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

A Quantum Production Model

Luís Tarrataca · Andreas Wichert.

Received: date / Accepted: date

Abstract The production system is a theoretical model of computation relevant to the artificial intelligence field allowing for problem solving procedures such as hierarchical tree search. In this work we explore some of the connections between artificial intelligence and quantum computation by presenting a model for a quantum production system. Our approach focuses on initially developing a model for a reversible production system which is a simple mapping of Bennett's reversible Turing machine. We then expand on this result in order to accommodate for the requirements of quantum computation. We present the details of how our proposition can be used alongside Grover's algorithm in order to yield a speedup comparatively to its classical counterpart. We discuss the requirements associated with such a speedup and how it compares against a similar quantum hierarchical search approach.

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

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

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

Keywords quantum computation, production system, artificial intelligence, tree search

Mathematics Subject Classification (2000) 81P68 · 68Q05 · 68Q10 · 68Q12

1 Introduction

The artificial intelligence community has since its inception focused on developing algorithmic procedures capable of modeling problem solving behaviour. Typically, this process requires the ability to translate into abstract terms environmental concepts and the set of appropriate actions that act upon them. This type of knowledge enables problem-solving agents to consider the environment and the sequence of actions allowing for a given goal state to be reached. This process is also commonly referred to as reasoning [25]. The production system is a formalism for describing the theory of computation. The initial set of ideas for the production system is due to the influential work of Emil Post [32]. Production system theory describes how to form a sequence of actions leading to a desired state. Production system theory also presents a computational theory of how humans solve problems [3]. Some of the best known examples of human cognition-based production systems include the General Problem Solver [29] [28] [17] [30], ACT [2] and SOAR [24] [23]. Recently, applications of quantum computation in artificial intelligence were examined in [41].

1.1 Production systems

A production system is composed of condition-action pairs, i.e. if-then rules, which are also called productions. A computation is performed with the aid of productions through the transformation of an initial state into a desired state. The state description at any given time is also referred to as working memory. A rule is applied when the conditional part is recognized to be part of a given state. The action describes the respective problem-solving behaviour. Applying an action results in the state of the problem instance changing accordingly. On each cycle of operation, productions are matched against the working memory of facts. At any given point, more than one production might be deemed to be applicable. This subset of productions represents the conflict set. A conflict resolution strategy is then employed to this subset in order to determine an appropriate production. Finally, the action of the selected rule is carried out, changing the state of the problem instance. The operational cycle is brought to a close when a goal state is reached or when no more rules can be triggered. This general architecture is illustrated in Figure 1.

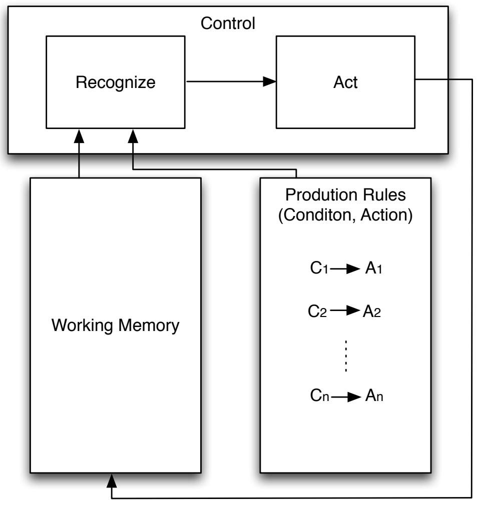

Fig. 1: General architecture for a production system (adapted from [25]).

1.2 On the power of production systems

The first half of the twentieth century saw the beginning of the first efforts to describe intelligence in computational terms. Not surprisingly, some of the first attempts focused on developing abstract models of computation and understanding their computational limits. Some of the best known models include the Universal Turing Machine [38] [39], Post's production system [32], followed closely by the thematically related Markov algorithms [27] and finally Church's lambda-calculus [10]. These computational formalisms were later shown to be equivalent in power [1] [11] [12]. This power equivalency translated into an ability by all models to compute the same set of functions. Notice that this is equivalent to stating that production systems are comparable in power to a Turing machine.

1.3 Objectives and Problems

In this work we propose an alternative model of quantum computation based on production system theory with a clear emphasis on problem-solving behaviour. Traditional approaches such as the quantum Turing machine are complex mechanisms oriented towards general purpose computation. A quantum production system model would be more suited to typical artificial intelligence tasks such as reasoning, inference and hierarchical search. From the onset it is possible to immediately pose some questions, namely: How should such a quantum production system model be developed? What are the requirements of quantum computation and its respective impact on the aforementioned model? How to develop the associated unitary operator? Additionally, what are the performance gains from employing quantum mechanics and how does such a proposition compare against similar strategies? Finally, are there any requirements that should be observed for those improvements? By employing such an approach we are able to (1) provide a detailed explanation of how to develop a quantum production system model; (2) assess the main differences between our proposition and its classical analog; and (3) provide an insight into better describing the power of quantum computation. However, it is not our intention to present an exact characterization of quantum computational models. Answering this question would have far-reaching consequences on complexity theory which are beyond the scope of this work.

The following sections are organized as follows: Section 2 presents the formal definitions for our proposition of a quantum production system; Section 3 presents an assessment comparing the performance of classical production systems against our quantum proposition. We present the concluding remarks of our work in Section 4.

2 Formal Definitions

In this section we present a modular approach of our quantum production system proposition. Accordingly, we choose to start by introducing the set of definitions incorporating traditional production system behaviour in Section 2.1. We then build on these notions to discuss reversibility requirements associated with quantum computation in Section 2.2. These concepts are then employed to enumerate the characteristics of a probabilistic production system in Section 2.3. The probabilistic model will serve as a basis for our quantum production system which will extend those concepts in Section 2.4.

