UNIVERSITY OF AGDER

On Novel Variants of the Hierarchical Stochastic Searching on the Line

By

Pariya Shahbazi

Thesis submitted in Partial Fulfillment of the
Requirements for the Degree Master of Science in
Information and Communication Technology

Thesis Supervisor: Professor Ole-Christoffer Granmo

Thesis Co-Supervisor: Dr. Anis Yazidi

University of Agder, 2012

Faculty of Engineering and Science

Department of Information and Communication Technology

Abstract

This research proposes two novel types of hierarchical search based solution to the Stochastic-Point Location (SPL) problem. In the SPL problem, placing a point on the line is the main goal of the robot. In order to find the best position of the unknown parameter (ψ^*) on the line, the robot communicates with an Environment which basically guides it with *p* (the "effectiveness" of the Environment) probability, which means that the robot has a probability of being given erroneous recommendations by the Environment for choosing right or left side of a given point. The first investigation on the SPL problem by using a hierarchical search space [36] executed a controlled random walk on a discretized space constructed as a binary tree to place the unknown parameter in an accurate and quick way. The main deficiency of the Hierarchical Stochastic Search on the Line (HSSL) solution [36] is the fact that the robot can still visit the rest of the nodes after finding the optimal action. In addition, HSSL approach makes transitions between consecutive levels in the tree structure of the search space.

To deal successfully with locating the unknown parameter on the line, two novel alternatives, *Learning Automata based solution to HSSL* and *Multilevel Jumps based solution to HSSL*, are proposed. The former is a new search method with a logic that resorts to *Reward-Inaction Learning Automata (L_{RI})* operation, and the latter is a simple search method that allows transitions across multiple levels, rather than only between consecutive levels. The advantages of using Learning Automata (LA) and Multilevel jumps that provide accurate and simple techniques are novel features of the proposed methods. The strategies proposed here can be applied to learn the best parameter to be used in the optimization. The solution has been simulated, with interesting results.

Preface

This master thesis is submitted in partial fulfillment of the requirements for the degree Master of Science in Information and Communication Technology at the University of Agder, Faculty of Engineering and Science. This work was carried out under the supervision of Professor Ole-Christoffer Granmo at the University of Agder, Norway.

First of all, I want to thank my supervisor Dr. Ole-Christoffer Granmo, for great assistance and inspiration throughout the project period. Especially for his patience, it is safe to say that without his assistance the thesis would have faced endless difficulties.

Secondly, I would also like to thank Dr. Anis Yazidi for introducing me to this interesting field of computer science and giving his valuable support, feedback and admirable patience during the entire project period. Under his guidance I learnt a lot about the project content and technical report writing.

Last but not the least, I would like to thank my family for supporting, helping and believing in me. Without their support and understanding, I have never made this achievement of this study possible.

Grimstad, May 2012
Pariya Shahbazi

Contents

List of Figures

List of Tables

1 Introduction

Nowadays, discrete optimization problem has become one of the essential problems by which engineers and decision makers collect information about stochastic systems. Handling these systems is performed by the discrete events with interactions between these events over time.

This project considers a problem in discrete optimization called Hierarchical Stochastic Searching on the Line (HSSL). The problem involves a robot (Learning Mechanism (LM)) walking along the real line planning to place a special point ψ*. It is assumed that LM can interact with an Environment ("Oracle"), which notifies it with information regarding the direction in which it should go.

The discrete optimization problem is cited as "deterministic point location problem", if the Environment is deterministic. In its pioneering type, Baeza-Yates *et al.* [19] offered the problem in a form that the Environment could allocate a cost to the robot that is corresponding to the distance it is from the point searched for. The question of having multiple interacting LMs place a point on the line has also been investigated by Baeza-Yates *et al.* [19, 20].

In the stochastic variant of the point location problem proposed by Oommen [15], [31], [16], the Learning Mechanism intends to place a point in an interval with stochastic (i.e., possibly erroneous) feedbacks from the Environment, rather than deterministic feedbacks. Hence, when it should really be going to the "right" it may be recommended to go to the "left" and vice versa, with a probability p ($p \neq 0$).

The Stochastic Point Location (SPL) problem is related to the field of Learning Automata (LA) problems [42–47], in which the LM aims to learn from a stochastic Environment. Particularly, unlike the pioneering type of LA model in which the LA intends to determine the optimal action proposed by the Environment, in this research it is considered the LM is

attempting to place an unknown point ψ^* on an interval by communicating with the random[1] Environment through a sequence of informed estimates.

The purpose of the optimization problems is to perform a given task with the minimum cost or with the maximum profits. If the basic cost function (or benefit function) is recognized, then the problem is normally one of minimizing (or maximizing) this function.

The algorithm for SPL executes its way iteratively from its current point to the optimal point, like many of the optimization approaches such as image processing, pattern recognition, and neural computing [15], [21- 27], [49]. Such algorithms have a key parameter that indicates the algorithm convergence to the optimal action. Selecting the value for this parameter is crucial to the algorithm. In many methods, the scheme parameter is related to the second derivative of the function, which ends in a technique analogous to a "Newton's" root solving method. The deficiency of the latter is the fact that it needs the additional computation for assessing the (matrix of) second derivatives [23, 24, and 26]. In order to solve this shortage, in the work of [36] it is declared that their solution solves the stochastic hierarchical search with the best parameter that can be used in any algorithm. It should be cited that their solution needs no additional computations for derivatives. Furthermore, in the method of [36] there exists a learning strategy in order to converge to a value arbitrarily close to the best parameter. Therefore, in our solutions, the method presented in [36] is utilized to develop an accurate technique with less computation.

The main shortages of the state-of-the-art solution to the Stochastic-Point Location (SPL) problem presented in [36] are two cases. First, in the [36] approach when the LM reaches the optimal action, it can still visit the rest of the nodes in the tree, so the scheme convergence is decreased. Second, it makes transitions between consecutive levels in the tree means

[1] - Random Environment will be briefly described in section 1.1.1.

that LM can move up to the immediate *Parent*, so finding a node placed at four (or more than four) levels upwards in the tree is time consuming.

Unlike the investigation of Yazidi *et al.* [36] in this research, designing two novel approaches of hierarchical learning schemes named as *LA based solution to HSSL* and *Multilevel Jumps based solution to HSSL* for solving the SPL problem is devised. In the former solution presented in our investigation *Reward-Inaction* LA converges to the best value of the unknown parameter ψ^* and never comes to another node, hence the scheme convergence is increased and in the latter approach the scheme allows the LM to move up for more than one level in the hierarchical search space, rather than only between consecutive levels.

1.1 Background

1.1.1 Learning Automata

A Learning Automata (LA) is an adaptive and stochastic decision making automaton[2] with relatively little initial knowledge that is placed in a random Environment and concurrently learns the optimal action through frequently interactions with its Environment and based on its achieved experience. The actions are selected corresponding to a particular probability distribution that is updated based on the response that automaton gains from the Environment by executing a specific action.

Learning Automata is associated with the design and programming of methods that allows computers to generate behavior based on experimental input data, such as sensor data or databases. The Learning Automata concept is created on the basis of psychologists' work in behavior modeling and statisticians' efforts in determining the choice of trials based on past experience. The aim of scientists is to produce intelligent decisions.

In the study of LA, the Environment is typically developed as one that rewards or penalizes the automaton randomly; the LA aims to find the optimal strategy, hence it maximizes the probability of being rewarded. The key point is that the success probability for each action is unknown to the automaton; it adapts itself to the Environment by learning the optimal action.

In other words, while interacting with the Environment, the Learning Automaton chooses an action and then the Environment response tells the LA whether the selected action was rewarded or penalized. Afterward, the automaton makes use of this information in order to decide which action to take next time, and the cycle continues.

[2]- "An automaton (plural: automata or automatons) is a self-operating machine. The word is sometimes used to describe a robot." [55]

The LA operates in either *random Environment* or *unknown Environment*. In the former case, an action does not present the same response each time it is executed and in the latter, the action does not require information about the impact of its action at the start point of the operation.

A strong feature of Learning Automata is that it can progress its performance by means of a learning process. LA combines quick and precise convergence with low computational complexity. Therefore, LA is employed in this research as one of our proposed solutions to solve a stochastic learning problem.

The automaton chooses an *action* a_i at instant i from a finite action set $\{a_i \mid i=1 \text{ to } R\}$ (see Figure 1). The chosen *action* acts as the input to the Environment that yields a response b_i at time i. The b_i is an element of $B = \{0, 1\}$. It is assumed that 1 is *Penalty* and 0 is *Reward*. When performing an *action* a_i, there is a certain probability that the Environment responds with a *Penalty:*

$$P(Penalty \mid Action = a_i) = c_i, \quad 1 \leq i \leq R$$

Remark: If the Environment does not respond with a *Penalty*, it responds with a *Reward* instead.

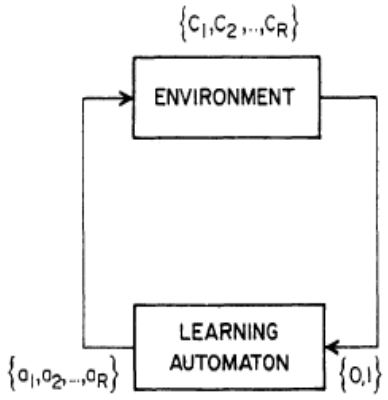

Figure 1: Learning Automata [35]

1.1.2 The Stochastic Point Location Problem

In this section, the main definition of the Stochastic Point Location (SPL) problem, which was first proposed by Oommen in [15], is briefly presented. In the SPL problem a Learning Mechanism (LM) is applied in order to determine the best value of a parameter, ψ. It is assumed that there exists an optimal value for ψ – an unknown value, called ψ* in a unique search interval, [0, 1). Learning the unknown parameter ψ* in an efficient way is the main goal of the SPL problem. Although, the value of ψ* is unknown to the scheme, it is supposed that it has feedbacks from an intelligent "*Environment*" ("*Oracle*") which is enable to inform it whether any value of ψ is too small or too big. It should be stated that the *Environment* feedback is assumed "faulty"; hence the problem is different from its deterministic counterpart. Indeed, the *Environment* may recommend the LM to increase ψ when it should be decreased, and vice versa, with a probability $1 - p$ that is bigger than zero. In this area, the quantity "*p"* indicates the "effectiveness" of the "*Environment" E* which is normally suggested bigger than 0.5. Thus, whenever the current ψ is smaller than ψ*, the *Environment* correctly responds that LM should increase ψ with probability *p*. It synchronously could have incorrectly replied that LM should decrease ψ with probability $(1 - p)$. The reverse is also true whenever ψ ≥ ψ* [36].

1.1.3 The Hierarchical Stochastic Search on the Line

In order to solve the SPL problem, the Hierarchical Stochastic Searching on the Line (HSSL) solution has been proposed in [36], which supplies much faster convergence compared to the work [15]. Besides, HSSL scheme is able to deal with non-stationary Environments[3]. Indeed, the Hierarchical

[3] - The Environment is cited as a non-stationary Environment when its feedback for placing a point in an interval to find the best value of ψ is both uncertain and time varying.

Stochastic Search is similar to the binary search[4], but in the HSSL technique, ψ^* changes over time, which means that the Environment is non-stationary. In addition, the quantity p in HSSL solution indicates the probability that Environment correctly suggests to LM moves in the tree, which confirms the non-deterministic property of HSSL. Hence, finding the target node in HSSL method is rigorous compared to simple binary search, since the Environment of the binary search generates deterministic feedback.

HSSL strategy executes a controlled random walk on a discretized search space constructed as a binary tree with maximum depth D. It is assumed that $[\alpha, \beta)$ is the current search interval containing the unknown parameter ψ^*. To each node in the hierarchy it is associated an interval containing three points, i.e., left and right boundary points (α and β) along with the middle point of the interval ($(\alpha+\beta)/2$).

In the HSSL solution, the LM searches for the target node including ψ^* by two kinds of random walk transitions. When the LM aims to reach the immediate *Parent* (a larger search interval), the *reverse transition* is applied; hence it corresponds to a lower level movement in the hierarchy. Besides, a *top-down transition* is employed whenever the LM performs a transition to a deeper level in the hierarchy by selecting a *Child* node. In fact, in *top-down transition* the search space becomes small, and so the LM will concentrate on one of the adjacent intervals at the next level of the tree that may include the unknown parameter ψ^*.

By making use of the HSSL solution in solving the SPL problem, the convergence speed of the scheme is remarkably increased compared to the original SPL solution reported in [15], since in the HSSL technique the search space is discretized and is structured as a binary tree. Therefore, the HSSL solution is applied in this research as our second solution to solve a stochastic learning problem.

[4] - For more information about binary search see Appendix.

1.2 Thesis Definition and Hypothesis

1.2.1 Thesis Definition

We formulate the thesis definition in the following way:

The main goal of this thesis is to determine how LA based solution to HSSL and Multilevel Jumps based solution to HSSL can be used to solve the SPL problem with high accuracy and low complexity convergence. To evaluate the performance and scalability empirically, a tree-based search space should be designed and developed. Further, novel proposed approaches should be tested with faulty feedbacks from the Environment.

1.2.2 Hypothesis

As the growing need of efficient ways to solve the SPL continues, it is in this research hypothesized that combining the *Reward-Inaction* LA (L_{RI}) and Multilevel jumps with the existing approach will drastically increase the efficiency of the solving SPL. The reason to this is that the *Reward-Inaction* LA is ε-optimal and it converges to the optimal action precisely [57]. Hierarchical Stochastic Search on the Line (HSSL) [36] is an existing approach which has been proved to be a very efficient method to solve the SPL problem, therefore this approach has been chosen to be combined with the *Reward-Inaction* LA (L_{RI}) and Multilevel jumps, in order to prove whether LA and Multilevel jumps will increase the efficiency to solve SPL or not. This combination will create the *LA based solution to HSSL* and *Multilevel Jumps based solution to HSSL* techniques.

