A new permutation model for solving the graph k-coloring problem

István Juhos[1], Attila Tóth[1], Masaru Tezuka[2], Phillip Tann[3], and Jano I. van Hemert[4]

[1] University of Szeged
[2] Hitachi East Japan Solutions, Ltd.
[3] University of Sunderland
[4] National Institute for Mathematics and Computer Science, Amsterdam (CWI)

Abstract

This paper describes a novel representation and ordering model, that is aided by an evolutionary algorithm, is used in solving the graph k-coloring. A comparison is made between the new representation and an improved version of the traditional graph coloring technique DSATUR on an extensive list of graph k-coloring problem instances with different properties. The results show that our model outperforms the improved DSATUR on most of the problem instances.

1 Introduction

The main goal of this paper to present a new effective algorithm for graph k-coloring problem, which is known to be *NP*-complete [2]. Furthermore, we perform an empirical study where we compare with an improved version [3] of the well known DSATUR [1]. Section 2 provides a brief overview of the graph k-coloring problem and techniques for solving it. In Section 3 we describe the new representation of the problem. Section 4 explains how to incorporate the representation into the evolutionary algorithm. Then, in Section 5 we present the experimental results on the competition graph database [4]. Finally, we draw conclusion in Section 6.

2 Graph k-coloring

The problem class known as graph k-coloring is defined as follows, given a graph $\langle V,E\rangle$ where $V = \{v_1, ..., v_n\}$ is a set of vertices and $E = \{(v_i, v_j)|v_i,v_j \in V\ i\neq j\}$ is a set of edges, where every edge lies between two vertices. In the graph k-coloring problem every vertex in V should be assigned one of k colors such that no two vertices connected with an edge in E have the same color. Many algorithms to solve this problem have been created and studied. Early algorithms are both complete and sound. However, as it was proven that such algorithms exhibit an exponential effort to execute when the problem is scaled up, many algorithms studied today use a stochastic process to guide them towards suboptimal solutions or hopefully towards an optimal solution. Examples are Tabu-search [5], simulated annealing [6] and hybrid techniques [7]. One popular approach to dealing with graph k-coloring in specific is evolutionary computation [8, 9, 10]. Unfortunately, evolutionary algorithms are not necessarily very good in solving such constraint satisfaction problems [11] as they may suffer from a number of flaws that keeps them from reaching optimal solutions [12]. One obstacle that is appropriate for graph k-coloring is when the problem at hand contains symmetry. There are a number of studies addressing this problem in general [13][14]. The size of the search space is determined by the representation of the solutions for the graph k-coloring problem. When symmetry is not taken under consideration this size is k^n. On the other hand, when symmetry is explicitly removed we may reduce the search space, when $n \geq k$ to the size as reported by [15] *(Eq. 1).*

$$\Pi(n,k) = \sum_{i=0}^{k} (-1)^k \binom{k}{i} (k-i)^n \tag{1}$$

The goal of graph k-coloring is to find the minimum number of colors, i.e., the lowest *k* that still yields a valid coloring. This *k* is then called the chromatic number.

3 Representing the Graph k-Coloring Problem

Merge Model (MM)

We represent the colored graph in a table format (*See Figure 3*) called Merge-Table (MT). Columns are assigned to the vertices and rows are assigned to the colors. From point of view of coloring edges are constraints. These are coded as values in the cells of the MT. One of three possible values must be assigned to each cell in the MT, 0, 1 or X. If the vertex *i.* is colored with color *c* then set the cell *MT(c,i)* to 1. A cell *MT(c,i)* is set to zero if the color *c* assigned to that vertex would violate a constraint, i.e., an edge exists between this vertex c and another vertex colored with c. If it does not matter whether we color a vertex *i* with color *c* than we set the cell *MT(c,i)* to *X.*

We initialize the MT by assigning color *i* to vertex *i* for each of the vertices, which always yields a valid coloring as any graph coloring problem with *n* vertices can be colored with *n* colors. Thus in our MT representation this relates to setting *MT(i,i) = 1*. Then for every edge *(i,j)* we set each cell in the MT to 0, thus *MT(c,j)=0* if and only if *c=i* and *(i,j)* in *E.* All the remaining cells of MT are set to *X.* An example of such a Merge-Table is in *Figure 3*, which is initialized from the graph in *Figure 1*.