2.1 Classical Production System

Any approach to a general quantum production system model needs to incorporate powerful computational abstractions, which are not bounded by input length, in a similar manner to the classical Turing machine [38] and its quantum counterparts. Accordingly, we choose to present the following definitions through set theory. As previously discussed each production system S consists of a set of production rules R and a control system C alongside a working memory W. The following definitions embody the production system behaviour discussed in Section 1.1.

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

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

Definition 3: The set of production rules R has the form presented in Expression 1.

$$\{(precondition, action) | precondition, action \in \Gamma\} \tag{1}$$

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

Definition 4: The formal definition of a production system S is a tuple (Γ, S_i, S_g, R, C) where Γ, R are finite nonempty sets and $S_i, S_g \subset \Gamma^*$ are, respectively, the set of initial and goal states. The control function C satisfies Expression 2.

$$C : \Gamma \rightarrow R \times \Gamma \times \{h, c\} \tag{2}$$

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

2.2 Reversible Requirements

In quantum computation, discrete state evolution of a closed system is achieved through mathematical maps known as unitary operators [31]. These maps correspond to injective and surjective functions, i.e. bijections. Bijections guarantee that every element of the codomain is mapped by exactly one element of the domain [7]. From a computational perspective, the bijection requirement can be obtained by employing reversible computation. Classical computation is an irreversible process since at its core the use of many-to-one binary gates makes it impossible to ensure a one-to-one and onto mapping. A computation is said to be reversible if given the outputs we can uniquely recover the inputs [36] [37]. Irreversible computational processes can be made reversible by (1) substituting irreversible logic elements by the adequate reversible equivalents; or (2) by accounting for the information that is traditionally lost.

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

It is possible to adapt Bennett's original set of definitions [4] in order to describe the behaviour of a production system by a finite set of transition formulas also referred to as quadruples, in an allusion to the form of Expression 2. Each quadruple maps the present state of the working memory to its successor. By introducing the tuple terminology it becomes simpler to present the following set of definitions:

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

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

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

These definitions contrast with Bennett's more elaborated model where information regarding the internal states of the control unit before and after the transition, alongside tape movement with the associated reading and writing information is maintained. In order to fully understand the exact impact of such requirements lets proceed by considering a production system responsible for sorting strings composed of letters *a*, *b*, *c*, *d*, and *e* based on [40]. The set of production rules is presented in Table 1. Whenever a substring of the original string matches a rule's condition the production is applicable. Applying a specific rule consists in replacing the original substring, *i.e.* precondition, by the action string. The sequence of rules that is applied when the working memory is initialized in state "edcba" is illustrated in Table 2, with the computation proceeding until the string is fully sorted.

Bennett [4] points to the fact that any irreversible computation can be made reversible by saving all the information that is typically erased. However, this reversible history needs to be saved into a resource. Reusing this resource would require the information to be erased or thrown away, merely postponing the problem. The solution relies on performing a computation, saving the intermediate information that is typically lost, and then using this information to backtrack to the original input. Since both forward and backward stages are done in a reversible manner, the overall process always preserves the original information.

However, before undoing the computation, care has to be taken in order to ensure that the output is preserved. This requires copying the output to an output register, an operation which has to be performed reversibly. Once the output copy has been completed it is possible to proceed with the backward stage, *i.e* reverse the consequences of each quadruple application. Eventually, the computation terminates, the production system returns to its original state and the result of the procedure is stored in the output medium. In Bennett's original work the reversible Turing machine is composed of three tapes, namely [4]:

– working tape - where the program's input is initially stored and computation is performed in order to obtain an output which is later reversed to the original input;

– history tape - where the information that is traditionally thrown away is kept, once the program's output has been copied the history information is used in order to revert the working tape to its original state;

– output tape - where the program's output is stored.

By observing Table 2 it is possible to see that in order to ensure that the original input is obtained, the sequence of rules leading from an initial state γ_i to a goal state γ_g needs to be accounted for. This sequence of rules can be used in order to "undo" each action. In doing so it is possible to obtain each precondition that led to a particular action being applied, up until an initial state $\gamma_i \in S_i$. Notice that the quadruples presented in Expression 2 effectively convey information about which production is applied when going from a certain condition to the appropriate action. Additionally, in production system theory there exists a strong emphasis on the sequence of rules leading up to a target state. This situation contrasts with the traditional interest of merely knowing the final state of the working memory. If we allow ourselves to change Bennett's original definitions of the reversible Turing machine then it becomes possible to obtain a mapping for a reversible production system. This process can be performed by requiring that

1. applying a production results in its addition to the history tape, instead of a new control-unit state. Since the quadruple and production rules are equivalent concepts we are basically storing the same transitional information employed by Bennett's model;

2. once the computation halts it is necessary to copy the contents of the history tape to the output tape, this contrasts with the original copying of the work-

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

Table 1: Rule set for sorting a string composed of letters a, b, c, d, and e (adapted from [25]).

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

Table 2: An example of the sequence of rules applied for sorting a string composed of letters a, b, c, d, and e.

ing tape. In order to do so the history tape's head needs to be place at the tape's beginning. Afterwards, the copy process from the history tape to the output tape can proceed.

3. upon the copying mechanism's conclusion, the output tape's head needs to be placed at the beginning. This process can be performed by shifting left the output tape until a blank symbol is found.

