Assignment of cells to switches in cellular mobile network: a learning automata-based memetic algorithm

Mehdi Rezapoor Mirsaleh[1] and Mohammad Reza Meybodi[2]

[1] *Department of Computer Engineering and Information Technology, Payame Noor University (PNU), P.O.BOX 19395-3697, Tehran, Iran*

[2] *Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran*

e-mail: (mrezapoorm, mmeybodi)@aut.ac.ir

Abstract

Handoff and cabling costs management plays an important role in the design of cellular mobile networks. Efficient assigning of cells to switches can have a significant impact on handoff and cabling cost. Assignment of cells to switches problem (ACTSP) in cellular mobile network is NP-hard problem and consequently cannot be solved by exact methods. In this paper a new memetic algorithm which is obtained from the combination of learning automata (LA) and local search is proposed for solving the ACTSP in which the learning automata keeps the history of the local search process and manages the problem's constraints. The proposed algorithm represents chromosome as object migration automata (OMAs), whose states represent the history of the local search process. Each state in an OMA has two attributes: the value of the gene (allele), and the degree of association with those values. The local search changes the degree of association between genes and their values. To show the superiority of the proposed algorithm several computer experiments have been conducted. The obtained results confirm the efficiency of proposed algorithm in comparison with the existing algorithms such as genetic algorithm, memetic algorithm, and a hybrid Hopfield network-genetic algorithm.

Keywords*: Cellular mobile networks, Assignment of cell to switch problem (ACTSP), Learning Automata (LA), Genetic Algorithm (GA), Memetic Algorithm (MA), Object Migration Automata (OMA)*

1 Introduction

In a typical cellular mobile network, the area of coverage is often geographically partitioned into hexagonal cells as illustrated in Fig. 1. Each cell contains a stationary base station (BS) covering a small geographic area and using an antenna for communications among users with pre-assigned frequencies. The calls first are transferred to switches by base stations and then routed either to another base station or to a public switched telephone network (PSTN) by switches[1]. For various reasons and especially because of users' mobility, the signals between the mobile unit and the base station may become weaker while interference from adjacent cells increases. When a user in communication crosses the line between adjacent cells, the base station of the new cell has the responsibility to relay this communication by assigning a new radio channel to the user. This operation is called a handoff and it occurs when the mobile network automatically support the transferring of a communication from one cell to another adjacent cell. When a handoff occurred between two cells associated to the same switch, it is called a simple handoff; because of there are few necessary update. This type of handoff is relatively easy to perform, and does not involve any location update in the databases that record the position of the user. For example, in Fig. 1, a simple handoff causes by moving user from cell 12 to cell 13. In this type of handoff, the network's database that records the position of users does not need an update. Also, Fig. 1 illustrates another type of handoff; complex handoff; which is caused by moving user from cell 12 to cell 8. A complex handoff refers to a handoff between two cells associated to different switches. In this type of handoff the process of handoff involves the execution of a complicated protocol between switch 1 and switch 2. It involves the updating of the location of the user in the network's database. Furthermore, if the original switch (switch 1 in this case) is in charge of the billing, the handoff cannot simply replace switch 1 with switch 2. Note that the complex handoff process consumes much more network resources than simple handoff process. Consequently, it is usually advantageous to connect cells which have a high frequency of handoffs between them to the same switch.

The network bandwidth management is another important issue that should be considered in the design of mobile networks. Network bandwidth is used not only through user traffic, but also through controlled traffic to ensure user mobility. As we do not control the volume of user traffic, all efforts to use efficiently bandwidth are based on frequency reuse and minimizing the control traffic. The registration area planning problem (RAP) seeks a partition

of the cells of the network into contiguous areas called registration areas so that the bandwidth used by control signals is minimized [2].

In this paper a new algorithm is proposed for solving ACTSP. This algorithm is obtained from combination of learning automata (LA) and local search method in which the local search method searches for high quality solutions with the minimum possible cost in terms of handoff and cabling cost and the LA keeps the history of the local search. Object migration automaton (OMA) represents chromosome in the proposed algorithm. Each state in an OMA has two attributes: the value of the gene, and the degree of association with its value. Information about the past history of the local search process shows the degree of association between genes and their values.

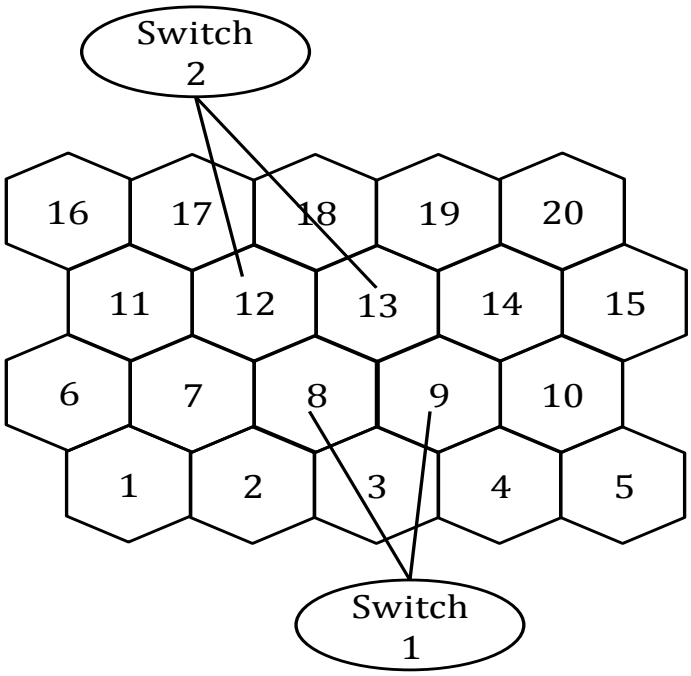


Fig. 1 : An example of simple handoff (from cell 12 to cell 13) and complex handoff (from cell 12 to cell 8)

The results obtained for the proposed algorithm are compared with the results of the best-known assignment of cells to switches algorithms such as: GA[3], MA[4], HI[5], HII[5] ABC[6] and API/TS[7]. The obtained results show the superiority of the proposed algorithm over the other algorithms in terms of the quality of solution. The rest of the paper is organized as follows. Background and related works on ACTSP is represented in section 2. The theory of automata is described in section 3. The new proposed algorithm is described in section 4. Section 5 is including of implementation considerations, simulation results, and comparison

with other algorithms to highlight the contributions of the new algorithm. Section 6 is the conclusion and directions for future works.

2 Background and related works

Let *n* be the number of cells and *m* be the number of switches in a cellular mobile network where the location of cells and switches are predetermined. Suppose that the cell *i* is assigned to switch *k* with cost of cabling c_{ik}. Let h_{ij} be the handoff cost per unit of time between cell *i* and cell *j*, if cell *i* and cell *j* are assigned to different switches, and $h_{ij} = 0$ when they are assigned to the same switch. The ACTSP can be described as finding matrix X that minimizes the function $Z(X)$ defined as Eq.(1). X is an $n \times m$ binary matrix, in which the element $x_{ik} = 1$ if cell *i* is assigned to switch *k*, and $x_{ik} = 0$ otherwise.

$$Z(X) = \sum_{i=1}^{n} \sum_{k=1}^{m} c_{ik} x_{ik} + \sum_{i=1}^{n} \sum_{j=1}^{n} h_{ij} \left(1 - \sum_{k=1}^{m} h_{ij} x_{ik} x_{jk}\right), \quad (1)$$

The first term of Eq.(1), represents the cost of cabling between the cells and the switches and the second term represents the handoff cost corresponding to handoffs between cells assigned to different switches. The base station in cell *i* manages λ_i calls per unit time whereas the call handling capacity of switch *k* is M_k. These limitations impose the following constraint:

$$\sum_{i=1}^{n} \lambda_i x_{ik} \leq M_k \qquad \forall k, \quad (2)$$

Also, every cell must be assigned to one and only one switch. This limitation imposes the following constraint:

$$\sum_{k=1}^{m} x_{ik} = 1 \qquad \forall i, \quad (3)$$

The wide variety of approaches has been reported for ACTSP in the literature. These approaches can be classified as exact or approximation approaches. Exact approaches are practically inappropriate to solve large-sized instances of this problem. Exact approaches exhaustively examine the entire search space in order to find the optimal solution and for this reason they are only efficient for small search spaces corresponding to small-sized instances of the problem. For example, for a network with *n* cells and *m* switches, m^n solutions should

be examined. Hence, different approximation algorithms have been reported in the literature for finding the near optimal solution for ACTSP. Approximation algorithms for solving ACTSP by itself can be classified into two classes: hybrid algorithms obtained by combining evolutionary algorithm with a local search such as tabu search or simulated annealing and PSO based algorithms. In the rest of this section, we briefly review some well-known algorithms for solving ACTSP from these two classes and some well-known algorithms for solving RAP.

Hybrid algorithms for solving ACTSP: In [4] two multi-populations memetic algorithms with migrations and two sequential memetic algorithms are proposed for ACTSP. In all of these algorithms, the local search used is tabu search or simulated annealing. In [5] a hybrid algorithm based on Hopfield network and genetic algorithm is introduced. In this algorithm a genetic algorithm searches for high quality solutions, and Hopfield network manages the problem's constraints. In [6], a kind of honeybee-inspired algorithm called Artificial Bee Colony (ABC) algorithm is utilized to solve ACTSP. The model is mathematically developed as a binary non-linear problem with the objective of minimizing cost, which includes handoff and cabling costs. In [7], another hybrid algorithm which is based on API ant algorithm and tabu Search (API/TS) is reported. API/TS algorithm performs a global and local search for high quality feasible solutions in terms of hand-off and cabling costs. A key element in the success of API/TS approach is the use simultaneously an intensified research on several areas of research space. In [8] two algorithms are reported, an ant colony based algorithm (ACO) and a hybrid algorithm based on the ACO and the k-opt local search. The hybrid algorithm works well for large-size problems and performs the same as ACO algorithm for small-size problems. In [9] Quintero and Samuel Pierre propose a hybrid evolutionary algorithm for ACTSP. This algorithm uses two local searches in order to combine the strengths of global and local search aspects. The local searches are tabu search and simulated annealing. In [10] a comparative study of three algorithms; parallel genetic algorithms with migrations (PGAM), a hybrid algorithm based on PGAM and simulated annealing (PGAM-SA) and a hybrid algorithm based on PGAM and tabu search (PGAM-TS) for solving ACTSP are conducted. The results show that the PGAM-SA and PGAM-TS improve the solution provided by PGAM. In [11] another hybrid algorithm which is based on pricing mechanism and simulated annealing is reported. The pricing mechanism provides a direction to proceed when trying to identify new solutions which can be a powerful tool in speeding up simulated annealing which can otherwise be extremely slow.

PSO based algorithms for solving ACTSP: In [12] two algorithms based on the barebones (BB) and the exploiting barebones (BBExp); variants of particle swarm optimization (PSO) are reported for solving ACTSP. In BB-PSO the standard PSO velocity equation is removed and replaced with samples from a normal distribution. Whereas, in BBExp-PSO, approximately half of the time velocity is based on samples from a normal distribution and the rest of the time, velocity is derived from the particle's personal best position. In [13] a discrete particle swarm optimization (PSO) based on estimation of distribution algorithms (EDA) is introduced for solving ACTSP. This algorithm incorporates the global statistical information collected from personal best solutions of all particles into the PSO, and therefore each particle has comprehensive learning and search ability.

well-known algorithms for solving RAP: in [14] a grouping genetic algorithm (GGARAP) is introduced to solve the RAP in PCS networks. This algorithm offers several benefits. First, grouping genetic algorithms are flexible enough to support any number of constraints by including penalties to the fitness function. Second, grouping genetic algorithms reduce redundant search of the solution space and guarantee children with characteristics of their parents. Third, GGARAP begins with an initial population that is generated using a problem-specific heuristic.

In [2] a hybrid grouping genetic algorithm, i.e., a steady-state grouping genetic algorithm (SSGGA) is introduced to solve the RAP. Solution encoding, selection method, crossover and mutation opearatores, population replacement policy applied in SSGGA are all different from other GAs. SSGGA use a local search to further improve the solution quality of certain selected solutions.

In [15] a hybrid artificial bee colony (ABC) algorithm based approach is reported for the RAP. The ABC approach is designed for the RAP keeping in mind its grouping nature. While generating neighboring solutions, the ABC approach tries to preserve the grouping property of the RAP to the maximum extent possible.

