Graph Coloring Problem Based on Learning Automata

J. Akbari Torkestani
Computer Engineering Department
Islamic Azad University- Arak Branch
Arak, Iran
j-akbari@iau-arak.ac.ir

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

***Abstract*— The vertex coloring problem is a well-known classical problem in graph theory in which a color is assigned to each vertex of the graph such that no two adjacent vertices have the same color. The minimum vertex coloring problem is known to be an NP-hard problem in an arbitrary graph, and a host of approximation solutions are available. In this paper, four learning automata-based approximation algorithms are proposed for solving the minimum (vertex) coloring problem. It is shown that by a proper choice of the parameters of the algorithm, the probability of approximating the optimal solution is as close to unity as possible. The last proposed algorithm is compared with some well-known coloring algorithms and the results show the efficiency of the proposed algorithm in terms of the color set size and running time of algorithm.**

Keywords-component; Graph coloring problem, vertex coloring problem, combinatorial optimization problem, learning automata

I. INTRODUCTION

The vertex coloring problem is a well-known combinatorial optimization problem in graph theory, which is widely used in real life applications like computer register allocation, air traffic flow management, timetabling, scheduling, frequency assignment, and light wavelengths assignment in optical networks. 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 $f:V\to C$ 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. It is shown in [1] 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 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. The approximation approaches reported in the literatures can be classified as local search approaches, genetic algorithms, fuzzy-based optimizations, evolutionary algorithms, simulated annealing methods, ant colony-based approaches, Markov chain approaches, and neural network approaches.

In this paper, we propose four learning automata-based approximation algorithms for finding near optimal solutions to the minimum (vertex) coloring problem. It is shown that by a proper choice of the parameters of the proposed algorithms, the probability of approximating the optimal solution is as close to unity as possible. The performance of the last proposed algorithm is measured through experimental results, in terms of the time and the number of colors required for coloring the graphs, and compared with CHECKCOL[8], GLS[7], ILS[4], TPA[5] and AMACOL[6]. The results show that the proposed algorithm outperforms the others.

The rest of the paper is organized as follows. The learning automata are described in the next section. In section III, the proposed learning automata-based algorithms are described. The performance of the proposed algorithms is evaluated through the simulation experiments in section IV, and section V concludes the paper.

978-0-7695-3595-1/09 $25.00 © 2009 IEEE
DOI 10.1109/ICIME.2009.106

II. LEARNING AUTOMATA, AND VARIABLE ACTION SET LEARNING AUTOMATA

A. Learning Automata

A learning automaton [2-3] 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. Learning automata can be classified into two main families: 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.

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

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

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

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.

B. 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 [3] 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 $q_i(k) = prob[A(k) = A_i \mid A_i \in A, 1 \le i \le 2^n - 1]$. $\hat{p}_i(k) = prob[\alpha(k) = \alpha_i \mid A(k), \alpha_i \in A(k)]$ 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 $\hat{p}_i(k) = p_i(k) / K(k)$, 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 $p_i(k) = prob[\alpha(k) = \alpha_i]$.

III. THE PROPOSED ALGORITHMS

In this section, we propose four intelligent approximation algorithms based on learning automata for coloring a given graph with the minimum expected number of colors (minimum coloring problem) as defined in section I. In each algorithm, a network of learning automata isomorphic to the graph is initially formed by assigning a learning automaton to each vertex of the graph. Such a network of learning automata can be described by a duple $< \underline{A}, \underline{\alpha} >$, where $\underline{A} = \{A_1, A_2, ..., A_m\}$ denotes the set of learning automata and $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_m\}$ denotes the set of actions in which $\alpha_i = \{\alpha_{i1}, \alpha_{i2}, ..., \alpha_{iri}\}$ defines the set of actions can be taken by learning automaton A_i, for each $\alpha_i \in \underline{\alpha}$. The set of colors with which each vertex v_i can be colored form the set of actions can be taken by learning automaton A_i. Each of the proposed algorithms contains of a number of stages, and at each stage, each learning automaton randomly selects one of its colors and assigns it to its corresponding vertex. Thus, a coloring is made at each stage. The set of colors to be selected at each stage is referred to as color-set. Then, the algorithm decides whether the coloring is a legal coloring or not (except algorithm 4. At each stage k, the cardinality of the minimum color-set that has yet been found is denoted by dynamic threshold T_k. The cardinality of the color-set of a given legal coloring is then compared with dynamic threshold T_k. If it colors the graph with the smaller number of colors, then it is rewarded by the environment, and penalized otherwise. The various legal colorings are iteratively produced and the action probability vectors are updated until a near optimal solution is found to solve the minimum coloring problem with a probability close enough to unity.