Table 3 illustrates this set of ideas for a reversible production simple based on the string sorting production system presented earlier (Table 1 and Table 2). As it is possible to verify the computation proceeds normally for iteration 0 through 10, also known as the forward computation stage. The only alteration to Bennett's model consists in adding the productions fired to the history tape. Once this stage has concluded the history tape's head needs to be properly placed at the beginning. This step is carried out in iteration 11. In this case we opted to represent the position of a tape's head by an underbar. The system then proceeds in iteration 12 by copying the contents of the history tape onto the output tape. Additionally, the output tape's head is placed at the beginning in iteration 13. The last stage of the computation consists in undoing each one of the applied productions, as illustrated from iteration 14 to 24. For this stage we opted to represent the inverse of a rule R mapping a precondition A into an action B, *i.e.* $R : A \to B$, by R^{-1} such that $R^{-1} : B \to A$. By inverting the rules applied we are for all purposes reversing the consequences of each associated quadruple.

2.3 Probabilistic Production System

Consider a production system whose control strategy chooses a rule to apply from set of production rules based on a probability distribution. This behaviour can be formalized with a simple reformulation of Expression 2 as illustrated by Expression 3, where $C(\gamma, r, \gamma', d)$ represents the probability of choosing rule r, substituting symbol γ with γ' and making a decision d on whether to continue or halt the computation if the memory contains γ.

$$C : \Gamma \times R \times \Gamma \times \{h, c\} - [0, 1] \tag{3}$$

Additionally, it would have to be required that $\forall \gamma \in \Gamma$ Expression 4 be observed

$$\sum_{\forall (r,\gamma',d) \in R \times \Gamma \times \{h,c\}} C(\gamma, r, \gamma', d) = 1 \tag{4}$$

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

Table 3: Operation of a reversible production system based on the example of Table 2 and Bennett's model for a reversible Turing machine. The underbar denotes the position of the head.

This modification to the deterministic production system allows the control strategy to yield different states with probabilities that must sum up to 1. In such a model, a computation can be perceived has having an associated probability which is simply the multiplication of each production's probability. If the several possibilities are accounted for the overall computational process presents a tree form. Figure 2 illustrates a production system whose set of production rules is binary, *i.e.* $\{p_0, p_1\}$. The root node A depicts the initial state in which the working memory is initialized. Each depth layer d is responsible for adding b^d nodes to the tree, where b is the branching factor induced by the production set cardinality. For this specific case $b = 2$. The remaining tree nodes represent states achieved by applying the sequence of productions leading up to that specific element, *e.g.* state J is achieved by applying sequence $\{p_0, p_1, p_0\}$.

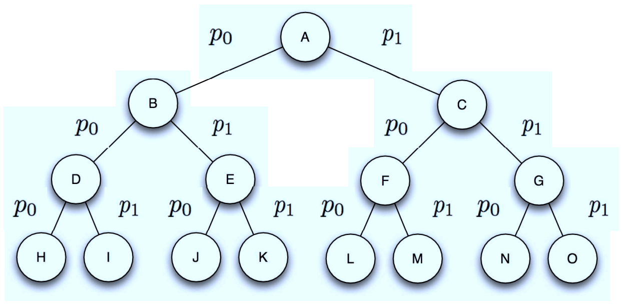

Fig. 2: Tree structure representing the multiple computational paths of a probabilistic production system.

2.4 Quantum Production System

A suitable model for a probabilistic production system enables a mapping between real-valued probabilities and complex-value quantum amplitudes. Specifically, the complex valued control strategy would need to behave as illustrated in Expression 5 where $C(\gamma, r, \gamma', d)$ provides the amplitude if the working memory contains symbol γ then rule r will be chosen, substituting symbol γ with γ' and a decision d made on whether to continue or halt the computation.

$$C : \Gamma \times R \times \Gamma \times \{h, c\} - \mathbb{C} \tag{5}$$

The amplitude value provided would also have to be in accordance with Expression 6, $\forall \gamma \in \Gamma$

$$\sum_{\forall (r,\gamma',d) \in R \times \Gamma \times \{h,c\}} |C(\gamma, r, \gamma', d)|^2 = 1 \tag{6}$$

Is it possible to elaborate on the exact unitary form that C should take? If we were to develop a classical computational gate for calculating Expression 2 then it would have a form as illustrated in Figure 3a. Since multiple arguments could potentially map onto the same element such a strategy would not allow for reversibility. Theoretically, any irreversible production system can be made reversible by adding some auxiliary input bits and through the addition modulo 2 operation [37], a process formalized in Expression 7 and shown in Figure 3b. Since the inputs are now part of the outputs, this mechanism allows for a bijection to be obtained.

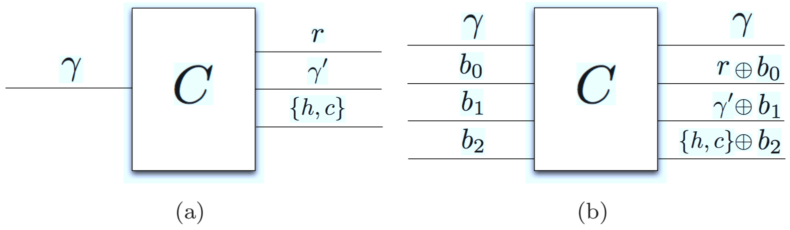

Fig. 3: An irreversible control strategy 3a can be made reversible 3b through the introduction of a number of constants and auxiliary input and output bits.

$$\underbrace{(\gamma, b_0, b_1, b_2)}_{\text{input vector } v_1} \xrightarrow{C} \underbrace{(\gamma, r \oplus b_0, \gamma' \oplus b_1, \{h, c\} \oplus b_2)}_{\text{output vector } v_2} \tag{7}$$