1.3 Importance of the Topic

Implementing capable techniques for solving discrete optimization problems such as the SPL problem is a significant area of engineering research. Such techniques could be utilized to assist in solving the stochastic resonance problem [40], the stochastic sampling problem in computer graphics [41], the problem of determining roads in aerial images by utilizing geometric-stochastic models [37], the stochastic and dynamic vehicle routing problem [38], and different variants of location problems [39].

Considering the above reasons, we hope that the reader is convinced that searching for an effective solution for such an underlying problem can be quite rewarding. Hence, in this research the HSSL [36] has been utilized since it is proved as an effective approach in stochastic optimization problem. In addition, the HSSL technique has been shown to supply orders of magnitude faster convergence than the traditional SPL solution [15] when implemented in non-stationary Environment where ψ^* changes over time. In this sense, LA has also been used since it converges accurately in learning schemes, so this research will introduce new and efficient ways of solving SPL by proposing the *LA based solution to HSSL* and *Multilevel Jumps based solution to HSSL* in hierarchical structure of the search space.

1.4 Motivation

If the problem is solved, the research will offer two novel and drastic techniques of solving SPL by applying *Reward-Inaction* Learning Automata and Multilevel jumps in the original HSSL solution in [36]. Adding a new algorithm to the collection of solver algorithms is a step further of solving complex problems that can be represented as SPL, and an advancement in the science of SPL.

1.5 Research Questions

In our research we will answer the following questions:

- How can *LA based solution to HSSL* be applied to design a new accurate strategy of solving the SPL problem in stationary Environments?

  The question covers one of the central research elements, i.e., to accurately solve the discrete optimization problem by using Learning Automata in the tree-based search space and in the stationary Environment where ψ^* is time-invariant. The question also defines a technique that should be applied to the problem. The technique is the Hierarchical Stochastic Searching on the Line (HSSL), which is presented by Yazidi, Granmo, Oommen and Goodwin [36]. The heart of the HSSL strategy includes executing a controlled random walk on a discretized space structured as a binary tree. The HSSL strategy is specially presented to determine the best parameter to be used in the optimization. The overall learning scheme has been shown to be optimal [36].

- How to incorporate Multilevel jumps with the HSSL solution for designing a new simple strategy of solving the SPL problem in non-stationary Environments?

  The question refers to the second central research element, i.e., how the LM can move upwards for more than one level in the tree-based search space to solve the discrete optimization problem with less complexity compared to the work [36]. It is interesting to observe how the solution behaves when residing in a wrong sub tree, i.e., one that does not include ψ^*. This alternative is executed in a non-stationary Environment where ψ^* is changing over the time.

- Will the integrated methods make locating the stochastic point more efficient, and how can we test it?

  After modeling the hierarchical structure of the search space and merging LA and Multilevel jumps with the original HSSL solution [36], we will compare our results to the algorithm introduced in [36] to disclose whether our results have been obtained in high enough efficiency or not. Therefore, we will find out whether the proposed alternatives can be useable in SPL solving or not.

1.6 Claim

In this thesis it is claimed that using a *LA based solution to HSSL* clearly provides more accurate convergence compared to the original HSSL solution offered in [36]. Further, we claim that Multilevel jumps in the hierarchical structure of the search space can be applied in order to find the best unknown parameter ψ^* by a less complex technique compared to the

original HSSL proposed in [36]. Lastly, we claim that the solution is scalable, but the total running time is increasing when adding more levels to the tree structure of the search area.

1.7 Limitations and key Assumptions

The implementation of the proposed algorithms, *LA based solution to HSSL* and *Multilevel Jumps based solution to HSSL*, will be done in the Java programming language. Although, two novel variants of HSSL proposed in this research may require much work to be implemented very efficiently in Java. Finding the best parameter in the optimization problem could be done by different algorithms, which each of them can affect the results. However, all of these algorithms will not be performed in this research due to the limited time span of the thesis. Besides, by increasing the scheme resolution N (or tree depth $D=\log_2(N)$), the convergence rate of our proposed solutions is also increased, but handling large values of N demands a high number of simulation runs which was not done in this project due to limitations in computational resources.

We have made two significant assumptions. First, the Learning Automata applied in the project is *Reward-Inaction* LA in which the effects of non-optimal actions are ignored. In other words, if the LA is rewarded by the Environment for choosing an action, it increases the probability of choosing this action at the next time instant, otherwise if LA is penalized then the probability of choosing this action remains unchanged. Secondly, the responses from the Environment to a LM in the tree structure of the search space are Binary (*Reward* or *Penalty*).

1.8 Contribution to the Knowledge

The outcomes of this project will contribute with a new experience in the computer research domain. Indeed, the results will assist discrete optimization solver algorithms. By making use of the results achieved in this research, the optimal value of the parameter will be found with a higher accuracy and less complexity, and therefore with the least computational complexity.

In the area of computer science, this project will test a different strategy by incorporating the HSSL technique with Learning Automata operations in solving SPL problem for the first time. The proposed solution in this project will work in a stationary Environment where ψ^* is time-invariant. Further, applying Multilevel jumps in the HSSL solution will be presented in this thesis research which provides a simple solution to the SPL problem when the Environment is non-stationary where ψ^* is time varying.

1.9 Target Audience

The target audience of this thesis is anyone that has interests within the hierarchical stochastic search for solving the SPL problem. Particularly, it is targeted at people who are interested in the usage of tree-based search technique along with LA concepts for finding the best value in optimization problems. Since the main contribution of the thesis is to apply a machine-learning technique called Hierarchical Stochastic Search on the Line (HSSL), Artificial Intelligence researchers and other people interested in this field may find this thesis interesting.

For the reader to fully understand the concepts and reasoning behind the proposed solutions, some knowledge of machine learning and a good

understanding of common basic elements within the field of computer science is recommended.

1.10 Thesis Report Outline

The rest of this thesis is organized as follows: Chapter 2 represents the previous research in the field of SPL problem solver methods. Chapter 3 is intended to present theoretical background of the Learning Automata concept, i.e., what LA is, how LA operates in a system and what the *Reward-Inaction* LA is. Besides, Chapter 3 introduces briefly concepts behind discrete stochastic optimization, hierarchical structure of the search space and the Stochastic Point Location (SPL) problem. Chapter 4 is aimed to explain the proposed solutions formally. A combination of the HSSL solution with the Learning Automata concept in a stationary Environment is thoroughly stated in this chapter. It also describes how *Multilevel jumps based solution to HSSL* will be executed in order to find the best value for the unknown parameter ψ^* when the Environment is non-stationary. In chapter 5 the results achieved from the experiments are delineated as commented plots. In Chapter 6 the main findings of the proposed solutions will be discussed. Chapter 7 is intended to wrap up, supply a conclusion and offer interesting aspects that may be pursued in further research.

2 Significant Prior Research

The focus in this chapter is on significant prior work. There have not been done many researches in solving the SPL problem using tree-based search space. We will therefore mention two important papers in solving the SPL problem in the following sections. The first paper is important for solving SPL in an efficient way without applying hierarchical structure and the second one is significant for applying hierarchical search space for solving SPL. In both papers the discretizing search space method is applied to solve the SPL problem, because this method takes benefit of the restricted accuracy available in practical implementations to limit the probability of selecting an action to only finitely many values from the unique search interval [0, 1].

In the first paper a "*one-dimensional"* controlled random walk in a discretized search space is executed in order to place the unknown parameter ψ^*. However, the second paper makes use of a discretized search space constructed as a binary tree; an efficient method that has been used to solve the Stochastic Point Location (SPL) problem.

2.1 Solving Stochastic Point Location by Discretizing the Space

The heart of the strategy proposed in [15] consists of discretizing the search space and executing a controlled random walk on this space. The scheme has been proved to be ε-optimal and to converge with probability 1. One difference among traditional learning systems and the discrete learning scheme is that *Discretized Point Location* algorithm *(DPL)* modifies the action probability in discrete steps and not continuously. The DPL algorithm restricts the action likelihood into finite values of the search interval. In contrast, in the concept of traditional learning algorithms is the fact that the probability of selecting an action may be any real number in the interval [0, 1].

The solution cited in [15] for the SPL problem performed as follows: the search space is partitioned by subdividing the unit interval into N steps $\{0, 1/N, 2/N, \ldots, (N-1)/N, 1\}$. The discretization is assigned as linear if the allowed values are equally subdivided in the interval [0, 1] and contains the points 0 and 1; otherwise, the discretization is cited as nonlinear. In this approach, a larger value of N will indicate a more precise convergence to the unknown ψ^* (target node). The algorithm then executes a controlled random walk on the discretized space. Whenever the Environment informs the LM to go to the right (or left), it moves to the right (or left) by a single step in the discretized space.

Indeed, the scheme alluded in [15] performs the following updating rules:

Assume $\psi(n)$ is the value at time step "n". In other words, $\psi(n)$ is a guess of the unknown value of ψ^* at time step "n" [15]. Then,

$\psi\ (n + 1) := \psi\ (n) + 1/N$ *;if E suggests to increase* ψ *and* $0 \leq \psi\ (n) < 1$

$\psi\ (n + 1) := \psi\ (n) - 1/N$ *;if E suggests to decrease* ψ *and* $0 < \psi\ (n) \leq 1$

At the end states the scheme performs:

$\psi\ (n + 1) := \psi(n)$ *;if* $\psi\ (n) = 1$ *and E suggests increasing* ψ

$\psi\ (t + 1) := \psi(n)$ *;if* $\psi\ (n) = 0$ *and E suggests decreasing* ψ

The analytical outcomes achieved in [15] determined that if the "Oracle"[5] was informative[6] then the discretized random walk learning was optimal. Hence, the mechanism would move toward a position arbitrarily close to the true target position with an arbitrarily high likelihood.

The main deficiency of the scheme explained in [15] is the fact that the steps made are always very conservative. If the size of each step is increased the convergence speed of the scheme is improves, but the accuracy is correspondingly diminished.

2.2 Solving Stochastic Point Location by Hierarchical Stochastic Search on the Line

The approach [15] which was stated in section 2.1 limited the parameter to be one of the finite number of values in the unique interval [0, 1], and then executed a "*one-dimensional*" controlled random walk on the discretized search space, where "*left*" and "*right*" are the only choices for the learning mechanism. Hence, the size of the scheme resolution, *N*, was proportional to the convergence speed. Meaning that as the scheme resolution got bigger, the scheme convergence performed slower. This deficiency was resolved by

[5] - The scheme is informed by "Oracle" or "Environment" *E* whether the present value of ψ is too small or too big.

[6] - The definition of the "informative" Environment is stated in section 3.1.

Yazidi *et al.* in their work in [36]. In this paper, a different approach is proposed for discretization, where the search operation is performed in a discretized search space constructed as a binary tree with maximum depth $D=log_2(N)$ where N is the resolution of the scheme, so the convergence speed is improved by using a tree-based search space. In this technique, a resolution is associated to each level of the tree, which becomes better at deeper levels of the tree. In fact, executing a managed random walk by the Learning Mechanism on the tree-based search space is the main part of this solution, which distinguishes this solution from the approach stated in [15]. It has been shown in this paper that Hierarchical Stochastic Search on the Line (HSSL) supplies a much more faster convergence compared to the original SPL solution presented in [15]. It has also been proved that the overall scheme is ε-optimal when the Environment effectiveness, p, is greater than the *golden ratio conjugate*[7] [10]. In this paper it has been demonstrated that the HSSL solution supplies a more accurate scheme than the original SPL approach in [15]. HSSL solution requires less iteration compare to the original SPL solution in order to obtain 95% of the unknown parameter ψ^*.

For the sake of brevity, the construction of the hierarchy and other concepts related to HSSL will be described later, since in this thesis research, similar concepts are applied.

[7] - The quantity of *golden ratio conjugate* is stated in section 3.4.3.

3 Research Methodology

3.1 Learning Automata

In order to put our work in the right perspective, this section is started by providing a brief review of the main concepts of the Learning Automata first introduced by Tsetlin in [14].

Learning Automata (LA) [11, 15-17, 28, 34] have been utilized in order to design and form biological learning systems. Learning Automata aim to find the optimal action[8] which is suggested by a random Environment. Learning is carried out by clearly communicating with the Environment, and processing its responses to the selected actions, while gradually converging toward a final goal. Two entities, the random Environment and a learning automaton are involved in the learning loop. As mentioned before, a strong feature of Learning Automata (LA) is that it can improve its performance by means of a learning process. LA combine quick and precise convergence with low computational complexity, so in this research LA is utilized as one of the variants of the Hierarchical Stochastic Search on the Line in order to precisely find the optimal value of ψ^*. There exist two main types of LA in the machine learning area, one that is deterministic and one that is stochastic [50]. The former type possesses fixed states, transition and action functions, while the state output of the stochastic LA is decided by certain likelihood. In this research the stochastic type of LA is used. The reader is referred to the books of Lakshmivarahan [11], Narenda and Thathachar [16], Najim and Poznyak [15], and Poznyak and Najim [28] for more information about Learning Automata.

As alluded to earlier, the purpose of the learning process is to discover the optimal value of some parameter $\psi^* \in [0, 1)$. However, the value of ψ^* is unknown to the learning process, it is supposed that the Learning

[8] - The action with the highest reward probability is called optimal action.

Mechanism possesses responses from the *Environment E*, which is able to notify it whether the current estimate ψ is too small or too big.

There exist two different types of Environment—*informative* and *deceptive*. An Environment is cited as "*informative*" if the probability *p* of the Environment, is greater than 0.5. If $p < 0.5$, the Environment is stated as "*deceptive*," this means that the Environment will generate erroneous feedback more than correct feedbacks. Finally, the Environment is a compulsive liar, if the probability of a correct feedback *p* of the Environment approaches zero.