In the first proposed algorithm, which we call Algorithm 1, it is assumed that the number of colors can be assigned to each vertex v_i is equal to the number of vertices

in the graph (i.e., n), and therefore the action set of each learning automaton A_i which is referred to as α_i, contains the same number of actions also. Stage k of algorithm 1 is briefly described in the following steps:

Step 1. Color selection phase

- For all learning automata do in parallel
  - Each automaton A_i chooses one of its actions according to its action probability vector.
  - Vertex v_i is colored with the color corresponding to the selected action.
  - The selected color (if it has not yet been added) is added to the list of colors (color-set) with which the graph may be legally colored at this stage.

Step 2. Feasibility checking

- If no two vertices have the same color, the coloring is legal, and otherwise step 1 is repeated to select a new set of colors.

Step 3. Updating the dynamic threshold and action probabilities

- If the cardinality of the color-set (in a legal coloring) created at stage k is less than or equal to dynamic threshold T_k, then
  - Threshold T_k is set to the cardinality of the color-set selected in this stage.
  - All learning automata reward their actions and update action probably vectors using a L_{R-I} reinforcement scheme.
- Otherwise,
  - Each learning automaton updates its probability vector by penalizing its chosen action.

Step 4. Stopping Condition

- The process of selecting legal colorings of the graph and updating the action probabilities are repeated until the product of the probability of choosing the colors of a legal coloring called PLC is greater than a certain threshold or the number of colorings exceeds a pre-specified threshold. The coloring which is chosen last before stopping the algorithm is the coloring with the smallest color-set among all proper colorings.

- In the first proposed algorithm, the number of actions available for each learning automaton is assumed to be equal to the number of vertices in the graph. Although the probability of finding a legal coloring is much higher when a large number of actions is available, such an action set significantly prolongs the convergence delay of each learning automaton to the optimal action as well as the running time of the algorithm. On the other hand, it is shown that an arbitrary graph can be colored with at most $\Delta+1$ colors, where Δ denotes the maximum degree of the vertices in the graph. To alleviate the convergence latency, we propose another algorithm in which the number of actions can be chosen by each learning automaton is equal to $\Delta+1$. Due to the reduction in number of actions, it is expected that the second proposed algorithm colors the graphs in less time, and with a smaller number of colors comparing algorithm 1. Algorithm 1 in which such a modification in number of actions is made is called algorithm 2.

- In the previous algorithms, all learning automata have the same number of actions, while some vertices of the graph can be easily colored with a much smaller number of colors (actions). This gives the same importance to all colors, and results in an excessive delay in finding the proper color for a large number of vertices. On the other hand, in these algorithms, each learning automaton locally decides to color itself and does it based on its neighbors' color. Furthermore, in the worst case, each vertex v_i and its neighbors can be colored with at most Δ_i+1 colors, where Δ_i is the degree of vertex v_i. Therefore, we propose another algorithm in which the number of actions available for each learning automaton A_i is locally determined based on the degree of vertex v_i. The algorithm in which the number of actions of each learning automaton is computed as the number of its adjacent automaton plus one is called algorithm 3.

The drawback of the first four proposed algorithms is that they can not guarantee to find a legal coloring of the graph at each stage. This weakness significantly increases the number of colorings needs to be computed until a legal coloring is found, and therefore increases the running time of algorithm. To eliminate the problem of improper colorings, we propose another learning automata-based algorithm in which, unlike the algorithms described earlier, the number of actions can be taken by each learning automaton varies with time. This property allows the learning automata to pick the only colors they have not yet been selected by the adjacent learning automata. In this algorithm, to avoid the improper colorings, each learning automaton A_i changes its number of actions by disabling the actions selected by its adjacent learning automata as described in subsection II on variable action set learning automata. Algorithm 3 in which, at each stage, a legal coloring of the graph is found is called algorithm 4. This algorithm differs from algorithm 3 in the action set defined for each learning automaton as well as the color selection phase (step 1). Since all colorings are found in step 1 of algorithm 4 are legal, step 2 (feasibility checking) is not required to be verified in this algorithm.

