, it is possible to uniquely recover ab through the operation $c \oplus (c \oplus (ab)) = ab$. This simple mapping mechanism allows for maintaining a detailed account of all inputs and outputs. Ergo, it is in accordance with the definition of reversible computing.

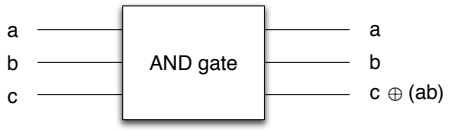


Figure 4: Reversible AND gate.

How can we build a unitary operator capable of expressing the inner-workings of the reversible AND gate? Unitary matrices perform one-to-one mappings of states. A quick analysis of Table 1 illustrates precisely this behaviour which, in its essence, can be interpreted as the set of possible mappings between all binary states. The majority of the state transitions map onto themselves, with the exception of the transitions associated with binary inputs “110” and “111”.

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

Table 1: Truth table for the reversible AND gate.

More specifically, binary state "110" maps onto "111", which we express as $110 \rightarrow 111$, or by employing an equivalent decimal representation, $6 \rightarrow 7$. Additionally, state "111" maps to "110", i.e. $7 \rightarrow 6$.

Constructing the unitary operator requires the ability to encode each one of these mappings. In order to do so, we must first focus on how to represent each state. This task can be performed by using an 2^n dimensional vector x, where n represents the number of bits. In quantum computation, the Dirac notation [44] [45] is employed in order to represent vector x as $|x\rangle$. For the specific case of the reversible AND, a vector with dimension 2^3 should be employed, with each state being represented by an entry set 1 at the corresponding dimension, and all remaining dimensions set to 0, e.g. state $|0\rangle = (1,0,0,0,0,0,0,0)^T$, state $|1\rangle = (0,1,0,0,0,0,0,0)^T$, ..., and state $|7\rangle = (0,0,0,0,0,0,0,1)^T$. From a mathematical point of view applying the reversible AND gate to an input corresponds to multiplying the matching unitary operator by an input vector. This operation is presented in Expression 2, where U denotes the unitary operator for the reversible AND gate, $|a\rangle$ the input state and $|b\rangle$ the output state.

$$U|a\rangle = |b\rangle \tag{2}$$

Notice that each input vector $|a\rangle$ specifies which columns of U should be taken into consideration in order to form the output vector. For instance, input vector $(1,0,0,0,0,0,0,0)^T$ effectively eliminates all but the first column from the multiplication process. Accordingly, in order to define the mapping $U|0\rangle = |0\rangle = (1,0,0,0,0,0,0,0)^T$ we may specify $(1,0,0,0,0,0,0,0)^T$ as the first column of U. As an additional example state consider the mapping $U|7\rangle = |6\rangle = (0,0,0,0,0,0,1,0)^T$, accordingly the eighth column of U should specify $(0,0,0,0,0,0,1,0)^T$ as the resulting transformation. Repeating this process of column permutation for the remaining columns allows one to obtain the complete form of U, which is presented in Expression 3 alongside with the associated transformations as superscript indexes.

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

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

In general, an irreversible circuit can be made reversible by substituting each irreversible gate by an equivalent reversible gate [38]. Additionally, if each element of the circuit is replaced with its respective inverse we are able to perform the inverse operation of the original circuit. In practice, this means that if we run the reversed circuit with an output, we will obtain the originating input bit register.

3 Sliding block puzzle

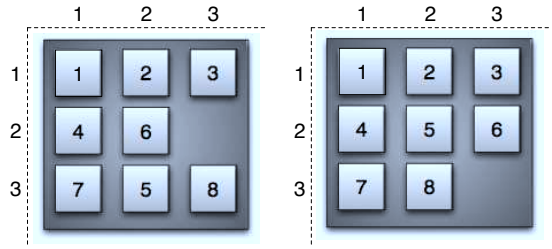
Our quantum production system will be based on the concepts of the sliding block puzzle. The sliding block puzzle is a familiar problem commonly approached in the artificial intelligence community, which conveniently showcases key problem-solving notions. The next couple of sections are organized as follows: Section 3.1 will start by describing a classical production system for the sliding block 8-puzzle; Section 3.2 will then build on these results and develop a simpler sliding block 3-puzzle; Section 3.3 presents the details for a reversible circuit capable of solving instances of the 3-puzzle; Section 3.4 describes the details surrounding the extensions to any n-puzzle.

3.1 Sliding block 8-puzzle

Many artificial intelligence applications involve composing a sequence of operations. The search space generated by an 8-puzzle sliding block puzzle is both complex enough to be interesting and small enough to be tractable. It also lends itself to solution using a production system [6].

According to [46] a sliding block puzzle challenges a player to shift pieces around on a board without lifting them to establish a certain end-configuration. This non-lifting property makes finding moves, and the paths opened up by each move important parts of solving sliding block puzzles . Typically, both the initial- and end-configuration might be chosen randomly, as illustrated by Figure 5. However, the end-configuration typically reflects some sort of logical arrangement, as exemplified in Figure 5b. The final logical arrangement is specific to individual problem instances of sliding block puzzles. The set of possible elements for the 8-puzzle, $S_{8-puzzle}$, is presented in Expression 4.

$$S_{8-puzzle} = \{One, Two, Three, Four, Five, Six, Seven, Eight, Blank\} \quad (4)$$



(a) Initial board configura-(b) End board configuration.
tion.

Figure 5: A sliding block puzzle example with a board of dimension 3×3.