3.1.1 The Operation of an Automaton

An automaton remembers which actions are "good" by maintaining a *state* $S_t \in \{ S_1, \ldots, S_n \}$.

Operation of an automaton is as follows:

*"*1. Selects and outputs an action based on its present state

2. Takes a response from the Environment as input

3. Changes its state based on (a) the response and (b) the action performed (see Figure2)*."* [54]

An automaton can be said to *learn* if it reduces the number of *Penalties* received as a result of interacting with the Environment. Indeed, the automaton makes use of the Environment feedback and the knowledge obtained in the past actions to decide which the next action is.

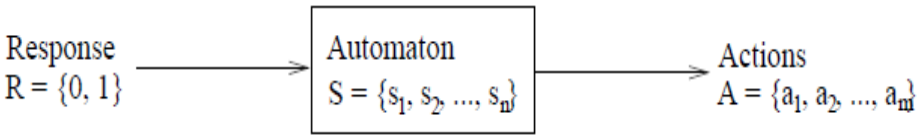

Figure 2: Operation of an Automaton [54]

In order to clarify the LA definitions, the ε-optimality concept of LA and two other different types of LA, namely discretized and continuous LA is described in following.

It is assumed that P_b is the probability that the learning automaton selects the best action [31]. A learning automaton is ε-optimal if $P_b(t) \rightarrow 1$ as $t \rightarrow \infty$ [31]. In other words, when the time goes to infinity, the learning automaton will finally find out the correct answer. Hence, a learning automaton is called ε-optimal if the probability of selecting the best action can be made as close to unity as desired, even if the automaton is not able to find the best action with probability of unity.

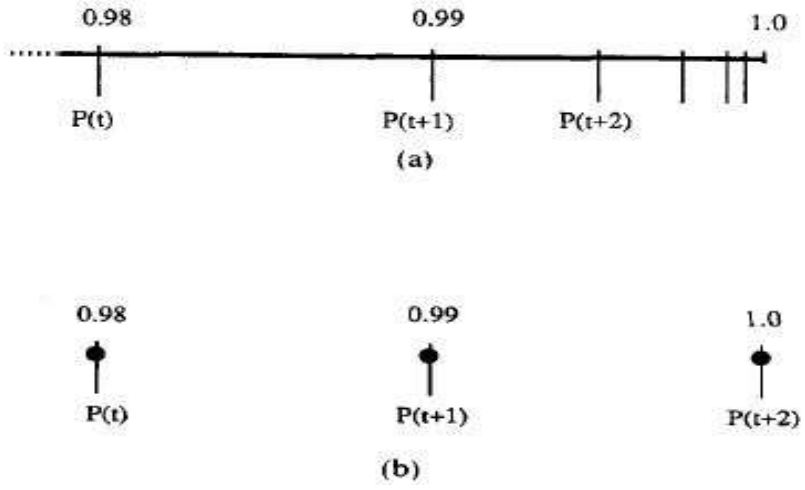

Figure 3: Comparison of probability of choosing optimal action for continuous and discretized learning algorithm. (a) Continuous case. (b) Discrete case. [31]

From Figure 3 it can be observed when the optimal action has been discovered, say a time *t*, and the corresponding action probability is close to unity, the discretized automaton will increase the probability of selecting that action to the value of unity directly, instead of coming near to the value unity asymptotically [31]. Figure 3 depicts that for the continuous case, *A*, if the probability of selecting the optimal action is at 0.98, and the automaton achieves reward for five times, then at time *t +5* the probability of selecting the optimal action will come near to unity. Indeed, the value will depend on the scheme parameters. In the discretized case, *B*, the probability of selecting the optimal action will be unity if the probability space is partitioned into intervals of width 0.01 [31]. In general, the speed of convergence utilizing this method is substantially improved. Discretization is also profitable when it concerns issues related to implementation and representation. Since this type of algorithms utilizes integer (as opposed to real number) representations, they allow addition (as opposed to multiplication) operations. Some of the existing outcomes about discretized automata are considered in [27, 29, 30, 31, 32, and 33]. Therefore, discretized learning automata are employed in this research.

In learning systems, a *Reward-Inaction* learning automaton (L_{RI}) is an effective type of LA compared to *Reward-Penalty* counterpart; hence it is chosen for this research and is described in the following section.

3.1.2 *Reward-Inaction* Learning Automata

Making use of *Reward-Inaction* model of Learning Automata causes the algorithms to approach the optimal action, since actions are chosen near to optimal point. In this model only the optimal actions influence the probabilities of action and non-optimal actions are disregarded.

Challenges included in biasing two components of *Reward* and *Penalty* in the *Reward-Penalty* technique has become one of the most significant reasons of the popularity of the *Reward-Inaction* solution. A normal *Reward-Penalty* algorithm can be specified as follow [18]:

$$Pi(n+1) = Pi(n) + a.(1-\beta(n))(1-Pi(n)) - b.\beta(n)\,Pi(n) \quad ; \text{for the chosen action}$$

$$Pj(n+1) = Pj(n) - a.(1-\beta(n))\,Pj(n) + b.\beta(n).[(r-1)-1-Pj(n)] \quad ; \text{for the other action}$$

where β is the response from the Environment in the interval [0, 1], and constants *a* and *b* are the *Reward* and *Penalty* parameters, respectively. In this formula if $b=0$ then the model is stated as *Reward-Inaction* in which the effects of non-optimal actions are ignored. In other words, if Learning Automata selects an *action* and achieves *Reward* response from the Environment then it increases the probability of choosing this *action* at the next time instant and decreases the probability of the other *action* that has not been chosen, otherwise if the Learning Automata obtains *Penalty* feedback from the Environment in the new *action*, then the probability of selecting this *action* remains unchanged, hence the algorithm is called *Reward-Inaction.*

3.2 Discrete Optimization Problem

Discrete optimization is a branch of optimization in applied mathematics and computer science. Unlike continuous optimization, the variables used in the mathematical program (or some of them) are restricted to assume only discrete values, such as the integers [51]. The main goal of a discrete optimization operation is finding the maximum (minimum) using the local information that is accessible.

Discrete optimization has a significant position in modeling and analyzing of discrete event systems. Configuration design of distributed computer systems, VLSI design, the routing design in commutation networks, and many scheduling schemes in communication networks are examples of the discrete optimization problem [53]. A common property of the discrete optimization problems in mentioned systems is that they are NP-complete[9]. In order to solve this deficiency one has to relax the objective somewhat. For instance, it should be only queried for algorithms that can supply a good model with high probability. In fact, this is a good compromise, since in most positions if a poor system is chosen by the algorithm, it normally can be identified by inspection.

In order to familiarize the reader with the definition of discrete stochastic optimization problems, one example is cited in the following.

Consider the following optimization problem

$$Min\ f(n) = E\{X_n\} \quad ; n \in \mathbb{N} \qquad (1)$$

where $\mathbb{N} = \{1, 2, \ldots\}$ and $\{ X_n \}$ is a sequence of random variables [52]. Suppose $\mathcal{S} \subset \mathbb{N}$ demonstrates the set of local minimizers of the function f. When the expected values of random variables, $E\{X_n\}$ where $n = 1, 2, \ldots,$, can be analytically estimated, standard integer developing methods can be applied in order to solve the optimization problem (1) [52]. A new

[9] - "NP-complete is a class of decision problems. A decision problem *L* is NP-complete if it is in the set of NP (Non-deterministic Polynomial time) problems, so that any given solution to the decision problem can be verified in polynomial time". [56]

technique is proposed in [52] that is modeled to place an element of the set $\mathcal{S}$ in the position where the expected values $E\{ Xn \}, n = 1, 2, ...$, cannot be analytically estimated. This new method is similar to a stochastic approximation algorithm in the following definition: when employed to resolve a minimization problem, a stochastic approximation algorithm, such as the Robbins-Monro algorithm (Robbins and Monro 1951) and the algorithms introduced by Andradottir (1995, 1996a) produces a sequence of the solution guesses, where each new guess is achieved from the previous guess by taking a small step through the direction of the negative gradient of the objective function [52]. As this is the way through which the objective function will be diminished the fastest, a logical description of these algorithms is that each step includes of determining a "good" direction, that is a direction through which researchers assume the objective function to diminish, and then taking a small step through this direction. One simple algorithm for solving discrete stochastic optimization is presented in the following, which is based on the same principle.

Since the objective function is specified on a discrete set, gradient information is not applied for determining a "good" direction. However, as the set of possible approaches is $\mathbb{N}$, natural integers set, so at each possible point, there exist two directions to select: *up* or *down*. Hence, in each iteration, the algorithm is expected to determine whether to take a small step up or down or to resides at the same position. Since the length of the smallest step that can be taken is one, if $n > 1$ is the current position, then it should be specified whether n or one of its neighbors $n - 1$ or $n + 1$ is the next position (e.g. if the current position is 1, then it should be identified whether 1 or its neighbor 2 is the next position). This is obtained by selecting at random either $n - 1$ or $n + 1$ as a next position, then running the scheme at n and its selected neighbor to specify at which position the objective function has a lower value, and then letting the next position be the better of the two. The reader is referred to [52] for more details about this method.

In the rest of this section, two more algorithms for solving discrete stochastic optimization are briefly described. The first algorithm is proposed by Yan and Mukai [53], which is a random search technique and is cited as the Stochastic Ruler (SR) algorithm. This algorithm compares observations of the objective function values with observations of a uniform random variable, called the "Stochastic Ruler," whose range contains the range of the observed objective function values. The number of such comparisons will be increased when the number of iterations grows.

For the sake of comparing observations, the Stochastic Ruler method proposes neighborhood structure. The candidate of the next step will be selected from the neighbor set of current decision parameter. The algorithm makes use of the elements in neighbour set to estimate the optimal solution. In other words, the Stochastic Ruler technique includes generating up to M_K observations of the objective function values in iteration k, where $M_K \to \infty$ *as* $k \to \infty$ [53]. One deficiency of this method in practical implementations is that the method is very sensitive to the rate at which the sequence $\{M_K\}$ is increased. If the sequence $\{M_K\}$ is increased quickly, then the Stochastic Ruler technique may end up at a local solution, whereas if the sequence $\{M_K\}$ is increased slowly, then the technique will take a long time in order to converge. However, Yan and Mukai [20] supply guidelines for how the sequence $\{M_K\}$ should be chosen, choosing these guidelines in practical implementations is hard, since they rely on parameters that are commonly unknown. Besides, the experimental results of Alrefaei and Andrado´ttir [2] imply that mentioned guidelines do not end to a good execution of the method, even if these guidelines guarantee the convergence of the Stochastic Ruler method.

The second algorithm is introduced by Gong, Ho, and Zhai (1992) called Stochastic Comparison (SC) method [53]. It utilizes a growing amount of computer work per iteration when the number of iterations is growing. Stochastic Comparison algorithm compares estimated objective function values at neighbouring points.

One shortage of this method is that if the objective function is definitely determined, then the SC algorithm converges insignificantly to the optimum. Moreover, the analysis of convergence will be complex when objective function has to be estimated.

Solving Stochastic Point Location (SPL) problem which is a sample in discrete stochastic optimization is the goal of the thesis project. Hence, in order to make the reader familiar with the SPL definition, next section is included below.

3.3 The Stochastic Point Location Problem

3.3.1 Defining SPL problem

As alluded to previously, the "Stochastic Point Location Problem," was first proposed in the work of Oommen *et al.* in [15]. The Stochastic Point Location (SPL) problem [15]–[17] is a general learning problem in which the Learning Mechanism (which is either a robot or a learning automaton) aims to find a "parameter," for instance, ψ^*, in a closed search interval. It is considered the problem of a Learning Mechanism (LM) walking on the line trying to place a special point. The LM communicates with an Environment that provides stochastic (i.e., erroneous with a given probability) feedbacks. Hence, when LM should really be going to the "*Right*" direction it may be recommended to go to the "*Left*" and vice versa [13, 16].

In Point Location problem, if the Environment is deterministic, the problem is called "Deterministic Point Location Problem," which has been investigated in [19, 20]. In fact, the SPL problem contains searching without certainty, since the target is unknown, and the only case that is known is the robot's movement direction.

SPL possesses potential applications in solving optimization problems. The aim of the optimization problems is to carry out a task with the minimum cost or with the maximum profits. If the cost function (or profit function) is known, the problem is normally one of minimizing (maximizing) this function. In many optimization solutions—for example in pattern recognition [15], [21- 27], [49], the algorithm executes its way from its "current" action to the optimal action based on its achieved information.

Indeed, the problem of a stochastic learning automation interacting with a random Environment is investigated in this research. The main parts of the learning schemes are executing a controlled random walk on the search interval and placing a position in this closed interval in order to find the optimal action with the highest probability of success.

3.3.2 Continuous Point Location with Adaptive d-ARY Search

In the investigation of [20], Oommen *et al.* presented the Continuous Point Location with Adaptive d-ARY Search (CPL-AdS), which is a generalization of a portion of the work in his previous paper [19]. In [19] Oommen *et al.* divided the search interval into three disjoint subintervals, while in CPL-AdS, the given search interval is partitioned into *d* partitions displaying *d* disjoint subintervals, where $d > 3$. In each interval, primarily, the midpoint of the given interval was determined as the estimate of the unknown ψ*. Each of the *d* partitions of the interval is independently traversed using an ε-optimal two-action Learning Automata (LA), where the two actions are those of choosing a point from the left or right half of the partition. Then, the paper researchers omit at least one of the subintervals from being searched further, and recursively seek the rest of the pruned adjacent interval until the seek interval is at least as small as the required resolution of estimation. This omission process basically makes use of the ε -optimality feature of the automata and the monotonicity of the intervals to ensure the convergence. At each epoch including of a definite number N_∞ of iterations, the algorithm "confidently" discard regions of the search space.

