The Problem of Robot Random Motion Tracking Learning and Quantum Algorithms

Carlos Rodríguez Lucatero

Universidad Iberoamericana Campus Ciudad de México,
Dirección de Análisis e Información Académica,
Prolongación Paseo de la Reforma 880 Lomas de Santa fe
Ciudad de México, México, 01210
carlosr.lucatero@uia.mx, profesor.lucatero@gmail.com

Abstract. The paper studies the problem of tracking a target robot that moves following a random walk strategy, by constructing in the observer robot a model of the behaviour of the target. The strategy of the target robot is supposed to be a random generator of movements. We make the assumption that the robot motion strategies can be modelled as uniform random generator of movements. We suppose that the observations are noise free. We will explore the hardness of the problem of trying to predict the numbers generated by a uniform random generator and relate this problem with our motion tracking problem. We explore too the use of quantum algorithms as a way to deal with some complexity hardness problems that arise when we try to predict the movements of a target robot that moves randomly.

1. Introduction

In a previous article [1] we have talked about the learning algorithms of robot motion tracking problem under the assumption that the strategy followed by the *target robot* as well as the *observer robot* was a DFA (DETERMINISTIC FINITE AUTOMATA). In this article, the agents observed the actions taken by the other agents, and try to predict the behaviour of them and react in the best possible way by means of the construction of a model of the behaviour of the *target* that was obtained by the automata learning algorithm obtained in this work. This was possible in a computationally tractable way because the DFA obtained was not minimal in the number of states. The fundamental assumption about the computational power of the agents was that they were limited in their computational power. Because of that we proposed in [2] to state the robot tracking problem as a repeated game. The solution obtained in [2] outperform the solutions proposed in [4] [5] and [7] given that our assumptions about the behaviour of the implicated agents were more general than the evading strategy of the *target* in the articles cited just before. In the seminal paper on complexity and bounded rationality written by Christos H. Papadimitriou and M. Yannakakis [17] it was analyzed the complexity of calculating the Nash equilibrium (the best strategy for the two players) of a two player game in the case of agents with limited rationality. In [17] Papadimitriou studied the Nash equilibrium of classical game theory prisioner's dilemma and observed that there is paradox on the non-

© G. Sidorov, B. Cruz, M. Martínez, S. Torres. (Eds.)
Advances in Computer Science and Engineering.
Research in Computing Science 34, 2008, pp. 209-221

Received 29/03/08
Accepted 26/04/08
Final version 30/04/08

cooperative Nash equilibrium strategies of the players that is opposed to the social experience of cooperative behaviour showed by Axelrod in [18]. Papadimitriou argues that a good measure of the limited computational power (limited rationality) can be the number of states of the automata executed by each player. Depending on that the players can tend to play cooperatively (cooperative Nash equilibrium). One computational complexity obstacle for obtaining efficient learning algorithms is related with the fact of being a passive or an active learner. In the first case it has been shown the impossibility to obtain in the worst case an efficient algorithm [20][21]. In the second case if we permit the learner to make some questions (i.e. to be an active learner) we can obtain efficient learning algorithms [3] . This work done on the DFA (DETERMINISTIC FINITE AUTOMATA) learning area has given place to many excellent articles on learning models of intelligent agents as those elaborated by David Carmel & Shaul Markovitch [23] [24] and in the field of Multi-agent Systems those written about Markov games as a framework for multi-agent reinforcement learning by M.L. Littman [6]. In the present work we will try to approach the same robot motion tracking problem stated in [1] [2] but now the assumption on the behaviour is that the robots follow a random walk strategy. When we talk about a random walk strategy we mean that the robots toss coins for calculating their next move. In the present work we are assuming again that observed moves are not noisy and that the calculation power of each agent is limited. So we have to talk about the computing complexity that we have to face when we want to predict the next move of an agent that has this type of behaviour. For this end we will start our exposition by making some definitions about pseudo-random functions and pseudo-random behaviour. Given that the agents behave randomly we will give a rough description of how to construct random functions as well as how to use them for constructing random strategies. Because of that we have to talk about the complexity challenges implied while one try to predict the next move of a *target robot* that moves randomly and as will be shown later it is a hard to solve problem, said otherwise, it doesn't exist a deterministic polynomial time algorithm for solving it. As we will see this problem is related to the problem of decryption of RSA (RIVEST, SHAMIR & ADELMAN ENCRYPTION PROTOCOL) that at the same time is related with discovering the prime factors of a number. It has not be discovered a deterministic polynomial time algorithm for this last problem and because of that it belongs to the NP complexity class, but at the same time it has not been proved that it is $NP-complete$ neither. Consequently we will propose a way to cope algorithmically with that complexity under some new theoretical computing model assumption.