In [16] a hybridized grouping genetic algorithm (HGGA) is introduced to obtain cell formations for the RAP. The hybridization is accomplished by adding a tabu search-based improvement operator to a traditional grouping genetic algorithm (GGA).

3 Theory of automata

In this section, learning automata (LA) is introduced in brief. Then object migrating automata (OMA) which is a type of fixed structure learning automata is presented.

3.1 Learning Automaton

A learning automaton (LA) [17] is an adaptive decision-making unit. It can be described as determination of an optimal action from a set of actions through repeated interactions with an unknown random environment. It selects an action based on a probability distribution at each instant and applies it on a random environment. The environment sends a reinforcement signal to automata after evaluating the input action. The learning automata process the response of environment and update its action probability vector. By repeating this process, the automaton learns to choose the optimal action so that the average penalty obtained from the environment is minimized. The environment is represented by a triple $< \underline{\alpha}, \underline{\beta}, \underline{c} >$. $\underline{\alpha} = \{\alpha_1, \dots, \alpha_r\}$ is the finite set of the inputs, $\underline{\beta} = \{0, 1\}$ is the set of outputs that can be taken by the reinforcement signal, and $\underline{c} = \{c_1, \dots, c_r\}$ is the set of the penalty probabilities, where each element c_i of $\underline{c}$ is corresponds to one input action α_i. The input $\alpha(n)$ to the environment belongs to $\underline{\alpha}$ and may be considered to be applied to the environment at discrete time $t = n\,(n = 0, 1, 2, \dots)$. The output $\beta(n)$ of the environment belongs to $\underline{\beta}$ and can take on one of two values 0 and 1. An $\beta(n) = 1$ is identified with a failure or an unfavorable response and $\beta(n) = 0$ with a success or favorable response of the environment. The element c_i of $\underline{c}$ which characterizes the environment may then be defined by $pr(\beta(n) = 1 | \alpha(n) = \alpha_i) = c_i \;\; (i = 1,2, \dots, r)$. When the penalty probabilities are constant, the random environment is said a stationary random environment. It is called a non stationary environment, if they vary with time. Fig. 2 shows the relationship between the LA and the random environment.

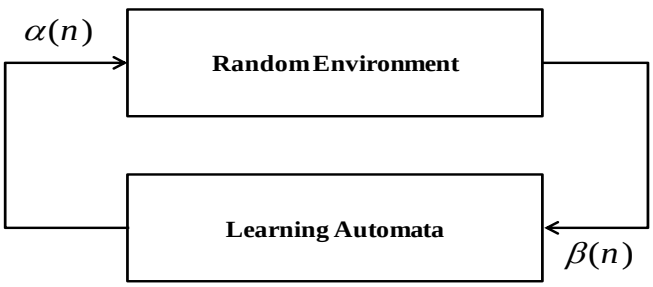


Fig. 2: The relationship between the LA and random environment

There are two main families of learning automata [18]: fixed structure learning automata and variable structure learning automata. First, we formally define fixed structure learning automata and then some of the fixed structures learning automata such as Tsetline, Krinsky, and Krylov automata are described.

A fixed structure learning automaton is represented by a quintuple $< \underline{\alpha}, \underline{\Phi}, \underline{\beta}, F, G >$ where:

- $\underline{\alpha} = \{\alpha_1, \dots, \alpha_r\}$ is the set of actions that it must choose from.
- $\underline{\Phi} = \{\varphi_1, \dots, \varphi_s\}$ is the set of internal states.
- $\underline{\beta} = \{0, 1\}$ is the set of inputs where 1 represents a penalty and 0 represents a reward.
- $F: \underline{\Phi} * \underline{\beta} \rightarrow \underline{\Phi}$ is a function that maps the current state and current input into the next state.
- $G: \underline{\Phi} \rightarrow \underline{\alpha}$ is a function that maps the current state into the current output. In other words, *G* determines the action taken by the automaton.

The operation of fixed learning automata could be described as follows:

At the first step, the selected action $\alpha(n) = G[\Phi(n)]$ serves as the input to the environment, which in turn emits a stochastic response $\beta(n)$ at the time n. $\beta(n)$ is an element of $\underline{\beta} = \{0, 1\}$ and is the feedback response of the environment to the automaton. In the second step, the environment penalize (i.e., $\beta(n) = 1$) the automaton with the penalty probability c_i, which is the action dependent. On the basis of the response $\beta(n)$, the state of automaton is updated by $\Phi(n+1) = F[\Phi(n), \beta(n)]$. This process continues until the desired result is obtained.

In the following paragraphs, we describe some of the fixed structure learning automata such as Tsetline, Krinsky, and Krylov automata.

The two-state automata ($L_{2,2}$): This automaton has two states, φ_1 and φ_2 and two actions α_1 and α_2. The automaton accepts input from a set of $\{0, 1\}$ and switches its states upon encountering an input 1 (unfavorable response) and remains in the same state on receiving an input 0 (favorable response). An automaton that uses this strategy is refereed as $L_{2,2}$ where the first subscript refers to the number of states and second subscript to the number of actions.

The Tsetline automata (The two-action automata with memory $L_{2N,2}$): Tsetline suggested a modification of $L_{2,2}$ denoted by $L_{2N,2.}$ This automaton has *2N* states and two actions and attempts to incorporate the past behavior of the system in its decision rule for choosing the sequence of actions. While the automaton $L_{2,2}$ switches from one action to another on

receiving a failure response from environment, $L_{2N,2}$ keeps an account of the number of success and failures received for each action. It is only when the number of failures exceeds the number of successes, or some maximum value *N*; the automaton switches from one action to another. The procedure described above is one convenient method of keeping track of performance of the actions α_1 and α_2. As such, *N* is called depth of memory associated with each action, and automaton is said to have a total memory of *2N*. For every favorable response, the state of automaton moves deeper into the memory of corresponding action, and for an unfavorable response, moves out of it. This automaton can be extended to multiple action automata. The state transition graph of $L_{2N,2}$ automaton is shown in Fig. 3.

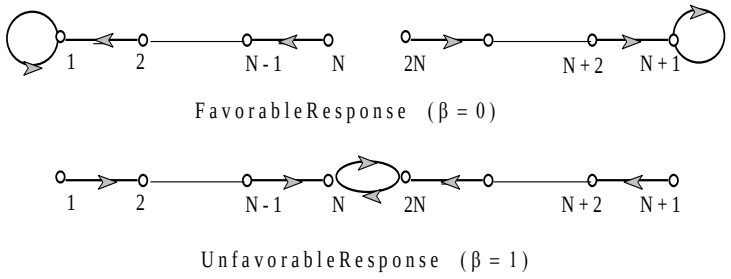


Fig. 3: The state transition graph for $L_{2N,2}$ (Tsetline Automaton)

The Krinsky automata: This automaton behaves exactly like $L_{2N,2}$ automaton when the response of the environment is unfavorable, but for favorable response, any state φ_i (for i = *1, ... N*) passes to the state φ_1 and any state φ_i (i = *N+1, 2N*) passes to the state φ_{N+1}. This implies that a string of *N* consecutive unfavorable responses are needed to change from one action to another. The state transition graph of Krinsky automaton is shown in Fig. 4.

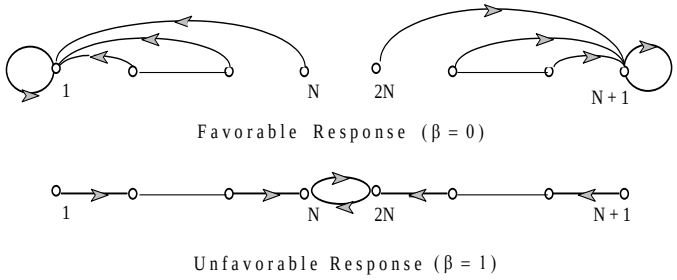


Fig. 4: The state transition graph for Krinsky Automaton

The Krylov automata: This automaton has state transitions that are identical to the $L_{2N,2}$ automaton when the output of the environment is favorable. However when the response of the environment is unfavorable, a state φ_i ($i \neq 1, N, N+1, 2N$) passes to a state φ_{i+1} with probability 1/2 and to state φ_{i-1} with probability 1/2, as shown in Fig. 5. When *i=1* or *N+1*, φ_i stays in the same state with probability 1/2 and moves to φ_{i+1} with the same probability. When *i=N*, φ_N moves to φ_{N-1} and φ_{2N} each with probability 1/2 and similarly, When *i=2N*,

φ_{2N} moves to φ_{2N-1} and φ_N each with probability 1/2. The state transition graph of Krylov automaton is shown in Fig. 5.

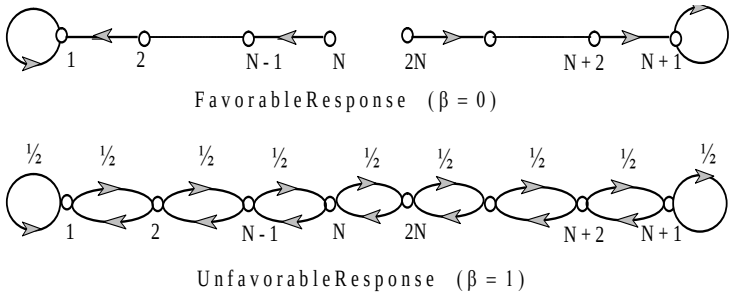


Fig. 5: The state transition graph for Krylov Automaton

Object migration automaton (OMA) that is an example of fixed structure learning automata is described in the next section. Learning automata have a vast variety of applications in combinatorial optimization problems [19-26], computer networks [22, 27-33], queuing theory [34, 35], signal processing [36, 37], information retrieval [38], adaptive control [39-41], neural networks engineering [42, 43] and pattern recognition [44-46].

3.2 **Object Migration Automata**

Object migration automata were first proposed by Oommen and Ma [47]. OMAs are a type of fixed structure learning automata, and are defined by a quintuple $< \underline{\alpha}, \underline{\Phi}, \underline{\beta}, F, G >$. $\underline{\alpha} = \{\alpha_1, \dots, \alpha_r\}$ is the set of allowed actions for the automaton. For each action α_k, there is a set of states $\{\varphi_{(k-1)N+1}, \dots, \varphi_{kN}\}$, where N is the depth of memory. The states $\varphi_{(k-1)N+1}$ and φ_{kN} are the most internal state and the boundary state of action α_k , respectively. The set of all states is represented by $\underline{\Phi} = \{\varphi_1, \dots, \varphi_s\}$, where $s = N * r$. $\underline{\beta} = \{0, 1\}$ is the set of inputs, where 1 represents an unfavorable response, and 0 represents a favorable response. $F: \underline{\Phi} * \underline{\beta} \rightarrow \underline{\Phi}$ is a function that maps the current state and current input into the next state, and $G: \underline{\Phi} \rightarrow \underline{\alpha}$ is a function that maps the current state into the current output. In other words, G determines the action taken by the automaton. W objects are assigned to actions in an OMA and moved around the states of the automaton, as opposed to general learning automata, in which the automaton can move from one action to another by environmental response. The state of objects is changed on the basis of the feedback response from the environment. If the object w_i is assigned to action α_k (i.e., w_i is in state ξ_i, where $\xi_i \in \{\varphi_{(k-1)N+1}, \dots, \varphi_{kN}\}$), and the feedback response from the environment is 0, α_k is rewarded, and w_i is moved toward the most internal state ($\varphi_{(k-1)N+1}$) of that action. If the feedback from the environment is 1, then

α_k is penalized, and w_i is moved toward the boundary state (φ_{kN}) of action α_k. The variable γ_k denotes the reverse of the state number of the object assigned to action α_k (i.e., degree of association between action α_k and its assigned object). By rewarding an action, the degree of association between that action and its assigned object will be increased. Conversely, penalizing an action causes the degree of association between that action and its assigned object to be decreased. An object associated with state $\varphi_{(k-1)N+1}$ has the most degree of association with action α_k, and an object associated with state φ_{kN} has the least degree of association with action α_k.

4 The proposed algorithm for assignment of cells to switches problem

In this section, we propose a new algorithm based on the learning automata (LA) for solving the assignment of cells to switches problem (ACTSP). In this section we first describe the solution representation for ACTSP and then the framework of the proposed algorithm is presented.

