A NEW VERTEX COLORING ALGORITHM BASED ON VARIABLE ACTION-SET LEARNING AUTOMATA

Javad Akbari Torkestani
Department of Computer Engineering, Islamic Azad University, Science and Research Branch, Tehran, Iran
j-akbari@iau-arak.ac.ir

Mohammad Reza Meybodi
Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran
mmeybodi@aut.ac.ir

Abstract. In this paper, we propose a learning automata-based iterative algorithm for approximating a near optimal solution to the vertex coloring problem. Vertex coloring is a well-known NP-hard optimization problem in graph theory in which each vertex is assigned a color so that no two adjacent vertices have the same color. Each iteration of the proposed algorithm is subdivided into several stages, and at each stage a subset of the uncolored non adjacent vertices are randomly selected and assigned the same color. This process continues until no more vertices remain uncolored. As the proposed algorithm proceeds, taking advantage of the learning automata the number of stages per iteration and so the required number of colors tends to the chromatic number of the graph since the number of vertices which are colored at each stage is maximized. To show the performance of the proposed algorithm we compare it with several existing vertex coloring algorithms in terms of the time and the number of colors required for coloring the graphs. The obtained results show the superiority of the proposed algorithm over the others.

Keyword: Graph coloring, vertex coloring, learning automata, iterative algorithms, legal coloring

1 Introduction

The *vertex coloring problem* is a well-known combinatorial optimization problem in graph theory [1], which is widely used in real life applications like computer register allocation [2], air traffic flow management [3], timetabling [4], scheduling [5], frequency assignment [6], and light wavelengths assignment in optical networks [7]. A *legal vertex coloring* of graph $G=(V,E)$, where $V(G)$ is the set of $|V|=n$ vertices and $E(G)$ is the edge set including $|E|=m$ edges, is a function $f:V\rightarrow C$ from the vertices of the graph G to the color-set $C=\{c_1,c_2,...,c_p\}$ such that $f(u)\neq f(v)$ for all edges $(u,v)\in E$.

That is, a legal vertex coloring of G is assigning one of p distinct colors to each vertex of the graph in such a way that no two endpoints of any edge are given the same colors. Formally, the vertex coloring problem can be either considered as an optimization problem or as a decision problem. The optimization version of the vertex coloring problem is intended to find the smallest number of colors by which the graph can be legally colored, and the decision problem aims at deciding for a given p whether or not the graph is p-colorable, and is called p-*coloring problem*.

A given graph G is p-*colorable*, if it can be legally colored with at most p different colors. The *chromatic number* $\chi(G)$ is the minimum number of colors required for coloring the graph, and a graph G is said to be p-*chromatic*, if $\chi(G) = p$. A *minimum coloring* of G is a legal coloring in which the smallest number of colors (i.e., chromatic number) to be assigned to the vertices. The *minimum coloring problem* is formally an NP-hard problem for general graphs as determining the chromatic number is known to be NP-hard [8]. It is shown in [9] that the decision problem of graph p-*colorability* (p-*coloring problem*) is an NP-complete problem for $p \geq 3$, and can be solved in polynomial time otherwise. Since the vertex coloring problem is known to be NP-hard for general graphs, all the exact algorithms [10-11] can only be applied on small graphs, while very large graphs often arise in a variety of applications. On the other hand, in applications, it often suffices to find a near optimal coloring of the graph. Hence, a large number of polynomial time algorithms have been proposed to approximate the near optimal solutions to the coloring problem [12]. The approximation approaches reported in the literatures can be classified as local search approaches [13], genetic algorithms [14], fuzzy-based optimizations [15], evolutionary algorithms [16], simulated annealing methods [17], ant colony-based approaches [18], Markov chain approaches [19], neural network approaches [20] and so on.

It is shown in [9] the graph coloring problem is an NP-hard problem in graph theory, an so a host of approximation and exact solutions have been proposed to solve it. For instance, in [21], Hertz and Werra proposed a tabu search algorithm in which a partition of the vertices of the graph is maintained at each iteration. In this algorithm, a different color is assigned to each block of the partition, which is not always guaranteed to be an independent set. Therefore, this algorithm works with the solutions which are not necessarily feasible. At each iteration, a sample of neighbors of each given configuration is generated. The set of neighbors generated for each vertex is restricted by a tabu list which prevents the algorithm from getting stuck in local optima.

Caramia and Dell'Olmo [22] proposed a local search algorithm, called HCD, based on tabu search. The basic idea behind HCD was to make use of tabu concepts without explicitly representing tabu lists. Instead, a dynamic assignment of priorities to the vertices in the graph performed the same task, avoiding repetitions in subsequent moves of the algorithm. They showed in [22] the experimental gain of HCD over tabu search. An iterated local search algorithm (ILS) was proposed by Lourenco et al.[23] to solve the graph coloring problem. The proposed algorithm is based on a randomized walk in the space of the local optima. This walk is built by iteratively perturbing a locally optimal solution, next applying a local search algorithm to obtain a new locally optimal solution, and finally using an acceptance criterion for deciding from which of these solutions to continue the search. Voudouris and Tsang [24] proposed a meta-heuristic search method known as *Guided Local Search* (GLS) to solve the combinatorial optimization problems. A GLS is a stochastic local search method in which the evaluation function is modified so as to escape from the local optima and plateaus. Chiarandin and Stutzle applied GLS to solve the graph coloring problem [25]. In this method, the objective function is modified using a specific scheme, when the local search algorithm settles in a local optimum. Caramia et al.[26] proposed a priority search algorithm, called CHECKCOL, in which the running time decreases by avoiding unnecessary searches in large portions of the graph without making any progress in the solution. To do this, they introduced the notion of check point, and forced the algorithm to stop at certain steps, to release all of its memory, and to start a new local search.

An adaptive algorithm, called AMACOL, proposed by Galiner et al.[27] for the solution of the graph coloring problem. The adaptive memory algorithm is a hybrid evolutionary heuristic in which a central memory is used to store the stable sets that originate from the colorings generated during the previous stages of the search. On each generation, a randomized greedy set covering heuristic is used to find a set of color classes that covers the set of vertices of the graph. This covering is transformed into a coloring in a straightforward way. Then, an iterative neighborhood technique is applied to the coloring. Eventually, the central memory is updated by using the color classes of the new obtained coloring. Caramia and Dell'Olmo [28] also proposed a two-phased local search for vertex coloring. The algorithm alternately executes two closely interacting functionalities, namely, a stochastic and a deterministic local search. The stochastic phase is based on a biased random sampling in which the feasible colorings are iteratively constructed. In deterministic phase, each vertex is assigned to the color which causes the lowest increase of the solution penalty. In this algorithm, the objective function tries to minimize the penalty function.