2. How to Implement Pseudo-random Strategies

Randomness has attracted the attention of many computer scientists over the last twenty years. One of the firsts subjects about that interested the researchers was how to measure the string randomness. This give place to the Kolmogorov string randomness notion, which can be defined as the length of the shortest description of a string. In a more recent approach, it has been emerged the computational complexity based notion of polynomial randomness measure of a string, that can be defined as follows. A set S of strings is polynomial random if programs running in polynomial time produce the same results when fed either with elements randomly selected in S or with string selected randomly from the set of all the strings. It means that there exist a polynomial time algorithm that, upon input of a *k*-bit string, outputs a poly(*k*)-bit string, such that, if one-way function exist, then the set of all output strings is poly-random. Under this approach of string randomness a function is called poly-random if no polynomial time algorithm, asking for the values of the function at chosen arguments, can distinguish the computed values of the function and values given by an independent coin flips. Based on the existence of a one way function, it can be defined the poly-random collection as a set of all functions $H_k : I_k \to I_k$ where I_k is the set of all k-bit strings. The cardinality of H_k is $2^{k \cdot 2^k}$ so we need $k2^k$ for the specification of this set which is impractical for moderate values of k. In [25] they cope with this problem by randomly selecting for all k a subset $\tilde{H}_k \subseteq H_k$ of cardinality 2^k that belongs to the collection $\tilde{H}$ in such a way that each element of this collection has a unique *k*-bit index function. The objective of [25] was to make random functions accessible for applications, being of easy evaluation and hard to distinguish from random chosen functions in H_k. They achieve this goal by choosing functions from a multiset F_k (whose elements are in H_k) where the collection $F = \{F_k\}$ has the properties of *indexing, Poly-time evaluation and pseudo-randomness*. The *pseudo-randomness property* means that no probabilistic polynomial time algorithm in k can distinguish the function in F_k from the functions in H_k. These functions just mentioned are equivalent to the cryptographically strong pseudorandom bit generators (CSB generators) defined in [22] but outperform them in the sense that they save coin tosses and storage in polynomial time computation with random oracle. A CSB generator is efficient deterministic program that stretch a random *k*-bit-long input seed to a k^t-bit-long output pseudorandom sequence, for some $t > 0$, indistinguishable from a true randomly generated string in polynomial time. The pseudo-random sequence must have some statistical properties present in true random sequences as for example, having the same number of 0's and 1's. In [22] Shamir presents a pseudorandom number generator for which computing the next number in the sequence from the preceding ones is as hard as inverting the RSA function. For the sake of clarity we will give some definitions and results obtained in [25] without demonstration.

Definition of Multiset: Let A be a be a multiset with distinct elements $a_1, \ldots a_n$ occurring with multiplicities $m_1, \ldots m_n$, respectively. Then $|A| = \sum_{i=1}^{n} m_i$. By writing $a \in_R A$, we mean that the element a has been randomly selected from the multiset A. That is, an element occurring in A with multiplicity m is chosen with probability $m/|A|$.

Definition of CSB Generator: Let P be a polynomial. A CSB generator G is a deterministic *poly(k)*-time program that stretches a *k*-bit long randomly selected seed into a *P(k)*-bit long sequence (called CSB sequence) that passes all *next-bit-tests:* Let P be a polynomial, S_k a multiset consisting of *P(k)*-bit sequences and $S = \cup_k S_k$. A *next-bit-set* for S is a probabilistic polynomial time algorithm T that on input k and the first i bits in a string $s \in_k S_k$ outputs a bit b. Let p_k^i denote the probability that b equals the $i+1$st bit of s. We say that S passes the *next-bit-test* T if, for all polynomials Q, for all sufficiently large k, and for all integers $i \in [0, P(k))$;

$$\left|p_k^i - \frac{1}{2}\right| < \frac{1}{Q(k)}$$

It exist a more general kind of test called polynomial-time statistical test where the condition change as follows

$$\left|p_k^S - p_k^R\right| < \frac{1}{Q(k)}$$

where p_k^S denotes the probability that T outputs 1 on $P_1(k)$ randomly selected strings in S_k, and p_k^R represents the probability that T outputs 1 on $P_1(k)$ random bit strings, each of length $P(k)$.