Generally, there exist two basic weaknesses of these later two algorithms [19, 20]. First, both algorithms work with the assumption that the parameter ψ* is time-invariant. In this case, they are not able to manage with non-stationary features where the parameter ψ* changes over time. Suppose that ψ* is a time-variant parameter. Hence, if ψ* modifies to a new value contained in an interval that was already eliminated, both the tertiary search [19] and CPL-AdS [20] will not succeed to converge to the optimal parameter ψ*. Contrary, our second novel variant of the HSSL, i.e., *Multilevel jumps based solution to HSSL*, proposed in this research is shown to be able to work with non-stationary features where ψ* is time varying.

Besides, the two algorithms stated in [19] and in [20] rely on the ε-optimal feature of the individual LA, so both of them are *error prone*. Indeed, at each epoch of the algorithms proposed in [19], [20], it is required to execute each individual LA for an *infinite* number of iterations in order to assure its convergence to its optimal action with probability 1. Thus, the smaller number of iterations may lead to the higher probability to converge to a wrong interval and discard the interval of interest that contains ψ^*. In fact, in order to increase the confidence of the search process at each epoch, a considerable number of iterations per epoch are necessitated, leading to a slow speed of convergence.

3.4 Combining the Field of Binary Search and the Stochastic Point Location Problem

Since in this research, our novel approaches for solving the Stochastic Point Location (SPL) problem are performed in the search space prepared in the form of a binary tree with depth $D = log_2 (N)$, where N is the resolution of the scheme, so in the rest of this chapter the construction of hierarchy, Environment feedbacks and transitions in the tree is described. By using a tree structure of the search space, the Learning Mechanism (LM) rapidly travels the search space and concentrates its movements on the district that includes ψ^*.

For convenience, in our proposed solutions some definitions and notations of [34] and [36] are used in order to describe the hierarchical structure of the search space and to index the nodes in the binary tree. These definitions and notations are explained in the following sections.

3.4.1 Definitions

Hierarchy Structure. Suppose $\Delta = [\alpha, \beta)$ is the current search interval including ψ^*. It is assumed that $\alpha = 0$ and $\beta = 1$, so LM organizes a controlled random walk on a unique search interval. The following parts describe the main concepts of the hierarchical search space. The hierarchy is managed as a binary search tree with maximum depth *D*. To each node of the binary search tree in the hierarchy it is associated an interval containing three values, i.e., left and right boundary values (α and β) along with the middle value of the interval ($0.5*(\alpha+\beta)$).

Root Node. The interval $\Delta = \Delta_{\{0,1\}} = [0, 1)$ is assigned the root node (at depth 0), which it is denoted by $I_{\{0,1\}}$. This interval is subdivided into two separate equi-sized intervals $\Delta_{\{1,1\}}$ and $\Delta_{\{1,2\}}$, that $\Delta_{1,1} = [0, 1/2)$ and $\Delta_{1,2} = [1/2, 1)$. It should be mentioned that $1/2 = mid\ (\Delta_{\{0,1\}})$, where $mid\ (\Delta_{\{0,1\}})$ determines the middle point of $\Delta_{\{0,1\}}$. Furthermore, the interval $\Delta_{\{1,1\}}$ is presented as the *Left Child* of the root node and $\Delta_{\{1,2\}}$ as the *Right Child*.

Leaf Node at Depth *D*. At depth *D*, which defines the maximum level of the binary search tree, the nodes have no children.

For a node *i* at level *d* connected to the specific interval $\Delta_{\{d,i\}}$, it can be inferred the left and right boundaries of the interval, i.e., $\alpha_{\{d,i\}} = (i-1)(1/2)^d$ and $\beta_{\{d,j\}} = i(1/2)^d$, for $i \in \{1, ..., 2^d\}$ where $0 \leq d \leq D$.

Intermediate Node at Depth *d*. Node $i \in \{1, ..., 2^d\}$ at depth *d*, denoted $I_{\{d,i\}}$, where $0 < d < D$, is assigned the interval $\Delta_{\{d,i\}} = [\alpha_{\{d,i\}}, \beta_{\{d,i\}})$ which is subdivided into two separate equi-sized intervals $\Delta_{\{d+1,2i-1\}}$ and $\Delta_{\{d+1,2i\}}$. Therefore, $\Delta_{\{d+1,2i-1\}}$ is the *Left Child* and $\Delta_{\{d+1,2i\}}$ is the *Right Child* of $\Delta_{\{d,i\}}$.

Remark. Since level "*D+1*" is not employed, so the *Right Child* and the *Left Child* of a leaf node are the same as the leaf node. It can be cited that *Left Child*($I_{\{D,i\}}$)=*Right Child*($I_{\{D,i\}}$)= $I_{\{D,i\}}$ for $i \in \{1, ..., 2^D\}$ depth. Besides, "-1" is not utilized in the binary tree, so the *Parent* of $\Delta_{\{0,1\}}$ is illustrated by $\Delta_{\{0,1\}}$. In other words, *Parent* of the interval $\Delta_{\{0,1\}}$ is $\Delta_{\{0,1\}}$.

Scheme Resolution. The scheme resolution is applied in order to determine the number of leaf nodes, i.e., $N = 2^D$.

Target Node. The leaf node that is associated with an interval includes ψ^* is stated as the target node.

Non-Target Node. The leaf nodes that associated intervals do not include ψ^* are presented as non-target nodes.

Using the middle point of each interval as an estimate of the unknown ψ^* is recommended to the learner which resides at a certain node in the binary tree. Indeed, when a LM is standing at a leaf node, the guess of ψ^* will select a discretized value among the N following values [36]:

$$\{mid(\Delta_{\{D,1\}}), mid(\Delta_{\{D,2\}}), \ldots, mid(\Delta_{\{D,N\}})\} = \{(1/2)^{D+1}, 3(1/2)^{D+1}, 5(1/2)^{D+1}, \ldots, (2N-1)(1/2)^{D+1}\}$$

When LM is standing at a node of depth d i.e., at intermediate node, where $0 < d < D$, a discretized value among the N_d (where $N_d = 2^d$) following discretized values will be taken as the guess of ψ^* [36]:

$$\{mid(\Delta_{\{d,1\}}), mid(\Delta_{\{d,2\}}), \ldots, mid(\Delta_{\{d,N_d\}})\} = \{(1/N_d)-(1/(2N_d)), (2/N_d)-(1/(2N_d)), (3/N_d)-(1/(2N_d)), \ldots (N_d/N_d)-(1/(2N_d))\}$$

3.4.2 Construction of the Search Space and Feedbacks from the Environment

As alluded to previously, in this research the search space is organized as a balanced binary tree, where an interval is assigned for each node. In the first step of scheme running, the middle point of the given interval is suggested as the estimate of the unknown ψ*. The LM looks for the best value of ψ* by executing a random walk on the tree-structured search space, going from one node of the tree to another. As delineated in Figure 4, each node of the tree is matched with an interval; e.g., the root is matched with the unique interval [0, 1). This interval is subdivided into two separate equi-sized parts. Therefore, the left child of the root is associated with [0, 1/2) interval and the right child with [1/2, 1), and so on.

At any running time instance, the LM stands at a node $I_{\{D,i\}}$ of the tree, where $i \in \{1, \ldots, 2^d\}$ and $0 \leq d \leq D$. Finding the next search interval that is suggested to include ψ* is the purpose of the LM which is done by generating a sequence of "informed" estimates. For each estimation, the Environment *E* informs the LM, perhaps erroneously (i.e., with likelihood *p*), which path it should go to achieve the best value of ψ*. Suppose $\Delta_{\{d,i\}}$ is the interval that is matched to the node where the LM stands at the current time instant. The "informed" estimates correspond to a sampling at the boundary values of the search interval $\Delta_{\{d,i\}}$, and at the middle point of the interval: $mid(\Delta_{\{d,i\}})$.

A vector $\vec{x} = [x^1, x^2, x^3]$ demonstrates the sampled points, where $x^1 = \alpha_{\{d,i\}} = (i-1)(1/2)^d$, $x^2 = mid\ \{\Delta_{\{d,i\}}\} = (2i-1)(1/2)^{d+1}$ and $x^3 = \beta_{\{d,j\}} = i(1/2)^d$ [36].

A tuple $\vec{\Omega} = [\Omega^1, \Omega^2, \Omega^3]$ illustrates the feedbacks of the Environment *E* [36]. When $k \in \{1, 2, 3\}$, Ω^k is a random variable which can select either "*Left*" or "*Right*" value. For convenience, *L* will be utilized for determining

the "*Left*" side of the sampled point and *R* indicates the "*Right*" side of the sampled point. As mentioned before, the Environment *E* is supposed faulty, so it recommends the correct path with a likelihood *p* and Ω^k when $k \in \{1, 2, 3\}$ is denoted as follows [36]:

If $\psi^* < x^k$

$$\Omega^k = \begin{cases} L & with\ probability\ p \\ R & with\ probability (1-p) \end{cases}$$

If $\psi^* \geq x^k$

$$\Omega^k = \begin{cases} L & with\ probability\ (1-p) \\ R & with\ probability\ p \end{cases}$$

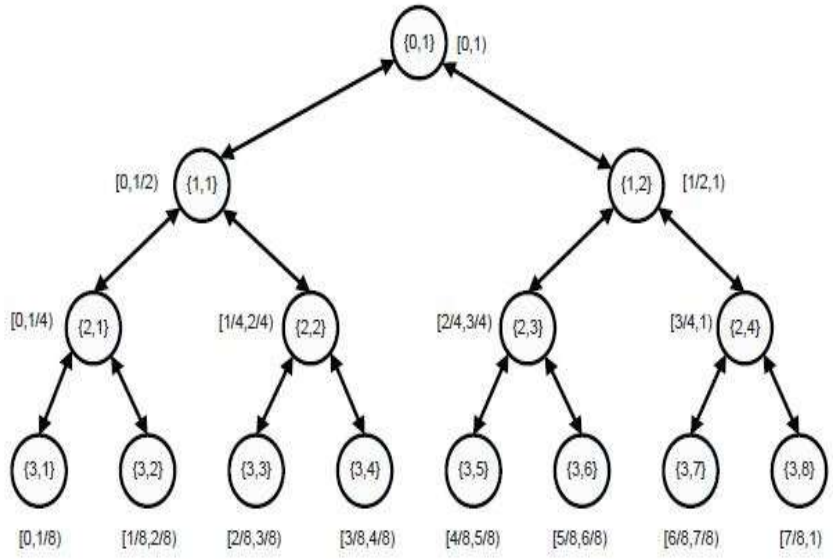

Figure 4: The search space is managed as a binary search tree. Each node $I_{\{i,j\}}$ is associated with an interval $\Delta_{\{i,j\}} = [\alpha_{\{i,j\}}, \beta_{\{i,j\}})$. This interval is sampled at $\alpha_{\{i,j\}}$, $\beta_{\{i,j\}}$, and $mid(\Delta_{\{i,j\}})$, generating one of the eight feasible Environment feedbacks: {[L, L, L], [L, L, R], [L, R, L], [L, R, R], [R, L, L], [R, L, R], [R, R, L], [R, R, R]} [36] .

Remark. In hierarchical stochastic search technique, [L, R, R], [L, L, R], [R, L, R] and [L, R, L] feedbacks are stated as inconsistent and the results are obtained when LM only utilizes the consistent responses. Hence, inconsistent responses are not considered.

3.4.3 Mapping the Feedbacks to Transitions in the Tree

As alluded to earlier, in order to find the predicted value for ψ^*, the middle point of the search interval which is corresponding to the current node in which the LM stands is considered. The main point is that of deciding how to modify the scheme guess ψ of the unknown ψ^* based on the faulty property of the Oracle's feedback. From this viewpoint, the goal is finding a method that determines the next search interval of the LM, which finally ends to defining a set of rules that manages the movements of the LM in way that it proceeds towards the next probable node in the tree (i.e., the one matched to an interval that is expected to include the unknown ψ^*).

The LM proceeds to another node based on these feedbacks, either to the current node's parent, or to one of its children (*Right Child* or *Left Child*). The moving rules in the tree are presented in Table 1 [36].

Next Search Interval	**Condition**
Parent($\Delta_{\{i,j\}}$)	*[R, R, R] ∨ [L, R, R] ∨ [L, L, R] ∨ [L, L, L]*
Left Child($\Delta_{\{i,j\}}$)	*[R, L, R] ∨ [R, L, L]*
Right Child($\Delta_{\{i,j\}}$)	*[R, R, L] ∨ [L, R, L]*

Table 1: Decision table to select the next interval based on the feedback vector $[\Omega^1, \Omega^2, \Omega^3]$**, when** $\Delta_{\{i,j\}}$ **is the current search interval [36].**

Two versions of random walk transitions in the tree are recommended in [36], which are used in our proposed solutions:

- "***Reverse transitions:*** This type of transition corresponds to a movement to a lower level in the hierarchy. This happens when the LM moves to the immediate *Parent* (a larger search interval), which, in turn, allows the LM to escape from getting trapped in a wrong subtree, i.e., one that does not contain ψ*."[36]
- "***Top-down transitions:*** This type of transition corresponds to a movement to a deeper level in the hierarchy. Whenever the LM performs a transition to a deeper level in the hierarchy by choosing a *Child* node, the search space shrinks, and will, hopefully, concentrate on one of the contiguous intervals at the next level of the tree that contains the unknown parameter ψ*."[36]

It has been shown in [36] that the Hierarchical Stochastic Search on the Line (HSSL) method is asymptotically optimal. This has been proved by analyzing the Markov chain properties along with the rules in Table 1. The proof is eliminated due to space limitation. Considering "informed" series of estimates, the LM will focus its movements within nodes in the tree that are corresponded to small intervals including the optimal value ψ* (if *p* is bigger than the conjugate of the golden ratio).