Typically, each sliding block puzzle has a blank cell, which can be perceived to move on a set of possible directions. This way we gain generality by thinking of "moving the blank cell" rather than moving a numbered tile. In the case of the sliding block puzzle exemplified in Figure 5 only diagonal movements are not allowed. Accordingly, the set of possible movements for the blank cell consists of actions *Up*, *Down*, *Right* and *Left*, as illustrated by Expression 5. A solution to the problem is an appropriate sequence of actions, such as "'move blank cell up, move blank cell left, ..., etc'"". Clearly, not all possible actions are applicable to all positions within the board. In fact, only position $(2, 2)$ is able to execute the full range of motions. If the blank cell is in any of the remaining positions of the board then only 2 to 3 moves can be executed.

$$\text{Possible Actions} = \{Up, Down, Left, Right\} \quad (5)$$

Before advancing any further it is important to focus on a few points regarding the complexity of solving sliding block puzzles. More precisely, is there any way of determining if an end-configuration is obtainable from an initial configuration? If so, what is the minimum set of movements for achieving the desired

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

Table 2: Production rules set for the 8-puzzle. (Source: [6])

state? As is to be expected a wide number of authors have tried to tackle this subject. Surprisingly, not a great deal of progress has been achieved. Short of trial and error, it is impossible to present an answer to these questions [47]. This apparent inability to solve efficiently sliding block puzzles stems from computational complexity theory. Sliding-block puzzles have been shown to belong to a class of problems known as PSPACE-complete [48] [49]. PSPACE consists of those problems which can be solved using few spatial resources but potentially requiring significant time. PSPACE is thought to be even harder than NP-complete, although this has never been proved.

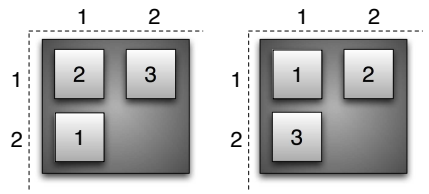
As previously stated, in order to solve a problem using a production system, we must specify the working memory, the productions and the control strategy. Lets say we initialize the working memory with the initial board configuration depicted in Figure 5a and we aim to reach the target board configuration illustrated in Figure 5b. We can then define an illustrating set of production rules as presented in Table 2. The only remaining issue is due to the control strategy employed, which might be defined as [6]

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

3.2 Sliding block 3-puzzle

In order to proceed with the development of our quantum production system we will concentrate on a board with dimension 2×2, i.e. a 3-puzzle. Figure 6 depicts a 2×2 sliding block puzzle with an initial board configuration (Figure 6a) and a target configuration (Figure 6b). The set of possible movements for the blank element remains the same as in the case of the 8-puzzle, respectively illustrated in Expression 5. However, for the 3-puzzle, at any given position only two movements are deemed possible to be executed. Since the blank cell always occupies a corner position, its movement can be perceived as performing a clockwise or counter-clockwise movement. Figure 7 illustrates the states obtained after performing counter-clockwise and clockwise movements for the initial board configuration presented in Figure 6a. The set of possible elements for the 3-puzzle, $S_{3-puzzle}$, is presented in Expression 6. We choose to focus on the concepts surrounding a 3-puzzle board merely for practical reasons, since such

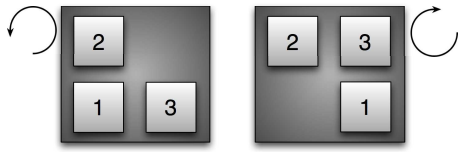
a problem requires fewer bits to encode and consequently demands a smaller-sized reversible circuit. However, as we will demonstrate later, our model can be extended in order to accommodate any N-puzzle, with $N \geq 3$.



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

Figure 6: A 3-puzzle example.

$$S_{3-puzzle} = \{One, Two, Three, Blank\} \tag{6}$$



(a) Blank-cell counter-clockwise movement. (b) Blank-cell clockwise movement.

Figure 7: Movement example for the blank cell given the board configuration depicted in Figure 6a.

3.3 Building the reversible circuit for the 3-puzzle

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

[1] Another possible strategy would consist in encoding each of the $|S_{3-puzzle}|! = 4! = 24$ possible board configurations. This strategy would require $\lceil \log_2 24 \rceil = 5$ bits, allowing for three bits to be saved. However, such an encoding mechanism would make it harder to understand the position of each element.

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

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

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

Table 4: Binary strings depicting the board configurations presented in Figure 6a and Figure 6b.

Board configurations can now be perceived as a binary string of length 8 containing the encodings for each position of the board. This process is illustrated in Table 4. The binary representation for board configurations will be crucial to the development of the reversible circuit.

In order to proceed with our analysis lets focus on a few design concepts. Namely, in order to develop a production system capable of tackling a sliding block puzzle we need the ability to:

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

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

When dealing with reversible computation, we find it helpful to first consider the desired operational behaviour in terms of a classical gate. In doing so we gain some useful insight into the inputs and outputs of the reversible operator. Let us begin by concentrating our efforts on the first requirement. Theoretically, we need to develop a gate capable of receiving as an argument a binary string depicting the state of the board to be tested. In classical computation, we would simply output a single bit, which would be set to 1 if the board presented was the target board configuration and 0 otherwise. This computational process can be represented as function f illustrated in Expression 7.

$$f(\underbrace{b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8}_{\text{board configuration}}) = \begin{cases} 1 & \text{if target board configuration} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

From the previously presented Expression 1 we know that the inputs should also be outputted, i.e. the board configuration should also be part of the outputs. The only issue is due to the result bit, which requires that a single control bit be provided as an input. Accordingly, our reversible gate will have a total of 9 input and output bits, 8 of which are required for representing the board and 1

Inputs									Outputs								
b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	c	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	$c \oplus f(b)$
0	0	0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0
0	0	0	1	1	0	1	1	1	0	0	0	1	1	0	1	1	1
0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	0	1	0	0	0	1	1	1	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	0	1	1	0	0	0	0	1	1	0	1	1	0	0	1
0	1	1	0	1	1	0	0	1	0	1	1	0	1	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	0	0	1	0	0	0	1	1	1	0	0	1	0	0	0
1	1	1	0	0	1	0	0	1	1	1	1	0	0	1	0	0	1

Table 5: A selected number of results from the truth table of the target board unitary operator.

bit serving as control. This gate, which we will label as the target board unitary operator, is illustrated in Figure 8 [2]. Table 5 showcases the gate's behaviour for a few board configurations, where $f(b)$ denotes $f(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8)$. Notice that when the gate determines that a board is a target board configuration it effectively switches the control bit, as highlighted in Table 5.

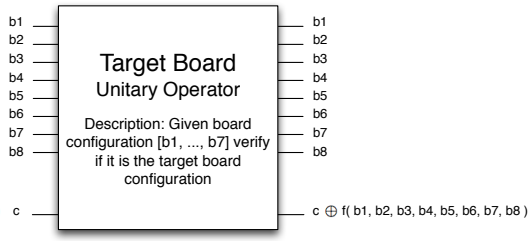


Figure 8: The target board unitary operator.

