University "Alexandru Ioan Cuza" of Iaşi
Faculty of Computer Science

TECHNICAL REPORT

An Ordering-Based Genetic Approach to Graph Coloring

Cornelius Croitoru
Ovidiu Gheorghieş
Adriana Gheorghieş

TR 05-03, July 2005

ISSN 1224-9327

Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. Berthelot 16, 700483 -Iaşi, Romania
Tel. +40-232-201090, email: bibl@infoiasi.ro

An Ordering-Based Genetic Approach to Graph Coloring

Cornelius Croitoru, Ovidiu Gheorghieş, and Adriana Gheorghieş

University "Al. I. Cuza" Iaşi, Romania
Faculty of Computer Science
{croitoru,ogh,adrianaa}@infoiasi.ro

Abstract. We introduce a new evolutionary formulation of the graph coloring problem, based on the interplay between orderings and colorings of vertices. The new formulation aims at breaking the symmetry in the solution space and provides opportunities for combining evolutionary and other search techniques. Our formulation is very simple compared to previous approaches which use the relationship between a graph's chromatic number and its acyclic orientations. We have experimented on graphs from the DIMACS Computational Challenge. The results provide evidence that the ordering approach can be used to obtain accurate colorings. We also give a general property which accounts for the computational difficulty of all approaches that attempt to iteratively reduce the number of colors in a coloring.

1 Introduction

If $G = (V, E)$ is a graph and k a positive integer, a k-coloring of (the vertices of) G is any assignment $c : V \to \{1, .., k\}$ with the property that for each $i \in \{1, ..., k\}$ the set $c^{-1}(i) = \{v | v \in V, c(v) = i\}$ is a stable set in G, that is a set of mutually non-neighbor vertices. The set $c^{-1}(i)$ is called the color class i of the coloring c and the list $(S_1, ..., S_k)$ of all color classes completely determines the coloring c.

It is well-known that it is NP-complete to decide whether, for a given graph G and an integer k there exists a k-coloring of G [10], [15]. The least possible number k of colors for which a graph G has a k-coloring is called the chromatic number of G and is denoted by $\chi(G)$.

The graph coloring problem is the problem of finding, for a given graph G, an optimal coloring, that is a coloring with $\chi(G)$ colors. Since this problem is not approximable within $|G|^{1/14-\epsilon}$ for any $\epsilon > 0$ [4], the graph coloring problem is one of the favorites to try new meta-heuristics for (e.g. simulated annealing [14], tabu search [13]). Other approaches include DSatur [5] and greedy strategies [6] which have been outperformed by hybrid techniques based on maximal stable sets [9].

We provide a new evolutionary formulation of the graph coloring problem which is based on the interplay between orderings and colorings of the vertices. This formulation gives the possibility of combining evolutionary and other search

techniques. We note that our formulation is very simple compared with that given in [3] which uses the well-known relationship that exists between a graph's chromatic number and its acyclic orientations [8].

We also give a general property that shows how a $k-1$ coloring can be obtained form an already constructed k coloring. This property is used to devise a graph coloring algorithm. The property can also be used to explain the computational difficulty of all approaches that attempt to iteratively reduce the number of colors in a coloring, thus leaving open the quest for yet more powerful techniques. However, the "no free lunch" theorem ([19], [20], [7]) states that there cannot be a single algorithm that prevails over others for all problem instances.

2 Orderings and Colorings

Let $G = (V, E)$ be a graph with the vertices set $V = \{1, 2, ..., n\}$. The set of all $n!$ permutations on the set V is denoted by $\mathcal{S}_n$ and each permutation $v \in \mathcal{S}_n$, $v = v_1 v_2 ... v_n$, is interpreted as an ordering of vertices of G: v_1 is the first vertex in the ordering v, v_2 is the second one and so on.

For $v_i \dots v_j \in \mathcal{S}_n$ and $i, j \in \{1, ..., n\},\ i < j$ we denote by $I[v, i, j]$ the interval determined by the two indices i and j in v, that is, $I[v, i, j] = \{v_{i+1}, ..., v_{j-1}\}$. We also denote by $G[v, i, j]$ the subgraph induced in G by $I[v, i, j]$.

Definition 1. *Let $e \in E$ be an edge of G and $v \in \mathcal{S}_n$ an ordering. We call e a bad edge with respect to v if $e = v_i v_j,\ i < j$ and $G[v, i, j] - e$ is a null subgraph of G.*

Note that by this definition any edge connecting two consecutive vertices in v, $e = v_i v_{i+1}$, is a bad edge with respect to v.