Theorem. "*The parameter learning algorithm specified by the rules summarized in Table 1 is asymptotically optimal if p is bigger than the conjugate of the golden ratio. Formally,*
$Lim_{N\to\infty}\ Lim_{n\to\infty}\ E\ [\psi\ (n)] \to \psi^*$*."* [36] (proof of theorem is available in [36])

Remark. “The golden ratio conjugate quantity [10] is determined by Φ, where $\Phi = (\sqrt{5}-1)/2 \approx 0.61803$. Suppose that the Environment is informative, but its effectiveness p is less than Φ. The issue that p is less than the Φ can be countered by applying a majority voting algorithm. If p is known to the LM, this reduces to finding a minimum number of queries one has to ask to the “Oracle” which ensures that the probability that the majority of responses are correct is bigger than the conjugate of the golden ratio. It should be obvious to the reader that similar reasoning can be applied if p is unknown to the LM, and we only know a lower bound p_{min} of p such that $p_{min} > 0.5$. It will not be elaborated on these ideas here.” [36]

4 Solution

In this section the two novel types of the Hierarchical Stochastic Searching on the Line (HSSL) will be presented, in the first type, the field of *Reward-Inaction* Learning Automata in HSSL is utilized when the Environment is stationary and in the second type, the Multilevel jumps in HSSL solution is applied when the Environment is non-stationary, so the non-consistency feature of the Environment presents the challenging problem. In the HSSL approach, the Learning Mechanism (LM) intends to learn a best value of the parameter ψ^* within a closed interval. In this research, for each estimate the mechanism is informed by the Environment with a probability p of being informed erroneously about which path it should select to achieve the optimum value of ψ^*. Indeed, the suggestion of the Environment for LM in both proposed solutions is uncertain.

The requirements of this project, which will be explained in this chapter, were to use a hierarchical structure of the search space, Multilevel jumps in the tree-based search space, and *Reward-Inaction* Learning Automata (L_{RI}) for selecting an action and searching in the binary tree in order to find the optimal value of the unknown parameter ψ^* (target node) accurately.

4.1 Merging the Field of *Reward-Inaction* Learning Automata and the Original HSSL

In this and next section, two novel approaches for discrete stochastic optimization is proposed and shown how the generalized types of the HSSL technique can be employed in order to solve discrete optimization problems.

The first solution proposed in this chapter is *Reward-Inaction LA based solution to HSSL*, which has the following feature: it converges almost certainly to the best value of the unknown parameter ψ^*. In other words, the proposed method spends most of the computational effort close to the optimal point of the ψ^* in the hierarchical structure of the search space.

Finding the optimal parameter problem in the hierarchical search space could be merged with the Learning Automata (LA) problem. While communicating with the Environment, the LA chooses one action from a set of actions, and the response from the Environment guides the LA if the selected action was rewarded or penalized. Then LA makes use of this information to determine which action to take next, and the cycle continues.

For convenience, in *LA based solution to HSSL*, the same definitions and notations as in [36] are used such as the sampled points of each search interval in hierarchical structure is demonstrated as a vector $\vec{x}=[\,x^1, x^2, x^3]$, where $x^1= \alpha_{\{d,i\}} = (i-1)(1/2)^d$, $x^2= mid\,\{\Delta_{\{d,i\}}\} =(2i-1)(1/2)^{d+1}$ and $x^3=\beta_{\{d,j\}} = i\,(1/2)^d$ and the Environment feedbacks can also be illustrated as a tuple $\vec{\Omega} = [\Omega^1, \Omega^2, \Omega^3]$.

Each node in the tree is associated with a Learning Automaton which has three actions for choosing such as "*Up*", "*Right Child*" or "*Left Child*". The action which has the highest *"*Reward*"* probability is chosen by the LA. However, two exceptions occur at the root node and leaf nodes, i.e., at the

root level, LA can stay at the same level rather than choosing "*Up*" or can move either to "*Right Child*" or "*Left Child*" and at leaf nodes, LA stays at the same node instead of moving to "*Right Child*" or "*Left Child*" or can move to its immediate *Parent.* In this approach, if LA chooses one action e.g. "*Right Child*" and the "Oracle" also responds "*Right*" for the next movement, then LA achieves *"Reward"* from the "Oracle". Since a *Reward-Inaction* LA is used in this research, the LA will increase the reward probability of chosen action when it is rewarded by the Environment.

In order to express our first solution in detail, it should be mentioned that each node is associated with an interval that contains sampled points and each sampled point of the current search interval, i.e., x^1, x^2 or x^3 is attached to one learning automaton. Each learning automaton correctly estimates the direction towards the unknown ψ^* with p and incorrectly with $1-p$ likelihood, where p is the probability of the "Oracle" correctly providing the responses. Hence, there exist three different LAs in the current search interval which select "*Right"* or "*Left*" action based on three different Environment feedbacks, i.e., tuple $\vec{\Omega} = [\Omega^1, \Omega^2, \Omega^3]$. Afterward, by considering the moving rules quoted in Table 1, the LA selects the next search interval and moves to another node, either to the current node's parent or to one of its children (*Right Child*/*Left Child*).

If a learning automaton chooses an action, either "*Right"* or "*Left*", and the Environment also responds the same action, the automaton is rewarded and so it increases the reward probability of chosen action. In other words, at each scheme running, the action with the highest "Reward" probability is chosen for the next step.

The algorithm for updating the probabilities is presented as follows:

Procedure HSSL_LA

Begin

Initialize number of tree levels and Reward parameter of LA

If LA chooses action ω, update P(t) as follows:

IF *$\beta(t)=0$* **THEN**

P(t+1)=P(t)+a(1-P(t))

ELSE

P(t+1)=P(t)

ENDIF

End If

End

where $\beta(t) = 0$ means that the LA is rewarded by the Environment for choosing the action ω at time t and a is the reward parameter of LA, like in the formula in section 3.1.2.

By making use of *Reward-Inaction* LA in our proposed solution, the scheme precisely converges to the interval which has the best value of ψ^* and afterward will not erroneously move to the lower or deeper level. Therefore, the scheme never modify the best obtained value for ψ^* after convergence, since the interval including ψ^* is chosen by LA with the highest "Reward" probability. However, in the original HSSL method introduced in [36], the scheme may move to lower or deeper level, after it finds the best value of the ψ^* and the convergence rate is decreased by the scheme fault.

4.2 Implementing HSSL with Multilevel Jumps

As previously mentioned, another novel approach proposed in this research involves studying how the Multilevel jumps can be made by the scheme. In this section, it is shown that using Multilevel jumps in a hierarchical construction of the search space could lead to a less complex scheme. Although schemes with Multilevel jumps can easily be described, they would, typically, be hard to develop.

In this new approach, the corresponding feedbacks of the Environment *E* can be presented as a tuple or unique. Tuple response means $\vec{\Omega}=[\Omega^1, \Omega^2, \Omega^3]$ quoted previously where Ω^k, for $k \in \{1, 2, 3\}$, is a random variable which can take either the value "*Left*" or "*Right*" and unique response means informing "*Left*" or "*Right*" to the LM. Like in the original HSSL technique [36] and *LA based solution to HSSL*, each node is associated with an interval, which is subdivided into two separate equi-sized parts. At each node the midpoint of the interval is considered as an estimate of the unknown ψ^*. The LM tries to find the next promising search interval that maybe contain ψ^* by generating a sequence of "informed" estimates. Similar to the original HSSL [36] and *LA based solution to HSSL*, in this new approach, the Environment guides the LM probably faulty (i.e., with *p* likelihood), which path it should select to find the best value of ψ^*.

The differences between the original HSSL solution [36] and *Multilevel jumps based solution to HSSL* are listed as follows:

- In the *Multilevel jumps based solution to HSSL* at root node (at depth zero) and at odd levels in the tree ($d= \{1,3,..., D-1\}$[10]), only the middle point (i.e., x^2 in the sampled points) is queried by the LM and the Environment response is unique, either "*Right*" or "*Left*" rather than tuple feedback.

[10] - It is assumed that the tree depth $D = log_2(N)$ is an even number.

- In the *Multilevel jumps based solution to HSSL* only at even levels in the tree (*d= {2,4,…, D-2}*[11]), all three points (i.e., x^1, x^2, x^3 in the vector of sampled points) are asked by the LM and the Environment response is a tuple, i.e., $\vec{\Omega}=[\Omega^1, \Omega^2, \Omega^3]$.
- Unlike the original HSSL solution [36], in this research, for the nodes at depth *d* where *0 < d < D* and *d* is an odd number (*d= {1,3,…, D-1}*), the LM possesses two choices for selecting either "*Right Child*" or "*Left Child*", hence at these levels the LM is not able to move "*Up*" . However, for the nodes at depth *d* where *0 < d < D* and *d* is an even number (*d= {2,4,…, D-2}*), the LM has three actions for choosing; "*Right Child*", "*Left Child*" and "*Up*", means that *reverse transition* for moving to a lower level in the hierarchy could be done at these levels.
- In the original HSSL [36], the *reverse transition* corresponding to the immediate *Parent* (a larger search interval) means one level movement upwards, while in our proposed solution the *reverse transition* is performed for more than one level reflects a Multilevel jump in the hierarchy.
- In the *Multilevel jumps based solution to HSSL*, for the nodes at depth *D* (leaf nodes), when *D* is an even number the LM could stay at the same level or jump to the lower level, while at depth *D* where *D* is an odd number the LM could only stay at the same level, which means that it could not escape from getting trapped in a wrong leaf node, i.e., one that does not include ψ^*. However, in the original HSSL solution [36] for nodes at depth *D*, which *D* could be either odd or even number the LM could stay at the same level or move up for one level when it resides at a non-target node.

[11] - It is assumed that the tree depth $D = log_2(N)$ is an even number.

Figure 5 delineates the hierarchical structure in our second solution; *Multilevel jumps based solution to HSSL* technique, along with the LM movements when the tree depth $D = log_2(N)$ is equal to 4.

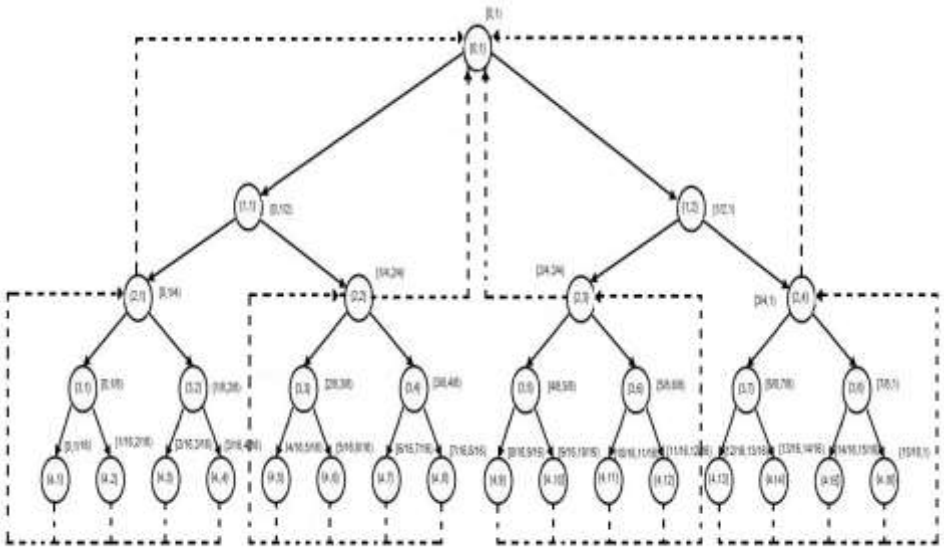

Figure 5: The figure depicts the tree structure of the search space in Multi-level Jumps based solution to HSSL with LM movements in the tree. Each node $I_{\{i,j\}}$ is associated with an interval $\Delta_{\{i,j\}} = [\alpha_{\{i,j\}}, \beta_{\{i,j\}})$. This interval is sampled at $\alpha_{\{i,j\}}$, $\beta_{\{i,j\}}$, and $mid(\Delta_{\{i,j\}})$, creating one type of these Environment feedbacks: *{[L, L, L], [L, L, R], [L, R, L], [L, R, R], [R, L, L], [R, L, R], [R, R, L], [R, R, R]} or {[L],[R]}.*

As alluded to previously, it can be observed from the figure above that a LM at root node (at depth 0) has two actions for moving, "*Right Child*" and "*Left Child*", which means that the LM is not able to move up. Likewise, at odd levels in the tree ($d= \{1,3,..., D-1\}$), the LM has two choices, while at even levels ($d= \{2,4,..., D-2\}$), the LM has three actions, "*Right Child*", "*Left Child*" and "*Up*". Indeed, if the LM moves to an even level in the binary tree, it could move up for two or more than two levels. Hence, our solution is called *Multilevel jumps based solution to HSSL*, since a LM can move up for more than one level when it resides at even levels. However, if

the LM jumps to an odd level in the tree structure, it is not able to move to a lower level and therefore the scheme will not find the optimal action.

5 Simulation Results

In this section, the performance of the proposed methods, *Reward-Inaction Learning Automata based solution to HSSL* and *Multilevel Jumps based solution to HSSL*, through numerical experiments is explained. In order to plot accurate diagrams, MATLAB software is utilized for our experiments in this chapter.

Since there is no previous information about the value ψ^*, at time instant 0, we initialize the first position of LM in our proposed solutions is the root node of the tree.

5.1 Results from Merging the Field of *Reward-Inaction* Learning Automata and the Original HSSL

The *LA based solution to HSSL* described in this research was experimentally assessed to identify the validity of the results and to investigate its convergence rate. To specify the power of the proposed solution and to display its effectiveness for various conditions, the simulation was performed for different values for *p*, the probability of the "Oracle" correctly supplying the response, and for different values for *N*, the scheme resolution.