From a mathematical point of view, how can we express the inner-workings of the reversible circuit illustrated in Figure 8? We have seen how to approach unitary operator construction in Section 2. Accordingly, we need to specify the set of column permutations. Let T denote the unitary operator responsible for implementing the behaviour of function f. T is a matrix with dimensions $2^9 \times 2^9$. From Table 5 it should be clear that only two input states map onto other states rather than themselves (when considering the target board configuration of Figure 6b). Namely, $T|216\rangle \rightarrow |217\rangle$ and $T|217\rangle \rightarrow |216\rangle$. Accordingly, the 217^{th} column of T should permute to $|217\rangle$, and the 218^{th} column map to state $|216\rangle$. All other remaining states would continue to map onto themselves.

Next, we draw our attention to the second requirement. As we have seen the

[2] We could extend this gate in order to receive as an input the desired target board configuration. This addition would be carried out at a cost of 8 additional input and output bits. However, since we are aiming for design simplicity, the target board configuration will be "hard coded" into the gate. In doing so, we loose generality but gain a simpler design.

original board configuration provided as input should also be part of the outputs. The new gate should also output a new board configuration. Also, in the context of a production system we are interested in applying the move blank cell unitary operator if and only if the inputted board configuration is not a target board configuration. Otherwise, the production system would systematically discard any potential solutions found. This process can be performed by including a reference to function f in the new function g's definition.

The only issue is due to how to output the new board configuration in a reversible manner. Expression 1 illustrated how this process could be performed for a single result bit. In this specific case we are interested in having 8 result bits representing the new board configuration. Accordingly, since the gate must respect the principles of reversible computation, an additional 8 control bits should be included as input. Expression 1 can be extended in order to accommodate any number of control bits, as illustrated by Expression 8 [43] where c_i are control bits, and $f(x) = (y_1, y_2, \cdots, y_n)$.

$$(x, c_1, c_2, \cdots, c_n) \mapsto (x, c_1 \oplus y_1, c_2 \oplus y_2, \cdots, c_n \oplus y_n) \tag{8}$$

Such a reversible gate formulation will require $2(\lceil \log_2 x \rceil + n)$ input and output bits, whilst its irreversible equivalent would be satisfied with $\lceil \log_2 x \rceil + n$ bits. Again, both versions differ by a constant factor $k = 2$. From a computational complexity perspective such a factor effectively doubles the space employed by a reversible formulation when compared against classical irreversible ones. However, since such growth is not exponential it can still be described as moderate.

Expression 8 makes it possible to define a function g responsible for producing the new board configuration. Function g should receive as inputs the current board configuration and a bit m indicating whether the blank cell should perform a clockwise ($m = 1$) or counter-clockwise movement ($m = 0$). By systematically checking for the position of the blank cell and with the movement described in bit m we can generate the new board configuration. Let $g : \{0,1\}^9 \to \{0,1\}^8$ with $g(b, m) = (y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8)$, where b denotes a valid board configuration $(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8)$, and m the type of movement. By valid board configuration we mean that a board configuration must be an un-ordered collection of size four composed of distinct elements taken from

Table 1. The computational behaviour of g is presented in Expression 9.

$$g(b,m) = \begin{cases} (b_5, b_6, b_3, b_4, b_1, b_2, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_1 = b_2 = 0 \text{ and } m = 0 \\ (b_3, b_4, b_1, b_2, b_5, b_6, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_1 = b_2 = 0 \text{ and } m = 1 \\ (b_3, b_4, b_1, b_2, b_5, b_6, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_3 = b_4 = 0 \text{ and } m = 0 \\ (b_1, b_2, b_7, b_8, b_5, b_6, b_3, b_4) & \text{if } f(b) = 0 \text{ and } b_3 = b_4 = 0 \text{ and } m = 1 \\ (b_1, b_2, b_3, b_4, b_7, b_8, b_5, b_6) & \text{if } f(b) = 0 \text{ and } b_5 = b_6 = 0 \text{ and } m = 0 \\ (b_5, b_6, b_3, b_4, b_1, b_2, b_7, b_8) & \text{if } f(b) = 0 \text{ and } b_5 = b_6 = 0 \text{ and } m = 1 \\ (b_1, b_2, b_7, b_8, b_5, b_6, b_3, b_4) & \text{if } f(b) = 0 \text{ and } b_7 = b_8 = 0 \text{ and } m = 0 \\ (b_1, b_2, b_3, b_4, b_7, b_8, b_5, b_6) & \text{if } f(b) = 0 \text{ and } b_7 = b_8 = 0 \text{ and } m = 1 \\ (b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) & \text{otherwise} \end{cases} \tag{9}$$

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

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

The reversible gate, which we will refer to as the move blank cell unitary operator M, is illustrated in Figure 9. M is a matrix of dimension $2^{8+1+8} \times 2^{8+1+8}$ which can be built in a similar way to T, *i.e.* from the corresponding truth table determine the mappings between states. These mappings can then be translated as column permutations.

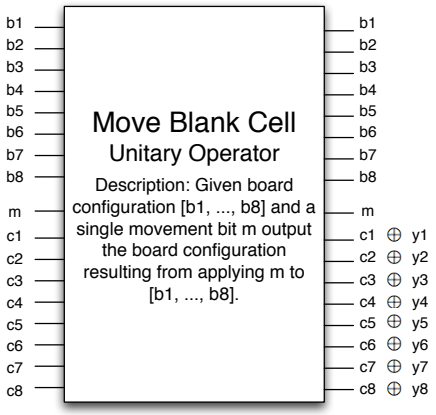


Figure 9: The move blank cell unitary operator.