In this paper, a learning automata-based algorithm is proposed for solving the vertex coloring problem. In the proposed algorithm, a learning automaton is assigned to each vertex of the graph. Each iteration of the proposed algorithm is divided into several stages, and at each stage a subset of the non adjacent vertices of the graph is selected by the learning automata and assigned the same color. The remaining uncolored vertices are selected and colored in the next stages. An iteration is completed if all the vertices are colored. The proposed algorithm guarantees that the graph can be legally colored at each iteration. Then, the coloring is evaluated by the random environment as follows. If the number of selected colors (the color-set size) is less than the size of color-sets selected earlier, the colorings is rewarded and penalized otherwise. As the proposed algorithm approaches to the end, each vertex learns how to select one of its non adjacent vertices (or one of its actions) so that the number of vertices which are selected in a given stage can be maximized. Therefore, as the algorithm proceeds, the number of stages per iteration decreases as the number of vertices colored at each stage increases.

The proposed algorithm is compared with several approximation vertex coloring algorithms like ILS[23], GLS[24], CHECKCOL[26], AMACOL[27] and TPA[28], and the obtained results show the superiority of the proposed algorithm over the others in terms of the color-set size (or the number of required colors) and running time of algorithm. The rest of the paper is organized as follows. The learning automata are described in the next section. In section 3, the proposed learning automata-based algorithm is described. The performance of the proposed algorithm is evaluated through the simulation experiments in section 4, and section 5 concludes the paper.

2 Learning Automata, and Variable Action-Set Learning Automata

2.1 Learning Automata

A learning automaton [29,30] is an adaptive decision-making unit that improves its performance by learning how to choose the optimal action from a finite set of allowed actions through repeated interactions with a random environment. The action is chosen at random based on a probability distribution kept over the action-set and at each instant the given action is served as the input to the random environment. The environment responds the taken action in turn with a reinforcement signal. The action probability vector is updated based on the reinforcement feedback from the environment. The objective of a learning automaton is to find the optimal action from the action-set so that the average penalty received from the environment is minimized [29].

The environment can be described by a triple $E \equiv \{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, ..., \alpha_r\}$ represents the finite set of the inputs, $\beta \equiv \{\beta_1, \beta_2, ..., \beta_m\}$ denotes the set of the values can be taken by the reinforcement signal, and $c \equiv \{c_1, c_2, ..., c_r\}$ denotes the set of the penalty probabilities, where the element c_i is associated with the given action α_i. If the penalty probabilities are constant, the random environment is said to be a stationary random environment, and if they vary with time, the environment is called a non stationary environment. The environments depending on the nature of the reinforcement signal $\underline{\beta}$ can be classified into P-model, Q-model and S-model. The environments in which the reinforcement signal can only take two binary values 0 and 1 are referred to as P-model environments. Another class of the environment allows a finite number of the values in the interval [0, 1] can be taken by the reinforcement signal. Such an environment is referred to as Q-model environment. In S-model environments, the reinforcement signal lies in the interval $[a,b]$. The relationship between the learning automaton and its random environment has been shown in figure 1.

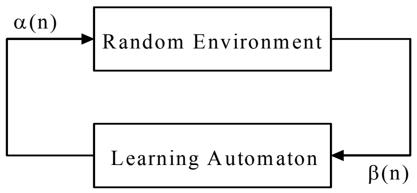

Figure 1. The relationship between the learning automaton and its random environment

Learning automata can be classified into two main families [29]: fixed structure learning automata and variable structure learning automata. Variable structure learning automata are represented by a triple $<\underline{\beta},\underline{\alpha},T>$, where $\underline{\beta}$ is the set of inputs, $\underline{\alpha}$ is the set of actions, and T is learning algorithm. The learning algorithm is a recurrence relation which is used to modify the action probability vector. Let $\alpha(k)$ and $\underline{p}(k)$ denote the action chosen at instant k and the action probability vector on which the chosen action is based, respectively. The recurrence equation shown by (1) and (2) is a linear learning algorithm by which the action probability vector $\underline{p}$ is updated. Let $\alpha_i(k)$ be the action chosen by the automaton at instant k.

$$p_j(n+1)=\begin{cases} p_j(n)+a[1-p_j(n)] & j=i \\ (1-a)p_j(n) & \forall j \ \ j\neq i \end{cases} \tag{1}$$

When the taken action is rewarded by the environment (i.e., $\beta(n)=0$) and

$$p_j(n+1)=\begin{cases} (1-b)p_j(n) & j=i \\ (b/r-1)+(1-b)p_j(n) & \forall j \ \ j\neq i \end{cases} \tag{2}$$

When the taken action is penalized by the environment (i.e., $\beta(n)=1$). r is the number of actions can be chosen by the automaton, $a(k)$ and $b(k)$ denote the reward and penalty parameters and determine the amount of increases and decreases of the action probabilities, respectively. If $a(k)=b(k)$, the recurrence equations (1) and (2) are called linear reward-penalty (L_{R-P}) algorithm, if $a(k)>>b(k)$ the given equations are called linear reward-ε penalty ($L_{R-\varepsilon P}$), and finally if $b(k)=0$ they are called linear reward-Inaction (L_{R-I}). In the latter case, the action probability vectors remain unchanged when the taken action is penalized by the environment.

2.2 Variable Action-set Learning Automata

A variable action-set learning automaton is an automaton in which the number of actions available at each instant changes with time. It has been shown in [30] that a learning automaton with a changing number of actions is absolutely expedient and also ε-optimal, when the reinforcement scheme is L_{R-I}. Such an automaton has a finite set of n actions, $\alpha=\{\alpha_1,\alpha_2,...,\alpha_n\}$. $A=\{A_1,A_2,...,A_m\}$ denotes the set of action subsets and $A(k)\subseteq\alpha$ is the subset of all the actions can be chosen by the learning automaton, at each instant k. The selection of the particular action subsets is randomly made by an external agency according to the probability distribution $q(k)=\{q_1(k),q_2(k),...,q_m(k)\}$ defined over the possible subsets of the actions, where $q_i(k)=prob[A(k)=A_i \mid A_i\in A,\ 1\leq i\leq 2^n-1]$. $\hat{p}_i(k)=prob[\alpha(k)=\alpha_i \mid A(k),\alpha_i\in A(k)]$ is the probability of choosing action α_i, conditioned on the event that the action subset $A(k)$ has already been selected and also $\alpha_i\in A(k)$. The scaled probability $\hat{p}_i(k)$ is defined as

$$\hat{p}_i(k) = p_i(k) / K(k) \tag{3}$$

where $K(k) = \sum_{\alpha_i \in A(k)} p_i(k)$ is the sum of the probabilities of the actions in subset $A(k)$,and $p_i(k) = prob[\alpha(k) = \alpha_i]$.