We say that a multiset $S = \cup_k S_k$ is *samplable* if there is a probabilistic polynomial-time algorithm that, given as input k, outputs $s \in_R S_k$.

Definition of one-way function: Let $D_k \subseteq I_k$. Let $f_k : D_k \to D_k$ be a sequence of functions and let the function f defined as follows: $f(x) = f_k(x)$ if $x \in D_k$. Let f^i denote f applied i times. Let $D_k^i \subseteq D_k$ such that $y \in D_k^i$ if $y = f^i(x)$ for some $x \in D_k$. f is a one-way function if

1) f polynomial time computable;

2) f is hard to invert; that is, for every probabilistic polynomial-time algorithm A and for all sufficiently large k, for every $1 \le i \le k^3$, $A(x) \ne f_k^{-1}(x)$ for at least a constant fraction of $x \in D_k^i$;

3) $\cup U_k$ is samplable.

3. Hardness of a Random Robot Motion Tracking

After all those definitions we can now resume the main results of [25] that will allow us to grasp the computational hardness on the prediction of the bits generated by uniform random function. This will enable us to formally base our statements about the computational complexity of the robot motion tracking problem when the *target* robot follows a random motion strategy. We will list the results obtained in [25] as follows.

Result 1: *Let $S = \cup_k S_k$ be a samplable multiset of bit sequences. The following statements are equivalent:*

i. S *passes the next-bit-test.*

ii. S *passes all polynomial-time statistical tests for strings.*

iii. S *passes all polynomial-time statistical tests whose input consists of a single string in* S *. (Rem. CSB sequences pass all polynomial time statistical tests)*

Result 2: *There exists a one-way function if and only if there exists a CSB generator.* This last result allow us to ensure the construction secure CSB generators. Given that a CSB generator can be constructed explicitly if one way functions exist, so can poly-random collections.

Result 3 *(main theorem): Let* F *be a collection of functions constructed using a CSB generator* G *. Then* F *passes all polynomial-time statistical tests for functions.* This lend us to the last result in [25] that we state as follows.

Result 4: *Let* $F = \{F_k\}$ *be a collection of functions satisfying the indexing, and the polynomial-time evaluation conditions of a poly-random collection. Then* F *cannot be polynomially inferred if and only if it passes all polynomial-time statistical tests for functions.* This last result give us the main argument concerning the computational complexity of random robot motion tracking problem. So based in the preceding random function construction results we are able to state formally the next affirmation.

Theorem 1: *The problem of tracking a target robot that behaves randomly cannot be learned in polynomial-time.*

Proof: The strategy followed by the target include a call to a random function. Then we can predict the next move of the target if and only if we can predict the next number generated by a random function. So given that it is not polynomially predictable then the theorem is proved.

As we have done in [1] we assume that each robot is aware of the other robot actions, i.e. Σ^o, Σ^t are common knowledge while the preferences u^o, u^t are private. It is assumed too that each robot keeps a model of the behaviour of the other robot. The strategy of each robot is adaptive in the sense that a robot modifies his model about the other robot such that the first should look for the best response strategy w.r.t. its utility function. Given that the search of optimal strategies in the strategy space is very complex when the agents have *bounded rationality* it has been proven in [10] that this task can be simplified if we assume that each agent follow a DFA strategy. For more details about our DFA learning algorithm see [1].