Notice that the reversible gate can be perceived as acting upon an input vector v_1 and delivering v_2. If we adopt a linear algebra perspective alongside the Dirac notation [15] [16], then such behaviour can be described as shown in Expression 8, where C is the required unitary operator.

$$C|v_1\rangle = |v_2\rangle \tag{8}$$

Based on Expression 7 and Expression 8 it becomes possible to develop a unitary operator C. Accordingly, C acts upon an input vector v_1 conveying specific information about the argument's state. From Expression 7 we can verify that any input vector $|v_1\rangle$ should be large enough to accommodate γ, b_0, b_1 and b_2. Since b_0, b_1 and b_2 will be used for bitwise addition modulo 2 operations with, respectively, r, γ' and $\{h, c\}$, we need to determine the appropriate dimensions for a binary encoding of these elements. Assume that:

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

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

A parallel can be established between C's behaviour and the truth table concept of classical gates. Truth tables are classical mechanisms employed to describe logic gates employed in electronics. The tables list all possible combinations of the inputs alongside the respective results [26]. In a similar manner, we can build unitary operator C by going through all possible combinations and decoding the information present in each combination. This procedure is illustrated through pseudo-code in Procedure 1. Lines 1-3 are employed in order to determine the required number of bits for our encoding mechanism. These values can also be used to determine the dimension $2^{\alpha+\beta+\delta} \times 2^{\alpha+\beta+\delta}$ of unitary operator C. This operator is initialized in line 4 as a matrix with all entries set to zero.

The cycle *for* from lines 5-15 is responsible for going through all possible combinations. Line 6 of the code obtains a string S which is the binary version of decimal combination λ, represented as $\lambda_{(2)}$ to illustrate base-2 encoding. Recall from Expression 3 that each input vector needs to convey information about γ, b_0, b_1 and b_2. Accordingly, for each λ we need to parse the different elements of the string in order to determine those values. This process is illustrated through lines 7-10 which are responsible for obtaining the binary substrings. For any string S, $S[i, j]$ is the contiguous substring of S that starts at position i and ends at position j of S [20]. Line 11 is responsible for invoking function mapBinaryEncoding which maps substring S_1 to a symbol $\gamma \in \mathit{\Gamma}$. This function can be easily calculated with the help of any trivial data structure.

Once the input symbol γ has been determined it is possible to calculate the transition depicted in Expression

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

Procedure 1 Pseudo code for building unitary operator C

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

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

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

3 Classical vs. Quantum Comparison

Deutsch described a universal model of computation capable of simulating Turing machines with inherent quantum properties such as quantum parallelism that cannot be found in their classical counterparts [13]. However, the number of computational steps required by Deutsch's model grew exponentially as a function of the simulated Turing's machine running time. Subsequently, a more efficient model for a universal quantum Turing machine was proposed in [6]. In the same work the authors questioned themselves if a quantum turing machine can provide any significant advantage over their classical equivalents. They proceeded by showing that a quantum turing machine described in [14] is capable of efficiently solving the Fourier sampling problem. However, care was also employed in order to emphasize that their result did not prove that quantum Turing machines are more powerful than probabilistic Turing machines, since the latter can sample from a distribution within ϵ total variation distance of the desired Fourier distribution [6]. Later, Shor's algorithm for fast factorization [33] presented further evidence on the power of quantum computation.

Naturally, the question arises: how does our quantum production system proposal fare against its classical counterpart? Namely, what do we stand to gain by applying quantum computation? And what are the requirements associated to those improvements? In order to answer these questions consider a unitary operator

C which is applied to an initial state $x \in S_i$. Additionally, assume that C needs to be applied a total of d times for a result to be obtained, where $d \in \mathbb{N}$ is chosen such that the computation is able to proceed until it stops. The result of applying C can be represented as $g(x)$ which in production system theory can be a simple output of the productions applied. As a consequence, the quantum register employed needs to convey information about the initial state and also be large enough to accommodate for $g(x)$. We opted to represent this requirement by employing a unspecified length register $|z\rangle$. Accordingly, we can represent the initial state of the system by the left-hand side of Expression 9. The right-hand side represents the result obtained after unitary evolution.

$$C^d|x, z\rangle = |x, z \oplus g(x)\rangle \tag{9}$$

In order to gain a quantum advantage over the classical version we need to employ the superposition principle. Accordingly, it is possible to initialize register $|x\rangle$ as a superposition, $|\psi\rangle$, of all starting states, a procedure illustrated in Expression 10, where $S_i \subset \Gamma^*$ is the set of starting states. This procedure is also depicted in Figure 4, where multiple binary searches are performed simultaneously, with the dotted line representing initial nodes that, for reasons of space, are not shown, but are still present in the superposition. Now consider a scenario where the production system definition only contemplates a single initial state, *i.e.* $|S_i| = 1$. Since it is not possible to explore the high levels of parallelism provided by the superposition principle, we would therefore not have any significant advantage over the sequential procedure by applying $|\psi_n\rangle$. However, if the productions set cardinality is greater than one, then there exist several neighbour states which which can be employed as initial states thus circumventing the problem.

$$|\psi\rangle = \frac{1}{\sqrt{|S_i|}} \sum_{s \in S_i} |s\rangle \tag{10}$$

This approach differs from other strategies of hierarchical search, namely [34] and [35], who, respectively, (1) evaluate a superposition of all possible tree paths up to a depth-level d in order to determine if a solution is present and (2) present an hierarchical decomposition of the quantum search space through entanglement detection schemes.

The following sections are organized as follows: Section 3.1 presents the main results on Grover's algorithm. These concepts will then be extended in Section 3.2 in order to present a system combining our production system proposal alongside the quantum search algorithm. Finally, we will conclude in Section 3.3 by discussing the performance gains achieved over the classical production system equivalent.