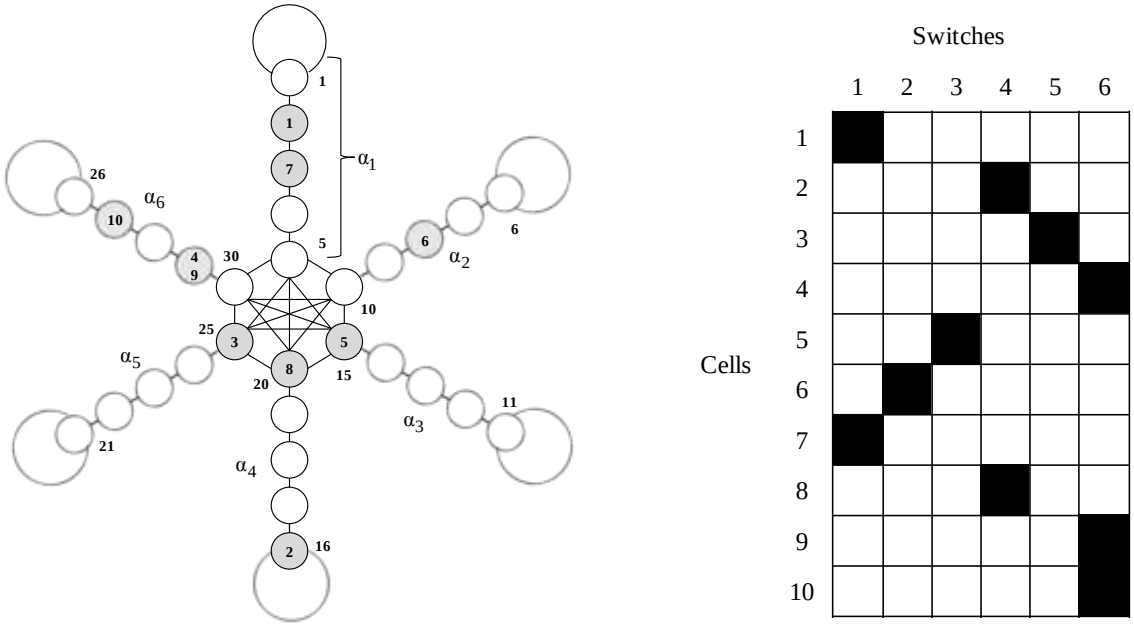
4.1 Solution representation

In proposed algorithm, the solution is represented by an object migration automaton (OMA) whose states keep information about the past history of the local search process. In the OMA-based representation, there are *m* actions corresponding to *m* switches. Furthermore, for each action, there are a fixed number of states *N*. Each state in an OMA has two attributes: the value of assigned object, and the degree of association with its value. The cells, as migratory objects in the automata are assigned to states of corresponding action (switch). Fig. 6(a) shows a representation of assigning 10 cells to 6 switches using the Tsetline automaton-based OMA with depth of memory of 5. In Fig. 6(a) there are 6 actions (switches), denoted by α_1, α_2, α_3, α_4, α_5 , and α_6. Cells 1, 4, 6, 7, 9 and 10 are assigned to actions α_1, α_6, α_2, α_1, α_6 and α_6 (switches 1, 6, 2, 1, 6 and 6) and located at internal states 2, 4, 3, 3, 4 and 2 of their actions, respectively. Cells 3, 5 and 8 are assigned to actions α_5, α_3 and α_4 (switches 5, 3 and 4) and located at boundary states of their actions, respectively. Consequently, there is a minimum degree of association between these actions and their corresponding object. The remaining cell, cell 2, is assigned to actions α_4 (switch 4) and located at the most internal state of its action. That is, it has the maximum degree of association with action 4.

After applying a local search, if the assignment of a cell to a switch is promising, then the automaton is rewarded and the assigned cell moves toward the most internal state of its switch; otherwise, the automaton is penalized and the assigned cell moves toward the boundary state of its switch. The rewarding and penalizing of automaton, changes the degree of association between cells and their switches. The initial assignment of cells to switches is created randomly and cells are located in the boundary state of their switches. We can transfer an OMA representation into the solution vector $S = [(s_1, t_1), (s_2, t_2), \dots, (s_n, t_n)]$ to show the obtained assignment, where s_i is the switch assigned to cell i and t_i is its state, specifying $1 \leq s_i \leq m$ and $(s_i - 1) * N + 1 \leq t_i \leq s_i * N$. If we have a solution vector *S*, the corresponding binary matrix *X* can be calculated as:

$$x_{ij} = \begin{cases} 1, & if\ S.s_i = j, \\ 0, & otherwise \end{cases} \quad (4)$$

Fig. 6(b) and (c) represent the corresponding binary matrix *X* and the solution vector *S* of the sample shown in Fig. 6(a), respectively.



(a) (b)

S = [(1,2),(4,16),(5,25),(6,29),(3,15),(2,8),(1,3),(4,20),(6,29),(6,27)]

(c)

Fig. 6: Example of the state transition graph of a Tsetline-based OMA with *N=5* (a), the binary matrix *X* with 6 switches and 10 cells (The black squares represent 1s and the white squares represent 0s) (b) and the solution vector *S* (c).

Representation of solution based on other fixed structure learning automata is also possible. In a Krinsky-based OMA representation, as shown in Fig. 4, the object will be associated with the most internal state (i.e., it gets the highest degree of association with the corresponding action) when it is rewarded, and moves according to the Tsetline automaton-based OMA when it is penalized. In the representation of a Krylov OMA shown in Fig. 5, the object moves either toward the most internal state, or toward the boundary state, with a probability 0.5 toward penalty, and moves according to the Tsetline automaton-based OMA upon reward.

4.2 The framework of the proposed algorithm

The proposed algorithm has a local search which is applied to all cells in current solution sequentially. Specifically, for a certain cell *k* which is assigned to switch *s* in current solution *X*, a temporary solution $X'$is created based on the current solution X in which the switch assigned to cell k (switch s) is replaced with another switch s' which is selected randomly. The temporary solution X' can be created as follows.

$$x'_{ij} = \begin{cases} x_{ij}, & \forall\, i,j, \quad 1 \le i \le n, \quad i \ne k, \quad 1 \le j \le m \\ 1, & i = k, \quad j = s' \text{ where } s' \text{ is a random number between } 1 \text{ and } m\,, s' \ne s \\ 0, & i = k, \quad j \ne s' \end{cases} \tag{5}$$

If the assignment of cell k to new switch s' does not exceed the capacity of switch s' and the total cost of temporary solution ($Z(X')$) is less than total cost of current solution ($Z(X)$), the assignment of cell k to switch s in current solution is penalized. Otherwise, it is rewarded. By rewarding a cell, the state of that cell will move toward the most internal state according to the OMA connection. This causes the degree of association between a cell, and its corresponding switch, to be increased. Fig. 7 illustrates an example of rewarding the assignment of cell 4 to switch 6.

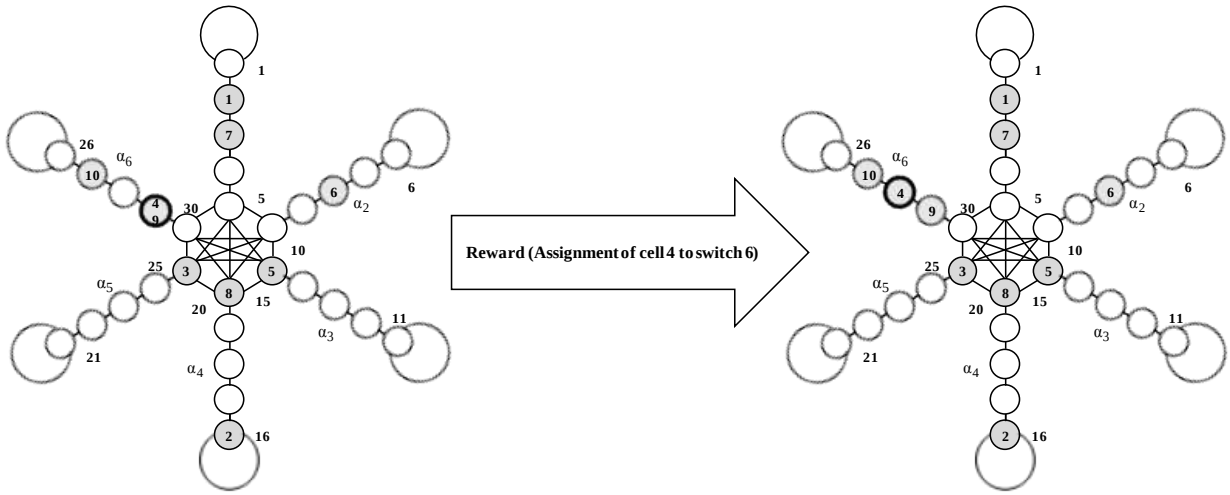


Fig. 7: An example of a reward function for assignmnet of cell 4 to switch 6

The state of a cell remains unchanged, if the cell is located at its most internal state, such as the assignment of cell 2 to switch 4, shown in Fig. 8.

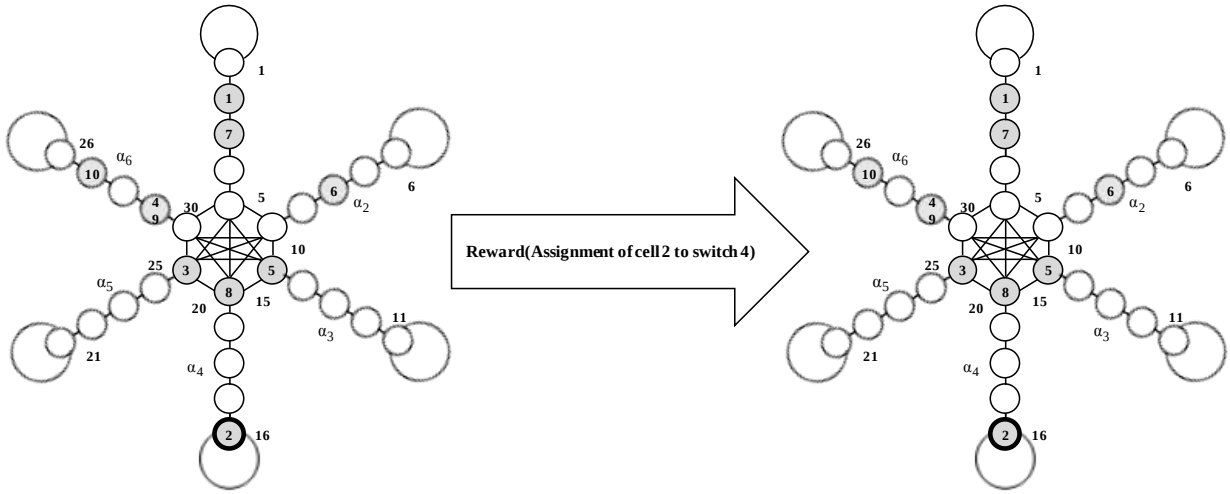


Fig. 8: An example of a reward function for assignment of cell 2 to switch 4

Fig. 9 provides pseudo code for rewarding the cell *i* in solution vector *S*.

```
Procedure Reward( S, i )
    If (S.t_i − 1) mod N <> 0 then
        Dec(S.t_i);
    End If
End Reward
```

Fig. 9: Pseudo code for a reward function

Penalizing a cell causes the degree of association between cell and its corresponding switch to be decreased. If a cell is not in the boundary state of its action, then penalizing causes the cell assigned to the switch to move toward the boundary state. This means that the degree of association between the cell and the corresponding switch will be decreased (Fig. 10).