The procedure of choosing an action and updating the action probabilities in a variable action-set learning automaton can be described as follows. Let $A(k)$ be the action subset selected at instant k. Before choosing an action, the probabilities of all the actions in the selected subset are scaled as defined in equation (3). The automaton then randomly selects one of its possible actions according to the scaled action probability vector $\hat{p}(k)$. Depending on the response received from the environment, the learning automaton updates its scaled action probability vector. Note that the probability of the available actions is only updated. Finally, the probability vector of the actions of the chosen subset is rescaled as

$$p_i(k+1) = \hat{p}_i(k+1) \cdot K(k) \tag{4}$$

for all $\alpha_i \in A(k)$ .The absolute expediency and $\varepsilon-$optimality of the method described above have been proved in [30].

3 The Proposed Vertex Coloring Algorithm

In this section, we propose a learning automat-based approximation algorithm called LAVCA for solving the minimum vertex coloring problem as defined in Section 1. the proposed algorithm is an iterative algorithm in which each iteration is divided into several stages. At each stage, exploiting the variable action-set learning automata, a group (subset) of the non adjacent vertices of the graph is selected and colored (with the same color). The remaining uncolored vertices are selected in the next stages. Each iteration stops if all the vertices are colored. The required number of colors will be minimized if the size of the subsets of the non adjacent vertices (which are colored at the same stage) are maximized. In the proposed algorithm, this can be done by the learning automata.

In the proposed algorithm, at first, each vertex of the graph is assigned a learning automaton, and so a network of learning automata isomorphic to the graph is initially constructed. This network of learning automata can be described by a duple $<\underline{A},\underline{\alpha}>$, where $\underline{A} = \{A_1, A_2, ..., A_n\}$ denotes the set of learning automata corresponding to the vertex-set, and $\underline{\alpha} = \{\underline{\alpha_1}, \underline{\alpha_2}, ..., \underline{\alpha_n}\}$ denotes the action-set in which $\underline{\alpha_i} = \{\alpha_i^1, \alpha_i^1, ..., \alpha_i^{ri}\}$ (for each $\underline{\alpha_i} \in \underline{\alpha}$)defines the set of actions can be taken by learning automaton A_i . It should be noted that a learning automaton is associated with a vertex (one-to-one), and so hereafter vertex v_i may be referred to as learning automaton A_i and vice versa. In this algorithm, the action-set of each learning automaton (say A_i) includes the vertices which can be colored with the color of vertex v_i . In other words, vertex v_i and the vertex corresponding to each action of automaton A_i can be assigned the same color. For example, as shown in Figure 2, vertices 4, 6, 7, 8, 9, 10, and 11 are the vertices to which the color of vertex 1 can be assigned, and so they form the action-set of automaton A_1 (or vertex v_1). To form such an action-set, each automaton A_i reserves an action for each of the vertices of the graph which are not adjacent to vertex v_i .

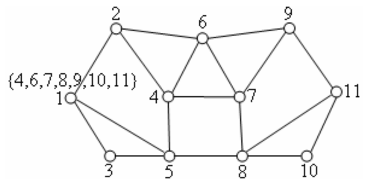

Figure 2. A sample graph

As described earlier, the proposed vertex coloring algorithm consists of a number of iterations, and at each iteration, the graph is thoroughly colored. Each iteration also contains a number of stages, and at each stage, a subset of vertices is assigned the same color. In what follows, we describe an iteration (e.g., iteration k) of the proposed algorithm shown in Figure 3 in detail.

As shown in Figure 3, graph $G(\mathrm{V},\mathrm{E})$ and threshold p are the input parameters, and chromatic number of the input graph is the output parameter. After the action-sets are formed (Lines 4 and 5), the first iteration begins. Let k denotes the number of iterations, Γ denotes the set of uncolored vertices, Z denotes the set of actions which must be removed from the action-set of the next automata, and χ denotes the color number which is assigned to the vertices at each stage. At the end of each iteration, χ also denotes the number of colors required for coloring the graph.

At the beginning of k th iteration, vertex $v_i \in \Gamma$ (such that $i = \min j$, for all $v_j \in \Gamma$) is selected, activated and assigned color χ . The colored vertex is removed from the set of uncolored vertices (Line 14). It is also must be removed from the action-set of all the automata which will be selected in the current iteration, if any (Line 16). From Line 15, it is concluded that the neighbors of the vertices which are colored in a stage must be also removed from the action-set of the automata which will be selected in that stage. These actions are removed from the action-set of the (currently) active learning automaton as described in section 2.2 on the variable action-set learning automata. This avoids assigning the same color to the neighboring nodes. Then, If the action-set of automaton A_i is a null set, the current stage in which the vertices are colored with color χ is terminated, and algorithm initiates a new stage to color the remaining vertices with another color (i.e., color $\chi+1$). Otherwise (i.e., $\underline{\alpha_i} \neq \varnothing$), learning automaton A_i (corresponding to the colored vertex v_i) randomly chooses one of its actions. The vertex corresponding to the selected action is activated, colored an denoted as v_j . Then, vertex v_j repeats the same operations as vertex v_i did (Lines 21-24). This process is repeated until no uncolored vertices remain (i.e., $\Gamma = \varnothing$). This process guarantees that the graph can be legally colored at each iteration. The set of colors with which the graph is colored is referred to as color-set. After all the vertices are colored, the number of used colors is compared with dynamic threshold T_k . Dynamic threshold T_k represents the cardinality of the smallest color-set which have been used to color the graph until iteration k . If the cardinality of the selected color-set (i.e., χ) is less than the dynamic threshold T_k , LAVCA rewards the actions chosen by all the activated learning automata, and sets dynamic threshold T_k to χ , otherwise it penalizes the selected actions. In this algorithm, learning scheme L_{R-P} is used to update the action probability vectors. Here, the k th iteration of LAVCA is completed, and the stopping condition of algorithm should be verified. The proposed algorithm stops if all the activated learning automata choose their action with a probability higher than pre-specified threshold P . This condition represents the stopping condition of algorithm. The proposed algorithm stops and returns the chromatic number χ if the stopping condition is met, and initiates a new iteration otherwise. The pseudo code of the proposed vertex coloring algorithm is shown in Figure 3.

```
Algorithm LAVCA
1: Input: Graph $G(\mathrm{V},\mathrm{E})$ , Thresholds $P$
2: Output: Chromatic number $\chi$
3: Assumptions:
4: Assign a learning automaton $A_i$ to each vertex $v_i$
5: Let $\underline{\alpha_i}$ denotes the action-set of automaton $A_i$
```

```
6: Begin Algorithm
7:  $k \leftarrow 0$ , $T_k \leftarrow |V|$
8:  Repeat
9:      $\Gamma \leftarrow V$ , $\chi \leftarrow 1$
10:     Repeat
11:         $Z \leftarrow V - \Gamma$
12:         Vertex $v_i \in \Gamma$ with the lowest ID is selected
13:         $v_i$ is assigned color $\chi$
14:         $\Gamma \leftarrow \Gamma - \{v_i\}$
15:         $Z \leftarrow Z + \{v_i\}$ +Neighbors of $\{v_i\}$
16:         $\underline{\alpha_i} \leftarrow \underline{\alpha_i} - Z$
17:         While $\underline{\alpha_i} \neq \varnothing$ Do
18:             Automaton $A_i$ (corresponding to $v_i$) randomly chooses one of its actions
19:             The vertex corresponding to the chosen action is denoted as $v_j$
20:             $v_j$ is assigned color $\chi$
21:             $\Gamma \leftarrow \Gamma - \{v_j\}$
22:             $Z \leftarrow Z + \{v_j\}$ +Neighbors of $\{v_j\}$
23:             $v_i \leftarrow v_j$
24:             $\underline{\alpha_i} \leftarrow \underline{\alpha_i} - Z$
25:         End While
26:         $\chi \leftarrow \chi + 1$
27:     Until $\Gamma = \varnothing$
28:     If $\chi \leq T_k$ Then
29:         $T_k \leftarrow \chi$
30:         All learning automata reward their chosen actions
31:     Else
32:         All learning automata penalize their chosen actions
33:     End If
34:     $k \leftarrow k + 1$
35: Until Stopping condition = TRUE
36: Return $\chi$
37: End Algorithm
```