3.1 The quantum search algorithm

Traditionally, production system theory is applied to problems devoid of an element of structure, and thus requiring the search space of all possible combinations to be exhaustively examined. The class NP consists of those problems whose possible configurations can be verified in polynomial-time. Grover's algorithm works by amplifying the amplitude of the solution states. The algorithm is able to "mark" a state as a solution by employing an oracle O which, alongside an adequate initialization of the answer register in a superposition state, effectively flips the amplitudes of those states. This behaviour is illustrated in Expression 11, where $|x\rangle$ and $|y\rangle$ represent, respectively, an n-bit query register and a single bit answer register. Function $f(x)$ simply verifies if x is a solution, as formalized in Expression 12. The quantum search algorithm [18] is ideally suited for solving NP problems and allows for a quadratic speedup relatively to classical algorithms. Classical algorithms require $O(N)$ time for N-dimensional search spaces, whilst Grover's algorithm requires $O(\sqrt{N})$ time, or in terms of $|x\rangle$'s dimension $O(\sqrt{2^n})$ time.

$$O : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle \tag{11}$$

$$f(x) = \begin{cases} 1 \text{ if } x \text{ is a solution} \\ 0 \text{ otherwise} \end{cases} \tag{12}$$

The amplification process is achieved by flipping the amplitude of the solution states and performing a inversion about the mean of the amplitudes. The overall effect of such a procedure, referred to as Grover's iterate, induces a higher probability of observing a solution when a measurement is performed over the superposition state. Grover's algorithm was experimentally demonstrated in [9]. The quantum search algorithm systematically increases the probability of obtaining a solution with each iteration. Upon conclusion a measurement is performed in a quantum superposition. The superposition state represents the set of all possible results. Grover's approach sparked interest by the

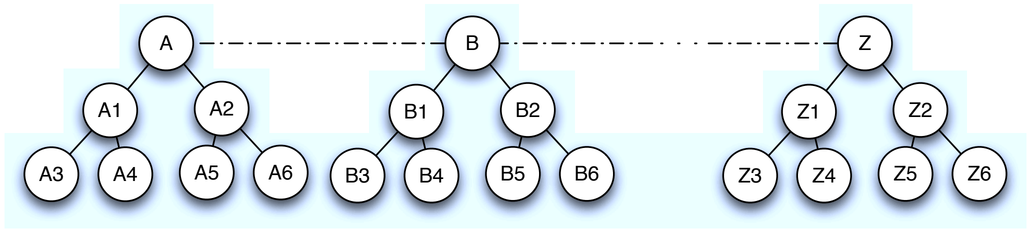

Fig. 4: Parallel search with $S_i = \{A, B, \cdots, Z\}$ and $|\psi_n\rangle = \frac{1}{\sqrt{|S_i|}} \sum_{s \in S_i} |s\rangle$. The dotted lines represent the initial states belonging to superposition $|\psi_n\rangle$

scientific community on whether it would be possible to devise a faster search algorithm. Subsequently, it was proved any procedure based on oracles employing total function evaluation will always require at least $\Omega(\sqrt{N})$ time [5]. Grover and Radhakrishnan [19] considered the speedup achievable if one was only interested in determining the first m bits of a n bit solution string. In practice, their approach proceed with analysing different sections of the quantum search space. The authors prove that it is possible to obtain a speedup, however, as m grows closer to n the computational gains obtained disappear [19]. This speedup was then improved in [21] and [22] and an extension to multiple solutions was presented in [8].

3.2 Oracle Extension

In this section we present an extension to the oracle operator employed by Grover's algorithm allowing it to be combined alongside our quantum production system proposal. As a result we need to determine what happens when two different functions f and g are combined into a single unitary evolution, as illustrated by Expression 13. In this case we opted to employ three quantum registers, namely $|x\rangle$ which is configured with the system's initial state, alongside registers $|y\rangle$ and $|z\rangle$ where, respectively, the output of functions $f(x)$ and $g(x)$ is stored. The original amplitude flipping process is a result of placing register $|y\rangle$ in the superposition state $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Accordingly, we need to verify if the amplitude flip still holds with the oracle formulation of Expression 13 alongside $|y\rangle$'s superposition initialization. This behaviour is shown in Expression 14. From Expression 15 we are able to conclude that despite the new oracle formulation the amplitude flipping continues to occur.

$$O|x, y, z\rangle = |x, y \oplus f(x), z \oplus g(x)\rangle \tag{13}$$

3.3 Performance Analysis

In order to proceed with our performance analysis lets consider we have a production system whose definitions are incorporated into a unitary operator C combining the results of Expression 9 and Expression 13. Accordingly, C will have the form presented in Expression 16, where $|x\rangle$ is initialized with a superposition of the production system starting states. In addition we employ register $|z\rangle$ which has an unspecified length in order to accommodate for the productions applied, *i.e* the output growth of function $g(x)$.