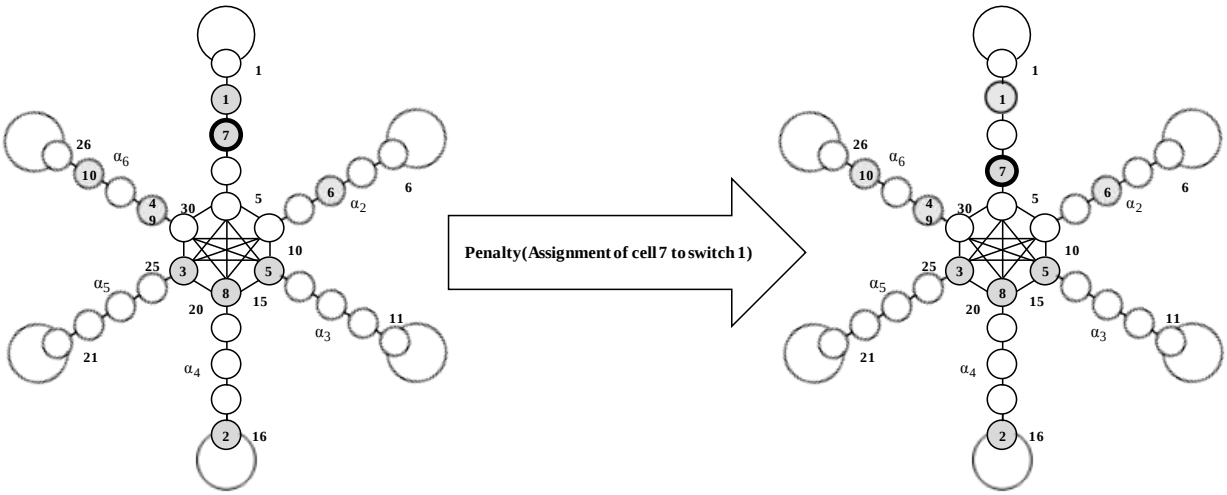


Fig. 10: An example of a penalty function for assignment of cell 7 to switch 1

If a cell is in the boundary state of its switch, then penalizing causes the cell is assigned to a new switch, and results the creation of a new solution. Fig. 11 shows the effect of the penalty function on assignment of cell 8 to switch 4.

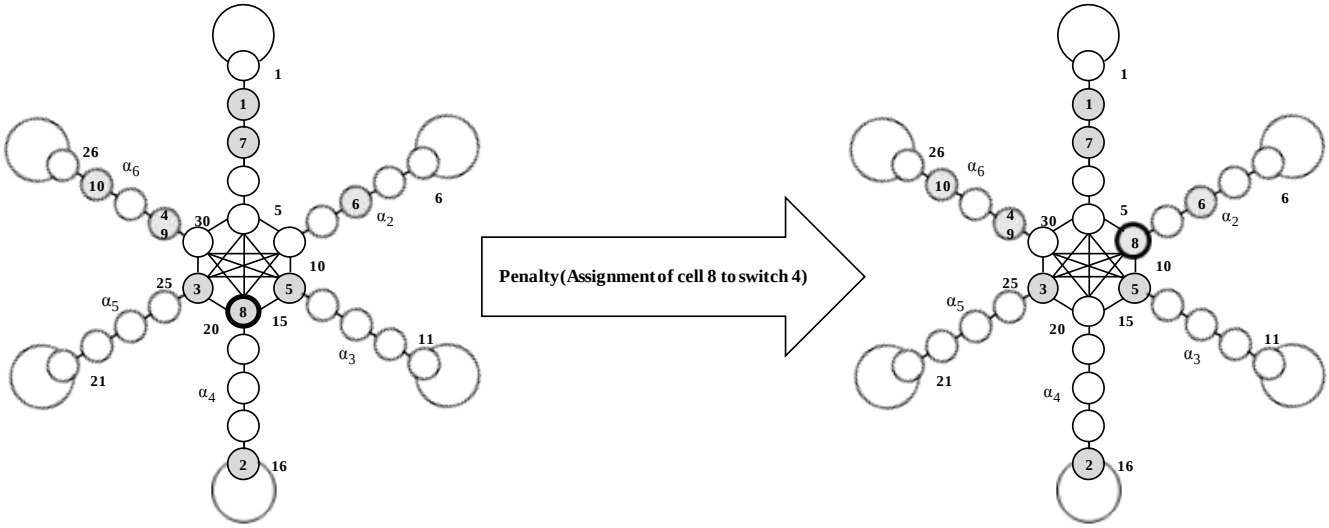


Fig. 11: An example of a penalty function for assignment of cell 8 to switch 4

The pseudo code for the penalty function is shown in Fig. 12.

```
Procedure Penalize( S, i )
  If (S.t_i) mod N <> 0 then
    Inc(S.t_i);
  Else
    Select switch j randomly where j ≠ S.s_i and 1 ≤ j ≤ m.
    S.s_i = j;
    S.t_i = S.s_i * N;
  End If
End Penalize
```

Fig. 12: Pseudo code for the penalty function

The pseudo code for proposed algorithm is shown in Fig. 13.

Learning automata based algorithm for assignment of cells to switches problem (LA-ACTSP)

Input: cells 1 to n with their coordinates, switches 1 to m with their coordinates, the number of states (*N*), the number of calls handled by cell *i* (λ_i) and the call handling capacity of switch *k* (M_k)
Output: The solution vector *S*, The binary matrix *X* and the corresponding assignment of cells to switches
Begin Algorithm
  For i=1 to *n* **do**
    $S.s_i = \text{random}(1..m)$; // Assign a switch to cell *i* randomly.
    $S.t_i = S.s_i * N$;
  End For
  Repeat
    $S \rightarrow X$; //Transfer solution vector *S* into binary matrix *X* according to Eq. (4);
    For i=1 to *n* **do**
      $X' = localsearch(X)$; //Create temporary binary matrix X' by exchanging the switch of cell *i* with new switch s' in binary matrix *X* according to Eq. (5).
      If $(Z(X') \leq Z(X) and \sum_{i=1}^{n} \lambda_i x_{is'} \leq M_{s'})$
        Penalize(*S*, *i*);
      Else
        Reward(*S*, *i*);
      End If
    End For
  Until all cells locate in the most internal state of their switches;
End Algorithm

Fig. 13: Pseudo code for LA-ACTSP

5 Experimental Result

In this section several experiments are conducted in order to study the efficiency of proposed algorithm. We consider a cellular mobile network such as Fig. 1 where switches are uniformly distributed randomly outside the grid and the antenna of the base station for each cell is located in the center of the cell. We assume that the cost of cabling c_{ik} between cell *i* and switch *k* is taken to be proportional to the geometric distance between the two. The rate of calls in cell *i* (λ_i) was generated from a Gamma distribution with mean one and coefficient of variation 0.25. The handoff rate h_{ij} is defined as:

$$\iota_{ij} = \lambda_i * r_{ij} \tag{6}$$

Where r_{ij} is the probability of transfer the call from cell *i* to cell *j*, $\sum_{j=1}^{k+1} r_{ij} = 1$, and *k* is the number of neighbors of cell *i*. At the end of the service period in cell *i*, the call could be either transferred to the j_{th} neighbor (*j* = 1, ..., *k*) with a probability r_{ij}, or ended with a probability r_{ik+1}. We assume that all switches have the same call handling capacity M_k calculated as:

$$M_k = \frac{1}{m}\left(1 + \frac{K}{m}\right)\sum_{i=1}^{n} \lambda_i, \tag{7}$$

where $10 \leq K \leq 50$ ensures a global excess of 10–50% of the switches' capacity compared to the cells' volume of calls [9]. For all experiments a Tsetline-based OMA was used for

chromosome representation. The algorithm terminates when all the objects in only chromosome are located in the most internal state of their actions. Each reported result was averaged over 50 runs. We performed a parametric test (t test) and a non-parametric test (wilcoxon rank sum test) at the 95% significance level to provide statistical confidence. The t tests were performed after ensuring that the data followed a normal distribution (by using the Kolmogorov–Smirnov test). The experiments are implementedby MATLAB 2009a running on a PC with a 2.83 GHz processor and4 GB memory.

5.1 **Experiment 1**

We studied the impact of the parameter *N* (depth of memory) on the total cost and running time (in second) of the proposed algorithm. The depth of memory was varied from 2 to 10 by increments of 2. Table 1 shows the results obtained by the proposed algorithm for 15 different cases, with respect to the average (Avg.), minimum (Min.) and maximum (Max.) of total cost and their standard deviation (Std.). In this table symbols P#, CN and SN stand for problem number, number of cells and number of switches, respectively. Experimentations have shown that for all cases the proposed algorithm finds a solution regardless of the depth of the memory and also for all cases the lowest cost solution is obtained when N=4.

Table 2 shows the results obtained by the proposed algorithm for 15 different cases, with respect to the average (Avg.), minimum (Min.) and maximum (Max.) of running time of the proposed algorithm and their standard deviation (Std.). Comparing the results reported in Table 2, we find that, in all cases, the proposed algorithm with depth of memory 2 outperforms the others in terms of running time, whiletheir the total costs are the worst in most cases.

TABLE 1: COMPARISON OF THE MINIMUM (MIN.), MAXIMUM (MAX.), AVERAGE TOTAL COSTS (AVG.) AND THEIR STANDARD DEVIATION (STD.) OF THE PROPOSED ALGORITHM WITH DIFFERENT DEPTHS OF MEMORY

P#	CN	SN	The Proposed Algorithm																	
			N=2				N=4				N=6				N=8				N=10			
			Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.
1	15	3	186.12	0.24	185.69	186.58	185.08	0.12	184.79	185.29	186.12	0.25	185.61	186.76	186.32	0.54	185.18	187.29	187.25	1.23	185.23	190.11
2	15	4	372.36	0.45	371.59	373.76	369.52	0.08	369.38	369.69	370.63	0.65	369.28	372.45	370.32	1.65	366.80	373.51	372.25	2.24	366.35	377.86
3	15	5	376.14	3.25	370.23	384.85	371.51	0.15	371.27	371.82	374.25	1.23	371.77	377.46	373.58	2.35	367.32	378.94	376.14	3.45	365.40	383.88
4	30	3	808.98	2.63	801.30	814.54	805.65	1.23	802.77	808.73	809.65	0.36	809.05	810.38	812.36	1.36	808.17	816.18	810.12	4.12	801.32	822.74
5	30	4	728.34	2.35	722.37	733.62	724.25	2.36	718.90	730.38	732.25	1.85	728.27	736.44	736.45	2.35	731.75	741.50	742.25	3.45	733.08	750.32
6	30	5	698.14	4.36	689.78	711.65	681.54	5.63	670.95	697.35	702.54	3.25	693.70	708.15	698.35	2.75	692.68	704.67	689.52	2.01	685.31	694.23
7	50	3	1431.25	3.56	1424.77	1439.12	1412.81	5.68	1402.07	1423.30	1415.36	4.65	1404.26	1427.68	1423.58	3.45	1415.92	1433.39	1435.25	4.63	1424.77	1448.06
8	50	4	1168.32	4.68	1156.63	1178.23	1137.64	6.12	1124.22	1149.66	1141.25	8.24	1118.76	1155.60	1142.36	8.14	1125.99	1160.76	1163.25	12.58	1127.37	1186.30
9	50	5	956.32	4.69	943.10	965.72	945.33	5.24	934.22	961.26	946.79	2.36	941.13	951.00	952.32	7.14	938.53	968.30	962.14	14.32	929.17	995.11
10	75	3	2465.24	6.24	2450.59	2479.50	2423.51	8.24	2404.77	2439.36	2431.54	9.14	2412.17	2450.91	2436.24	12.24	2408.12	2462.04	2463.25	21.25	2414.34	2500.05
11	75	4	1764.32	5.14	1755.41	1778.86	1758.39	6.89	1743.43	1777.41	1777.33	4.35	1766.88	1786.44	1768.47	8.14	1752.62	1785.97	1768.36	12.09	1737.09	1795.89
12	75	5	2954.12	9.65	2938.37	2976.85	2875.36	15.63	2850.10	2912.81	2865.48	12.36	2842.39	2899.44	2898.54	9.47	2879.19	2917.37	2883.24	15.47	2863.78	2923.29
13	100	3	2963.15	10.01	2942.13	2985.45	2879.36	12.78	2852.67	2916.14	2886.32	14.36	2851.16	2912.69	2912.32	12.35	2880.71	2939.83	2924.15	16.32	2869.25	2959.85
14	100	4	3245.12	7.25	3230.25	3259.98	3224.57	8.42	3202.68	3240.13	3265.35	8.14	3236.56	3278.67	3245.14	7.63	3229.19	3258.76	3229.32	7.25	3210.64	3242.83
15	100	5	2778.23	12.36	2756.16	2803.66	2719.49	7.52	2701.29	2736.12	2735.89	9.12	2714.36	2756.55	2778.65	14.36	2743.38	2817.59	2776.36	12.35	2751.80	2804.10

TABLE 2: COMPARISON OF THE MINIMUM (MIN.), MAXIMUM (MAX.), AVERAGE RUNNING TIME IN SECOND (AVG.) AND THEIR STANDARD DEVIATION (STD.) OF THE PROPOSED ALGORITHM WITH DIFFERENT DEPTHS OF MEMORY