Figure 3. The pseudo code of the proposed algorithm

As given in Lines 28-33, at each iteration, the actions which form a smaller color-set are rewarded. Therefore, in the forthcoming iterations, these actions are selected with a higher probability as the choice probability of the other actions decrease. As the algorithm proceeds, the number of stages (i.e., the number of colors required for coloring the graph) decreases. This is due to the fact that the number of vertices which are colored at each stage increases as the algorithm approaches to the end. Indeed, after a number of iterations, the network of learning automata converge to an optimal action selection strategy in which at each stage the maximum subset of the non adjacent vertices can be colored.

3.1 An Example

In this section, we illustrate the proposed vertex coloring algorithm step by step for a sample graph. Figures 4(a)-4(l) show a possible execution scenario of the proposed algorithm for coloring the sample graph shown in Figure 2. This graph comprises 11 vertices and 18 edges. In these figures, a white circle represents an uncolored vertex, and a colored circle represents a vertex which has been assigned a color. The number beside a vertex represents the vertex ID number. The set beside a vertex represents the action-set of the learning automaton corresponding to the vertex. For each vertex, the set of vertices which are not adjacent to the vertex form its action-set. For instance in Figure 4(a), vertices 4, 6, 7, 8, 9, 10 , and 11 belong to the action-set of vertex 1. A crossed action (or vertex) represents a disabled action which can not be temporarily chosen by the learning automaton (e.g., actions 1 and 3 in Figure 4(b)). In these figures, we assume that color 1 is blue, color 2 is red and color 3 is green.

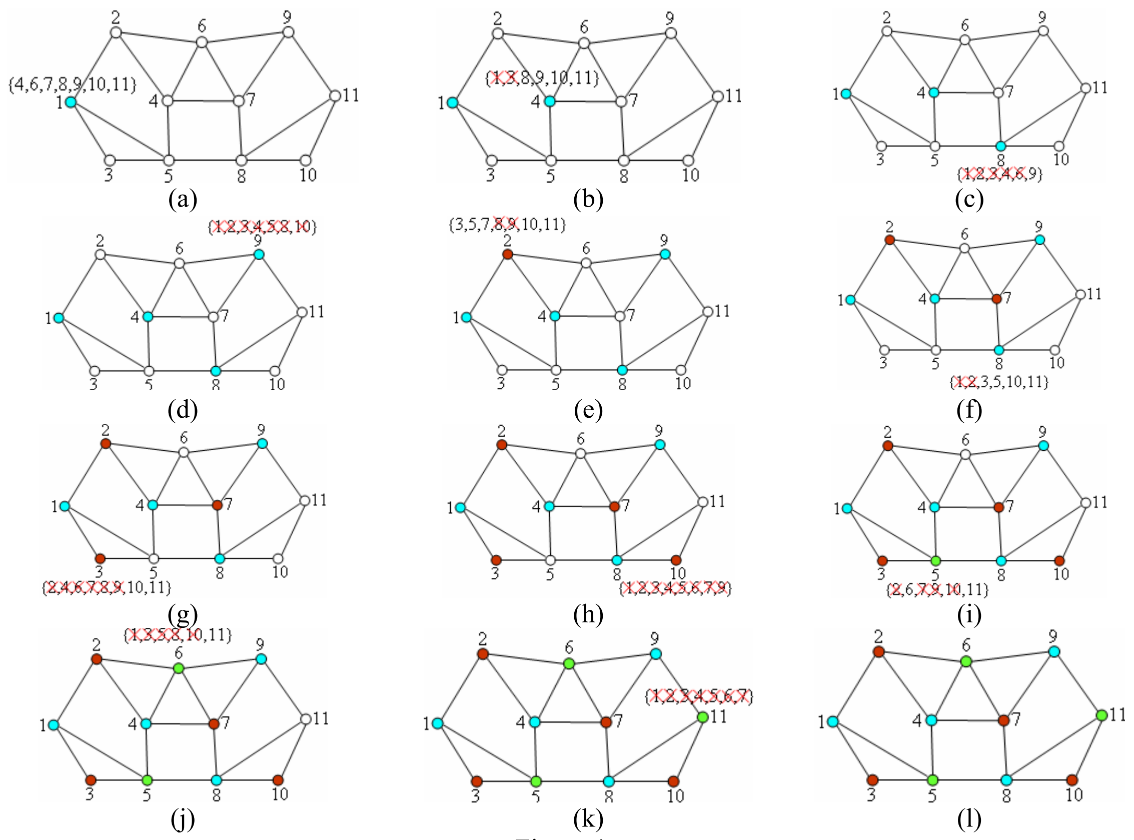

Figure 4.