4. Relation between CSB Generators and Decryption of RSA

In this section we will roughly describe the relation that exist between CSB generators and RSA (RIVEST, SHAMIR & ADELMAN ENCRYPTION PROTOCOL) as a formal tool to base our algorithmic proposed solution to deal wit the random robot motion tracking problem. For this end we will mention some issues that were studied in [22] concerning the generation of CBS sequences. In the seminal work on cryptography done by Adi Shamir in [22] he shows how to generate from a short random seed a long sequence of pseudo-random numbers that he called CSB sequences, based on the RSA cryptosystem. He related the notion of unpredictability with the property of the sequences of being cryptographically strong. He defined additionally the notion of cryptographic knowledge as computed knowledge, that is, as the ability to compute the desired value within certain time and space complexity bounds. He related the one-way functions with the easy to compute permutations on some finite universe U that are everywhere difficult to invert. That is, given a one-way function f, generate a long pseudo-random sequence of elements of U, by the application of f to a standard sequence of arguments derived from some initial seed S, for example $S, S+1, S+2, \ldots$. The difficulty of extracting S from a single value $f(S+i)$ is guaranteed by the one-way nature of f. In [22] is given as an example of a good one-way function, the RSA encryption function $E_K(M) = M^K (\bmod N)$. Concerning this function Shamir shows that it can give degenerate results if it is applied to the sequence $M = 2,3,4,5,6,\ldots$ and that this can be corrected if applied to the sequence $M = 2,3,5,7,\ldots$. For the sake of avoiding degenerancies he proposed an iterated application of f to the secret seed S in conjunction of the **XOR** operator denoted by $\oplus$. Based on that Shamir stated the following lemma.

Lemma 1: ***If f*** *is a one-way function, then a new element of the sequence cannot be computed from a single known element.*

Given that the $\oplus$ operator scrambles the sequences making impossible the proofs in more complex situations, he proposed the RSA public-key encryption function with modulus N that maps the secret cleartext M under the publicly known key K to $M^K \bmod N$. The corresponding decryption function recovers the cleartext by taking the K-th root of the ciphertext $(\bmod N)$. The cryptographic security of RSA cryptosystem is thus equivalent by definition to difficulty of taking root $\bmod N$. When N is a large composite number with unknown factorization, this root problem is believed to be very difficult, but when his factorization (or the Euler function $\varphi(N)$) is known an K is relative prime to $\varphi(N)$, there is a fast algorithm for solving it. Each pseudo-random sequence generator consists of a modulus N and some standard easy-to-generate sequence of keys $K_1, K_2, \ldots$ such that $\varphi(N)$ and all the K_i's are pairwise relative prime. In order actually to generate a pseudo-random sequence of values $R_1, R_2, \ldots$ the two parties choose a random seed S and use their knowledge of $\varphi(N)$ to compute the sequence of roots

$$R_1 = S^{1/K_1} (\operatorname{mod} N), R_2 = S^{1/K_2} (\operatorname{mod} N), \ldots$$

The security of this scheme depends only on the secrecy of the factorization of N. Because of that we can take state the equivalence between the factorization of a number N and the numbers generated by CSB generators. That is, if we consider that the random robot motion strategies use CSB generators for calculating the next move, we will be able to predict or learn the next move of the target robot in polynomial-time, if and only if we can obtain in polynomial-time the prime factorization of a number.

5. Discretization of the Workspace and Pseudo-random Actions

First of all we have discretize the directions to 9 possibilities (N, NW, W, SW, S, SE, E, NE, STOP). The second constraint is on the discretization of the possible situations that will become inputs to the automata of both robots. It must be clearly defined for each behavior what will be the input alphabet to which will react both robots. This can be done without modifying the algorithm The size of the input alphabet impact directly the learning algorithm performance, because it evaluates for each case all possible course of action. So, the table used for learning grows proportionally to the number of elements of the input alphabet. For more details about the construction algorithm see [1]. The discretization mentioned in the preceding paragraph constrains the set of values that can be given as output of a random robot motion strategy.