P#	CN	SN	The Proposed Algorithm																	
			N=2				N=4				N=6				N=8				N=10			
			Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.
1	15	3	1.5069	0.04	1.4430	1.6118	1.5889	0.10	1.3696	1.8416	3.3678	0.16	2.9591	3.7160	6.3556	0.13	6.0063	6.6001	11.9302	0.90	10.1208	13.7608
2	15	4	1.4990	0.01	1.4845	1.5215	1.6648	0.06	1.5452	1.7667	3.2592	0.20	2.8037	3.6552	5.8090	0.37	4.9268	6.7398	12.0915	0.65	10.7777	13.7920
3	15	5	1.4753	0.12	1.2296	1.7036	1.5430	0.02	1.5035	1.6017	3.5250	0.32	2.8168	4.1881	5.8535	0.30	5.1431	6.4246	12.3149	0.67	10.5536	14.3644
4	30	3	3.9780	0.10	3.7347	4.1410	4.2119	0.21	3.7668	4.6249	8.7712	0.65	7.4275	10.3094	14.5079	0.24	13.9772	14.9271	31.3367	1.81	26.9830	34.7427
5	30	4	3.9447	0.37	3.0490	4.6642	4.0736	0.33	3.3991	4.8110	8.6929	0.12	8.4594	8.9602	15.7923	1.15	13.0741	18.2265	30.5455	2.69	24.9188	37.2883
6	30	5	3.7254	0.36	2.8108	4.4910	4.1799	0.20	3.8118	4.6020	8.4545	0.33	7.9384	9.3758	15.4119	1.54	11.8794	19.7218	29.2681	0.17	28.8424	29.6220
7	50	3	11.6697	0.00	11.6634	11.6755	11.5692	0.57	10.5100	12.7490	25.3191	0.24	24.8078	25.9424	43.4142	1.32	40.7872	46.0076	89.9643	3.53	82.6128	97.6065
8	50	4	11.5748	0.36	10.9467	12.4927	12.1471	0.97	10.1781	14.3729	25.2868	0.83	23.4096	27.0066	43.7917	0.05	43.6365	43.9326	89.0585	4.75	74.6786	97.7145
9	50	5	11.4496	0.71	10.2383	13.5088	12.1086	0.83	10.6575	14.5880	26.7625	0.68	24.5979	27.8919	45.4897	3.58	35.3832	53.3807	89.7898	4.83	81.7612	104.0500
10	75	3	14.9836	0.69	13.5393	16.2972	16.6066	0.60	15.0429	17.6882	33.4766	2.72	27.6153	39.8800	62.3224	1.21	59.8431	65.6886	123.3322	6.93	110.7059	137.4709
11	75	4	14.8094	0.60	13.3560	16.2414	15.5934	1.18	12.9964	18.4925	33.3296	2.94	25.8707	39.3347	59.4056	4.15	51.4558	69.8307	118.3633	0.45	117.4398	119.4320
12	75	5	14.8669	0.33	14.1120	15.4523	15.2816	0.59	13.9627	16.7042	33.2125	0.77	31.4402	34.5629	59.0420	4.86	48.0718	69.5291	122.3261	1.22	120.0203	124.6613
13	100	3	19.1214	1.55	14.9580	21.6983	21.0362	1.27	18.2369	23.8174	43.7375	1.13	40.6816	45.8135	72.5148	3.46	64.7179	83.2308	159.7161	7.92	146.1927	175.8707
14	100	4	18.9830	0.12	18.6712	19.2188	20.1493	0.26	19.4861	20.9032	42.2680	0.66	40.2874	43.6991	74.7360	5.52	59.9642	89.3165	157.3577	4.05	148.5370	165.5135
15	100	5	19.4937	0.86	17.7803	21.5856	21.1796	0.83	19.2419	22.7810	41.0512	1.13	38.5344	43.1522	73.7131	4.01	62.9681	81.9195	160.3828	1.36	157.7723	163.0077

Fig. 14 shows the comparison of results reported in Table 1.

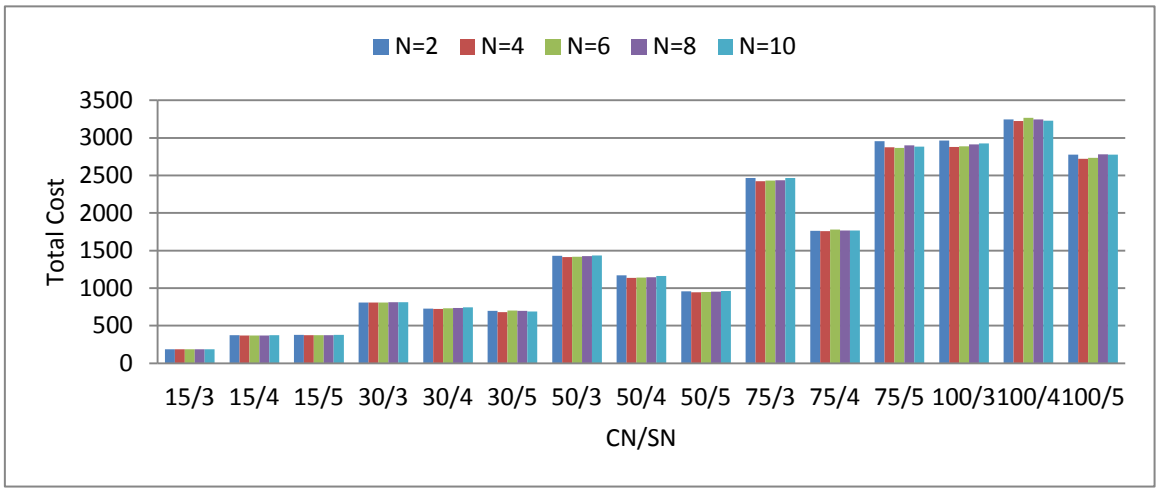


Fig. 14: Comparison of the average total cost in proposal algorithm with different depths of memory

Table 3 shows the p-values of the two-tailed t test and the p-values of the two-tailed wilcoxon rank sum test. The results reported in Table 3 show that for both types of statistical tests (wilcoxon and t test), the difference between the performance of the proposed algorithm with depth of memory 4 and the performance of the proposed algorithm with other depths of memory is statistically significant (p-value<0.05) in all cases.

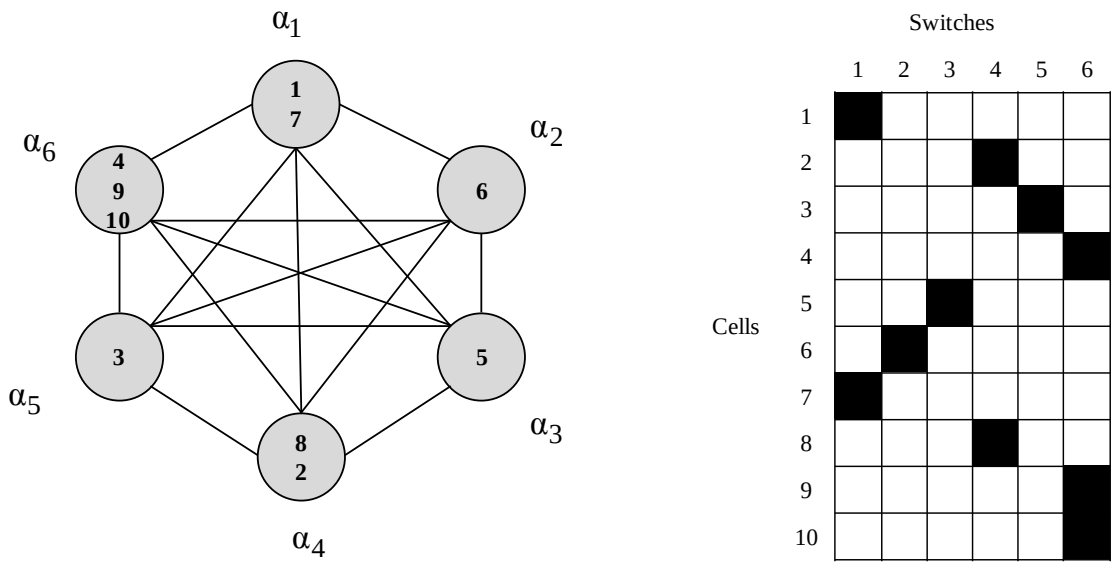
TABLE 3: THE RESULTS OF STATISTICAL TESTS FOR PROPOSED ALGORITHM WITH DEPTH OF MEMORY 4 VS. PROPOSED ALGORITHM WITH OTHER DEPTHS OF MEMORY

P#	CN	SN	Proposed Algorithm							
			N=2		N=6		N=8		N=10	
			t test	wilcoxon	t test	wilcoxon	t test	wilcoxon	t test	wilcoxon
1	15	3	4.15E-39	0.00E+00	5.99E-38	0.00E+00	7.36E-22	0.00E+00	6.96E-17	0.00E+00
2	15	4	8.54E-43	0.00E+00	1.79E-16	4.44E-16	1.27E-03	1.92E-03	2.08E-11	5.79E-11
3	15	5	1.52E-13	7.11E-15	6.15E-21	0.00E+00	1.02E-07	1.52E-07	1.07E-12	5.43E-12
4	30	3	1.20E-11	9.69E-12	9.10E-30	0.00E+00	1.89E-45	0.00E+00	7.84E-10	4.36E-09
5	30	4	8.45E-14	1.45E-11	1.65E-33	0.00E+00	1.30E-45	0.00E+00	4.21E-48	0.00E+00
6	30	5	3.15E-29	0.00E+00	2.18E-36	0.00E+00	1.60E-29	0.00E+00	1.49E-13	1.24E-12
7	50	3	1.52E-32	0.00E+00	1.58E-02	1.96E-02	1.24E-18	1.09E-14	2.40E-38	0.00E+00
8	50	4	5.98E-47	0.00E+00	1.48E-02	5.52E-03	1.47E-03	2.34E-03	2.33E-20	1.29E-14
9	50	5	7.52E-19	2.64E-14	7.76E-02	7.82E-02	2.51E-07	7.84E-07	9.19E-11	9.59E-11
10	75	3	2.72E-47	0.00E+00	1.20E-05	4.35E-05	2.93E-08	2.25E-07	1.70E-18	2.58E-14
11	75	4	4.69E-06	7.67E-06	9.27E-28	0.00E+00	1.59E-09	1.78E-08	2.71E-06	1.37E-06
12	75	5	3.43E-46	0.00E+00	7.00E-04	1.71E-03	9.44E-14	1.98E-11	1.29E-02	2.71E-02
13	100	3	8.76E-57	0.00E+00	1.20E-02	5.35E-03	2.80E-23	8.88E-16	4.67E-27	6.66E-16
14	100	4	4.74E-23	2.22E-16	9.59E-44	0.00E+00	1.42E-22	2.22E-16	3.20E-03	7.48E-03
15	100	5	3.43E-44	0.00E+00	4.34E-16	7.52E-13	9.62E-39	0.00E+00	3.56E-43	0.00E+00

5.2 **Experiment 2**

In this experiment we studied the impact of the memory depth (*N*) on the amount of reward received by assignments of cells to switches during a generation when proposed algorithm is

used. For this purpose we calculated the number of assignments of cells to switches which lead to reward function per generation for different values of depth of memory. The depth of memory was varied from 0 to 10 by increments of 2. Fig. 16 illustrates result of this experiment for test case #15 (CN=100 and SN=5). The obtained results show that the depth of memory has a large impact on the performance of the proposed algorithm and the maximum amount of reward received is obtained when $N \geq 4$. The proposed algorithm behaves like a random algorithm when *N=0*. Note that when the actions of OMA don't have memory, the history of the local search cannot be maintained by the algorithm and the algorithm behaves randomly (Fig. 15).



(a) (b)

S = [(1,0),(4,0),(5,0),(6,0),(3,0),(2,0),(1,0),(4,0),(6,0),(6,0)]

(c)

Fig. 15: Example of the state transition graph of a Tsetline-based OMA with *N=0* (a), the binary matrix *X* (b) and the solution vector *S* (c).