By employing such a formulation for our production system C we are able to employ it alongside Grover's algorithm in order to speedup the computation. In our particular case we are interested in changing $f(x)$'s definition in order to check if a goal state $s \in S_g$ is achieved after having applied d productions. *E.g.* consider that state M shown in Figure 2 is a goal state, then, assuming no backtracking occurs, such state can be reached by applying productions p_1, p_0 and p_1. As a consequence we can express such state evolution as $C^3|A, 0, \underline{\mathbf{0}}\rangle = |x, 1, \{p_1, p_0, p_1\}\rangle$, where $\underline{\mathbf{0}}$ represents a vector of zeros. Function f new definition is presented in Expression 17. The state of the system is described by a unit vector in a Hilbert space $\mathcal{H}_{2^m} = \mathcal{H}_{2^n} \otimes \mathcal{H}_2 \otimes \mathcal{H}_{2^p}$.

$$C^d|x, y, z\rangle = |x, y \oplus f(x), z \oplus g(x)\rangle \tag{16}$$

$$f(x) = \begin{cases} 1 \text{ if } C^d|x\rangle \in S_g \\ 0 \text{ otherwise} \end{cases} \tag{17}$$

Grover's original speedup was dependent on superposition $|\psi\rangle$ and the associated number of possible states. More concretely, the dimension of the space spanned is dependent on the dimension of the query register $|x\rangle$ employed. However, by applying an oracle C whose

$$O|x\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}|z\rangle = \frac{1}{\sqrt{2}}\left(|x\rangle|f(x)\rangle|z\oplus g(x)\rangle - O|x\rangle|1\oplus f(x)\rangle|z\oplus g(x)\rangle\right) \tag{14}$$

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

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

$$= (-1)^{f(x)}|x\rangle\frac{|0\rangle-|1\rangle}{\sqrt{2}}|z\oplus g(x)\rangle \tag{15}$$

behaviour mimics that of Expression 16 the elements present in superposition $|\psi\rangle$ will interact with registers $|y\rangle$ and $|z\rangle$. Typically, register $|y\rangle$ is ignored when evaluating the running time, producing an overall superposition $|\xi\rangle$ which will no longer span the original 2^n possible states but 2^{n+p}. From an algebraic perspective, the interaction process is due to the tensor product employed to describe the overall state between $|x\rangle$, $|y\rangle$ and $|z\rangle$. As a result, it is possible to pose the following question: what can be said about the growth of $|z\rangle$ and its respective impact on overall system performance?

Assume that a solution state can always be found after d computational steps, either by indeed finding a goal state or by applying an heuristic function to determine an appropriate state selection. Classically, a sequential procedure would require $C = |S_i| \times d$ iterations, one for each initial state in need of processing. Is it possible to do any better with our proposition? Answering this question requires determining appropriate boundary conditions on the exact dimensions of $|z\rangle$ for which it is still possible to obtain a speedup over classical procedures.

By employing Grover's algorithm we know that the search procedure will span the dimension of $|\xi\rangle$ which varies between $[2^n, 2^{n+p}]$. Accordingly, in the very unlikely best case scenario, we will be able to search all elements in $O(\sqrt{|S_i|})$ time. With each Grover iterate we need to apply oracle C a total of d times, which implies an overall number of invocations equal to $Q = \sqrt{|S_i|} \times d$. Therefore, a comparison is required between the classical and quantum number of iterations, respectively, C and Q, as illustrated in Expression 18. The ratio presented in Expression 18 allows us to conclude that C and Q differ by a factor of $\sqrt{|S_i|}$, effectively favoring the quantum proposal.

$$\frac{C}{Q} = \frac{|S_i|d}{\sqrt{|S_i|}d} = \sqrt{|S_i|} \tag{18}$$

However, such a ratio does not take into account the dimension of register $|z\rangle$. Therefore, we need to determine what happens when $|z\rangle$ grows and how it affects overall performance. Let m denote the number of bits employed by registers $|x\rangle$ and $|z\rangle$, then the number of quantum iterations will be $Q = \sqrt{2^m} \times k$. Accordingly, Expression 18 can be restated in terms of m, as depicted in Expression 19 which effectively conveys the notion that each additional bit added to $|z\rangle$ impacts the $\frac{C}{Q}$ ratio negatively by a factor of $\frac{1}{\sqrt{2}}$. If register $|z\rangle$ is composed by p bits this means that the overall decrease in performance will be $\frac{p}{\sqrt{2}}\frac{|S_i|}{\sqrt{2^n}}$, where n is the number of bits required to encode the set of initial states. This result can be restated as $\frac{p}{\sqrt{2}}\sqrt{|S_i|}$ if we consider Grover's speedup in light of the dimension of S_i.

$$\frac{C}{Q} = \frac{|S_i|}{\sqrt{2^m}} \tag{19}$$

Additionally, we are also interested in determining when is the number of quantum iterations Q smaller than the number of classical iterations C, as shown in Expression 20.

$$\begin{aligned} & Q && < && C && (20)\\ \Leftrightarrow\ & \sqrt{2^m}k && < && |S_i|k \\ \Leftrightarrow\ & 2^m && < && |S_i|^2 \\ \Leftrightarrow\ & m && < && \log_2|S_i|^2 && (21)\end{aligned}$$

Expression 21 needs to be further refined since we are trying to determine $m \in \mathbb{N}$ but the right-hand side may produce a value belonging to $\mathbb{R}$. This output is a consequence of having to deal with initial state sets S_i whose cardinality is not a power of 2. Notice that the measurement of performance we have chosen, respectively, the ratio C/Q will eventually be 1 when $m = \log_2|S_i|^2$. Accordingly, if a larger number of bits is employed it effectively yields $C/Q < 1$ which will no longer translate into a speedup by the quantum version. That being the case, we choose to restrict our model to those cases where $m < \lfloor \log_2 |S_i|^2 \rfloor$. Furthermore, m should also be large enough to contain the set of possible binary encodings of S_i, *i.e.* $m \geq \lceil \log_2 |S_i| \rceil$. The general boundary conditions are presented in Expression 22.

$$\lceil \log_2 |S_i| \rceil \leq m \leq \lfloor \log_2 |S_i|^2 \rfloor \tag{22}$$

Figure 5 illustrates the three-dimensional plot of Expression 19 as a function of a number of initial nodes in the range $[1, 2^{13}]$ alongside the required boundary conditions described by Expression 22. The plot presents the characteristic ladder effect associated with employing logarithmic functions in conjunction with functions that map real domains to the integer set. As a consequence, a plateau is reached for some combinations where a number of different cardinality S_i sets can be mapped by the same number of bits, thus presenting the same C/Q ratio. Relinquishing the floor and ceiling functions allows us to obtain a crude comparison between the lower and upper limits of Expression 22. More concretely, we are able to verify that these limits differ by a $\log_2 |S_i|$ factor. This means that the system, besides requiring $\lceil \log_2 |S_i| \rceil$ bits for register $|x\rangle$, can still employ an additional $\log_2 |S_i|$ bits to encode g's output and in the process still perform better than its classical counterpart.