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

In general, for the production system to work we need to verify if a target board configuration has been reached after applying a production rule. The move blank cell operator M incorporates in its design a test for determining if the gate should be applied or not. Therefore, it is only required to check if the final board configuration corresponds to a target board configuration. This process is illustrated in Figure 10 where res has the value presented in Expression 10. Alternatively, one could first employ the target board unitary operator and then redirect the output containing the result to a modified move blank cell gate. The modified version of the movement gate would employ that outcome in order to determine if the input board was in a target configuration or not. However, this would result in an unnecessary level of complexity to be added to the solution. Since we are aiming for simplicity we chose the first approach.

$$res = c_9 \oplus f(c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7, c_8 \oplus y_8) \quad (10)$$

Notice that the concept of working memory is contemplated through the initial board configuration bits $b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$ and the result bits $c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7, c_8 \oplus y_8$. Also, the circuit design translates the control strategy (although a very primitive one), *i.e.* if it is possible move the blank cell and test if the new board is a target board. It is worthy to draw attention to the fact that Figure 10 illustrates the application of a single movement operator M alongside a target board operator T. Algebraically, we can express this operation as presented in Expression 11, where $I^{\otimes 9} = I \otimes I \otimes \cdots \otimes I$ repeated 9 times, since operator T should only take into consideration bits $c_1, c_2, \cdots, c_9$. The unitary operator presented in Expression 11 would act on Hilbert space $\mathcal{H} = H_b \otimes H_m \otimes H_c$, where H_b is the Hilbert space spanned by the basis states employed to encode the board configuration bits $b = b_1, b_2, \cdots, b_8$, H_m is the Hilbert space spanned by the basis states employed to represent the set of productions, and H_c is the Hilbert space spanned by the auxiliary control bits.

$$(I^{\otimes 9} \otimes T)M|b_1, b_2, \cdots, b_8, m, c_1, c_2, \cdots, c_8, c_9\rangle \quad (11)$$

The previous strategy can be extended in order to apply any number of movement operators, where the output of a movement gate is provided as input to another movement operator. In doing so, we add a guarantee that, if possible, another production rule is applied to the initial board configuration. This process is illustrated in Figure 11 where two movement gates, *i.e.* productions, are applied to an initial board configuration $b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8$. Accordingly, two movement bits are required, namely m_1 and m_2. The former of which is fed as input to a first movement gate M_1, whilst the latter is provided as input to a second movement gate M_2. Consequently, res has the value presented in Expression 12. Notice that applying M_2 requires that its application by "shifted" by a total of 9 positions, whilst operator T should be applied after input bit 18. Algebraically, the overall circuit behaviour can be described as presented in Expression 13.

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

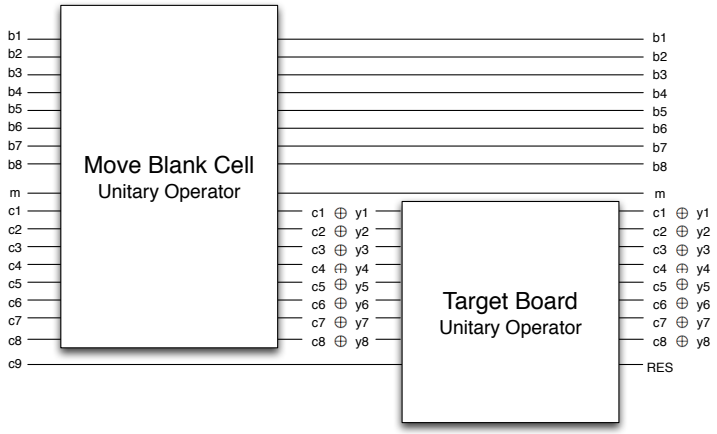


Figure 10: A reversible circuit incorporating the application of a single production rule for the 3-puzzle and a test condition in order to determine if the final board is a target configuration board.

$$(I^{\otimes 18} \otimes T)(I^{\otimes 9} \otimes M)M|b_1, b_2, \cdots, b_8, m_1, c_1, \cdots, c_8, m_2, c_9, \cdots, c_{16}, c_{17}\rangle \tag{13}$$

3.4 Extending for any n-puzzle

How can we build on this result in order to accommodate any n-puzzle? Let E be the set of possible elements for an n-puzzle, then the number of bits required

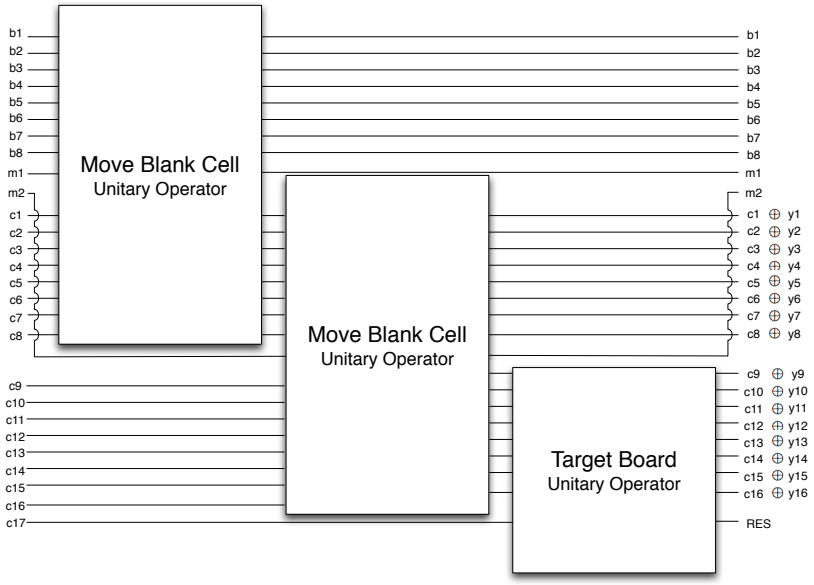


Figure 11: A reversible circuit illustrating the application of two move blank cell operators for the 3-puzzle and a target board gate in order to determine if the final board is a target configuration board.

to encode each element is $e = \lceil \log_2 |E| \rceil$. This implies that the number of bits required to encode a board configuration is $b = |E|e$. Additionally, let P be the set of possible productions for the same n-puzzle. Accordingly, $p = \lceil \log_2 |P| \rceil$ bits will be required for each production. Each movement operator M will require a total of $b + p + b = 2b + p$ input and output bits, and each target operator T will require a total of $b + 1$ input and output bits.