6. The Quantum Algorithm for the Random Robot Motion Tracking

As we have seen in the section 3 and 4 if take into account that the random movements of the target are generated by CSB generators there is no hope to predict in polynomial-time the next move of the target. The problem here is that under the standard Turing machine theoretical calculation model (TM for short), it has not yet been discovered a polynomial-time algorithm for the prime factorization problem, and it is conjectured that it doesn't exist. By other side, there is relatively new theoretical computer science field called Quantum Computing. Roughly speaking, this new field propose to replace the Newton physics operation based Turing Machine theoretical model by a Quantum physics operation based Turing Machine denoted as QTM (QUANTUM TURING MACHINE). The QTM has some extra features as for example, the capability of being in many states simultaneously. These new features give place to some hopes concerning the possibility to solve in polynomial-time with a QTM (QUANTUM TURING MACHINE) some very complex unsolvable in polynomial time under the standard TM model. For the moment it has not yet been proved that a QTM can solve NP-complete or NP-hard problems in polynomial-time but some progress has been made in the last twenty years in the conception of polynomial-time algorithms for solving some less hard problems that belong to the NP complexity class. Such is the case of the prime factorization problem. This algorithm was proposed by Peter W. Shor in [9] and solves the prime factorization problem in expected polynomial-time. Quantum algorithms are methods using quantum networks and processors to solve algorithmic problems. Quantum algorithms have been developed utilizing the power of quantum evolution, especially of such quantum phenomena as quantum superposition, parallelism and entanglement. For solving the prime factorization problem with a QTM algorithm, some number theory problems as integer factorization and the calculation of the *order* or *period* of a function $E_K(M) = M^K (\bmod N)$, that is, to find the smallest K *s.t.* $M^K \equiv 1 \bmod N$, must be solved in polynomial-time. For the first problem the *gcd* solve it in $O(\log N)$, and for the second we have to exploit the parallelism of a QTM and use the QFT (Quantum Fourier transform). The key idea of the Shor's algorithm is to relate the calculation of the period of a function with the prime factorization problem using the QFT.

In [9] Shor uses as computational model the *quantum acyclic circuit* that can compute in polynomial time the same functions that can be calculated by a *Quantum Turing Machine model* if a small probability errors allowed. This means that the class of functions polynomial time computable with a small probability error does not depend on the exact architecture of the quantum computer, what implies that the complexity class BPQ (by analogy with the classical complexity class Bounded Probabilistic Polynomial time or BPP) is robust. In a system with n components having two states can be described in classical physics completely by n bits whereas in quantum physics it requires $2^n - 1$ complex numbers. Formally talking, we are dealing with a 2^n- dimensional space having associated a linearly independent set of vectors called a base, in terms of which any vector or pure state $|x\rangle$ can be expressed. This kind of vector space is know as *Hilbert space*. The states of the system in *Hilbert space* are represented by unit length vectors. The superposition of states is represented as

$$\sum_{i=0}^{2^n-1} a_i |S_i\rangle,$$

where a_i are complex numbers representing the amplitudes such that $\sum_i |a_i|^2 = 1$ and each $|S_i\rangle$ is a basis vector of that space. So at each step of the computation the probability of seeing a state $|S_i\rangle$ is $|a_i|^2$.For being able to use the physical system for the computation we need to apply only unitary transformations to the state vectors by means of unitary matrices (whose conjugate transpose is equal to its inverse) because the sum of all the possible outcomes must give as result 1. The quantum circuits can only perform local 2-bit transformations but this is not a problem because it can be proved that the n-bit transformations can be implemented using as building blocks the 2-bit transformations. The 2-bit transformations or quantum gates can be expressed as a truth table as follows:

$$|00\rangle \rightarrow |00\rangle$$

$$|01\rangle \rightarrow |01\rangle$$

$$|10\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|10\rangle + |11\rangle\right)$$

$$|11\rangle \rightarrow \frac{1}{\sqrt{2}}\left(|10\rangle + |11\rangle\right)$$

The truth table above can be expressed with a matrix as follows

In	Out	$\|00\rangle$	$\|01\rangle$	$\|01\rangle$	$\|11\rangle$	
$\|00\rangle$		1	0	0	0	
$\|01\rangle$		0	1	0	0	
$\|10\rangle$		0	0	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{2}}$	
$\|11\rangle$		0	0	$\frac{1}{\sqrt{2}}$	$-\frac{1}{\sqrt{2}}$	

Assuming as an example that the machine is in the following superposition of states

$$\frac{1}{\sqrt{2}}|10\rangle - \frac{1}{\sqrt{2}}|11\rangle$$

and that we apply the unitary transformation shown in the matrix above. The machine will go to the next transposition of states

$$\frac{1}{\sqrt{2}}\left(|10\rangle + |11\rangle\right) - \frac{1}{\sqrt{2}}\left(|10\rangle - |11\rangle\right) = |11\rangle$$