As shown in Figure 4(a), vertex 1 is initially selected and colored with color 1 (blue) since it is the vertex with lowest ID in $\Gamma = V$. Vertex 1 is removed from the set of uncolored vertices Γ. This vertex and its neighboring vertices are added to Z. The set of actions which are in Z are disabled in the action-set of vertex 1. Vertex 1 randomly chooses one of its available actions (i.e., {4, 6, 7, 8, 9, 10, 11}). Let us assume that vertex 4 is chosen by vertex 1. Vertex 4 is also colored blue. It is removed from the set of uncolored vertices Γ, and added to Z. The neighbors of vertex 4 (i.e., vertices 2, 5, 6 and 7) are also added to Z. Now the action-set of vertex 1 is updated by removing the members of Z. As shown in Figure 4(b) vertices 1 and 3 are in Z, and must be removed from the action-set of vertices 4. The crossed vertices 1 and 3 represents the disabled actions. Now, vertex 4 chooses one of its remaining actions (i.e., {8, 9, 10, 11}). We assume that vertex 4 chooses vertex 8. Vertex 8 is colored blue, and the sets Γ and Z are updated as described earlier (Lines 21-24). It can be seen that vertices 1, 2, 3, 4, and 6 should be removed from the action-set of vertex 8. From Figure 4(c), it is clear that the updated action-set of vertex 8 only includes vertex 9. Therefore, vertex 9 is chosen by vertex 8 with probability one. Vertex 9 is colored blue too. After updating Γ, Z, and the action-set of vertex 9, we find that no more actions can be taken by vertex 9 and so the first (current) stage stops (see Figure 4(d)). This means that no more vertices can be colored blue. Since there are some uncolored vertices, a new stage is initiated. A new color (color 2) is selected for coloring the vertices which are chosen in the second stage (Line 26). Vertex 2 which has the lowest ID number is initially selected and colored red (i.e., color 2). After updating Γ, Z, and the action-set of vertex 2, we assume that it selects vertex 7 among the possible choices (i.e., {3, 5, 7, 10, 11}) (see Figure 4(e)). Let us assume that vertex 7 chooses vertex 3, and vertex 3 chooses vertex 10 (Figures 4(f)-4(h)). As shown in Figure 4(h), vertex 10 has no more possible actions, and so no more vertices can be colored red. The third stage starts by choosing and coloring vertex 5 with color 3 (i.e., green). As sown in Figures 4(i)-4(k), we assume that vertex 5 chooses vertex 6, and vertex 6 chooses vertex 11. Here, the third stage is completed since vertex 11 has no more available actions (see Figure 4(k)). Figure 4(k) shows that each vertex has been assigned a (legal) color (i.e., the set of uncolored vertices $\Gamma = \varnothing$), and so the current iteration terminates. It can be seen that three colors are used to color the graph in this iteration. Indeed, the number of stages is equal to the number of used colors. Flowchart of the proposed algorithm has been shown in Figure 5.

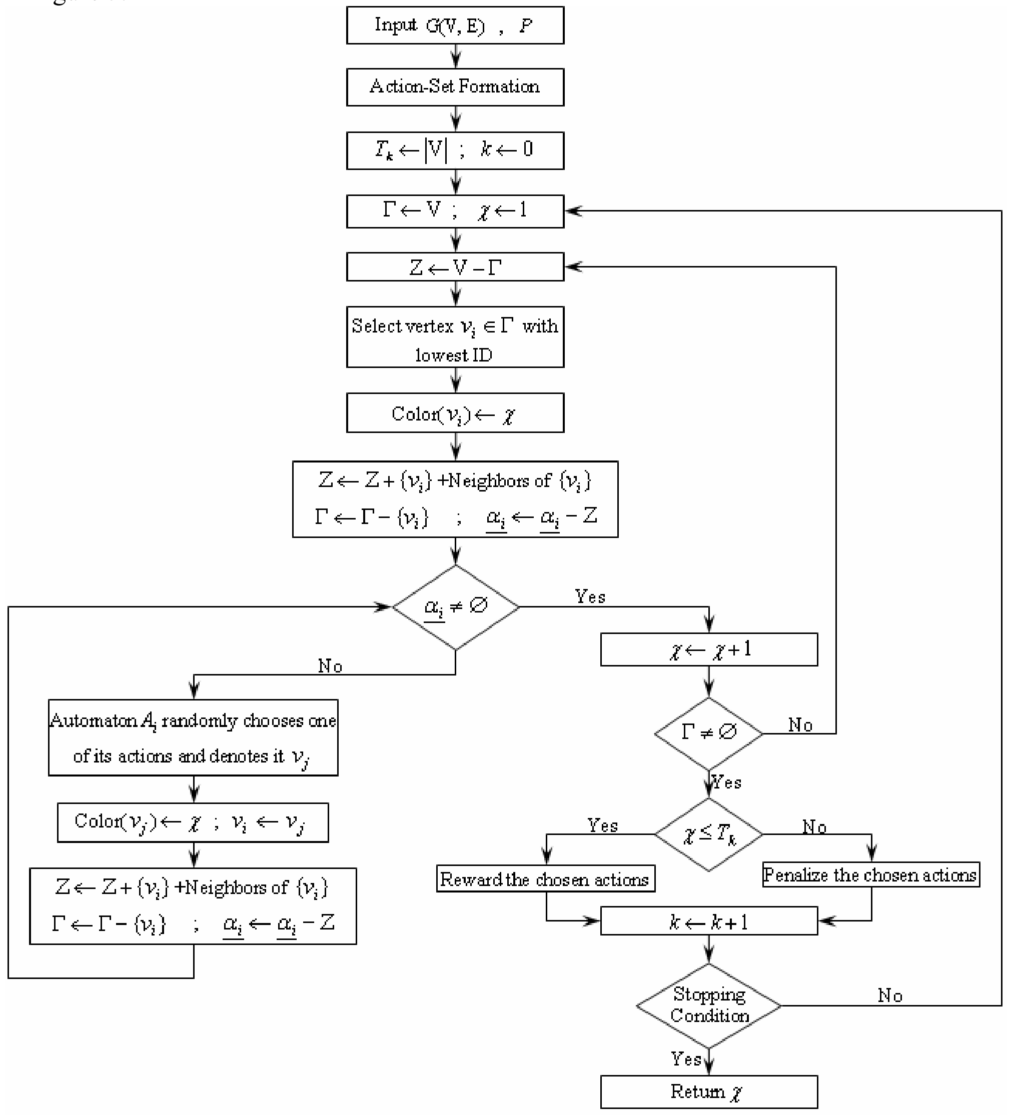
Input G(V, E) , P
Action-Set Formation
$T_k \leftarrow |V|$; $k \leftarrow 0$
$\Gamma \leftarrow V$; $\chi \leftarrow 1$
$Z \leftarrow V - \Gamma$
Select vertex $v_i \in \Gamma$ with lowest ID
Color(v_i) $\leftarrow \chi$
$Z \leftarrow Z + \{v_i\}$ +Neighbors of $\{v_i\}$
$\Gamma \leftarrow \Gamma - \{v_i\}$; $\underline{\alpha}_i \leftarrow \underline{\alpha}_i - Z$
$\underline{\alpha}_i \neq \varnothing$
Yes
No
Automaton A_i randomly chooses one of its actions and denotes it v_j
Color(v_j) $\leftarrow \chi$; $v_i \leftarrow v_j$
$Z \leftarrow Z + \{v_i\}$ +Neighbors of $\{v_i\}$
$\Gamma \leftarrow \Gamma - \{v_i\}$; $\underline{\alpha}_i \leftarrow \underline{\alpha}_i - Z$
$\chi \leftarrow \chi + 1$
$\Gamma \neq \varnothing$
No
Yes
$\chi \leq T_k$
Yes
No
Reward the chosen actions
Penalize the chosen actions
$k \leftarrow k+1$
Stopping Condition
No
Yes
Return χ