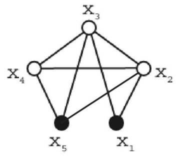

Figure 1

If the graph is not a full graph than it might be possible to decrease the number of colors needed to color the graph. In our representation this amounts to reducing rows in the MT. To make reduction of rows possible we introduce a merge operation (MO). This operation merges two rows only when assigning the same color to both vertices does not create an invalid coloring, i.e., the two vertices are not connected by an edge. Two rows can not be merged if at least one column exists where one cell is set to 0 and one cell is set to 1. If two rows can be merged than the set of rules in *Figure 2* determine how the new row is created.

1 and X = 1	*0 and X = 0*	*X and X = X*
1 and 1 = 1	*0 and 0 = 0*	***1 and 0 = row merge is cancelled***

Figure 2
Merge rules of Merge Operation and results.

We give an example of a merging operation and reduce two rows in MT “a” in *Figure 3* to produce a new table MT “b”. The rows belonging to colors y_1 and y_5 are candidate to a merge operation. After merging the two rows are replaced by the row *$MO(y_1,y_5)$*, which means that vertices x_1 and x_5 are now colored by the same color.

MT “a”

	x_1	x_2	x_3	x_4	x_5
y_1	1	0	0	X	X
y_2	0	1	0	0	0
y_3	0	0	1	0	0
y_4	X	0	0	1	0
y_5	X	0	0	0	1

MT “b”

	x_1	x_2	x_3	x_4	x_5
$MO(y_1,y_5)$	1	0	0	0	1
y_2	0	1	0	0	0
y_3	0	0	1	0	0
y_4	X	0	0	1	0

Figure 3
Merge example of two rows. Examination of the MT “a” clearly shows that in y_5 row, x_5 got y_5 color (there is 1 in the cell) but x_2, x_3 or x_4 cannot be colored by y_5 (meaning of 0 in the cells). Although x_1 can be colored by same color as x_5 (meaning of X in the cell). MT “b” shows merge results, $\boldsymbol{y_1}$ and $\boldsymbol{y_5}$ was merged.

Permutation Merge Model (PMM)

The results, i.e., the coloring of a graph, after two or more merge operations depend on the order in which these operations were performed. Consider the simple hexagon as a graph, presented in *Figure 4*. Now let the sequence of the rows *1,4,2,5,3,6* in the merge operations and consider the following merging procedure. Take the second row of sequence *4* and try to merge with previous one *1* the result comes from *MO(1,4)* merge operation. Taking next row *2*, *MO(MO(1,4),2)* is cancelled, thus *2* leaves in its original place. Continue the merging with the next rows *5,3,6* . The allowed merges are the *MO(1,4)*, *MO(2,5)* and *MO(3,6)*. This sequence of merge operations results a 3-coloring of the graph. However, if we use the sequence *1,3,5,2,4,6* than we end up with a 2-coloring of the graph with *MO(1,3), MO(MO(1,3),5)), MO(2,4)* and *MO(MO(2,4),6)* merges.

Initial MT					
1	0	X	X	X	0
0	1	0	X	X	X
X	0	1	0	X	X
X	X	0	1	0	X
X	X	X	0	1	0
0	X	X	X	0	1

MO	***Merge order 1,4,2,5,3,6***					
(1,4)	1	0	0	1	0	0
(2,5)	0	1	0	0	1	0
(3,6)	0	0	1	0	0	1

MO	***Merge order 1,3,5,2,4,6***					
(1,3,5)	1	0	1	0	1	0
(2,4,6)	0	1	0	1	0	1

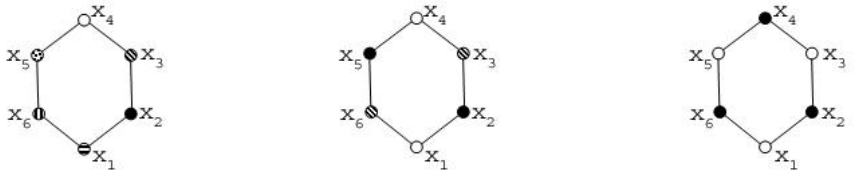

Figure 4
Hexagon graph's initial Merge-Table is shown on the left side. Middle table shows the result of the 1,4,2,5,3,6 merge order of the initial MT and right side is the result of the 1,3,5,2,4,6 merge order. "MO" column contains the rows used by Merge-Operation.