In this case, the value of the parameter ψ^* was supposed to be unknown to the LA. Although several experiments have been executed, for the sake of brevity, we report the results achieved for one set of experiments including the unknown parameter $\psi^* = 0.947$.

The true value of $E[\psi(\infty)]$ is demonstrated in Table 2. The results achieved from the scheme running after 10^6 iterations and for different values of p and the scheme resolution $N=2^D$ (D, the tree depth is equal to $log_2(N)$), when the reward parameter of LA, a, is 0.05 and value of $\psi^* = 0.947$.

In each case the $E[\psi(\infty)]$ converges accurately. For instance, when p is 0.72 and is N is equal to 64 (i.e., the tree depth, D=6), the value of $E[\psi(\infty)]$ is 0.915. It grows to 0.942 when N = 512 (i.e., the tree depth, D=9). The results are more "accurate" for larger values of p. Hence, when p is 0.93 and N=64, the value of $E[\psi(\infty)]$ is 0.946. The final terminal value when N = 4096 (i.e., the tree depth, D=12) and p is 0.93 demonstrates an error less than 0.0004%. The ε-optimality feature is empirically verified through the simulation, whether the value of p is 0.72 or 0.93, $E[\psi(\infty)]$ infinitely converges to the optimal ψ^* as the scheme resolution is increased.

Scheme Resolution (N)	*P=0.72*	*P=0.93*
64	0.915796875	0.946247195
128	0.92960498	0.947014632
256	0.94155459	0.946710170
512	0.942451001	0.946454641
1024	0.944934351	0.946423120
2048	0. 946055675	0.946737844
4096	0.946134351	0.946954688

Table 2: True value of $E[\psi(\infty)]$ for different values of p, the Environment effectiveness and different resolutions, N, when the value of ψ^* is 0.947 and a=0.05 in LA based solution to HSSL.

In order to determine the convergence of $E[\psi(n)]$ with time, "*n*", in *LA based solution to HSSL*, several experiments were executed in stationary Environment where ψ^* is constant over time. For the sake of brevity, the results of the four experiments are sketched in Figure 6. From the following diagrams it is inferred that different values for *p*, the effectiveness of the Environment, and *N*, the scheme resolution, will affect the scheme convergence.

In Figure 6, it is observed that the *LA based solution to HSSL* converges to a value that is 98% of the optimal value ψ^* and it takes 50 time instants to obtain the optimal action when the N=4096 (scheme resolution) and p (the effectiveness of the Environment) is 0.93, while in the case that N is 256 and p=0.93, the algorithm approaches 98% of the optimal value of ψ^* after 40 time instants. It can be seen from Figure 6, when the resolution of the scheme, *N*, is increased and the effectiveness of the Environment, *p* remains unchanged, then the required time instant for finding the optimal action is also increased, e.g., when N=256 and p=0.82 the LA requires 80 time instants for finding the best value of ψ^*, while in another experiment when N=4096 and p=0.82 the LA needs 110 time instants. However, by increasing the value of *p*, the required time instant for finding the best value of ψ^* is diminished when the scheme resolution remains unchanged, e.g., when N=256 and p=0.82 the LA needs 80 time instants and when N=256 and p=0.93 the LA needs 40 time instants. In these four experiments the reward parameter of LA, *a*, was 0.05.

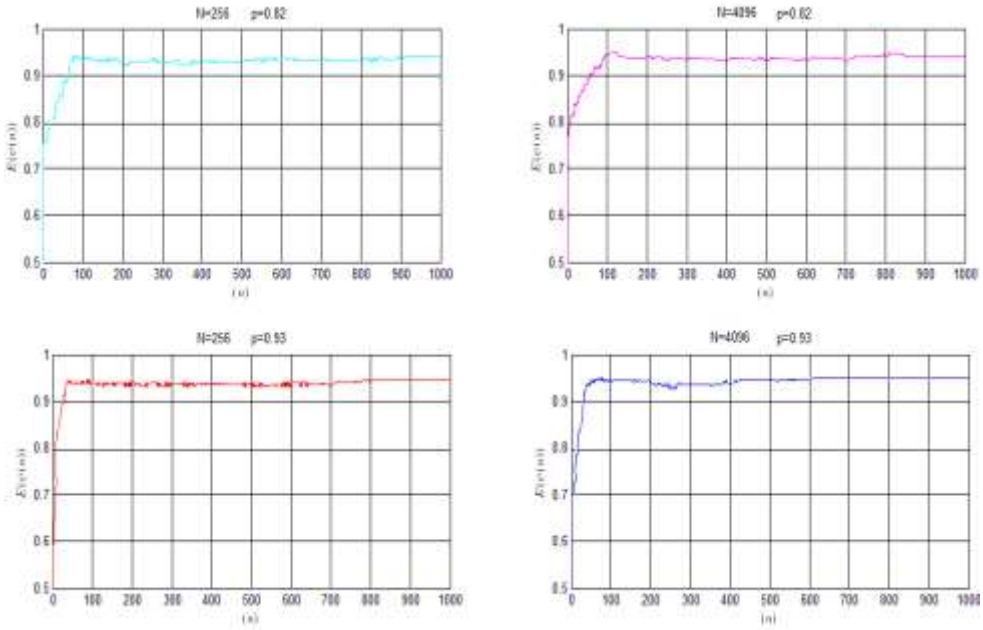

Figure 6: The plots show the convergence rate of *E[ψ(n)]* with time, *"n"*, in LA based solution to HSSL for different values of *p*, the Environment effectiveness and different values of the resolution parameter *N* when ψ*=0.947 and the reward parameter of LA is 0.05.

The result of another performed experiment for *LA based solution to HSSL* is delineated in Figure 7 when N=4096 and p=0.93. In this experiment for finding the best value of ψ*=0.947, different values for a, reward parameter of LA, was tested.

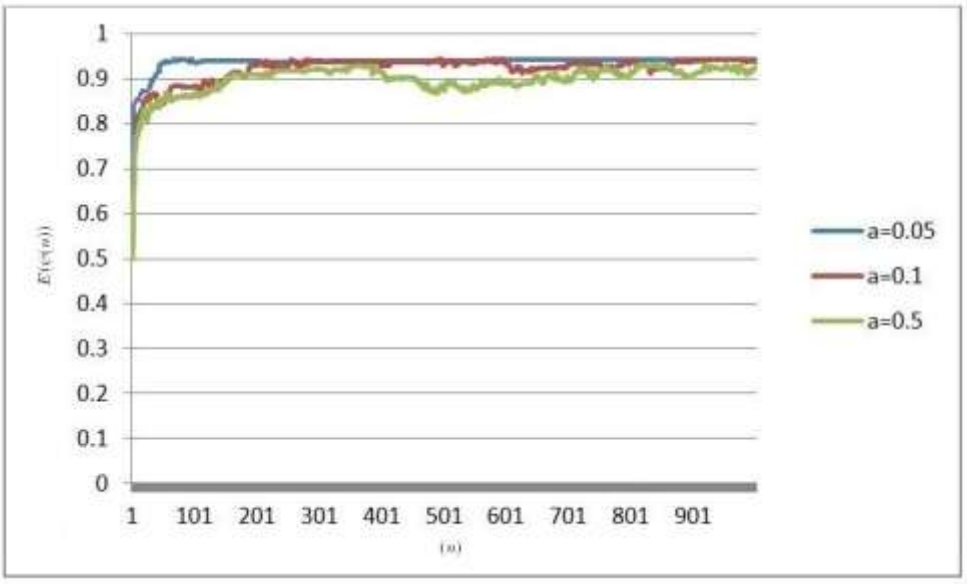

Figure 7: The figure shows the convergence rate of $E[\psi(n)]$ with time, "*n*", in LA based solution to HSSL for different values of *a*, reward parameter of LA, when ψ^*=0.947, *N*=4096, and *p*=0.93.

From Figure 7, it is obvious that the convergence rate of the *LA based solution to HSSL* is diminished by increasing the value of the reward parameter of LA, *a*. Hence, in this research in order to achieve precise results for our experiments of *LA based solution to HSSL*, the minimum value of *a*, i.e., *a*=0.05 is considered.

5.2 Results from Multilevel Jumps Based Solution to HSSL

In order to achieve a clear understanding as to how the *Multilevel jumps based solution to HSSL* converges with time, various simulations were executed to assess the performance of the algorithm under a variety of constraints. As previously mentioned, *Multilevel jumps based solution to HSSL* in this research was executed in a non-stationary Environment where the value of ψ^* is changing over time, and therefore the problem becomes more complex than in a stationary Environment.

As mentioned in previous chapter, in *Multilevel jumps based solution to HSSL,* for the nodes at depth *D* (leaf nodes), when *D* is an odd number the LM is not able to move "*Up*" and is not able to escape from getting trapped in a wrong sub tree, i.e., one that does not include ψ^*. Therefore, in our experiments, it has been tried to use an even number as the tree depth (*D*) for obtaining more precise results.

As in the first experiment, the true value of the $E[\psi(\infty)]$ was tested by using the original HSSL [36], *LA based solution to HSSL* and *Multilevel jumps based solution to HSSL* after 10^6 iterations when the resolution was equal to 4096 (i.e., the tree depth, D=12) and the unknown parameter $\psi^* = 0.947$. In this case, *p*, the probability of the "Oracle" correctly supplying the feedback was 0.93. The results are illustrated in Table 3. In this experiment, for running the *LA based solution to HSSL* the reward parameter of LA, *a*, was equal to 0.05.

Here is a comparative brief overview of the results of the executed experiment:

Scheme Resolution (N)	*LA based solution to HSSL*	*Multilevel jumps based solution to HSSL*	*Original HSSL*
64	0.946247195	0.94279687	0.94524748
128	0.947014632	0.941604981	0.94900677
256	0.946710170	0.945554593	0.946723564
512	0.946454641	0.945251001	0.946300683
1024	0.946423120	0.946134351	0.946163106
2048	0.946737844	0. 946335675	0.946507753
4096	0.946954688	0.946434351	0.946644342

Table 3: Compare the true value of $E[\psi(\infty)]$ by using different variants of HSSL when the value of ψ^* is 0.947 and p=0.93.

It is obvious from Table 3 that using *LA based solution to HSSL* approach yields a more accurate scheme, since it represents an error less than 0.0004% when the scheme resolution is 4096 (i.e., the tree depth is 12). In addition, it is observed from Table 3 that the size of the binary tree was proportional to the performance of HSSL. Meaning that as the scheme resolution, N, (or tree depth, D) got bigger, the better hierarchical search approaches performed, and vice versa. Therefore, the larger number of N leads to more accurate convergence.

We now present the outcomes of the experiments in which we have tried to determine the convergence of $E[\psi(n)]$ with time, "n" by using Multilevel jumps in the original HSSL solution [36]. In this case, the unknown parameter ψ^* changes periodically between the values 0.947 and $1 - 0.947$ means the Environment is non-stationary. From Figure 8, in the first 250 iterations, it can be found out that the *Multilevel jumps based solution to HSSL* takes 70 time instants to converge to a value that is 93% of the optimal value ψ^* when the N=4096 (the scheme resolution) and p (the effectiveness of the Environment) is 0.93, while in the case that N is 256

and p=0.93, the algorithm reaches 93% of optimal action after 65 time instants. In order to clarify the effect of the convergence rate, like in Figure 6 in the previous section, Figure 8 is plotted for different values of N and p.

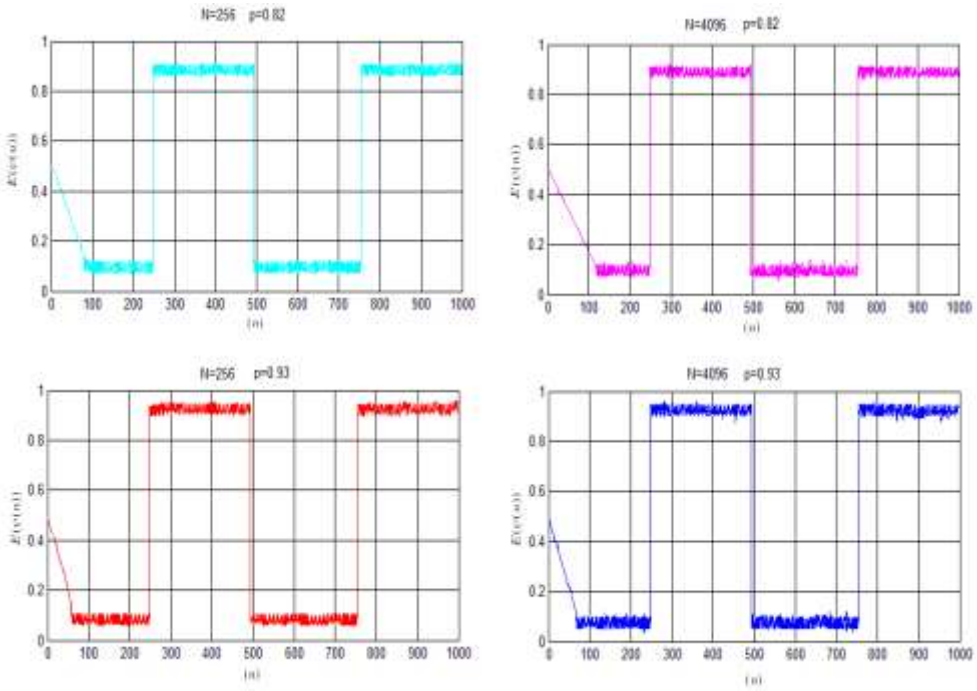

Figure 8: The plot shows the case when ψ* switches between the values 0.947 and 1-0.947 every 250th iteration in Multilevel jumps based solution to HSSL for different values of *p,* the Environment effectiveness and different values of the resolution parameter *N.*

From Figure 8, it can be seen that like Figure 6 when the resolution of the scheme, N, is increased and the effectiveness of the Environment, p, remains unchanged, then the required time instant for finding the optimal action is also increased, e.g., when N=256 and p=0.82 the LM requires 85 time instants for finding the best value of ψ^*, while in another experiment when N=4096 and p=0.82 the LM takes 120 time instants. However, by increasing the value of p, the required time instant for obtaining the best

value of ψ^* is decreased when the scheme resolution remains unchanged, e.g., when N=256 and p=0.82 the LM needs 85 time instants and when N=256 and p=0.93 the LM needs 65 time instants.