How many bits will be required by the circuit? Suppose we wish to apply the M operators a total of m times. The first operator M_1 requires $2b + p$ bits. Since a part of M_1 outputs will be provided as input to M_2 an additional $b + p$ bits will be added to the circuit. This means that $2b + p + (m - 1)(b + p)$ bits of the circuit are required just to move the blank element. Since operator T requires a single control bit this implies that the overall circuit employs a total of $n = 2b+p+(m-1)(b+p)+1$ bits. Of these n bits $c = n-(b+mp) = mb+1$ bits are control, or auxiliary, bits. These control bits can be perceived as the working memory of the production system. Furthermore, the sequence of bit indexes after which a movement operator M should be applied is $V = \{0, b + p, 2(b + p), 3(b + p), \cdots, (m - 1)(b + p)\}$.

Based on these statements we can describe a general formulation for an n-puzzle circuit C employing adequate M and T operators. This process is presented in Expression 14. Unitary operator C would act on an input register $|x\rangle$ en-

compassing the initial board configuration, the set of productions and also the auxiliary control bits. Accordingly, operator C would act upon a Hilbert space $\mathcal{H}$ spanned by the computational basis states required to encode x.

$$C = (I^{\otimes m(b+p)} \otimes T) \prod_{k \in V} \left(I^{\otimes k} \otimes M \right) \tag{14}$$

Finally, the number of movement operators M grows linearly with the depth of the search, which is a key component of the input register. The reversible circuits presented in Figure 10 Figure 11 apply, respectively, one and two movement gates, the former can be perceived as performing a depth-limited search of level 1, whilst the latter performs a search up to depth 2. This is a common strategy in computer science where a cutoff is performed at a depth-level d in order to provide an answer in a constant time. Traditionally, some type of heuristic function is employed in order to help decide which state may be closer to providing a solution. In addition, our proposition only contemplates tree-search involving a constant branching factor. However, it is not unusual for tree-search to deal with non-constant branching factors. Since potential encoding strategies are always required to reflect the complete range of actions this fact can adversely affect the performance of the system when compared against classical equivalent. This is due Grover's speedup being a function of the number of bits n employed. For more on these issues we refer the reader to [50] which presents a detailed examination.

4 Quantum extensions

Hitherto, we have only focused on how to build a reversible circuit for the 3-puzzle. However, reversible computation by itself does not provide any computational advantage. In order to gain a quantum advantage we need to employ quantum superpositions alongside Grover's algorithm. The following sections are organized as follows: Section 4.1 presents the quantum superposition principle; Section 4.2 introduces Grover's algorithm and Section 4.3 builds on these results to illustrate how the proposed reversible circuit can be extended in order to reap the benefits provided by quantum computation.

4.1 Quantum superpositions

The quantum superposition principle allows a register to be in several states at the same time. Associated with each quantum state i is an amplitude $\alpha_i \in \mathbb{C}$. By requiring the vector $(\alpha_1, \cdots, \alpha_n)$ to be unit-length preserving we ensure that $|\alpha_1|^2 + \cdots + |\alpha_n|^2 = 1$, where n is the number of states. Since we have a set of values that sum up to 1 then the $|\alpha_i|^2$ values may be considered as translating the probability of observing state i. However, due to the effects of

a process known as quantum decoherence only one of the states conveying the answer can be obtained. This collapse from a multitude of states into a single one takes into account the probabilities associated with each state [40]. Accordingly, states with a higher probability are more likely to be obtained, whilst states with smaller probabilities are less likely to be obtained. Notice that this does not imply that only those states with higher probabilities will be obtained. The Hadamard gate allows one to configure an input register configured to state $|00\cdots0\rangle$ in an uniform superposition of 2^n states, where n is the number of bits of the input register. This behaviour is presented in Expression 15. Traditionally, the $|\psi\rangle$ symbol is employed to denote a superposition of values. Superposition $|\psi\rangle$ can be employed alongside unitary operator C, a procedure translated as $C|\psi\rangle$, which effectively evaluates all possible states in a single computational step.

$$|\psi\rangle = H^{\otimes n}\underbrace{|00\cdots0\rangle}_{n \text{ bits}} = \underbrace{\frac{1}{\sqrt{2^n}}}_{\text{amplitude}} \sum_{x=0}^{2^n-1} |x\rangle \tag{15}$$

In order to proceed with our analysis lets concentrate on the circuit presented in Figure 10. Conceptually, we can differentiate between three inputs, respectively:

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

Let U_g refer to the unitary operator characterizing the reversible circuit presented in Figure 10. The behaviour of U_g can be described as illustrated by Expression 16.

$$U_g : |b\rangle|m\rangle|c\rangle \mapsto |b\rangle|m\rangle|c_1 \oplus y_1, c_2 \oplus y_2, c_3 \oplus y_3, c_4 \oplus y_4, c_5 \oplus y_5, c_6 \oplus y_6, c_7 \oplus y_7, c_8 \oplus y_8, res\rangle \tag{16}$$

The input register $|b\rangle|m\rangle|c\rangle$ can be configured to a specified value and by applying U_g it becomes possible to obtain the respective result. In the specific case of the unitary operator U_g we are interested in evaluating a combined superposition containing all possible board configurations and productions. The control bits not need not to be placed in a superposition since they are only employed in order to assist the overall process. Accordingly, the control bit register can be configured to $|0\rangle^{\otimes c}$, where c is the number of control bits. Let $|\psi\rangle$ denote the superposition of all board configurations, $|\psi_b\rangle$, and productions, $|\psi_m\rangle$, as illustrated in Expression 17.

$$
\begin{aligned}
|\psi\rangle &= |\psi_b\rangle|\psi_m\rangle|0\rangle^{\otimes c} \\
&= \frac{1}{\sqrt{2^8}} \sum_{x=0}^{2^8-1} |x\rangle \frac{1}{\sqrt{2^1}} \sum_{x=0}^{2^1-1} |x\rangle|0\rangle^{\otimes c} \\
&= \frac{1}{\sqrt{2^9}} \sum_{x=0}^{2^9-1} |x\rangle|0\rangle^{\otimes c}
\end{aligned} \tag{17}
$$