Lemma 1.

An arbitrary valid k-coloring of a graph G can be associated to a merged Merge-Table.

Proof. (constructive)

1. Consider an $k \times n$ Merge-Table, where n is the number of the vertices and k is the number of colors used in the coloring. Let a cell MT(c,i) cell contain 1 if color c is assigned to vertex i, let it contain 0 if and only if (i,j) is in E and MT(c,j)=1, otherwise let it contain X.

Proposition 1.

If O is an optimal coloring of the graph G, i.e., it uses the minimal amount of colors required, than there exists a sequence a merge operations P that reduces the initial Merge-Table such that O is obtained after applying P.

Proof. (constructive)

Suppose O is an optimal solution. Construct a merged Merge-Table according to O, this exists because of Lemma 1. Take apart Merge-Table rows that contain more than one 1 until the number of rows is equal to the number of vertices of G. Take apart a row i with more than one 1, choose a cell M(i,j) that contains a 1 and make a new row m. Set MT(m,j)=1 to M(i,j)=X. Set all MT(i,r) cells to X where vertex r is not connected to vertex i. Fill row m in the usual way, vertex j's neighbors are set to 0 (keep in mind MT(m,j)=1) and the remaining cells to X. By *repeating this until the MT contains n rows will provide the initial MT with one 1 in every row. By reversing the sequence we get the sequence of merge operators P.*

To find a minimal coloring for a graph k-coloring problem using the MT table representation and merge operations we need to find the sequence of merge operations that leads to that coloring. Thus the problem is a permutation of candidate reduction steps. We call this the Permutation Merge Model (PMM). The search space of this model is *n!* and contains many local optima. We will search in this space by using an evolutionary algorithm.

4 Evolutionary Algorithm for PMM (PMM-EA)

Representation

The phenotype is a candidate solution of the coloring, which stems from a merge table after applying a sequence of merge operations. The genotype is a permutation of the identifiers of MT rows, e.g., 4,3,2,1,5. The initial population contains randomly generated permutations.

Genetic operators

Mutation is done by swapping two row identifiers in the permutation (an example is in *Figure 5*) and the crossover is the order-based crossover for permutations (an example is in *Figure 6*) [16].

P_1	1	**4**	3	2	**5**	6
P_1’	1	**5**	3	2	**4**	6

Figure 5
Mutation of a permutation is changing two elements 4 and 5 in the order.

P_1	**1**	**4**	3	*2*	6	*5*
P_2	*2*	*5*	**4**	6	2	**1**
P_1’	**4**	**1**	3	2	6	5
P_2’	*2*	*5*	**4**	6	2	**1**

Figure 6
Crossover is an order-based crossover; reorder the head according to the other permutation provided order

Fitness

The k-χ defines a measurement that shows how far we are from an optimal coloring, under the assumption that we know the chromatic number. We should extend this measurement with a more fine property. The number of zeros in the reduced MT gives us a degree of how compact the MT is, which we can use to smooth the search landscape. We use the following function, which needs to be minimized, to evaluate candidate solutions,

$$f(P) = (k - \chi) \cdot z \tag{2}$$

P: a permutation of the row identifiers
k: remained rows after merge
χ: chromatic number
z: number of zeros in the merged MT

Stop condition

The algorithm is stopped if the maximum number of generations of 1000 is reached or the fitness of an individual is equal to zero, which means it used the same number of colors as the chromatic number.