IV. NUMERICAL RESULTS

To study the efficiency of the proposed algorithms, we have conducted two groups of simulation experiments. The first group is concerned with investigating the impact of the

learning rate on the size of the color set, and running time of the proposed algorithms. In the second group of experiments, algorithm 4, which outperforms the other proposed algorithms, is tested on a subset of hard-to-color benchmarks like Leighton, DSJ and Wap. The performance of algorithm 4 is measured in terms of the time and the number of colors required for coloring the graphs, and compared with CHECKCOL[8], GLS[7], ILS[4], TPA[5] and AMACOL[6].

In all experiments presented in this paper, the reinforcement scheme used for updating the action probability vectors is L_{R-I}. Each algorithm is terminated when the probability of the chosen color set (PLC) is 0.9 or greater, or the number of colorings exceeds a pre-defined threshold.

To study the impact of the learning rate on the size of the color set and running time of the proposed algorithms, the learning rate is varied from 0.06 to 0.5 and each of the proposed algorithms is tested on the standard random graph DSJC125.1. The average color set size and the average running time of each algorithm for coloring the graph is given in table I. Each result reported in table I is averaged over 100 runs. The results show that, in all algorithms, the size of the color set decreases as the learning rate decreases. Furthermore, for all proposed algorithms, it can be seen that the time required for coloring the graph increases as the learning rate decreases. For instance, in algorithm 1, the average size of the color set is 14 and the running time (in seconds) of algorithm is 0.0030, when the learning rate is 0.30, while the color set decreases to 5 and the running time increases to 16.437, as the learning rate decreases to 0.07.

The results reported in table I also reveal that algorithm 4 outperforms the others, and algorithm 1 has the worst performance comparing algorithms 1 and 2. Comparing the results of algorithms 1 and 2, we find that, due to the reduction in number of actions of each learning automaton, algorithm 2 is capable of coloring the graph with a smaller color set in a shorter time. It also can be seen that, algorithm 3 slightly outperforms algorithm 2. This is due to the fact that the number of actions available for each learning automaton is determined based on the local density of the graph, which results in a higher convergence rate for most learning automata.

In the second group of experiments, algorithm 4 is tested on three different classes of the hard-to-color benchmark graphs, and compared with those of CHECKCOL[8], GLS[7], ILS[4], TPA[5] and AMACOL[6]. The results are summarized in tables II, III, and IV. 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). In these experiments, the learning rate of algorithm 4 is set to 0.1, and the algorithm is terminated when the probability of the chosen color set is 0.9 or greater, or the number of colorings exceeds a pre-defined threshold.

Comparing the results of the other algorithms, reported in table II, 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 algorithm 4 with AMACOL, we find that the color sets chosen by algorithm 4 are as small as those of AMACOL, while the running time of algorithm 4 is considerably shorter than AMACOL. On the other hand, comparing algorithm 4 with GLS, it can be seen that the running time of algorithm 4 is as close to GLS as possible, while the size of the color set, in algorithm 4, is significantly smaller comparing GLS.

Comparing the results reported in table III, 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 algorithm 4 with the chromatic number of the benchmarks, we observe that the size of the color sets constructed by algorithm 4 is equal to the chromatic number. It is shown that, CHECKCOL is also capable of finding the optimal solution, but the running time of algorithm 4 is much less than that of CHECKCOL.

Comparing the results reported in table IV, 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 results of algorithm 4 show that the size of the color set constructed by algorithm 4 is close enough to the best results. On the other hand, the running time of algorithm 4 is considerably shorter comparing GLS and TPA.

V. CONCLUSION

In this paper, four learning automata-based approximation algorithms were proposed to solve the minimum (vertex) coloring problem. The proposed algorithms iteratively find the different possible colorings of the graph, and depending on the response received from the environment, the color-sets are rewarded or penalized. Finally, the algorithms learn how to find the minimum coloring of the graph, among all the legal colorings, with the highest probability. The last proposed algorithm was compared with some well-known coloring algorithms and the results showed the efficiency of the proposed algorithm in terms of the color set size and running time of algorithm.

REFERENCES

[1] 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.

[2] K. S. Narendra and K. S. Thathachar, "Learning Automata: An Introduction," New York, Printice-Hall, 1989.

[3] 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.

[4] 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.
[5] 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.
[6] 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.
[7] 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.
[8] 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.