Where each $|x\rangle$ should be interpreted as a state of the combined input register $|b\rangle|m\rangle$. Since unitary operators obey linearity principles we are now in a position to apply unitary operator U_g to the superposition register. In practice this process means that all possible board configurations and productions encoded in the superposition register are processed simultaneously. This operation is illustrated in Expression 18.

$$
U_g|\psi\rangle = \frac{1}{\sqrt{2^9}} \sum_{x=0}^{2^9-1} U_g|x\rangle|0\rangle^{\otimes c} \tag{18}
$$

4.2 Grover's search

Not surprisingly a great deal of scientific research has focused on altering the amplitudes, and consequently the probabilities, associated with each solution state. Perhaps the best known example of such an amplitude amplification process is Grover's search algorithm [27] [51]. Grover's algorithm was later experimentally demonstrated in [52]. The algorithm provides a polynomial speed-up when compared with the best-performing classical search algorithms. Any such classical algorithm requires $O(N)$ time in order to search N elements. Grover's algorithm requires $O(\sqrt{N})$ time, providing a quadratic speedup, which is considerable when N is large. Suppose we wish to search through a problem's search space of dimension N and that it is possible to efficiently perceive potential solutions to a problem. This is similar to the NP class of problems whose solutions are verifiable in polynomial time $O(n^k)$ for some constant k, where n is the size of the input to the problem [53]. Grover's search algorithm employs quantum superposition and reversible computation in order to query many elements of the search space simultaneously. An oracle [54] representing a unitary operator U_g is employed in order to mark the solution states. This process can be performed by adding an additional input bit c to a unitary operator combined with a function $g(x)$ which outputs 1 when x is a solution and 0 otherwise, as shown in Expression 19.

$$
U_g : |x\rangle|c\rangle \mapsto |x\rangle|c \oplus g(x)\rangle \tag{19}
$$

The algorithm then employs a process of amplitude amplification, known as Grover's iterate, in order to amplify the amplitudes of the solutions and in the process diminish those of the non-solutions. This process is performed by setting the control register c to a specified value, which, when combined with Grover's iterate can be mathematically proven to perform an inversion about the mean of the amplitudes [43]. As a direct result of Grover's iterate, the probability of the solution states increases. However, the amplitude of the solution value is amplified only in a linear way. If the function g is provided as a black box, then $\Omega(\sqrt{N})$ applications of the black box are necessary in order to solve the search problem with high probability for any input [54] A number of improvements have been proposed since Grover's original work [55] [56]. These improvements essentially targeted reduced time complexity bounds for non-query operations and overall robustness. For a number of several novel search related applications please refer to [51] [57] [58].

4.3 Oracle development

Ideally, in order to take advantage of Grover's algorithm our reversible circuit approach towards production systems should mimic the behaviour illustrated in Expression 19 as opposed to the one presented in Expression 16. In order to perform such a mapping we start with a simple observation, namely that Expression 19 effectively means that all the inputs, excluding bit c, should also be part of the outputs. Accordingly, the circuits presented in Figure 10 and Figure 11 should somehow undo their computation. As previously stated in Section 2, this operation can be performed by building a "mirror" circuit, where each component is the inverse operation of original circuit. Then, with both circuits developed, it is just a matter of establishing the appropriate connections, *i.e.* the outputs of the original circuit are provided as inputs to the mirror. The application of these operations to the reversible circuit presented in Figure 10 is illustrated in Figure 12 whose unitary operator computes $U_g : |b\rangle|m\rangle|c\rangle \mapsto |b\rangle|m\rangle|c\rangle$. Logically, this overall operation has the unpleasant, but also coveted, consequence of ending up in the same place where it started. Equivalently, it is possible to state this result in terms of the unitary operator C of Expression 14 as $C^{-1}C|x\rangle = |x\rangle$.

Consequently, an additional form of control has to be incorporated into the circuit design in order for the circuit's overall computation to be saved, respectively, the res value of Expression 12. This operation can be performed with the introduction of another control bit alongside a controlled-NOT gate, denoted CNOT, which is a famous gate in quantum computation. The gate acts on two bits, labelled the control bit and the target bit. The control bit is always unaffected by the CNOT gate. The target bit is switched, *i.e.* applied the NOT operation, if the control bit is set to 1. Otherwise, if the control bit has value 0, the gate does nothing. The truth table for the CNOT gate is presented in Table 6. The introduction of the CNOT gate allows the result to be saved in a re-

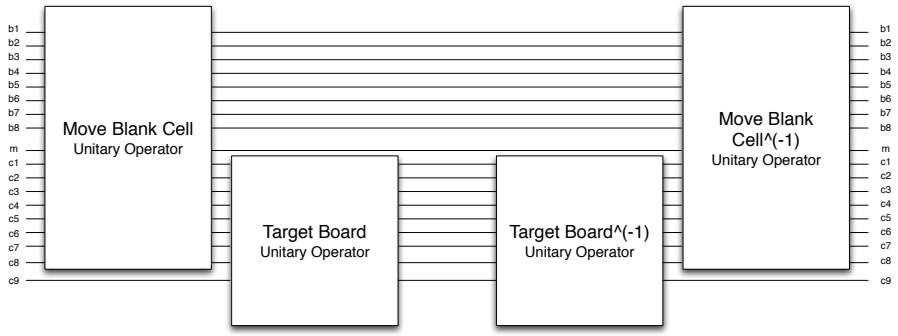


Figure 12: A reversible circuit showcasing the application of a single movement gate for the 3-puzzle, and then undoing the previously performed computations.