The below table shows the applied parameters of evolution algorithm:

mutation probability	0.3
crossover probability	0.8
population size	depends on problem 1-100
maximum number of generation	1000

5 Experiments

Reference algorithm

DSATUR from Brèlaz [1] uses a heuristic to dynamically change the ordering of the vertices and then applies the greedy method to color the vertices. It works as follows, one vertex with the highest saturation degree, i.e., number of adjacent colors, is selected and it is assigned the lowest color that still yields a valid coloring. In case of a tie, the vertex with the highest degree, i.e., number of neighboring vertices, that are still in the uncolored subgraph is selected. In case another tie a random vertex is selected. Because of the random tie breaking, DSATUR becomes a stochastic algorithm and similar for the EA, results of several runs need to be averaged to obtain useful statistics.

TRICK-DSATUR is an improved version of the DSATUR was introduced in [3], which first tries to find the largest clique, i.e., a subgraph that is a full graph, in the graph. It allocates colors for the vertices in each clique. Then, uncolored vertices are dynamically ordered by saturation of color and subproblems are created as in the basic DSATUR algorithm The last step is to solve the subproblems in depth-first manner.

Means of comparison

The basis of the comparison is the computational cost. We measure this as the number of constraint checks. It means we count that how many times an algorithm checks if two vertices are connected or not until it reaches either the chromatic number or maximum running time. The reason for the name "constraint checks" lies in that edges make constraints in the graph k-coloring problem. Constraint examination takes the most computational costs in constraint solving algorithms.

Problem instances

The test graphs are from [4], which is a standard competition graph repository. Which also contains the reference algorithm TRICK-DSATUR.

Results

The PMM-EA model is implemented in C++ and uses the EASEA [17] framework and EO evolutionary library [18]. We use TRICK-DSATUR code from [4] with an extension for keeping track of the number of constraint checks. The results are presented in *Figure 7*.

When the chromatic number χ was found we stopped the TRICK-DSATUR similar to the evolutionary algorithm. The results are the average of 10 independent runs with a constant population size and each run uses one of 10 randomly generated seeds for the random generator. The same set of random seeds was used for each graph. For PMM-EA the population size was adjusted to match the problem size. The two algorithms always find the chromatic number in every run.

Graph	**|V|**	**|E|**	χ	**T-DSATUR**	**PMM-EA**	
				constraint checks	*constraint checks*	*population size*
anna	138	493	11	89,024	**23,526**	1
david	87	406	11	34,557	**12,959**	1
homer	561	1629	13	1,484,278	**1,233,263**	5
huck	74	301	11	25,783	**10,732**	1
jean	80	254	10	29,939	**18,724**	2
fpsol2.i.1	496	11654	65	1,311,037	**668,504**	1
fpsol2.i.2	451	8691	30	1,078,861	1,757,187	6
fpsol2.i.3	425	8688	30	973,200	**700,479**	2
mulsol.i.1	197	3925	49	292,942	**176,316**	1
mulsol.i.2	188	3885	31	229,907	**116,938**	1
mulsol.i.3	184	3916	31	224,934	**118,185**	1
mulsol.i.4	185	3946	31	228,474	**128,585**	1
mulsol.i.5	186	3973	31	232,835	**117,443**	1
zeroin.i.1	211	4100	49	289,622	304,632	2

zeroin.i.2	211	3541	30	289,191	**95,763**	1
zeroin.i.3	206	3540	30	277,504	**208,941**	3
games120	120	638	9	101,070	**40,969**	1
miles250	128	387	8	76,961	185,460	8
miles500	128	1170	20	89,025	844,256	13
miles750	128	2113	31	136,866	12,383,472	80
miles1000	128	3216	42	462,986	14,803,819	10
miles1500	128	5198	73	404.051	1,487,468	6
queen5_5	25	160	5	8,797	25,184	3
queen6_6	36	290	7	223,804	18,366,697	50
myciel3	11	20	4	798	**176**	1
myciel4	23	71	5	4,273	**829**	1
myciel5	47	236	6	22,104	**4,318**	1
myciel6	95	755	7	117,163	**15,104**	1
myciel7	191	2360	8	638,536	**121,708**	2

Figure 7
Comparison PMM-EA with TRICK-DSATUR (T-DSATUR) by constraint checks. χ is the chromatic number.