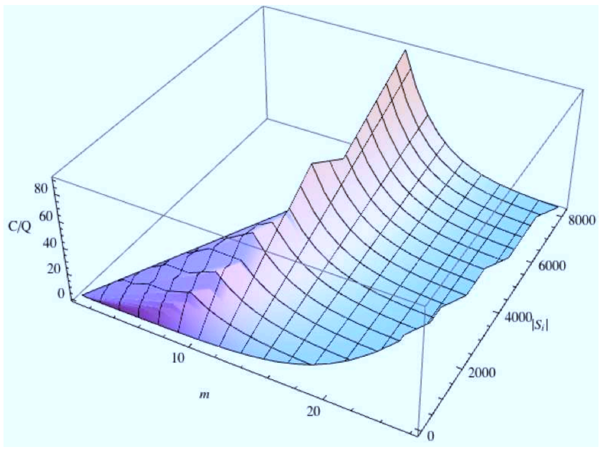

Fig. 5: The performance measurement ratio C/Q for $|S_i| \in [1, 2^{13}]$ illustrating the logarithmic growth $\lceil \log_2 |S_i| \rceil \leq m \leq \lfloor \log_2 |S_i|^2 \rfloor$ alongside the associated $\frac{p}{\sqrt{2}}\sqrt{|S_i|}$ decrease in performance.

3.3.1 On the growth of g's output

Consider a production system with a constant branching factor where a set of productions is applied then there will exist a total of $|R|^d$ possible tree paths at depth level d, who will require $\lceil \log_2 |R|^d \rceil$ bits for an adequate encoding. Clearly, if $\lceil \log_2 |R|^d \rceil \leq \lceil \log_2 |S_i| \rceil$ then g's output can encode the sequence of productions applied. If this is not the case we may opt to encode an unspecified number of productions applied according to some previously chosen strategy. As a consequence, our proposal may be more appropriate when dealing with large S_i sets since this would automatically imply that we would have at our disposition a large set of working bits. Even if this is not the case it is still possible to employ as initial states the set of nodes that can be found at a depth d which, as previously mentioned, typically grow in an exponential fashion.

3.3.2 Comparison with an hierarchical search model

Finally, it is important to compare our quantum production system performance against a similar proposal described in [34]. In their work the authors also employ Grover's algorithm alongside an hierarchical search oracle where a superposition consisting of all possible paths up to depth-level d is evaluated. The authors chose to build a binary string encoding in a logical fashion the sequence of actions taken. As a consequence a total of $p_1 = d \times \lceil log_2 |R| \rceil$ bits is required. This approach contrasts with the $p_2 = \lceil \log_2 |R|^d \rceil$ bits employed to encode all possible paths. However, if $|R|$ is a power of 2, then the use of the ceiling functions is no longer required and it is possible to conclude that $p_1 = p_2 = p$. For that reason, the number of bits m required by our current proposition will always be bigger than p, since $m = n + p + 1$. This implies that the number of Grover iterations to apply in [34], respectively, $O(\sqrt{2^p})$ will also be less than the $O(\sqrt{2^{n+p+1}})$ time required with this method. If a ratio is performed between both procedures then we are able to verify that they differ by a factor of $\frac{\sqrt{2^p}}{\sqrt{2^{n+p+1}}} = \sqrt{\frac{1}{2^{n+1}}}$ in favor of [34]. However, despite loosing in performance terms, our model differs significantly in nature. The authors original proposal focused on exploiting hierarchical search through polynomial time verification of paths, whilst we propose building a generic mechanism for hierarchical search through quantum operators.

4 Conclusions

In this work we presented an artificial intelligence inspired quantum computational model based on production system theory. Quantum computation is an inherently reversible process and as a consequence the proposed model would also allow for a reversible decision process. Since production systems share some key characteristics with classical tree search the proposed model also allows for an hierarchical quantum search mechanism. By formalizing the theoretical foundations of our approach we were able to enumerate the reversible and quantum requirements of our model. These requirements enabled us to present a method focusing on the construction of the unitary operator associated with our quantum production system. We then extended our proposition in order to combine with Grover's algorithm. Doing so allowed us to adequately study the performance of our system and enumerate those cases in which our model outperforms its classical counterpart. Although our proposition is able to compute faster the $\frac{p}{\sqrt{2}}\sqrt{|S_i|}$ performance penalty associated with each additional bit required is expensive, favoring the choice of models that rely exclusively on exploiting the class of problems NP through polynomial time verifications and superpositions of all possible paths.