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

Table 6: Truth table for the CNOT gate.

versible manner, which, as previously mentioned, is pre-requisite for describing operations in quantum computation. The combination of the reversible circuit presented in Figure 12 alongside the CNOT gate is shown in Figure 13. The circuits overall computation is presented in Expression 20 where res has the value shown in Expression 10. If we label the input register $|b\rangle|m\rangle|c\rangle$ as $|x\rangle$ then Expression 20 is equivalent to Expression 19.

$$U_g : \underbrace{|b\rangle|m\rangle|c\rangle}_{\text{input}} \quad \underbrace{|c_{10}\rangle}_{\text{oracle's control bit}} \quad \mapsto |b\rangle|m\rangle|c\rangle|c_{10} \oplus res\rangle \tag{20}$$

This result can be stated in more general terms if the previously constructed operator C of Expression 14 is used, as illustrated by Expression 21.

$$U_g = C^{-1}(I^{\otimes 2b+p+(m-1)(b+p)}CNOT)C|b\rangle|m\rangle|c\rangle|c_{mb+2}\rangle \tag{21}$$

In both cases the Hilbert space $\mathcal{H}$ of the input register is augmented with the basis states required to encode the additional auxiliary control bit, accordingly $\mathcal{H} = \mathcal{H}_b \otimes \mathcal{H}_m \otimes \mathcal{H}_c \otimes \mathcal{H}_{c_{mb+2}}$. The reversible circuit presented in Figure 13 in conjunction with Grover's algorithm provides for a quantum mechanism capable of performing a hierarchical search of depth level 1. As a consequence it becomes possible to, given a given board configuration b and a single production, verify

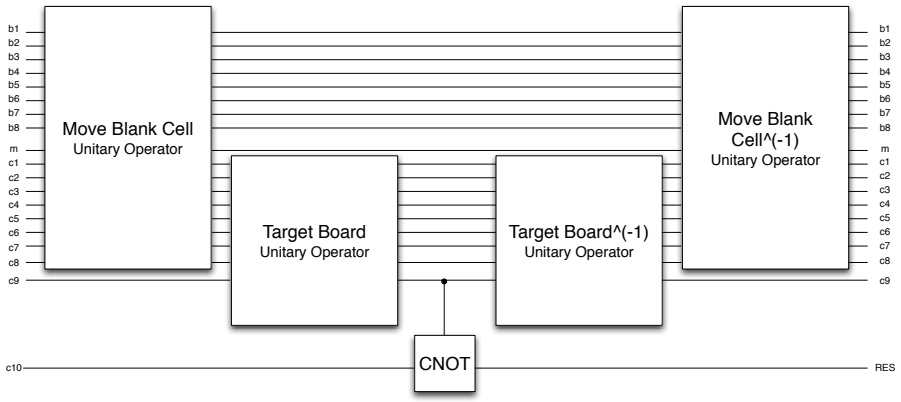


Figure 13: A reversible circuit showcasing the application of a single movement gate for the 3-puzzle whilst incorporating the principles of an oracle.

if a target board configuration is obtained. This behaviour is equivalent in function to that of a classical production system. However, there is a crucial difference since with our quantum proposition we are able to query all possible combinations simultaneously, and in the process obtain a solution quadratically faster.

5 Conclusions

In this work we presented a possible model for a quantum production system capable of solving instances of the n-puzzle. The proposed model can be viewed as an hybrid proposition combining a quantum search mechanism alongside production system theory adjusted so as to convey a clear emphasis on reversible computation. Developing the adequate reversible circuit requires a modest overhead on the number of bits employed relative to classical irreversible versions. These concepts, combined with the quantum superposition principle, can be used in order to query the search space spanned by all possible combinations of initial board configurations and paths up to depth-level d quadratically faster than its classical counterparts. Classical search strategies require $O(b^d)$ time, while a hierarchical search mapping to quantum computation employing Grover's algorithm allows this time to be reduced to $O(\sqrt{b^d})$. From a practical point-of-view such an improvement translates as cutting search depth in half.

Furthermore, the proposed model hints at the possible existence of a universal quantum production system. Hence, it would be interesting to determine if such a system exists and study the consequent implications. Namely, questions regarding the performance of the system and how well it would compare against its

classical counterpart would be relevant. In addition, the dynamics of production system theory make it well suited for exploitation by classical learning mechanisms. This makes it plausible for the unitary operator, and consequently the reversible circuit associated, to be developed by such mechanisms. Additional issues may focus on determining appropriate choices for search-depth d and if a related technique is used during problem solving by human cognition.

6 Acknowledgments

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

References

[1] Busemeyer JR, Wang Z, Townsend JT. Quantum dynamics of human decision-making. Journal of Mathematical Psychology. 2006;50(3):220 – 241. Jean-Claude Falmagne: Part II.

[2] Busemeyer JR, Trueblood J. Comparison of Quantum and Bayesian Inference Models. In: Bruza P, Sofge D, Lawless W, van Rijsbergen K, Klusch M, editors. Quantum Interaction. vol. 5494 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg; 2009. p. 29–43. 10.1007/978-3-642-00834-4-5.

[3] Busemeyer JR, Wang Z, Lambert-Mogiliansky A. Empirical comparison of Markov and quantum models of decision making. Journal of Mathematical Psychology. 2009;53(5):423 – 433. Special Issue: Quantum Cognition.

[4] Pothos EM, Busemeyer JR. A quantum probability explanation for violations of 'rational' decision theory. Proceedings of the Royal Society B: Biological Sciences. 2009;.

[5] Trueblood J, Busemeyer JR. A Comparison of the Belief-Adjustment Model and the Quantum Inference Model as Explanations of Order Effects in Human Inference. In: COGSCI 2010 The annual meeting of the cognitive science society; 2010. p. 1166–1171.

[6] Luger GF, Stubblefield WA. Artificial Intelligence: Structures and Strategies for Complex Problem Solving: Second Edition. The Benjamin/Cummings Publishing Company, Inc; 1993.

[7] Campbell M, Hoane Jr AJ, Hsu Fh. Deep Blue. Artificial Intelligence. 2002;134:57–83.

[8] Manin YI. Classical computing, quantum computing, and Shor's factoring algorithm. ArXiv Quantum Physics e-prints. 1999 Mar;.

[9] Ying M. Quantum computation, quantum theory and AI. Artificial Intelligence. 2010;174:162–176.

[10] Post E. Formal reductions of the general combinatorial problem. American Journal of Mathematics. 1943;65:197–268.

[11] Brownston L, Farell R, Kant E, Martin N. Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Addison-Wesley; 1985.

[12] Newell A, Shaw JC, Simon HA. Report on a general problem-solving program. In: Proceedings of the International Conference on Information Processing; 1959. p. 256–264.

[13] Newell A. A guide to the general problem-solver program GPS-2-2. Santa Monica, CA, USA: RAND Corporation; 1963. RM-3337-PR.