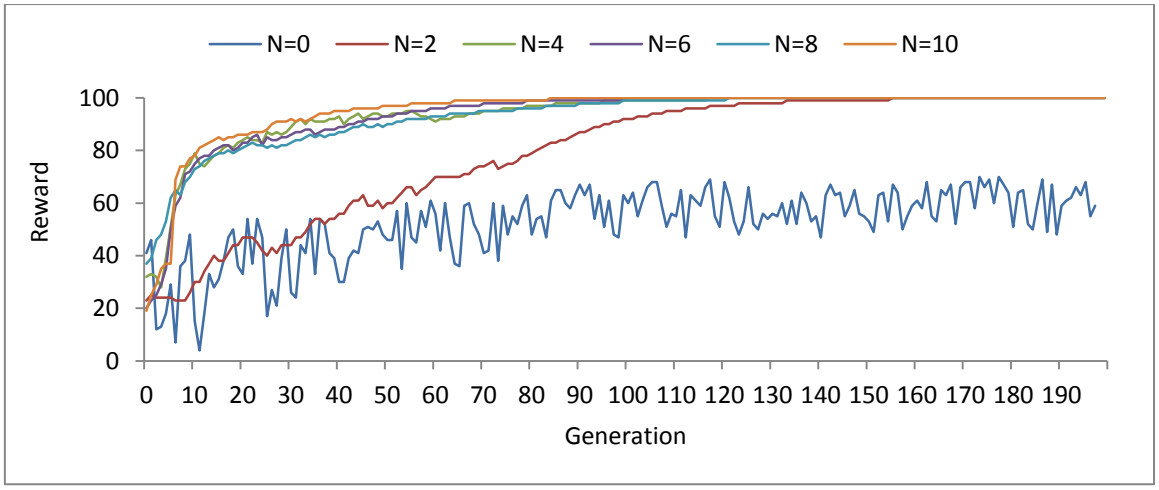


Fig. 16: The impact of depth of memory on the convergence rate of proposed algorithm for test case #15

5.3 **Experiment 3**

In this experiment, we compared the results obtained from the proposed algorithm with depth of memory 4 with the results of six other algorithms, a genetic algorithm (GA) [3], a memetic algorithm (MA) [4], two hybrid algorithm based on Hopfield network and genetic algorithm (HI and HII) [5], an Artificial Bee Colony algorithm (ABC) [6] and a hybrid algorithm based on API ant algorithm and tabu Search (API/TS) [7], for the ACTSP, in terms of the quality of solutions and speed of convergence. Table 4 presents the results of the different algorithms for 15 different cases with respect to the average (Avg.), minimum (Min.) and maximum (Max.) of total cost and their standard deviation (Std.). The results reported in Table 4 show that the proposed algorithm with depth of memory 4 performs better than other algorithms in all cases.

Table 5 shows the results obtained by the proposed algorithm for 15 different cases, with respect to the average (Avg.), minimum (Min.) and maximum (Max.) of running time of algorithms and their standard deviation (Std.). Comparing the results reported in Table 5, we find that, in all cases, the proposed algorithm with depth of memory 4 outperforms the others in terms of total cost and running time.

TABLE 4: COMPARISON OF THE MINIMUM (MIN.), MAXIMUM (MAX.), AVERAGE TOTAL COSTS (AVG.) AND THEIR STANDARD DEVIATION (STD.) OF THE PROPOSED ALGORITHM WITH N=4 AND OTHER ALGORITHMS

P#	CN	SN	GA				MA				HI				HII				ABC				API/TS				The proposed algorithm with N=4			
			Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.
1	15	3	189.40	4.72	176.99	201.14	187.41	8.46	171.96	207.20	185.95	1.08	183.91	188.23	185.28	0.42	184.53	186.33	195.21	5.05	184.55	209.27	195.12	5.41	184.48	207.93	185.08	0.12	184.79	185.29
2	15	4	379.22	11.70	358.02	399.85	374.93	6.71	361.01	388.55	371.28	2.15	365.43	376.94	369.86	0.85	367.44	371.62	384.41	12.23	364.52	405.90	383.41	12.64	363.86	407.32	369.52	0.08	369.38	369.69
3	15	5	394.78	16.10	341.21	421.66	376.65	5.46	367.65	387.87	379.44	5.12	368.65	389.84	376.66	2.63	371.95	382.01	400.17	16.22	346.23	428.05	400.02	16.34	347.80	430.60	371.51	0.15	371.27	371.82
4	30	3	840.28	23.23	798.41	895.70	824.61	11.44	801.08	847.71	832.26	21.44	788.64	876.30	811.27	10.34	785.01	833.58	845.41	23.33	800.86	901.13	844.69	23.75	804.47	901.11	805.65	1.23	802.77	808.73
5	30	4	770.18	20.96	710.45	814.82	757.24	16.17	721.31	795.98	741.76	11.89	718.63	768.48	744.22	12.34	720.68	765.98	776.68	21.24	719.36	821.89	776.26	21.22	717.36	823.17	724.25	2.36	718.90	730.38
6	30	5	740.20	18.55	691.00	780.65	754.48	36.13	699.03	836.99	733.45	25.24	669.29	774.85	721.73	11.94	694.43	747.07	745.50	18.00	699.72	782.29	745.57	18.08	703.52	785.49	681.54	5.63	670.95	697.35
7	50	3	1457.17	29.10	1385.38	1524.65	1447.81	30.67	1380.49	1516.94	1450.63	17.27	1410.91	1493.63	1436.19	14.54	1408.92	1479.06	1462.60	29.28	1393.16	1530.88	1462.45	30.01	1388.72	1529.78	1412.81	5.68	1402.07	1423.30
8	50	4	1205.36	29.38	1149.12	1290.53	1182.49	15.78	1139.33	1221.34	1221.62	48.45	1122.12	1317.71	1178.85	21.08	1136.88	1229.58	1211.52	29.46	1155.89	1298.07	1211.27	29.72	1157.36	1297.11	1137.64	6.12	1124.22	1149.66
9	50	5	994.61	31.21	915.41	1069.47	980.38	35.38	900.36	1057.30	979.74	15.89	943.31	1012.49	976.73	13.20	948.71	1007.41	1000.39	31.34	921.07	1075.89	1000.24	31.09	921.05	1074.95	945.33	5.24	934.22	961.26
10	75	3	2635.50	93.16	2449.31	2822.20	2588.28	141.46	2219.51	2956.00	2612.91	79.83	2395.42	2817.65	2529.58	40.58	2452.51	2661.35	2640.79	92.85	2452.13	2825.99	2640.09	93.14	2450.97	2829.15	2423.51	8.24	2404.77	2439.36
11	75	4	1884.77	47.32	1750.27	1973.48	1880.82	70.86	1702.74	2058.62	1870.14	50.26	1711.98	1976.50	1841.66	41.61	1734.57	1931.77	1890.25	47.45	1754.69	1978.98	1889.40	47.76	1753.51	1977.96	1758.39	6.89	1743.43	1777.41
12	75	5	3065.96	72.81	2942.28	3276.21	3062.93	143.48	2719.24	3395.76	3062.32	66.29	2920.73	3195.03	3017.16	54.75	2909.46	3164.07	3071.54	72.78	2950.38	3281.96	3070.72	73.54	2949.50	3284.54	2875.36	15.63	2850.10	2912.81
13	100	3	3118.51	75.35	2872.16	3268.71	3080.36	98.19	2840.89	3276.32	3114.76	94.12	2872.81	3327.86	3030.88	66.37	2914.58	3159.54	3123.90	75.32	2879.99	3272.18	3123.11	75.84	2875.57	3275.65	2879.36	12.78	2852.67	2916.14
14	100	4	3308.87	32.25	3248.29	3370.61	3337.11	96.06	3137.69	3492.03	3301.32	31.68	3219.32	3366.06	3287.58	31.21	3215.50	3375.32	3314.12	32.19	3251.83	3373.07	3314.07	32.31	3253.47	3372.53	3224.57	8.42	3202.68	3240.13
15	100	5	2942.27	68.60	2782.50	3074.31	2910.61	77.84	2763.69	3091.37	2857.51	52.41	2729.49	2968.01	2799.73	41.76	2703.34	2922.48	2947.79	68.73	2790.25	3082.07	2947.29	68.27	2793.13	3084.18	2719.49	7.52	2701.29	2736.12

TABLE 5: COMPARISON OF THE MINIMUM (MIN.), MAXIMUM (MAX.), AVERAGE RUNNING TIME IN SECOND (AVG.) AND THEIR STANDARD DEVIATION (STD.) OF THE PROPOSED ALGORITHM WITH N=4 AND OTHER ALGORITHMS

P#	CN	SN	GA				MA				HI				HII				ABC				API/TS				The proposed algorithm with N=4			
			Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.
1	15	3	1.8323	0.06	1.7176	1.9960	1.9900	0.00	1.9868	1.9932	4.5409	0.40	3.7136	5.4500	4.8737	0.02	4.8029	4.9033	9.9904	0.35	9.3363	10.6175	7.4667	0.37	6.8142	8.3511	1.5889	0.10	1.3696	1.8416
2	15	4	1.8694	0.07	1.7673	2.0712	2.7259	0.09	2.5564	2.9225	4.6388	0.48	3.4773	5.6441	4.9750	0.40	3.8000	5.7174	11.4876	0.32	10.6270	12.2374	9.9813	0.62	8.8560	11.0762	1.6648	0.06	1.5452	1.7667
3	15	5	1.8125	0.12	1.5613	2.0699	2.8049	0.14	2.4409	3.1912	4.8257	0.05	4.7132	4.9570	4.6524	0.49	3.1795	5.8759	13.9946	0.98	12.2592	16.5130	11.7125	0.67	10.4848	13.1309	1.5430	0.02	1.5035	1.6017
4	30	3	4.7060	0.39	3.9596	5.4944	6.8317	0.55	5.7901	8.8981	13.8450	1.23	10.9915	16.6513	14.8839	0.26	14.3939	15.6226	14.8094	0.28	13.9795	15.3262	14.0191	0.83	11.2472	15.4095	4.2119	0.21	3.7668	4.6249
5	30	4	4.9059	0.03	4.8285	4.9689	6.9612	0.31	6.3477	7.8915	13.8352	0.53	12.6892	14.9447	14.4611	1.19	10.6352	17.1277	16.5586	1.51	12.3985	19.0403	17.0129	0.24	16.5726	17.5958	4.0736	0.33	3.3991	4.8110
6	30	5	4.7567	0.19	4.3320	5.1088	6.4159	0.37	5.7540	7.1439	14.4120	1.26	11.5132	17.0840	14.4120	1.37	11.9615	17.2764	19.5694	1.31	16.0805	22.0345	19.0490	1.08	15.9769	21.3449	4.1799	0.20	3.8118	4.6020
7	50	3	13.8070	0.91	11.7885	15.6636	14.5316	1.27	11.8469	17.9614	18.8326	0.08	18.6523	19.0265	19.4879	0.72	17.9052	21.1244	42.2387	1.61	38.1263	45.7903	32.8369	2.81	25.9051	39.3530	11.5692	0.57	10.5100	12.7490
8	50	4	14.1500	0.52	13.0040	15.5265	16.9542	0.92	14.7821	19.4149	18.3618	1.43	15.5447	20.9673	19.6787	0.09	19.4764	19.8499	63.8283	0.54	62.3118	64.8978	35.8825	1.17	33.3177	38.2984	12.1471	0.97	10.1781	14.3729
9	50	5	14.1658	1.02	11.2402	16.8971	16.9080	0.64	15.5024	17.9455	23.2468	2.21	16.4059	26.4473	24.1592	1.85	20.3049	28.5949	68.0362	4.94	54.8925	79.5484	38.9360	3.54	30.4345	45.5714	12.1086	0.83	10.6575	14.5880
10	75	3	19.6508	1.61	16.2294	23.2452	19.8945	1.34	16.7988	22.1567	27.8595	0.84	26.0939	29.2527	27.2607	2.55	21.6695	33.0028	94.5721	6.41	78.3870	112.0879	67.6354	2.42	61.9766	72.7865	16.6066	0.60	15.0429	17.6882
11	75	4	19.8591	0.06	19.7206	20.0072	23.0901	1.68	17.9671	26.2190	27.8743	1.45	25.0715	31.6499	28.4584	2.01	22.9684	33.4010	98.7390	6.77	81.0952	116.3324	68.5551	3.67	61.5343	79.1870	15.5934	1.18	12.9964	18.4925
12	75	5	18.2150	0.73	16.2335	19.5364	24.4183	0.43	23.4097	25.1492	32.8675	0.72	31.1508	34.1062	32.1582	3.02	25.3317	38.7207	107.9417	9.41	85.4045	129.7670	81.5414	6.52	64.9909	97.1863	15.2816	0.59	13.9627	16.7042
13	100	3	22.6427	0.09	22.4848	22.8458	34.6613	1.82	29.5651	38.8595	38.5479	2.73	33.0162	46.3425	36.5431	1.37	33.9237	39.1470	327.2466	7.97	307.8109	343.1508	139.4553	8.76	121.2197	155.6807	21.0362	1.27	18.2369	23.8174
14	100	4	23.2519	2.28	16.0330	28.2700	33.6673	0.44	32.7809	34.7927	37.5229	1.87	33.8067	41.1858	39.1323	1.31	36.5331	42.0740	397.3316	18.09	359.7714	426.5101	157.8620	12.56	136.2829	188.4163	20.1493	0.26	19.4861	20.9032
15	100	5	24.9875	1.43	21.7167	27.6380	35.5675	2.89	29.3097	42.0003	40.9939	3.31	33.8331	46.5412	43.4745	4.26	32.2291	50.5305	464.3327	40.72	387.4756	558.8969	184.1800	8.83	166.5403	201.8684	21.1796	0.83	19.2419	22.7810

Fig. 17 shows the comparison of results reported in Table 4.