With the results shown in the Figure 8, it is observed that in each of the four experiments the required time instant for finding the optimal action is increased by using the *Multilevel jumps based solution to HSSL* compare to the results achieved by *LA based solution to HSSL* shown in the Figure 6.

In another experiment, the estimated value of $\psi(n)$ at time "n" was tested by using the *Multilevel jumps based solution to HSSL* when the scheme resolution, N, was equal to 64 (i.e., the tree depth, D=6) and the unknown parameter ψ^* = 0.947. In this case, p, the probability of the "Oracle" correctly supplying the feedback was 0.75. The simulation result presented in Table 4 obtained by running the learning scheme for 10^4 iterations. Although numerous iterations have been conducted, in the interest of brevity the results achieved for the first 22 iterations is reported in Table 4. Since there is no previous information about the value of ψ^*, at time instant 0, the initial position of LM in the *Multilevel jumps based solution to HSSL* technique is the root node (depth 0) of the binary tree, which is associated with the interval [0,1) and therefore the current interval in Table 4 is also [0, 1). In this step, the estimated value for ψ^* is the middle point of the interval [0, 1). This interval is subdivided into two separate equi-sized parts and in the next time instant, one of the parts is selected as the next interval, e.g., the right child of the root node is associated with [0.5,1) and the left child with [0,0.5). The important issue that is addressed in Table 4 is that of determining how to change the estimated value of the unknown ψ^* based on the erroneous suggestions (i.e., with p probability) of the Environment for choosing right or left child of the node to obtain the best value of the unknown parameter ψ^*. By using the moving rules quoted in Table 1, our scheme decides the next promising node in the tree (i.e., the node associated

with an interval that maybe include the unknown ψ*). Afterward, the LM moves to the deeper or lower level of the tree. As it can be seen from Table 4 and Figure 5, the Multilevel jumps can be performed only at even levels (*d= {2,4,…, D-2}*) of the tree, e.g., at time instant 4, the LM moves upwards (a larger search interval) from level 4 to level 2 in order to escape from getting trapped in a wrong sub tree, i.e., one that does not involve ψ*. In this experiment, from depth 0 (root level) to depth 6 (maximum level), five jumps are performed by the LM in order to obtain the unknown point.

Time instant *(n)*	Estimate value for ψ*	Current interval	The LM movements in the binary tree
0	0.5	[0,1)	The LM resides at depth 0 (at root level) (*d*=0)
1	0.6225	[0.5,1)	The LM moves to level 1 (*d*=1)
2	0.68525	[0.5,0.75)	The LM moves to level 2
3	0.66525	[0.625,0.75)	The LM moves to level 3
4	0.6975	[0.625, 0.6875)	Jump from level 4 to level 2
5	0.737	[0.5,0.75)	Jump from level 2 to level 0
6	0.5	[0,1)	The LM resides at depth 0
7	0.7085	[0.5,1)	The LM moves to depth 1
8	0.791562	[0.75,1)	The LM moves to depth 2
9	0.771562	[0.75, 0.875)	The LM moves to depth 3
10	0.722531	[0.75, 0.8125)	The LM moves to depth 4
11	0.7926562	[0.75, 0.78125)	The LM moves to depth 5
12	0.827687	[0.75, 0.765625)	Jump from level 6 to level 4
13	0.8487343	[0.75, 0.8125)	Jump from level 4 to level 2
14	0.866625	[0.75,1)	The LM resides at depth 2
15	0.874445312	[0.875,1)	The LM moves to depth 3
16	0.902953123	[0.875, 0.9375)	Jump from level 4 to level 2
17	0.911601562	[0.75,1)	The LM resides at depth 2
18	0.920265625	[0.875,1)	The LM moves to depth 3
19	0.9401875	[0.9375,1)	The LM moves to depth 4
20	0.941320314	[0.9375, 0.96875)	The LM moves to depth 5
21	0.942179687	[0.9375, 0.953125)	The LM moves to depth 6

Table 4: The estimated value for ψ* at time, "*n*", in Multi-level jumps based solution to HSSL when the Environment effectiveness, *p*, is 0.75, the resolution parameter, *N, is 64,* and ψ*=0.947.

In Table 4 there exist different values for the current interval at the same level of the tree, e.g., at time instants 11 the LM resides at depth 5 and selects [0.75, 0.78125) as search interval, while for the time instant 20 it chooses [0.9375, 0.96875) at tree depth 5. The reason for this is that for each estimate the Environment informs the LM probably erroneously (i.e., with *p* probability), which path it should move to obtain the unknown point and therefore the LM moves to a different interval at the same depth.

Another plot of the asymptotic value of $E[\psi(\infty)]$ as a function of p (the effectiveness of the Environment) in *LA based solution to HSSL*, *Multilevel jumps based solution to HSSL* and the original HSSL [36], is illustrated in Figure 9. The Figure 9 delineates the accuracy rate of the three mentioned algorithms when the scheme resolution is 4096 (i.e., the tree depth, D=12). In the Figure 9, it is observed that the scheme obtains a higher accuracy by using *Reward-Inaction* LA in HSSL to find the best value of the unknown parameter, ψ^*, since the LA converges to the interval containing the best value of ψ^* and the reward probability for the chosen action is close to one. In other words, *LA based solution to HSSL* converges to the optimal action at target node and never comes to lower or deeper levels in the hierarchical structure after convergence, while in Multilevel jumps approach and in the original HSSL [36] the LM will probably move to lower or deeper levels after obtaining the optimal action and therefore the scheme accuracy is diminished and the running time is increased. The simulation result depicted in Figure 9 was achieved by running the learning scheme for 10^6 iterations. It should be mentioned that in Figure 9, all three algorithms were executed in a stationary Environment and the reward parameter of LA, *a*, in *LA based solution to HSSL* was 0.05. The experiment illustrates how $E[\psi(\infty)]$ will change as the effectiveness of the Environment, *p*, modifies from $p = 0.65$ to $p = 1$.

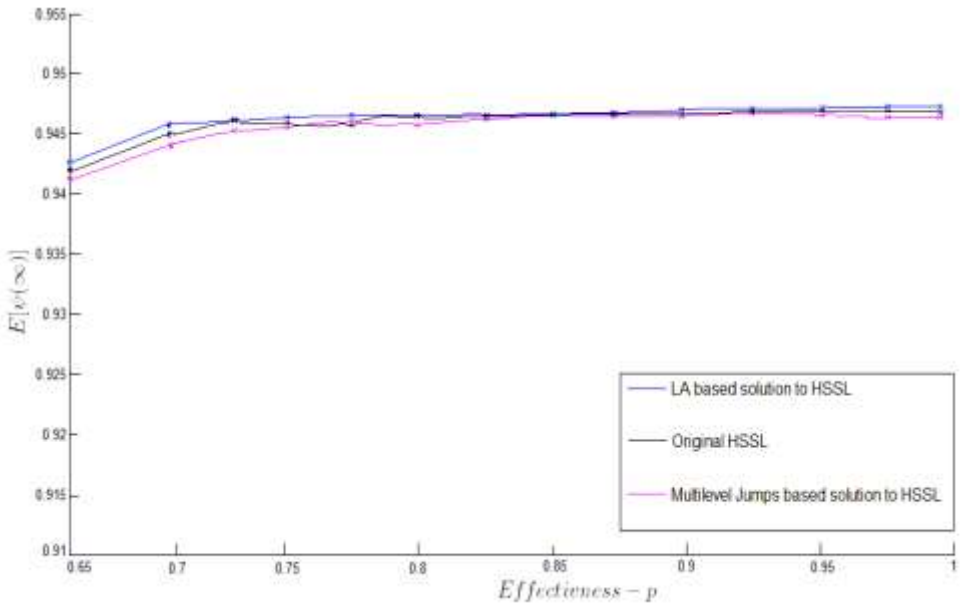

Figure 9: Comparison of asymptotic value of $E[\psi(\infty)]$ with the effectiveness of the Environment, *p*, for different variants of HSSL in a stationary Environment when the value of ψ* is 0.947 and *N*=4096.

Finally for the sake of determining the convergence rate of $E[\psi(n)]$ with time "*n*", in *LA based solution to HSSL*, *Multilevel jumps based solution to HSSL* and *the original HSSL* [36] another plot is depicted in Figure 10. In this experiment the effectiveness of the stationary Environment, *p*, was 0.93 and the scheme resolution, *N*, was 4096 (tree depth, *D*, is equal to 12). Like in the previous graph, the reward parameter of LA, *a*, was 0.05 and ψ*=0.947.

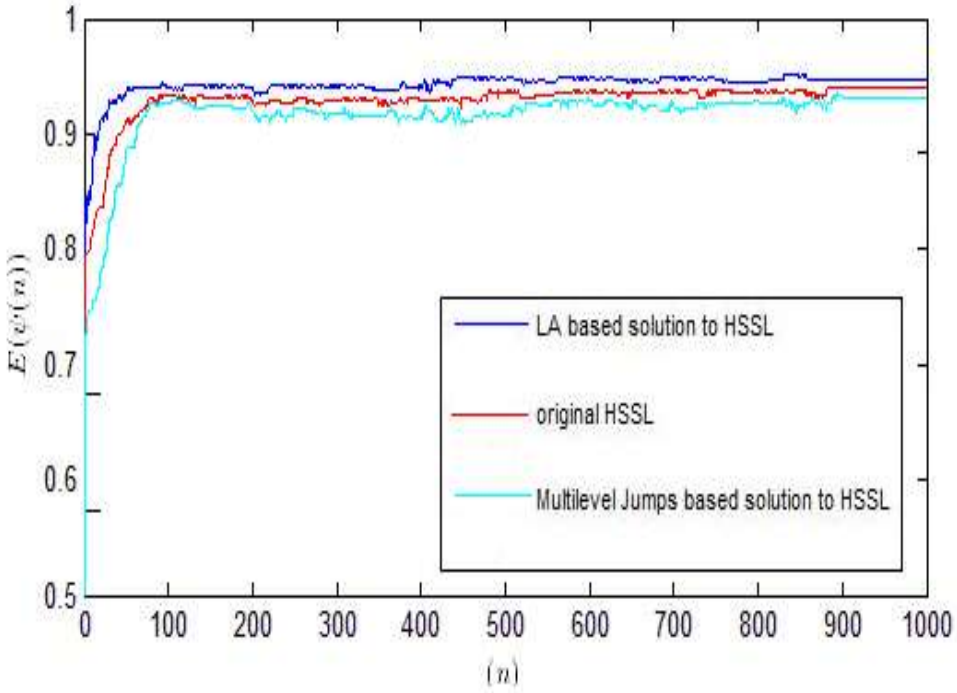

Figure 10: Comparison of the $E[\psi(n)]$ convergence rate with time, "n", for different variants of HSSL in a stationary Environment when the value of ψ^* is 0.947, N=4096 and p=0.93.

6 Discussion

By using *LA based solution to HSSL* and *Multilevel jumps based solution to HSSL* algorithms in this research, rather interesting results for solving the discrete stochastic optimization problem was achieved. Discretizing the search space into a binary tree and executing a controlled random walk on this search space was an efficient way to find the optimal action.

The new *LA based solution to HSSL* excelled primarily in solving the SPL problem. This observation is expected initially due to the Learning Automata operation's unique mechanism of accurately finding the optimal value of the unknown parameter, ψ^*.

Indeed, the *LA based solution to HSSL* is a clear winner over the two other algorithms when it comes to convergence accuracy. While *Multilevel jumps based solution to HSSL* and the original HSSL [36] approach the optimal action with less precision, since the LM still can visit the rest of the nodes in these algorithms after finding the optimal action. In other words, the most significant feature of *LA based solution to HSSL* is that it can converge to the optimal action at the target node, and never moves on to another node.

However, a drawback of *LA based solution to HSSL* is that LA is not able to cope with a non-stationary Environment, while *Multilevel jumps based solution to HSSL* and the original HSSL [36] are performed in non-stationary Environment where ψ^* is time varying.

Multilevel jumps based solution to HSSL is a strategy which adopts Multilevel jumps in the hierarchical structure of the search space. The advantage of using Multilevel jumps in HSSL is that the problem becomes less complex than the *LA based solution to HSSL* and the original HSSL [36]. Since, in the Multilevel jumps alternative, for each estimate at odd levels in the tree ($d= \{1,3,\ldots, D-1\}$[12]), the LM queries from the midpoint

[12] - It is assumed that the tree depth $D = log_2(N)$ is an even number.

and the Environment essentially guides the mechanism with one response rather than three feedbacks in the *LA based solution to HSSL* and the original HSSL [36].

However, using Multilevel jumps in HSSL solution leads to slower convergence of the scheme. The reason for this is that the LM should move up for two or more levels to find a larger search interval and start searching from that level which is an even level ($d= \{2,4,\ldots, D-2\}$[13]). Indeed, the LM should jump to an even level, so an extra attempt must be done for moving to the contiguous node at the same level of the tree when the target node is not achieved.