[14] Ernst GW, Newell A. GPS: a case study in generality and problem solving. Academic Press; 1969.

[15] Newell A, Simon HA. Human problem solving. 1st ed. Prentice Hall; 1972.

[16] Anderson JR. The Architecture of Cognition. Cambridge, Massachusetts, USA: Harvard University Press; 1983.

[17] Laird JE, Rosenbloom PS, Newell A. Chunking in Soar: The anatomy of a general learning mechanism. Machine Learning. 1986 03;1(1):11–46.

[18] Laird JE, Newell A, Rosenbloom PS. SOAR: An architecture for general intelligence. Artificial Intelligence. 1987;33(1):1–64.

[19] Markov A. A theory of algorithms. USSR: National Academy of Sciences; 1954.

[20] Turing AM. On Computable Numbers, with an Aoolication to the Entscheidungsproblem. ProcLondonMathSoc. 1936;42(2):230–265.

[21] Newell A, Simon HA. Human Problem Solving. Prentice-Hall; 1972.

[22] Anderson JR. The Architecture of Cognition. Harvard University Press; 1983.

[23] Klahr P, Waterman DA. Expert Systems: Techniques, Tools and Applications. Addison-Wesley; 1986.

[24] Newell A. Unified Theories of Cognition. Harvard University Press; 1990.

[25] Hart TP, Edwards DJ. The tree prune (TP) algorithm. Artificial Intelligence project memo 30. Cambridge, Massachusetts: Massschusetts Institute of Technology; 1961.

[26] Shor PW. Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science; 1994. p. 124–134.

[27] Grover LK. A fast quantum mechanical algorithm for database search. In: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. New York, NY, USA: ACM; 1996. p. 212–219.

[28] Hogg T, Huberman BA, Williams CP. Phase transitions and the search problem. Artif Intell. 1996;81(1-2):1–15.

[29] Hogg T. Quantum computing and phase transitions in combinatorial search. J Artif Int Res. 1996;4(1):91–128.

[30] Hogg T. A Framework for Structured Quantum Search. PHYSICA D. 1998;120:102.

[31] Hogg T. Quantum search heuristics. Phys Rev A. 2000 Apr;61(5):052311.

[32] Feynman RP. Simulating Physics with Computers. International Journal of Theoretical Physics. 1982;21(6):467–488.

[33] Feynman RP. Quantum mechanical computers. Foundations of Physics. 1986;16(6):507–531.

[34] Deutsch D. Quantum theory, the Church-Turing principle and the universal quantum computer. In: Proceedings of the Royal Society of London- Series A, Mathematical and Physical Sciences. vol. 400; 1985. p. 97–117.

[35] Deutsch D. Quantum Computational Networks. In: Proceedings of the Royal Society of London A. vol. 425; 1989. p. 73–90.

[36] Bennett CH. Logical Reversibility of Computation. IBM Journal of Research and Development. 1973 November;17:525–532.

[37] Toffoli T. Computation and construction universality of reversible cellular automata. Journal of Computer and System Sciences. 1977;15(2):213–231.

[38] Toffoli T. Reversible Computing. In: Proceedings of the 7th Colloquium on Automata, Languages and Programming. London, UK: Springer-Verlag; 1980. p. 632–644.

[39] Fredkin E, Toffoli T. Conservative logic. International Journal of Theoretical Physics. 1982;21:219–253.

[40] Hirvensalo M. Quantum Computing. Berlin Heidelberg: Springer-Verlag; 2004.

[41] Mano MM, Kime CR. Logic and Computer Design Fundamentals: 2nd Edition. Prentice Hall; 2002.

[42] Toffoli T. Reversible Computing. Massschusetts Institute of Technology, Laboratory for Computer Science; 1980.

[43] Kaye PR, Laflamme R, Mosca M. An Introduction to Quantum Computing. USA: Oxford University Press; 2007.

[44] Dirac PAM. A New Notation for Quantum Mechanics. In: Proceedings of the Cambridge Philosophical Society. vol. 35; 1939. p. 416–418.

[45] Dirac PAM. The Principles of Quantum Mechanics - Volume 27 of International series of monographs on physics (Oxford, England) Oxford science publications. Oxford University Press; 1981.

[46] Hordern E. Sliding Piece Puzzles. Recreations in Mathematics, No 4. Oxford University Press, USA; 1987.

[47] Gardner M. The hypnotic fascination of sliding-block puzzles. Scientific American. 1964;210:122–130.

[48] Hearn RA, Demaine ED. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. Theoretical Computer Science. 2005;343(1-2):72 – 96. Game Theory Meets Theoretical Computer Science.

[49] Hearn RA. The Complexity of Sliding-Block Puzzles and Plank Puzzles. In: Tribute to a mathemagician. A K Peters; 2005. p. 1–11.

[50] Tarrataca L, Wichert A. Tree search and quantum computation. Quantum Information Processing. 2010;p. 1–26. 10.1007/s11128-010-0212-z. Available from: `http://dx.doi.org/10.1007/s11128-010-0212-z`.

[51] Grover LK. A framework for fast quantum mechanical algorithms. In: STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing. New York, NY, USA: ACM; 1998. p. 53–62.

[52] Chuang IL, Gershenfeld N, Kubinec M. Experimental Implementation of Fast Quantum Searching. Phys Rev Lett. 1998 Apr;80(15):3408–3411.

[53] Edmonds J. Paths, Trees, and Flowers. Canadian Journal of Mathematics. 1965;17:449–467.

[54] Nielsen MA, Chuang IL. Quantum Computation and Quantum Information. Cambridge University Press; 2000.

[55] Grover LK. Trade-offs in the quantum search algorithm. Phys Rev A. 2002 Nov;66(5):052314.

[56] Grover LK. Fixed-Point Quantum Search. Phys Rev Lett. 2005 Oct;95(15):150501.

[57] Grover LK. Quantum Computers Can Search Rapidly by Using Almost Any Transformation. Phys Rev Lett. 1998 May;80(19):4329–4332.

[58] Grover LK. Quantum Search on Structured Problems. Chaos, Solitons & Fractals. 1999;10(10):1695 – 1705.