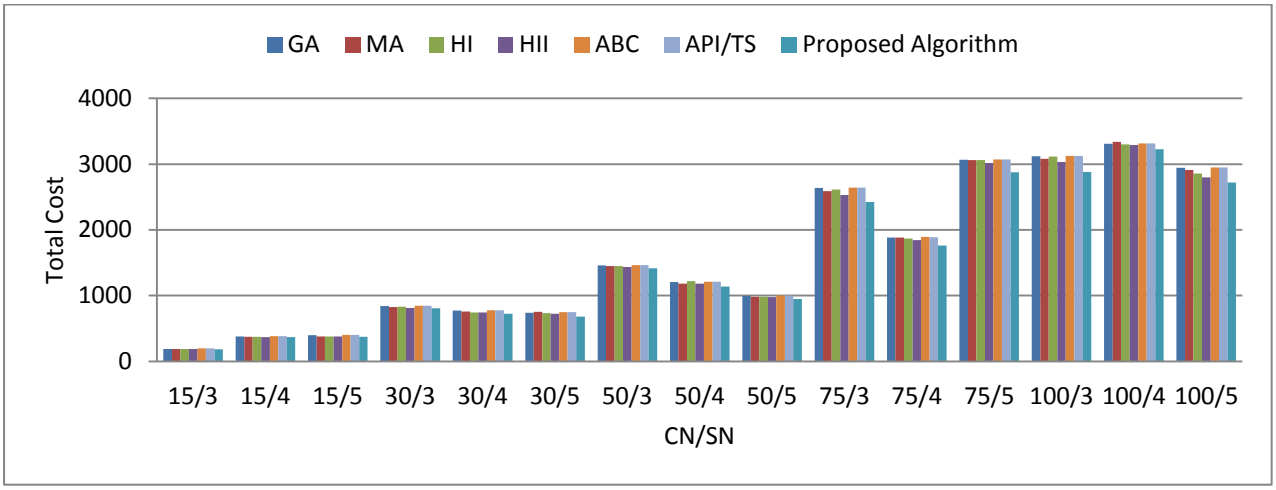


Fig. 17: Comparison of the total cost in proposed algorithm with depths of memory 4 with other algorithms

Table 6 shows the p-values of the two-tailed t test and the p-values of the two-tailed wilcoxon rank sum test. The results reported in Table 6 shows that for both types of statistical tests (wilcoxon and T test), the difference between the performance of the proposed algorithm with depth of memory 4 and the performance of the other algorithms is statistically significant (p-value<0.05) in all cases.

TABLE 6: THE RESULTS OF STATISTICAL TESTS FOR PROPOSED ALGORITHM WITH DEPTH OF MEMORY 4 VS. OTHER ALGORITHMS

P#	CN	SN	GA		MA		HI		HII		ABC		API/TS	
			t test	wilcoxon	t test	wilcoxon	t test	wilcoxon	t test	wilcoxon	t test	wilcoxon	t test	wilcoxon
1	15	3	4.41E-08	5.49E-10	5.78E-02	1.20E-01	1.11E-05	6.47E-05	1.70E-03	6.60E-03	5.82E-	2.22E-15	1.17E-	2.22E-16
2	15	4	3.76E-07	3.53E-05	6.79E-07	1.40E-06	6.92E-07	1.44E-07	6.90E-03	1.07E-03	2.29E-	4.64E-09	4.29E-	3.49E-08
3	15	5	9.69E-14	4.54E-13	2.31E-08	7.43E-06	7.10E-15	3.38E-14	1.34E-18	0.00E+00	7.53E-	3.38E-14	1.20E-	3.38E-14
4	30	3	3.31E-14	1.78E-15	7.00E-16	7.84E-14	1.20E-11	1.27E-12	3.71E-04	5.67E-05	2.79E-	2.22E-16	1.05E-	0.00E+00
5	30	4	1.17E-20	2.22E-16	1.82E-19	2.22E-16	4.41E-14	3.52E-13	1.38E-15	1.42E-14	7.40E-	2.22E-16	1.03E-	2.22E-16
6	30	5	3.35E-29	0.00E+00	2.65E-19	0.00E+00	7.79E-20	4.44E-16	1.56E-32	0.00E+00	5.99E-	0.00E+00	7.34E-	0.00E+00
7	50	3	1.20E-14	6.04E-14	1.54E-10	1.07E-11	2.15E-21	2.22E-16	1.11E-15	2.44E-14	2.03E-	1.22E-14	5.95E-	1.42E-14
8	50	4	6.13E-22	0.00E+00	1.52E-27	0.00E+00	1.17E-16	2.89E-15	4.08E-19	4.44E-16	1.43E-	0.00E+00	2.52E-	0.00E+00
9	50	5	3.59E-15	2.44E-14	6.94E-09	1.58E-09	3.16E-21	0.00E+00	1.49E-23	0.00E+00	6.46E-	3.77E-15	5.25E-	3.11E-15
10	75	3	2.74E-21	0.00E+00	8.33E-11	1.23E-11	4.25E-22	2.22E-16	2.27E-24	0.00E+00	8.41E-	0.00E+00	1.10E-	0.00E+00
11	75	4	1.74E-24	2.22E-16	1.54E-16	3.38E-14	5.70E-21	2.22E-16	3.61E-19	4.44E-16	2.93E-	0.00E+00	5.30E-	0.00E+00
12	75	5	1.81E-24	0.00E+00	2.48E-12	1.67E-12	4.16E-26	0.00E+00	1.05E-24	0.00E+00	4.67E-	0.00E+00	9.47E-	0.00E+00
13	100	3	4.58E-28	0.00E+00	1.74E-19	4.44E-16	3.39E-23	0.00E+00	1.09E-21	0.00E+00	1.58E-	0.00E+00	2.58E-	0.00E+00
14	100	4	9.73E-25	0.00E+00	7.04E-11	8.36E-11	3.49E-23	0.00E+00	1.17E-19	2.00E-15	4.89E-	0.00E+00	6.18E-	0.00E+00
15	100	5	3.69E-28	0.00E+00	1.05E-22	0.00E+00	3.42E-24	0.00E+00	1.73E-18	1.78E-15	1.31E-	0.00E+00	1.06E-	0.00E+00

5.4 **Experiment 4**

This experiment aimed to find the impact of learning automata and local search on the proposed algorithm performance. For this purpose we compared the proposed algorithm with depth of memory 4 (LA-ACTSP(*N=4*)) with the proposed algorithm without memory (LA-ACTSP(*N=0*)) and the proposed algorithm in which the rewrad and penalty functions is

replaced by a pure chance function (LA-ACTSP-PC) , in terms of quality of solution and speed of convergence. In pure chance function a state φ_i passes to a state φ_{i+1} with probability 1/2 and to state φ_{i-1} with probability 1/2.

Table 7 presents the results of these algorithms for 15 different cases with respect to the average (Avg.), minimum (Min.) and maximum (Max.) of total cost and their standard deviation (Std.). Also, Table 8 presents the results of these algorithms for same cases with respect to the average (Avg.), minimum (Min.) and maximum (Max.) of running time of different algorithms and their standard deviation (Std.).

From the results described in Table 7 and Table 8 we report the following:

- The LA-ACTSP(*N=4*) algorithm can find solutions with the lowest cost for all cases.
- However, the algorithm LA-ACTSP(*N=0*) is faster than algorithm LA-ACTSP(*N=4*) due to lack of memory and according to the termination condition of the algorithm (locating all cells in the most internal state of their switches), but the cost of the solutions found in this algorithm is greater than the cost of the solutions found in algorithm LA-ACTSP(*N=4*).
- Due to the random nature of algorithm LA-ACTSP-PC, this algorithm has a weaker performance than in algorithm LA-ACTSP(*N=4*) both in terms of total cost and running time.

TABLE 7: COMPARISON OF THE MINIMUM (MIN.), MAXIMUM (MAX.), AVERAGE TOTAL COSTS (AVG.) AND THEIR STANDARD DEVIATION (STD.) IN LA-ACTSP(N=4), LA-ACTSP(N=0) AND LA-ACTSP-PC

P#	CN	SN	LA-ACTSP(N=4)				LA-ACTSP(N=0)				LA-ACTSP-PC			
			Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.
1	15	3	185.08	0.12	184.79	185.29	240.42	11.90	218.71	266.55	241.29	8.21	221.49	255.59
2	15	4	369.52	0.08	369.38	369.69	494.29	20.94	434.83	534.07	494.93	14.8	451.01	518.50
3	15	5	371.51	0.15	371.27	371.82	495.54	32.84	427.75	583.74	468.34	22.3	417.08	528.32
4	30	3	805.65	1.23	802.77	808.73	1057.9	44.07	923.21	1153.8	1182.0	18.4	1127.3	1225.5
5	30	4	724.25	2.36	718.90	730.38	970.57	27.81	916.36	1023.8	970.41	20.9	932.44	1013.1
6	30	5	681.54	5.63	670.95	697.35	931.64	32.82	835.46	996.08	940.29	35.6	856.94	1018.5
7	50	3	1412.8	5.68	1402.0	1423.3	1814.5	149.8	1519.4	2194.0	1820.4	30.3	1749.4	1890.4
8	50	4	1137.6	6.12	1124.2	1149.6	1555.7	83.13	1398.1	1730.6	1556.3	40.4	1491.5	1656.1
9	50	5	945.33	5.24	934.22	961.26	1244.1	70.07	1075.6	1405.0	1247.3	34.4	1168.1	1318.2
10	75	3	2423.5	8.24	2404.7	2439.3	3318.2	110.9	3042.7	3638.4	3313.1	45.2	3208.5	3411.5
11	75	4	1758.3	6.89	1743.4	1777.4	2452.2	155.8	2111.6	2887.7	2453.7	53.0	2336.3	2549.0
12	75	5	2875.3	15.6	2850.1	2912.8	3906.7	180.7	3462.7	4278.0	3931.6	40.3	3836.2	4033.9
13	100	3	2879.3	12.7	2852.6	2916.1	4056.7	165.5	3769.1	4381.5	4056.3	33.5	3985.8	4134.5
14	100	4	3224.5	8.42	3202.6	3240.1	4194.4	143.9	3736.4	4460.2	4195.4	48.7	4100.3	4303.5
15	100	5	2719.4	7.52	2701.2	2736.1	3732.3	153.5	3345.1	4168.3	3706.8	34.9	3603.0	3776.4

TABLE 8: COMPARISON OF THE MINIMUM (MIN.), MAXIMUM (MAX.), AVERAGE RUNNING TIME (AVG.) AND THEIR STANDARD DEVIATION (STD.) OF LA-ACTSP(N=4), LA-ACTSP(N=0) AND LA-ACTSP-PC