Figure 5. Flow chart of the proposed algorithm

4 Numerical Results

To study the efficiency of the proposed algorithm, we have conducted several simulation experiments. In these experiments, the proposed vertex coloring algorithm is tested on a subset of hard-to-color benchmarks like DSJ[31], Leighton[32], and Wap[28]. The performance of the proposed algorithm is measured both in terms of the time and the number of colors required for coloring the benchmarks, and compared with those of ILS[23], GLS[24], CHECKCOL[26], AMACOL[27] and TPA[28]. In all experiments presented in this paper, the reinforcement scheme by which the action probability vectors are updated is L_{R-P}, and the reward and penalty parameters (i.e., a and b) are 0.1. In these experiments, threshold P is set to 0.95, and so the proposed algorithm stops if all the activated learning automata choose their action with a probability higher than 0.95. The results are summarized in Tables 2, 4, and 6. In these tables, the first column includes the number of colors required for coloring the graph (CN), and the second column includes the running time of each algorithm in seconds (RT).

DSJ [31] benchmark graphs are the first class of graphs on which the proposed algorithm is tested. This class comprises a set of uniform (n, p) random graphs which are denoted as DSJC $n.p$, where $n = \{125, 250, 500, 1000\}$ is the number of vertices, and $p = \{0.1, 0.5, 0.9\}$ denotes the probability of connecting every pair of nodes in the graph. DSJR graphs are geometric graphs were introduced by Johnson et al.[31] to provide benchmarks for heuristics. The number of nodes in these graphs changes from 125 to 1000, and so, it is not easy to find good solutions, especially with a density of 0.5 and 0.9, and with a number of nodes greater than 125. The characteristics (i.e., density and number of vertices and edges) of DSJ benchmark graphs are given in Table 1.

Table 1. The characteristics of DSJ benchmark graphs

Graph Name	*Class*	*Number of Vertices*	*Number of edges*	*Density*
DSJC125.1	DSJ	125	736	0.09
DSJC125.5	DSJ	125	3891	0.50
DSJC125.9	DSJ	125	6961	0.90
DSJC250.1	DSJ	250	3218	0.10
DSJC250.5	DSJ	250	15668	0.50
DSJC250.9	DSJ	250	27897	0.90
DSJC500.1	DSJ	500	12458	0.10
DSJC500.5	DSJ	500	62624	0.50
DSJC500.9	DSJ	500	112437	0.90
DSJR500.1	DSJ	500	3555	0.03
DSJR500.1c	DSJ	500	121275	0.97
DSJR500.5	DSJ	500	58862	0.47
DSJC1000.1	DSJ	1000	49629	0.10
DSJC1000.5	DSJ	1000	249826	0.50
DSJC1000.9	DSJ	1000	449449	0.90

Comparing the results of the other algorithms, we observe that, in most cases, GLS has the shortest running time, and CHECKCOL and AMACOL have the worst running time. It also can be seen that, in almost all cases, AMACOL picks the smallest number of colors to color the graphs, and GLS uses the most number of colors. Comparing the results of the LAVCA with AMACOL, we find that the color-sets chosen by the proposed algorithm are as small as those of AMACOL, while the running time of the proposed algorithm is considerably shorter than AMACOL. On the other hand, comparing the proposed algorithm with GLS, it can be seen that the running time of the proposed algorithm is as close to GLS as possible, while the size of the color-set, in the proposed algorithm, is significantly smaller as compared with GLS.

Table 2. A performance comparison of the proposed algorithm and the most effective coloring algorithms (Tested on hard-to-color DSJ benchmark graphs)

Graph	*Best*	*TPA[28]*		*AMACOL[27]*		*ILS[23]*		*CHECKCOL[26]*		*GLS[24]*		*LAVCA*	
		CN	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*
DSJC125.1	*5*	5	0	5	0	5	0	5	0	5	0	5	0.0090
DSJC125.5	*17*	19	289	17	125	17	2	17	110	18	0	17	17.040
DSJC125.9	*44*	44	5	44	57	44	0	44	4	44	0	44	41.170
DSJC250.1	*8*	8	10	8	12	8	0	8	28	9	0	8	16.015
DSJC250.5	*28*	30	3282	28	64	28	34	28	557	30	1	28	42.010
DSJC250.9	*72*	72	155	72	2604	72	6	72	182	73	6	72	67.300
DSJC500.1	*12*	12	0	12	9	13	0	12	4	13	0	12	32.200
DSJC500.5	*48*	48	124	48	326	50	106	48	1789	52	81	48	80.230
DSJC500.9	*126*	127	1268	126	1710	128	82	126	2045	130	154	127	114.50
DSJC1000.1	*20*	21	28	20	969	21	6	21	142	22	1	20	43.107
DSJC1000.5	*84*	84	2386	84	9235	91	303	84	7025	93	546	84	205.10
DSJC1000.9	*224*	226	3422	224	4937	228	2245	226	12545	234	1621	224	423.00

The second class of the benchmark graphs on which the proposed algorithm is tested is Leighton [32] which is a set of large random graphs, with 450 nodes, and denoted as Le450_ xy , where $x = \{5, 15, 25\}$ is the chromatic number of the graph and equal to the maximum clique size, and extension $y = \{a, b, c, d\}$ denotes the edge density of the graph. Finding a solution to the graphs with extensions c and d is, in general, more difficult than that with extensions a and b . The characteristics (i.e., density and number of vertices and edges) of Leighton benchmark graphs are given in Table 3.