Acknowledgements

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

References

1. Abramsky, S., S., A., Shore, R., Troelstra, A.: Handbook of computability theory. Elsevier (1999)
2. Anderson, J.R.: The Architecture of Cognition. Harvard University Press, Cambridge, Massachusetts, USA (1983)
3. Anderson, J.R.: Cognitive Psychology and its Implications, fourth edn. W. H. Freeman and Company (1995)
4. Bennett, C.: Logical reversibility of computation. IBM Journal of Research and Development **17**, 525–532 (1973)
5. Bennett, C.H., Bernstein, E., Brassard, G., Vazirani, U.: Strengths and weaknesses of quantum computing (1997). URL `http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/9701001`
6. Bernstein, E., Vazirani, U.: Quantum complexity theory. In: STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pp. 11–20. ACM, New York, NY, USA (1993). DOI http://doi.acm.org/10.1145/167088.167097
7. Bourbaki, N.: Elements of mathematics: theory of sets. No. vol. 1 in Elements of mathematics. Springer (2004). URL `http://books.google.pt/books?id=IL-SI67hjI4C`
8. Choi, B., Korepin, V.: Quantum Partial Search of a Database with Several Target Items. ArXiv Quantum Physics e-prints (2006)
9. Chuang, I.L., Gershenfeld, N., Kubinec, M.: Experimental implementation of fast quantum searching. Phys. Rev. Lett. **80**(15), 3408–3411 (1998). DOI 10.1103/PhysRevLett.80.3408
10. Church, A.: The Calculi of Lambda-Conversion. Annals of Mathematics Studies. Princeton University Press (1941)
11. Davis, M.: The Universal Computer: The Road from Leibniz to Turing. Norton (2000)
12. Davis, M.: Engines of logic: mathematicians and the origin of the computer. Norton (2001)
13. 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, pp. 97–117 (1985)
14. Deutsch, D., Jozsa, R.: Rapid Solution of Problems by Quantum Computation. Royal Society of London Proceedings Series A **439**, 553–558 (1992)
15. Dirac, P.A.M.: A new notation for quantum mechanics. In: Proceedings of the Cambridge Philosophical Society, vol. 35, pp. 416–418 (1939)
16. Dirac, P.A.M.: The Principles of Quantum Mechanics - Volume 27 of International series of monographs on physics (Oxford, England) Oxford science publications. Oxford University Press (1981)
17. Ernst, G., Newell, A.: GPS: a case study in generality and problem solving. Academic Press (1969)
18. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 212–219. ACM, New York, NY, USA (1996). DOI http://doi.acm.org/10.1145/237814.237866
19. Grover, L.K., Radhakrishnan, J.: Is partial quantum search of a database any easier? (2004). URL `http://www.citebase.org/abstract?id=oai:arXiv.org:quant-ph/0407122`
20. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press (1997). URL `http://books.google.pt/books?id=STGlsyqtjYMC`
21. Korepin, V., Grover, L.: Simple algorithm for partial quantum search. Quantum Information Processing **5**, 5–10 (2006). URL `http://dx.doi.org/10.1007/s11128-005-0004-z`. 10.1007/s11128-005-0004-z
22. Korepin, V.E., Xu, Y.: Hierarchical Quantum Search. International Journal of Modern Physics B **21**, 5187–5205 (2007). DOI 10.1142/S0217979207038344
23. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial Intelligence **33**(1), 1–64 (1987)
24. Laird, J.E., Rosenbloom, P.S., Newell, A.: Chunking in soar: The anatomy of a general learning mechanism. Machine Learning **1**(1), 11–46 (1986)
25. Luger, G.F., Stubblefield, W.A.: Artificial Intelligence: Structures and Strategies for Complex Problem Solving: Second Edition. The Benjamin/Cummings Publishing Company, Inc (1993)
26. Mano, M., Kime, C.R.: Logic and Computer Design Fundamentals: 2nd Edition. Prentice Hall (2002)
27. Markov, A.: The theory of algorithms. National Academy of Sciences, USSR (1954)

28. Newell, A.: A guide to the general problem-solver program gps-2-2. Tech. Rep. RM-3337-PR, RAND Corporation, Santa Monica, CA, USA (1963)
29. Newell, A., Shaw, J., Simon, H.A.: Report on a general problem-solving program. In: Proceedings of the International Conference on Information Processing, pp. 256–264 (1959)
30. Newell, A., Simon, H.A.: Human problem solving, 1 edn. Prentice Hall (1972)
31. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2000)
32. Post, E.: Formal reductions of the general combinatorial problem. American Journal of Mathematics **65**, 197–268 (1943)
33. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994). DOI 10.1109/SFCS.1994.365700
34. Tarrataca, L., Wichert, A.: Tree search and quantum computation. Quantum Information Processing pp. 1–26 (2010). URL `http://dx.doi.org/10.1007/s11128-010-0212-z`. 10.1007/s11128-010-0212-z
35. Tarrataca, L., Wichert, A.: Can quantum entanglement detection schemes improve search? Quantum Information Processing pp. 1–8 (2011). URL `http://dx.doi.org/10.1007/s11128-011-0231-4`. 10.1007/s11128-011-0231-4
36. Toffoli, T.: Reversible computing. In: Proceedings of the 7th Colloquium on Automata, Languages and Programming, pp. 632–644. Springer-Verlag, London, UK (1980)
37. Toffoli, T.: Reversible computing. Tech. rep., Massschusetts Institute of Technology, Laboratory for Computer Science (1980)
38. Turing, A.: On computable numbers, with an application to the entscheidungsproblem. In: Proceedings of the London Mathematical Society, vol. 2, pp. 260–265 (1936)
39. Turing, A.: Computing machinery and intelligence. Mind **59**, 433–460 (1950)
40. Winston, P.H.: Artificial Intelligence (Third Edition). Addison-Wesley (1992)
41. Ying, M.: Quantum computation, quantum theory and ai. Artificial Intelligence **174**, 162–176 (2010)