Generally, based on the obtained results in this investigation, the conclusive answer to the question stated in the research questions section (section 1.5), which was about whether the integrated methods, using LA and Multilevel jumps in HSSL, make the stochastic point locating more efficient or not is "Yes". Besides, using *LA based solution to HSSL* for solving SPL problem is definitely recommendable. The results from this technique are truly near to the unknown parameter ψ^* which indicates that the algorithm is quite stable and the results are not widely spread around the optimal point, but close to it.

[13] - It is assumed that the tree depth $D = log_2(N)$ is an even number.

7 Conclusion and Further Work

In this work, the problem was to solve the Stochastic Point Location (SPL) problem by applying a hierarchical stochastic search technique - HSSL - and combining the latter with two novel approaches - Learning Automata and Multilevel Jumps. Afterward, investigating whether these combinations provide better results in terms of solution accuracy and computational efficiency while solving SPL. SPL is a Nondeterministic Polynomial time complete (NP-complete) problem which LM attempts to learn a parameter ψ^* within a closed interval. In this research, for each estimate, the Environment guided probably erroneously (i.e., with *p* likelihood) the LM, which way it should select to attain the best value of the unknown parameter ψ^*. Hence, unlike the deterministic binary search, an interesting case was examined where the feedback of the Environment was supposed to be faulty.

In both proposed solutions the search space was implemented in a hierarchical construction like a binary tree and a controlled random walk was performed on this search space. The introduced solutions have been simulated, with interesting results.

LA based solution to HSSL was investigated in this research as our first novel alternative which converged almost certainly to the best value of the unknown parameter ψ^* in a stationary Environment. Indeed, *LA based solution to HSSL* provided a more accurate convergence than the original approach [36] for solving the discrete stochastic optimization. In this technique, the Environment informed the LA with three responses, each maybe erroneous with probability *p*.

Our second new solution proposed in this investigation was *Multilevel jumps based solution to HSSL* which was shown to supply a less complex algorithm than the SPL solution introduced in [36] when tested in non-stationary Environment where ψ^* is changing over time. In this novel

approach, a LM at the even levels ($d= \{2, 4,..., D-2\}$[14]) in the tree structure based search space, has three choices for moving down or up (i.e., "*Right Child"*, "*Left Child"* or "*Parent"*). The main difference between Multilevel jumps solution and the original HSSL in [36] is that the former allows transitions across multiple levels, rather than only between consecutive levels, hence the LM is able to move up for more than one level. The other difference is that the scheme complexity diminishes by using Multilevel jumps in the hierarchical search space. Therefore, at the odd levels ($d= \{1,3,..., D-1\}$) in the hierarchical structure, LM asks one question from the midpoint of the search interval in order to select either "*Right Child"* or "*Left Child"* and therefore there exists only one feedback for LM from the Environment.

Finally, based on the results achieved, our hypothesis got significantly strengthened; combining the LA operation and Multilevel jumps with existing approach did indeed increase the efficiency of solving SPL problem. As steps for further work, it is worth mentioning that combining *LA based solution to HSSL* with Multilevel jumps in a non-stationary Environment will show great promise in order to implement an accurate and simple scheme for solving the SPL problem, and is definitely worth investigating. Furthermore, it is recommended to research the use of the *LA based solution to HSSL* and *Multilevel jumps based solution to HSSL* in order to solve practical stochastic optimization problems.

[14] - It is assumed that the tree depth $D = log_2(N)$ is an even number.

Bibliography

[l] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines.* John Wiley & Sons, Inc., 1989.

[2] B. Hajek. *A Tutorial Survey of Theory and Applications of Simulated Annealing* . In *Proc.* of the 24th *CDC,* Ft.Lauderdale, Florida, December 1985.

[3] D.J.C MacKay. Bayesian Model Comparison and Backprop Nets. In J. E. Moody, S. J. Hanson, and R. P. Lippman editors, *Advances in Neural Information Processing Systems 4*. Morgan kaufmann, April 1992.

[4] D. L. Isaacson and R. W. Madsen. *Markov Chains: Thery and Applications.* John Wiley & Sons, Inc., 1976.

[5] D. Mitra, F. Romeo, and A. Sansiovanni Vincentelli. *Convergence and Finite-Time Behavior of Simulated Annealing*. In Proc. of the 24th CDC , Ft.Lauderdale, Florida, December 1985.

[6] D. Yan and H. Mukai. *Stochastic Discrete Optimization*. SIAM Journal on Control and Optimization, 30, May 1992.

[7] D. A. Cohn, Z.Ghahhramani, and M. I. Jordan. *Active learning with statistical models*. In G. Tesauro, D. Touretzky, and T. Leen, editors, Advances in Neural Information Processing Systems 7. MIT Press, 1995.

[8] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Pecipes in C*. Cambridge University Press, 1992.

[9] http://stackoverflow.com/questions/700241/what-is-the-difference-between-linear-search-and-binarysearch

[10] M. Livio. *The Golden Ratio: The Story of Phi, the World's Most Astonishing Number*.

[11] http://en.wikipedia.org/wiki/Global_optimization

[12] http://en.wikipedia.org/wiki/Annealing_(metallurgy)

[13] B. J. Oommen, S.W. Kim, M.T. Samuel, and O-C. Granmo, *A solution to the stochastic point location problem in metalevel nonstationary Environments,* IEEE Transactions on Systems, Man and Cybernetics, Part B, 38(2):466 –476, April 2008.

[14] -, *A learning automaton solution to the stochastic minimum spanning circle problem*, IEEE Trans. Syst. Man Cybem., pp. 598-603, July/August 1986.

[15] B. J. Oommen. *Stochastic searching on the line and its applications to parameter learning in nonlinear optimization*, IEEE Transactions on Systems, Man and Cybernetics, SMC-27B:733–739, 1997.

[16] B. J. Oommen and G. Raghunath, *Automata learning and intelligent tertiary searching for stochastic point location*, IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 28, no. 6, pp. 947–954, December 1998.

[17] B. J. Oommen, G. Raghunath, and B. Kuipers, *Parameter learning from stochastic teachers and stochastic compulsive liars*, IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 36, no. 4, pp. 820–836, August 2006.

[18] P. Lalbakhsh, B. Zaeri, A. Lalbakhsh, M. N. Fesharaki, *AntNet with Reward-Penalty Reinforcement Learning*, IEEE Transactions on Systems, Man, and Cybernetics, 978-0-7695-4158-7/10, 2010)

[19] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins, *Searching with uncertainty*, In Proceedings of Scandinavian Workshop Algorithms and Theory (SWAT), pages 176–189, Halmstad, Sweden, 1998.

[20] R. A. Baeza-Yates and R. Schott, *Parallel searching in the plane*, Comput. Geom. Theory Appl., 5:143–154, October 1995.

[21] R. L. Kashyap and B. J. Oommen, *Scale preserving smoothing of polygons*, *IEEE Trans. Pattern Anal. Machine Intell.* pp. 667–671, November 1983.

[22] Y.-H. Pao, *Adaptive Pattern Recognition and Neural Networks.* Reading, MA: Addison-Wesley, 1989.

[23] T. Pavlidis, *Structural Pattern Recognition.* New York: Springer-Verlag, 1977.

[24] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing.* Cambridge, U.K.:Cambridge Univ. Press, 1986.

[25] S. S. Rao, *Optimization: Theory and Applications,* 2nd ed. New Delhi,India: Wiley, 1984.

[26] P. D. Wasserman, *Neural Computing: Theory and Practice.* New York: Van Nostrand Reinhold, 1989.

[27] J. K. Lanctˆot and B. J. Oommen, *Discretized estimator learning automata*, IEEE Trans. Syst., Man, Cybern., vol. 22, pp. 1473–1483, Nov./December 1992.

[28] K. S. Narendra and M. A. L. Thathachar, *Learning Automata*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[29] M. A. L. Thathachar and B. J. Oommen. *Discretized reward-inaction learning automata*, Journal of Cybernetics and Information Science, pages 24–29, Spring 1979.

[30] B.J. Oommen and M. Agache. *Continuous and discretized pursuit learning schemes: various algorithms and their comparison*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 31(3):277 –287, June 2001.

[31] B. J. Oommen and J. K. Lanctˆot. *Discretized pursuit learning automata*, IEEE Transactions on Systems, Man, and Cybernetics, SMC-20(4):931–938, July/August 1990.

[32] B. J. Oommen and E. Hansen. *The asymptotic optimality of discretized linear reward-inaction learning automata*, IEEE Transactions on Systems, Man, and Cybernetics, SMC-14(3), May/June 1986.

[33] M. Agache and B. J. Oommen. *Generalized pursuit learning schemes: New families of continuous and discretized learning automata*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 32(6):738–749, December 2002.

[34] O-C. Granmo and B. J. Oommen, *Solving stochastic nonlinear resource allocation problemsusing a hierarchy of twofold resource allocation automata*, IEEE Transactions on Computers, 59:545–560, 2009.

[35] B. J. Oommen, *Absorbing and Ergodic Discretized Two-Action Learning Automata*, IEEE Transactions on Systems, Man, and Cybernetics, VOL. SMC-16, NO. 2, March/April 1986

[36] A. Yazidi, O- C. Granmo, B. J. Oommen, and M. Goodwin, *A Hierarchical Learning Scheme for Solving the Stochastic Point Location Problem.* To Appear in Proceedings of the 25th International Conference on Industrial, Engineering and Other Applications of Applied Intelligence Systems (IEA-AIE 2012), Springer, Dalian, China, June 2012.

[37] M. Barzohar and D. B. Cooper, *Automatic finding of main roads in aerial images by using geometric-stochastic models and estimation*, IEEE Trans. Pattern Anal. Mach. Intell., vol. 18, no. 7, pp. 707–722, July 1996.

[38] D. J. Bertsimas and G. Van Ryzin, *Stochastic and dynamic vehicle routing in the Euclidean plane with multiple capacitated vehicles*, Oper.Res., vol. 41, no. 1, pp. 60–76, January/February 1993.

[39] M. L. Brandeau and S. S. Chiu, *An overview of representative problems in location research*, Manage. Sci., vol. 35, no. 6, pp. 645–674, June 1989.

[40] J. J. Collins, C. C. Chow, and T. T. Imhoff, *Aperiodic stochastic resonance in excitable systems*, Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top., vol. 52, no. 4, pp. R3 321–R3 324, October 1995.

[41] R. L. Cook, *Stochastic sampling in computer graphics*, ACM Trans. Graph., vol. 5, no. 1, pp. 51–72, Jane 1986.

[42] S. Lakshmivarahan, *Learning Algorithms Theory and Applications*, Springer-Verlag, New York, 1981.

[43] K. Najimand A. S. Poznyak, *Learning Automata: Theory and Applications*, Pergamon Press, Oxford, 1994.

[44] Kumpati S. Narendra and Mandayam A. L. Thathachar, *Learning Automata: An Introduction*, Prentice-Hall, Inc., 1989.

[45] M.S. Obaidat, G.I. Papadimitriou, and A.S. Pomportsis, *Guest editorial learning automata: theory, paradigms, and applications*, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 32(6):706 – 709, December 2002.

[46] A. S. Poznyak and K. Najim, *Learning Automata and Stochastic Optimization*, Springer-Verlag, Berlin, 1997.

[47] M. A. L. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*, Kluwer Acadmic Publishers, Boston, MA, 2004.

[48] Ronald J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement Learning, *Machine Learning*, 8:229–256, 1992.

[49] Jon Louis Bentley and Andrew Chi-Chih Yao, *An Almost Optimal Algorithm for Unbounded Searching*, *Inform. Proc. Lett.*, 5:82–87, 1976.

[50] T. Bradland and T. Norheim, *Empirical evaluation of the bayesian learning automaton family*, 2009.

[51] http://en.wikipedia.org/wiki/Discrete_optimization

[52] S. Andradottir, *A Method for Discrete Stochastic Optimization*, Management Science/Vol. 41, No. 12, December 1995.

[53] W. Gong, Y. Ho, W. Zhai, *Stochastic Comparison Algorithm for Discrete Optimization With Estimation*, IEEE Transactions on Systems, Man, and Cybernetics, CH3229-2/92/0000-0795, December 1992.

[54] O-C. Granmo, Learning Problem Part II: Learning Automata, University of Agder IKT507 Lecture, 2010.

[55] http://en.wikipedia.org/wiki/Automaton#cite_note-0

[56] http://en.wikipedia.org/wiki/NP-complete

[57] B. J. Oommen, J. P. R. Christensen, *ε-Optimal Discretized Linear Reward-Penalty Learning Automata*, IEEE Transactions on Systems, Man, and Cybernetics, VOL. 18, NO. 3, May/June 1988.

[58] http://en.wikipedia.org/wiki/NP_(complexity)

[59]

http://wiki.answers.com/Q/What_are_the_differences_between_linear_search_and_binary_search

Appendix

Linear Search

A linear search algorithm is performed by considering each element in an array of data without jumping, and it continues its considering until finding the target node or obtaining the end of array. The time complexity of linear search is *O(n)* on a given array.

Binary Search

A binary search (hierarchical search) algorithm is performed in a sorted data set and it is started with the middle element of the data set, so the sorted data set is essential in binary search. Afterward, it considers whether that an element is bigger than or less than the target node, which indicates whether the element is in the first or second half of the array. It jumps to the half way of the sub array and repeats comparing. By using binary search algorithm just the left part of the elements is considered. Therefore, each time the algorithm searches for the target and misses, then half of the remaining elements are eliminated. *O(log n)* is the time complexity of this algorithm which is determines binary search is an efficient method since the number of search operations grows more slowly than the linear search. Halving the search space with each operation is the main reason for the binary search time complexity.

The differences between the linear and binary search are listed as follows:

- Sorted input data is required for the binary search; while for linear search simple input data.
- An ordering comparison is required in binary search; while in linear search, just equality comparison is needed.
- The complexity of binary search is *O(log n)*; but for linear search, it is *O(n)*.
- Random access to the data is required in binary search; while sequential access is needed in linear search.