TABLE I. THE AVERAGE SIZE OF THE COLOR SET AND THE AVERAGE RUNNING TIME OF THE PROPOSED ALGORITHMS

Learning rate	Algorithm 1		Algorithm 2		Algorithm 3		Algorithm 4	
	CN	RT(Sec.)	CN	RT(Sec.)	CN	RT(Sec.)	CN	RT(Sec.)
0.06	5	20.718	5	7.7100	5	6.1143	5	0.0907
0.07	5	16.437	7	3.2763	5	2.9050	5	0.0309
0.08	6	11.016	7	1.1003	5	0.9056	5	0.0231
0.09	9	3.2300	8	0.4530	6	0.1020	5	0.0100
0.10	9	1.2029	9	0.1309	7	0.0469	5	0.0139
0.15	9	0.9210	9	0.1250	8	0.0160	5	0.0069
0.20	10	0.0469	10	0.0470	9	0.0149	6	0.0060
0.25	12	0.0112	12	0.0094	8	0.0053	6	0.0005
0.30	14	0.0062	12	0.0030	10	0.0025	6	0.0000
0.35	14	0.0030	12	0.0032	10	0.0009	7	0.0000
0.40	15	0.0047	14	0.0011	10	0.0000	7	0.0000
0.45	15	0.0024	13	0.0007	11	0.0000	8	0.0000
0.50	15	0.0015	14	0.0000	12	0.0000	8	0.0000

TABLE II. A PERFORMANCE COMPARISON OF ALGORITHM 4 AND THE MOST EFFECTIVE COLORING ALGORITHMS (TESTED ON HARD-TO-COLOR DSJ BENCHMARK GRAPHS)

Graph	Best	TPA[5]		AMACOL[6]		ILS[4]		CHECKCOL[8]		GLS[7]		Algorithm4	
		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	0.0139
DSJC125.5	17	19	289	17	125	17	2	17	110	18	0	17	27.654
DSJC125.9	44	44	5	44	57	44	0	44	4	44	0	44	53.567
DSJC250.1	8	8	10	8	12	8	0	8	28	9	0	8	22.455
DSJC250.5	28	30	3282	28	64	28	34	28	557	30	1	28	60.045
DSJC250.9	72	72	155	72	2604	72	6	72	182	73	6	72	89.340
DSJC500.1	12	12	0	12	9	13	0	12	4	13	0	12	50.450
DSJC500.5	48	48	124	48	326	50	106	48	1789	52	81	49	100.23
DSJC500.9	126	127	1268	126	1710	128	82	126	2045	130	154	127	170.90
DSJC1000.1	20	21	28	20	969	21	6	21	142	22	1	20	61.120
DSJC1000.5	84	84	2386	84	9235	91	303	84	7025	93	546	84	305.12
DSJC1000.9	224	226	3422	224	4937	228	2245	226	12545	234	1621	225	765.25

TABLE III. PERFORMANCE OF ALGORITHM 4 AND COMPARISON WITH THE MOST EFFECTIVE COLORING ALGORITHMS (TESTED ON HARD-TO-COLOR LEIGHTON BENCHMARKS)

Graph	Best	TPA[5]		AMACOL[6]		ILS[4]		CHECKCOL[8]		GLS[7]		Algorithm4	
		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	47.250
Le450_15b	15	15	1655	15	345	15	0	15	2756	15	0	15	53.230
Le450_15c	15	15	82	15	2	15	19	15	4534	15	6	15	90.625
Le450_15d	15	15	34	15	4	15	20	15	4576	15	8	15	87.245
Le450_25c	25	26	44	26	93	26	2	25	3477	26	18	25	91.110
Le450_25d	25	26	22	26	10	26	1	25	4524	26	2	25	97.345

TABLE IV. A PERFORMANCE COMPARISON OF ALGORITHM 4 AND THE MOST EFFECTIVE COLORING ALGORITHMS (TESTED ON HARD-TO-COLOR WAP BENCHMARKS)

Graph	Best	TPA[5]		AMACOL[6]		ILS[4]		CHECKCOL[8]		GLS[7]		Algorithm4	
		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	23.490
Wap02a	41	41	1618	44	802	43	251	43	486	41	160	42	45.730
Wap03a	44	44	17	53	245	46	365	46	689	44	782	44	101.21
Wap04a	43	43	95	48	45	44	484	44	23	43	834	43	90.420
Wap06a	41	41	348	44	545	42	1	42	25	41	8	41	11.230
Wap07a	42	42	541	45	89	44	1	44	182	42	215	42	33.415
Wap08a	42	42	200	45	446	43	56	44	22	42	41	42	39.200