6 Discussion

We presented a new model for an evolutionary algorithm to tackle the graph k-coloring problem, which provides success on a set of various problem instances. The evolutionary algorithm PMM-EA has a better average performance, i.e., is faster, than TRICK-DSATUR for 20 problem instances out of 29.

PMM-EA has substantial problems when solving queen6.6. The clique finding step of the TRICK-DSATUR before starting to color the graph eliminates some symmetry of the search space. Also, TRICK-DSATUR creates a clique of size six, which is close to the lower bound of number of colors in the optimal coloring of queen6.6. PMM-EA may be extended with a clique finding method, which may improve its performance for this and similar graph k-coloring problems.

For a number of problem instances a solution is found using one individual from PMM-EA. This means the representation of the problem is able to solve these problem instances without any global search. This is makes the PMM model a powerful local search operator.

7 Acknowledgement

We would like to thank EVONET Summer School which put us in touch and Professor JÁNOS CSIRIK, the head of the Department of Informatics in University of Szeged for the valuable advises.

8 References

[1] D. Brèlaz. New methods to color the vertices of a graph. Communications of the ACM, 22(4):251--256, 1979.

[2] Cheeseman, P., B. Kenefsky, andW. Taylor (1991). Where the really hard problems are. In J. Mylopoulos and R. Reiter (Eds.), Proceedings of 12th International Joint Conference on AI (IJCAI-91),Volume 1, pp. 331–337. Morgan Kauffman.

[3] A. Mehrotra and MA Trick, A Column Generation Approach for Graph Coloring, INFORMS Journal on Computing, 8:344-354, 1996.

[4] Michael Trick: Network Resources for Coloring Graphs, http://mat.gsia.cmu.edu/COLOR/color.html , Last revision: November 3, 1994

[5] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. Computing, 39:345–351, 1987.

[6] Johnson, D. S., C. R. Aragon, L. A. McGeoch, and C. Schevon (1991, May-June). optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. Operational Research 39(3), 378–406.

[7] P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 1998.

[8] J.I. van Hemert. Application of Evolutionary Computation to Constraint Satisfaction and Data Mining. PhD thesis, Leiden University, 2002.

[9] A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. Journal of Heuristics, 4(1):25–46, 1998.

[10] E. Falkenauer. Genetic Algorithms and Grouping Problems. John Wiley & Son Ltd., 1998.

[11] J.I. van Hemert. Comparing classical methods for solving binary constraint satisfaction problems with state of the art evolutionary computation. In Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther Raidl, editors, Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim, volume 2279 of lncs, pages 81–90, Kinsale, Ireland, 3–4 April 2002. Springer-Verlag.

[12] J.I. van Hemert and T. Bäck. Measuring the searched space to guide efficiency: The principle and evidence on constraint satisfaction. In J.J. Merelo, A. Panagiotis, H.-G. Beyer, Jose-Luis Fernandez-Villaca˜nas, and Hans-Paul Schwefel, editors, Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, number 2439 in lncs, pages 23–32, Berlin, 2002. Springer.

[13] Clarissa Van Hoyweghen, Bart Naudts, and David E. Goldberg. Spin-flip symmetry and synchronization. Evolutionary Computation, 10(4):317–344, 2002.

[14] Anna Marino and Robert I. Damper. Breaking the symmetry of the graph colouring problem with genetic algorithms. In Darrell Whitley, editor, Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference, pages 240–245, Las Vegas, Nevada, usa, 8 July 2000.

[15] Even, S. (1973). Algorithmic Combinatorics. Collier-Macmillan.

[16] Michalewicz, Z. Genetic agorithms + data structures = evolution programs, Springer-Verlag, 219.

[17] "Take it EASEA," Parallel Problem Solving from Nature VI, vol 1917, Springer pp 891-901, Paris, September 2000.

[18] Maarten Keijzer, Juan J. Merelo Guervós, Gustavo Romero, Marc Schoenauer: Evolving Objects: A General Purpose Evolutionary Computation Library. Artificial Evolution 2001: 231-244