P#	CN	SN	LA-ACTSP(N=4)				LA-ACTSP(N=0)				LA-ACTSP-PC			
			Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.	Avg.	Std.	Min.	Max.
1	15	3	1.5889	0.10	1.3696	1.8416	1.21	0.4551	0.42	2.03	43.57	4.8638	32.81	52.16
2	15	4	1.6648	0.06	1.5452	1.7667	1.31	0.5580	0.02	2.46	57.71	6.0563	41.73	67.99
3	15	5	1.5430	0.02	1.5035	1.6017	1.41	0.4271	0.34	2.24	52.69	6.4538	40.97	69.70
4	30	3	4.2119	0.21	3.7668	4.6249	1.57	0.5676	0.55	3.32	139.17	8.8912	118.97	155.53
5	30	4	4.0736	0.33	3.3991	4.8110	3.19	0.8773	1.44	5.53	161.18	12.0285	134.47	189.78
6	30	5	4.1799	0.20	3.8118	4.6020	2.80	1.0407	0.59	5.24	184.56	13.4548	151.29	211.53
7	50	3	11.5692	0.57	10.5100	12.7490	2.19	0.6909	0.70	3.49	108.35	8.8544	83.42	124.16
8	50	4	12.1471	0.97	10.1781	14.3729	2.61	0.6971	1.13	4.17	137.50	11.2139	109.61	161.95
9	50	5	12.1086	0.83	10.6575	14.5880	2.67	0.9854	0.78	5.88	173.23	11.5343	141.43	197.83
10	75	3	16.6066	0.60	15.0429	17.6882	3.44	0.8818	1.37	4.75	260.96	19.7335	217.74	304.99
11	75	4	15.5934	1.18	12.9964	18.4925	2.88	1.4050	0.04	7.26	279.82	20.8664	227.34	321.24
12	75	5	15.2816	0.59	13.9627	16.7042	3.52	1.4502	0.77	7.91	236.42	19.2859	168.12	263.19
13	100	3	21.0362	1.27	18.2369	23.8174	3.68	1.2047	1.13	6.18	337.13	23.3097	271.72	392.06
14	100	4	20.1493	0.26	19.4861	20.9032	4.23	1.4322	0.76	7.12	488.66	19.4861	445.15	529.28
15	100	5	21.1796	0.83	19.2419	22.7810	3.77	1.4640	0.59	6.86	509.39	23.8852	466.79	583.68

6 Conclusion

A new algorithm is proposed for solving the assignment of cells to switches problem in cellular mobile network in this paper. The proposed algorithm is obtained from the combination of learning automata and local search method. In this algorithm, solution is represented by OMAs, in which the OMA actions correspond to switches and the OMA states keep information about the history of the local search process. The number of states in each action determines the depth of memory. Each state in the OMA has two attributes: the value of assigned object, and the degree of association with its value. The cells, as migratory objects are assigned to states of the learning automaton. The local search changes the degree of association between the assigned objects and their values. Experimental results have shown the superiority of the proposed algorithm in terms of quality of solution and speed of convergence. Also, the obtained results have shown that the depth of memory has a large impact on the performance of the proposed algorithm and the best results are obtained when the depth of memory is set to 4. All algorithms reported for solving ACTSP are designed under the assumption that the number of cells or switches are fixed (off-line version). The on-line version of ACTSP where the number of cells or switches changes due to the environment change or communication traffics may also be pursued.

References

[1] M. Toril, P. Guerrero-García, S. Luna-Ramírez, and V. Wille, "An efficient integer programming formulation for the assignment of base stations to controllers in cellular networks," *Computer Networks,* vol. 56, pp. 303-314, 2012.

[2] S. N. Chaurasia and A. Singh, "A hybrid evolutionary approach to the registration area planning problem," *Applied Intelligence,* vol. 41, pp. 1127-1149, December 01 2014.

[3] S. Khuri and T. Chiu, "Heuristic algorithms for the terminal assignment problem," in *Proceedings of the 1997 ACM symposium on Applied computing*, 1997, pp. 247-251.

[4] A. Quintero and S. Pierre, "Sequential and multi-population memetic algorithms for assigning cells to switches in mobile networks," *Computer Networks,* vol. 43, pp. 247-261, 2003.

[5] S. Salcedo-Sanz and X. Yao, "Assignment of cells to switches in a cellular mobile network using a hybrid Hopfield network-genetic algorithm approach," *Applied Soft Computing,* vol. 8, pp. 216-224, 2008.

[6] A. Vafadarnikjoo, S. M. A. K. Firouzabadi, M. Mobin, and A. Roshani, "A meta-heuristic approach to locate optimal switch locations in cellular mobile networks," 2015.

[7] F. Debbat and F. T. Bendimerad, "Assigning cells to switches in cellular mobile networks using hybridizing API algorithm and Tabu Search," *International Journal of Communication Systems,* vol. 27, pp. 4028-4037, 2014.

[8] J. R. Fournier and S. Pierre, "Assigning cells to switches in mobile networks using an ant colony optimization heuristic," *Computer Communications,* vol. 28, pp. 65-73, 2005.

[9] A. Quintero and S. Pierre, "Evolutionary approach to optimize the assignment of cells to switches in personal communication networks," *Computer Communications,* vol. 26, pp. 927-938, 2003.

[10] A. Quintero and S. Pierre, "Assigning cells to switches in cellular mobile networks: a comparative study," *Computer Communications,* vol. 26, pp. 950-960, 2003.

[11] S. Menon and R. Gupta, "Assigning cells to switches in cellular networks by incorporating a pricing mechanism into Simulated annealing," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* vol. 34, pp. 558-565, 2004.

[12] S. K. Goudos, K. B. Baltzis, C. Bachtsevanidis, and J. N. Sahalos, "Cell-to-switch assignment in cellular networks using barebones particle swarm optimization," *IEICE Electronics Express,* vol. 7, pp. 254-260, 2010.

[13] J. Wang, Y. Cai, Y. Zhou, R. Wang, and C. Li, "Discrete particle swarm optimization based on estimation of distribution for terminal assignment problems," *Computers & Industrial Engineering,* vol. 60, pp. 566-575, 2011.

[14] M. Vroblefski and E. C. Brown, "A grouping genetic algorithm for registration area planning," *Omega,* vol. 34, pp. 220-230, 2006/06/01/ 2006.

[15] S. N. Chaurasia and A. Singh, "A hybrid swarm intelligence approach to the registration area planning problem," *Information Sciences,* vol. 302, pp. 50-69, 2015/05/01/ 2015.

[16] T. James, M. Vroblefski, and Q. Nottingham, "A hybrid grouping genetic algorithm for the registration area planning problem," *Computer Communications,* vol. 30, pp. 2180-2190, 2007/07/31/ 2007.

[17] K. S. Narendra and M. A. L. Thathachar, *Learning automata: an introduction*: Prentice-Hall, Inc., 1989.

[18] M. A. L. Thathachar and P. S. Sastry, "Varieties of learning automata: an overview," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics,* vol. 32, pp. 711-722, 2002.
[19] M. Rezapoor Mirsaleh and M. R. Meybodi, "A new memetic algorithm based on cellular learning automata for solving the vertex coloring problem," *Memetic Computing,* vol. 8, pp. 211-222, 2016.
[20] B. Oommen and E. Hansen, "The asymptotic optimality of discretized linear reward-inaction learning automata," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 14, pp. 542-545, 1984.
[21] B. Johnoommen, "Absorbing and ergodic discretized two-action learning automata," *IEEE Transactions on Systems, Man and Cybernetics,* vol. 16, pp. 282-293, 1986.
[22] J. Akbari Torkestani and M. R. Meybodi, "Learning automata-based algorithms for finding minimum weakly connected dominating set in stochastic graphs," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems,* vol. 18, pp. 721-758, 2010.
[23] M. Rezapoor Mirsaleh and M. R. Meybodi, "A Learning Automata-based Memetic Algorithm," *Genetic Programming and Evolvable Machines,* vol. 16, pp. 399-453, 2015.
[24] J. Akbari Torkestani, "An adaptive focused Web crawling algorithm based on learning automata," *Applied Intelligence,* vol. 37, pp. 586-601, 2012.
[25] R. Vafashoar, M. R. Meybodi, and A. H. Momeni Azandaryani, "CLA-DE: a hybrid model based on cellular learning automata for numerical optimization," *Applied Intelligence,* vol. 36, pp. 735-748, 2012.
[26] M. Rezapoor Mirsaleh and M. R. Meybodi, "Balancing exploration and exploitation in memetic algorithms: A learning automata approach," *Computational Intelligence,* 2017.
[27] J. Akbari Torkestani and M. R. Meybodi, "An efficient cluster-based CDMA/TDMA scheme for wireless mobile ad-hoc networks: A learning automata approach," *Journal of Network and Computer applications,* vol. 33, pp. 477-490, 2010.
[28] J. Akbari Torkestani and M. R. Meybodi, "Mobility-based multicast routing algorithm for wireless mobile Ad-hoc networks: A learning automata approach," *Computer Communications,* vol. 33, pp. 721-735, 2010.
[29] J. Akbari Torkestani and M. R. Meybodi, "An intelligent backbone formation algorithm for wireless ad hoc networks based on distributed learning automata," *Computer Networks,* vol. 54, pp. 826-843, 2010.
[30] H. Beigy and M. R. Meybodi, "Learning automata based dynamic guard channel algorithms," *Computers & Electrical Engineering,* vol. 37, pp. 601-613, 2011.
[31] J. Akbari Torkestani and M. R. Meybodi, "LLACA: An adaptive localized clustering algorithm for wireless ad hoc networks," *Computers & Electrical Engineering,* vol. 37, pp. 461-474, 2011.
[32] M. Jahanshahi, M. Dehghan, and M. Meybodi, "LAMR: learning automata based multicast routing protocol for multi-channel multi-radio wireless mesh networks," *Applied Intelligence,* vol. 38, pp. 58-77, 2013.
[33] M. Rezapoor Mirsaleh and M. R. Meybodi, "A Michigan memetic algorithm for solving the community detection problem in complex network," *Neurocomputing,* vol. 214, pp. 535-545, 2016.
[34] M. R. Meybodi, "Learning automata and its application to priority assignment in a queueing system with unknown characteristics," Ph.D. thesis, Departement of Electrical Engineering and Computer Science, University of Oklahoma, Norman, Oklahoma, USA, 1983.

[35] M. L. Tsetlin, *Automaton theory and modeling of biological systems* vol. 102: Academic Press New York, 1973.
[36] A. Hashim, S. Amir, and P. Mars, "Application of learning automata to data compression," *Adaptive and learning systems*, pp. 229-234, 1986.
[37] B. Manjunath and R. Chellappa, "Stochastic learning networks for texture segmentation," in *Twenty-Second Asilomar Conference on Signals, Systems and Computers*, 1988, pp. 511-516.
[38] M. Rezapoor Mirsaleh and M. R. Meybodi, "A Michigan memetic algorithm for solving the vertex coloring problem," *Journal of Computational Science*, 2017.
[39] G. P. Frost, "Stochastic optimisation of vehicle suspension control systems via learning automata," Ph.D. Thesis, Department of Aeronautical and Automotive Engineering, Loughborough University, Loughborough, 1998.
[40] M. Howell, G. Frost, T. Gordon, and Q. Wu, "Continuous action reinforcement learning applied to vehicle suspension control," *Mechatronics*, vol. 7, pp. 263-276, 1997.
[41] C. Unsal, P. Kachroo, and J. S. Bay, "Multiple stochastic learning automata for vehicle path control in an automated highway system," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 29, pp. 120-128, 1999.
[42] H. Beigy and M. R. Meybodi, "A learning automata-based algorithm for determination of the number of hidden units for three-layer neural networks," *International Journal of Systems Science*, vol. 40, pp. 101-118, 2009.
[43] M. R. Meybodi and H. Beigy, "Neural network engineering using learning automata: determining of desired size of three layer feed forward neural networks," *Journal of Faculty of Engineering (University of Tehran)*, vol. 34, pp. 1-26, 2001.
[44] B. J. Oommen and D. S. Croix, "String taxonomy using learning automata," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 27, pp. 354-365, 1997.
[45] A. G. Barto and M. I. Jordan, "Gradient following without back-propagation in layered networks," in *1st Int. Conference Neural Nets, San Diego*, 1987.
[46] M. Thathachar and V. V. Phansalkar, "Learning the global maximum with parameterized learning automata," *IEEE Transactions on Neural Networks*, vol. 6, pp. 398-406, 1995.
[47] B. J. Oommen and D. C. Y. Ma, "Deterministic learning automata solutions to the equipartitioning problem," *IEEE Transactions on Computers*, vol. 37, pp. 2-13, 1988.