From the example we can see that the interference effect that cancel the observation of some states of the initial superposition. Another feature that we have un quantum computing and that it absent int classical computing is the reversibility of the calculation. As a consequence a deterministic computation is performable on a quantum computer only if it is reversible. There exist universal gates for computing reversibly called Toffoli gates and Fredkin gates that use extra input (controlled NOT) and output wires and that need to reset to 0 some output wires as a method for avoiding to have interference at the output wires. So we can replace classical AND or NOT gates by Fredkin gates or NAND gates by Toffoli gates and turn a non-reversible gate array into reversible one. For this end it is needed to duplicate some input wires and keep the extra output bits in a register. For more details on how to do that efficiently see [9]. The modular exponentiation part of the factoring algorithm is the place where more time and space is consumed. The best classical method to solve the modular exponentiation is the *repeated squaring* that takes $O(l^2 \log l \log\log l)$ time and $O(l \log l \log\log l)$ space for l-bit numbers. This method uses the FFT method for the multiplications and don't scales up well for small length numbers and in that case it is used the standard multiplication algorithm and the time taken by modular exponentiation becomes of order $O(l^3)$ and the space becomes of order $O(l)$ for computing $(a, x^a (\bmod n))$. Given that the quantum computations need unitary transformations and that the discrete Fourier transformation is a unitary transformation that that allow us to transform states corresponding to integers in binary representation on a computer, we can represent it as matrix. So consider a number a with $0 \le a < q$ where the number of bits of q is polynomial. We can transform a state $|a\rangle$ to a state $\frac{1}{q^{1/2}} \sum_{c=0}^{q-1} |c\rangle \exp(2\pi iac/q)$.

That is that we apply the unitary matrix whose (a,c) entry is $\exp(2\pi iac/q)$. We have to take $q = 2^l$ and the binary representation of a as $|a_{l-1}a_{l-2} \mathrm{K}\ a_0\rangle$. Then we can build the transformation matrix A_q using two type of quantum gates that will operate on the jth bit. These gates can be defined as follows

$$R_j = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \text{ and}$$

$$S_{j,k} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta_{k-j}} \end{pmatrix}$$

that operates on bits in positions j and k for $j < k$ .and where $\theta_{k-j} = \pi/2^{k-j}$

To perform the Fourier transformation the matrix multiplications have to be applied in general as follows

$$R_{l-1}S_{l-2,l-1}R_{l-2}S_{l-3,l-1}S_{l-3,l-2}R_{l-3} \mathrm{K}\ R_1 S_{0,l-2} \mathrm{K}\ S_{0,2}S_{0,1}R_0$$

If we take $q = 2^l$ for the Fourier transform A_q we will need $l(l-1)/2$ quantum gates.
The prime factorization problem have interested the mathematicians since the Euclid times and even before. From the number theory field it is know that any number n has a unique prime factor decomposition. The best factoring algorithm takes $\exp\left(c(\log n)^{1/3}(\log\log n)^{1/3}\right)$ for some constant , and given that the input n can be represented by $\log n$ bits, this algorithm is exponential. By other side, it is known that the factorization problem can be reduced to the problem of finding the order of an element by means random algorithms. Because of that the Shor algorithm proposed in [9] in order to do factorization for breaking RSA cryptosystem, it compute order or period of a function several times using th QFT (Quantum Fourier Transform). The basic idea of the algorithm is:

1. to create a state with a period we need to determine
2. to apply QFT to get rid of the offset
3. to extract the period by computation

So given that the random robot motion strategies use CSB generators for calculating the next move, and that it is equivalent to the prime factorization of a number, we can enounce our second result as follows.

Theorem 2: *The problem of tracking a target robot that behaves randomly can be learned in expected polynomial-time using the Shor prime factorization QTM algorithm.*

We have supposed along the present article that the *target robot* follows all the time a random strategy of movements. We can try to explore the case where the robot follows for some time a DFA (DETERMINISTIC FINITE AUTOMATA) and suddenly when it get stuck in a local minima, he starts a random walk, as is the case of robots following a path calculated by a planner of the kind proposed by Barraquand & Latombe in [11] or Erdmann in [16]. If we apply our DFA learning algorithm we can loss the *target robot*. So what we propose as solution is to run in parallel our DFA learning algorithm and quantum algorithm.

7. Conclusions and Future Work

As we have shown in the present paper, the random robot motion tracking problem is not polynomial-time solvable under the standard TM by relating it with prime factorization problem. As consequence we have proposed an alternative to cope with this negative result by means of the application of quantum algorithms. As a future work we can explore what happen in the case of using quantum versions of a CSB generators. Another possible future research can be the mixed situation mentioned at the end of the section 6 of the present article.