Table 3. The characteristics of Leighton benchmark graphs

Graph Name	*Class*	*Number of Vertices*	*Number of edges*	*Density*
Le450_5a	LEI	450	5714	0.06
Le450_5b	LEI	450	5734	0.06
Le450_5c	LEI	450	9803	0.10
Le450_5d	LEI	450	9757	0.10
Le450_15a	LEI	450	8168	0.08
Le450_15b	LEI	450	8169	0.08
Le450_15c	LEI	450	16680	0.17
Le450_15d	LEI	450	16750	0.17
Le450_25a	LEI	450	8260	0.08
Le450_25b	LEI	450	8263	0.08
Le450_25c	LEI	450	17343	0.17
Le450_25d	LEI	450	17425	0.17

Table 4. Performance of the proposed algorithm and comparison with the most effective coloring algorithms (Tested on hard-to-color Leighton benchmarks)

Graph	*Best*	*TPA[28]*		*AMACOL[27]*		*ILS[23]*		*CHECKCOL[26]*		*GLS[24]*		*LAVCA*	
		CN	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*
Le450_15a	15	15	1444	15	345	15	0	15	2145	15	2	15	31.290
Le450_15b	15	15	1655	15	345	15	0	15	2756	15	0	15	40.140
Le450_15c	15	15	82	15	2	15	19	15	4534	15	6	15	77.120
Le450_15d	15	15	34	15	4	15	20	15	4576	15	8	15	59.150
Le450_25c	25	26	44	26	93	26	2	25	3477	26	18	25	74.200
Le450_25d	25	26	22	26	10	26	1	25	4524	26	2	25	80.103

Comparing the results reported in Table 4, we find that, in almost all cases, ILS outperforms the others in terms of running time. It can be seen that CHECKCOL always selects the smallest color-sets for coloring the graphs, while its running time is the worst. Comparing the results of the proposed algorithm with the chromatic number of the benchmarks, we observe that the size of the color-sets constructed by the proposed algorithm is equal to the chromatic number. It can be seen that, CHECKCOL is also capable of finding the optimal solution (Best results). However, comparing the running time of our proposed algorithm with that of CHECKCOL, we find that the proposed algorithm significantly outperforms CHECKCOL in terms of the running time.

The third class of the benchmarks we consider includes Wap[28] graphs which arise in the design of transparent optical networks and are denoted by Wap0ma, where $m = \{1,\dots,8\}$. In this class, the graphs have a large number of vertices between 905 and 5231, and all instances have a clique of size 40. The characteristics (i.e., density and the number of vertices and edges) of Wap benchmark graphs are given in Table 5.

Table 5. The characteristics of Wap benchmark graphs

Graph Name	*Class*	*Number of Vertices*	*Number of edges*	*Density*
Wap01a	KOS	2368	110871	0.04
Wap02a	KOS	2464	111742	0.04
Wap03a	KOS	4730	286722	0.03
Wap04a	KOS	5231	294902	0.02
Wap05a	KOS	905	43081	0.11
Wap06a	KOS	947	43571	0.10
Wap07a	KOS	1809	103368	0.06
Wap08a	KOS	1870	104176	0.06

Table 6. A performance comparison of the proposed algorithm and the most effective coloring algorithms (Tested on hard-to-color Wap benchmarks)

Graph	*Best*	*TPA[28]*		*AMACOL[27]*		*ILS[23]*		*CHECKCOL[26]*		*GLS[24]*		*LAVCA*	
		CN	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*	*CN*	*RT*
Wap01a	42	42	245	45	345	44	2	44	568	42	55	42	16.186
Wap02a	41	41	1618	44	802	43	251	43	486	41	160	41	32.057
Wap03a	44	44	17	53	245	46	365	46	689	44	782	44	90.321
Wap04a	43	43	95	48	45	44	484	44	23	43	834	43	68.250
Wap06a	41	41	348	44	545	42	1	42	25	41	8	41	3.2300
Wap07a	42	42	541	45	89	44	1	44	182	42	215	42	20.195
Wap08a	42	42	200	45	446	43	56	44	22	42	41	42	28.302

Comparing the results reported in Table 6, we observe that, GLS and TPA outperform the others in terms of the size of the color-set, and ILS in terms of the time required for coloring the Wap benchmark graphs. The obtained results show that the size of the color-sets constructed by the proposed algorithm is always equal to the best reported results .On the other hand, the running time of the proposed algorithm is considerably shorter as compared with GLS and TPA.

5 Conclusion

Vertex coloring problem is a combinatorial optimization problem in which a color is assigned to each vertex of the graph so that no two adjacent vertices have the same color. The minimum vertex coloring is an NP-hard problem which a host of algorithms have been proposed to solve it. In this paper, an approximation algorithm was proposed for solving the minimum vertex coloring problem based on learning automata. The proposed algorithm iteratively finds different possible colorings of the graph. Each iteration of the proposed algorithm is subdivided into several stages. The number of stages per iteration represents the required number of colors. At each stage of algorithm, a subset of the non adjacent vertices of the graph is randomly selected and assigned the same color. This process continues until no vertices remain uncolored. The proposed algorithm guarantees that the graph can be legally colored at each iteration. As the proposed algorithm approaches to the end, each vertex learns how to select one of its non adjacent vertices so that the number of selected vertices per stage is maximized. Therefore, as the proposed algorithm proceeds, the number of stages and so the required number of colors tends to the chromatic number of the graph. To show the performance of the proposed algorithm we compared it with several vertex coloring algorithms in terms of the time and the number of colors required for coloring the graphs. The obtained results show the superiority of the proposed algorithm over the others.

References

[1]. T. R. Jensen, and B. Toft, "Graph coloring problems," *John Wiley & Sons*, USA, 1994.

[2]. G. J. Chaitin, M.A. Auslander, A.K. Chandra, J. Cocke, M.E. Hopkins, and P.W. Markstein, "Register allocation via coloring," *Computer Languages*, Vol. 6, 1981, pp. 47-57.

[3]. N. Barnier, and P. Brisset, "Graph coloring for air traffic flow management," *In Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques*, Le Croisic, France, 2002, pp. 133–147.

[4]. D. de Werra, "An introduction to timetabling," *European Journal of Operational Research*, Vol. 19, 1985, pp. 151–162.

[5]. F. T. Leighton, "A graph coloring algorithm for large scheduling problems," *Journal of Research of the National Bureau of Standards*, Vol. 84, No. 6, 1979, pp. 489–506.

[6]. A. Gamst, "Some lower bounds for a class of frequency assignment problems," *IEEE Transactions of Vehicular Technology*, Vol. 35, No. 1, 1986, pp. 8–14.

[7]. A. Zymolka, A. M. C. A. Koster, and R. Wessaly, "Transparent optical network design with sparse wavelength conversion," *In Proceedings of the 7th IFIP Working Conference on Optical Network Design a nd Modelling*, Budapest, Hungary, 2003, pp. 61–80.

[8]. R. M. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, Plenum Press, USA, 1972, pp. 85–103.

[9]. M.R. Garey, and D.S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *W.H. Freeman and Co.,* San Francisco, CA, 1979.

[10]. D. Brelaz, "New methods to color the vertices of a graph," *Communications of the ACM*, Vol. 22, No. 4, 1979, pp. 252–256.

[11]. J. R. Brown, "Chromatic scheduling and the chromatic number problem," *Management Science*, Vol. 19, No. 4, 1972, pp. 456–463.

[12]. J. Bttcher, and D. Vilenchik, "On the tractability of coloring semi-random graphs," *Information Processing Letters*, Vol. 108, 2008, pp. 143–149.

[13]. P. Galinier, and A. Hertz, "A survey of local search methods for graph coloring," *Computers & Operations Research*, Vol. 33, 2006, pp. 2547–2562.

[14]. B. B. Mabrouk, H. Hasni, and Z. Mahjoub, "On a parallel genetic–tabu search based algorithm for solving the graph coloring problem," European Journal of Operational Research, 2008, doi:10.1016/j.ejor.2008.03.050

[15]. H. Asmuni, E. K. Burke, J. M. Garibaldi, B. McCollum, and A. J. Parkes, "An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables," *Computers & Operations Research*, Vol. 36, 2009, pp. 981-1001.

[16]. A. E. Eiben, J.K. Vanderhauw, and J.I. Vanhemert, "Graph coloring with adaptive evolutionary algorithms," *Journal of Heuristics*, Vol. 4, 1998, pp. 25–46.

[17]. J. Thompson, K. Dowsland, "A robust simulated annealing based examination timetabling system," *Computers & Operations Research*, Vol. 25, 1998, pp. 637–48.

[18]. K. A. Dowsland, and J. M. Thompson, "An improved ant colony optimization heuristic for graph coloring," *Discrete Applied Mathematics,* Vol. 156, 2008, pp. 313 – 324.

[19]. C. Cooper, M. Dyer, and A. Frieze , "On Markov chains for randomly H-coloring a graph," *Journal of Algorithms*, Vol. 39, 2001, pp. 117-134.

[20]. P. M. Talavan, and J. Yanez, "The graph coloring problem: A neuronal network approach," *European Journal of Operational Research*, Vol. 191, 2008, pp. 100-111.

[21]. A. Hertz, and D. Werra, "Using tabu search techniques for graph coloring," *Computing*, Vol. 39, 1987, pp. 345–351.

[22]. M. Caramia, and P. Dell'Olmo, "A fast and simple local search for graph coloring," *Lecture Notes in Computer Science, Algorithm Engineering*, Vol. 1618, 1999, pp. 319–333.

[23]. H. R. Lourenco, and O. Martin, T. Stutzle, "Iterated local search," *In Handbook of Metaheuristics, F. Glover, G. Kochenberger (Eds.), Kluwer Academic Publishers*, USA, 2002, pp. 321–353.

[24]. C. Voudouris, and E. P. K. Tsang, "Guided local search," In Handbook of Metaheuristics, F. Glover (Eds.), Kluwer Academic Publishers, USA, 2003, pp. 185-218.

[25]. M. Chiarandin, and T. Stutzle, "Stochastic local search algorithms for graph set T-coloring and frequency assignment," Constraints, Vol. 12, No. 3, 2007, pp. 371-403.

[26]. M. Caramia, P. Dellolmo, and G. F. Italiano, "CHECKCOL: Improved local search for graph coloring," *Journal of Discrete Algorithms*, Vol. 4, 2006, pp. 277-298.

[27]. P. Galinier, A. Hertz, and N. Zufferey, "An Adaptive Memory Algorithm for the K-coloring Problem," *Discrete Applied Mathematics*, Vol. 156, 2008, pp. 267–279.

[28]. M. Caramia, and P. Dell'Olmo, "Embedding a novel objective function in a two-phased local search for robust vertex coloring," *European Journal of Operational Research*, Vol. 189, 2008, pp. 1358–1380.

[29]. K. S. Narendra and K. S. Thathachar, "Learning automata: An introduction," New York, *Printice-Hall*, 1989.

[30]. M. A. L. Thathachar and B. R. Harita, "Learning automata with changing number of actions," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMG17, 1987, pp. 1095-1100.

[31]. D.R. Johnson, C.R. Aragon, L.A. McGeoch, C. Schevon, “Optimization by simulated annealing: an experimental evaluation,” part II, graph coloring and number partitioning, *Operations Research*, Vol. 39, 1991, pp. 378–406.

[32]. F. T. Leighton, “A graph coloring algorithm for large scheduling problems,” *Journal of Research of the National Bureau of Standards*, Vol. 84, 1979, pp. 489–505.

Javad Akbari Torkestani received the B.S. and M.S. degrees in Computer Engineering from Arak University in Iran, in 2001 and 2004, respectively. He is currently pursuing the Ph.D. degree in Computer Engineering at Science and Research University, Tehran, Iran. He joined the faculty in Computer Engineering Department at Arak Azad University, Arak, Iran, in 2004. His research interests include wireless networks, mobile ad hoc networks, fault tolerant systems, learning systems, parallel algorithms, and soft computing.

Mohammad Reza Meybodi received the B.S. and M.S. degrees in Economics from Shahid Beheshti University in Iran, in 1973 and 1977, respectively. He also received the M.S. and Ph.D. degree from Oklahoma University, USA, in 1980 and 1983, respectively in Computer Science. Currently, he is a full professor in Computer Engineering Department, Amirkabir University of Technology, Tehran, Iran. Prior to current position, he worked from 1983 to 1985 as an assistant professor at Western Michigan University, and from 1985 to 1991 as an associate professor at Ohio University, USA. His research interests include wireless networks, fault tolerant systems, learning systems, parallel algorithms, soft computing and software development.