References

1. Carlos Rodríguez Lucatero & Rafael Lozano Espinosa, Application of automata learning algorithms to robot motion tracking, *Proceedings* ISPRA2005 Austria, 2005.
2. C.Rodríguez Lucatero, A. de Albornoz & R. Lozano E., A game theory approach to the robot tracking problem, *WSEAS Transactions on Computers*, Issue 4, Volume 3, ISSN 1109-2750, 862-868, October 2004.
3. Dana Angluin, A note on the number of queries needed to identify regular languages, *Information and Control*, 51, 76-87, 1981.
4. S.M. La Valle, H.H. González Baños, Craig Becker, & J.C. Latombe, Motion Strategies for Maintaining Visibility of a Moving Target, *Proceedings of the IEEE International Conference on Robotics and Automation*, 731-736,April 1997.
5. S.M. La Valle, David Lin, Leonidas J. Guibas, J.C. Latombe & Rajeev Motwani, Finding an Unpredictable Target in a Workspace with Obstacles, *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1997.
6. M.L. Littman, Markov games as a framework for multiagent reinforcement learning, *Proceedings of the eleventh International Conference on Machine Learning*, 157-163 ,1994.
7. R. Murrieta-Cid, H.H. González-Baños & B. Tovar, A Reactive Motion Planner to Maintain Visibility of Unpredictable Targets, *Proceedings of the IEEE International Conference on Robotics and Automation*, 2002.
8. Christos H. Papadimitriou & John N.Tsitsiklis, The complexity of Markov decision processes, *Mathematics of Operations Research*, Vol. 12, No. 3, August 1987.
9. Peter W. Shor, Polynomial time algorithms for prime factorization and discrete logarithms on quantum computer. *SIAM* Journal on computing, 26 (5):1484-1509, 1997.
10. Ariel Rubinstein, Finite Automata Play the Repeated Prisioner's Dilemma, *Journal of Economic Theory*, 39, 83-96, 1986.
11. J.Barraquand & J.C. Latombe, Robot Motion Planning: A distributed representation approach, *STAN-CS-89-1257*, Technical report CS. Dept. Stanford University , 1989.
12. Dana Angluin, Jeffery Westbrook, & Wenhong Zhu, Robot Navigation with range Queries, *ACM STOC 96*, 469-478, 1996.
13. Ronald L. Rivest, & Robert E. Schapire, Inference of Finite Automata using Homing Sequences, *Information and Computation*, 103, 299-347, 1993.
14. A. Blum, P. Raghavan & B. Schieber, Navigating in unfamiliar geometric terrain, *ACM STOC 91*, 494-504, 1991.
15. V. Lumelsky & A. Stepanov, Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape, *Algoritmica*, 2, 403-430, 1987.
16. Erdmann, M.A., On Probabilistic Strategies for Robot Tasks, *MIT Artificial Intelligence Laboratory*, Technical Report, 1990.
17. C.H.Papadimitriou & M. Yannakakis, On Complexity as bounded rationality, *STOC94 ACM*, Montreal, Quebec, Canada, 726-733, 2004.
18. R. Axelrod, *The Evolution of Cooperation*, Basic Books, 1984.

19. M.Kearns & L. Valiant Cryptographic limitation on learning Boolean formulae and finite automata , *Proc. 21th ACM Symposium on Theory of Computing* ,pag 433-444, May 1989.
20. L. Pitt & M.K. Warmuth, The minimal consistent DFA problem cannot be approximated within any polynomial, *JACM* 40(1):95-142, January 1993.
21. Shamir A. , On the generation of cryptographically strong pseudorandom sequences, *ACM Trans. Comput. Syst.* 1, 1:38-44, Feb. 1983.
22. David Carmel & Shaul Markovitch, Learning Models of Intelligent Agents, *Technical Report* CIS9606 Technion, 1996.
23. David Carmel & Shaul Markovitch, How to explore your oponent's strategy (almost) optimally, *Proceedings ICMAS98* Paris France, 1998.
24. Oded Goldreich, Shafi Goldwasser & Silvio Micali , How to construct random functions, *JACM Vol. 33, No. 4*, pp 792-807, October 1986.