For each ordering $v \in \mathcal{S}_n$ we denote by $b(v)$ the number of all bad edges in G with respect to v.

The following theorem and its proof show that determining an optimal coloring in a graph is equivalent to finding an ordering of its vertices with a minimum number of bad edges.

Theorem 1.

$$\chi(G) = 1 + \min_{v \in \mathcal{S}_n} b(v)$$

Proof. 1. Let $v^0 \in \mathcal{S}_n$ be an ordering with minimum number of bad edges. Starting with v_1^0 construct a maximal stable set of consecutive vertices in v^0: $\{v_1^0, ..., v_i^0\}$. Clearly, $i \geq 1$ and either $i + 1 > n$ or v_{i+1}^0 is adjacent with at least one vertex from $\{v_1^0, ..., v_i^0\}$. Call this stable set S_1 and repeat the above construction, starting with v_{i+1}^0 and obtaining $S_2, S_3, ...$, until all vertices of v^0 have been considered. We obtain a k-coloring $(S_1, S_2, ..., S_k)$ of G with the property that for each $t \in 2, ..., k$, if v_j^0 is the first vertex (with respect to the

ordering v^0) of S_t, then there is a last vertex v_i^0 in S_{t-1} such that $v_i^0 v_j^0 \in E$. Clearly, by the choice of v_i^0, the edge $v_i^0 v_j^0$ is a bad edge with respect to v^0. Thus, the number k of colors in the coloring associated with v^0 satisfies the inequality $k - 1 \leq b(v^0)$. Since $k \geq \chi(G)$, we have

$$\chi(G) \leq k \leq 1 + b(v^0) \tag{1}$$

2. Let $(S_1, S_2, ..., S_k)$ be an optimal ($k = \chi(G)$) coloring of G. Each stable set $S_i,\ 1 \leq i \leq k-1$ can be extended to become a maximal stable set in the subgraph induced in G by $S_i \bigcup S_{i+1} \bigcup ... \bigcup S_k$. The number of stable sets obtained remains k since $k = \chi(G)$. The obtained k-coloring $(S_1, S_2, ..., S_k)$ has the property that each vertex $u \in S_i$ has at least one neighbor in $S_{i-1},\ 2 \leq i \leq k$. Using this property we construct an associated ordering $v^0 \in \mathcal{S}_n$ in the following way:

(a) the last vertices in the ordering v^0 are the vertices of S_k in an arbitrary order; if $S_k = \{s_1, ..., s_p\}$ then $v_n^0 = s_1,\ v_{n-1}^0 = s_2, ..., v_{n-p+1}^0 = s_p$;
(b) the vertex before v_{n-p+1}^0 (i.e. v_{n-p}^0) is a vertex $u \in S_{k-1}$ which is a neighbor of v_{n-p+1}^0; there is such a vertex by the above property of the coloring. Put $v_{n-p} = u$ and the remaining vertices of S_{k-1} are added to v^0 in an arbitrary order (i.e. $S_{k-1} - \{u\} = \{u_1, ..., u_t\}$, put $v_{n-p-1}^0 = u_1,\ v_{n-p-2}^0 = u_2, ..., v_{n-p-t}^0 = u_t$);
(c) repeat (b) for $S_{k-2}, S_{k-3}, ...$ until all vertices of S_1 (and therefore of G) are added to v^0.

The ordering v^0 constructed above has only $k - 1$ bad edges, namely those obtained in the step (b) of the algorithm, connecting consecutive vertices in v^0 which have also consecutive colors. Therefore we have $b(v^0) + 1 = k = \chi(G)$ and since $b(v^0) \geq \min_{v \in \mathcal{S}_n} b(v)$ we obtain

$$\chi(G) = b(v^0) + 1 \geq 1 + \min_{v \in \mathcal{S}_n} b(v) \tag{2}$$

By (1) and (2) the theorem holds. □

Let us note that by (1) and (2) above the coloring and the ordering constructed in the proof are optimal. In other words, the two constructions described in the proof show how to efficiently obtain an optimal coloring from an ordering with minimum number of bad edges and conversely.

3 Description of the Genetic Algorithm

This section describes a genetic algorithm which employs ordering based approach to graph coloring.

Genetic algorithms [16] are probabilistic techniques which mimic the natural evolutionary process. A genetic algorithm maintains a population of candidate solutions. This is one important feature that differentiates them from conventional heuristics like hill climbing, simulated annealing, tabu search etc. Genetic

algorithms have thus the potential to better explore the search space; moreover, they provide tools for avoiding "premature convergence".

Each candidate solution in the population is encoded into a structure called chromosome. To each chromosome, a value (called fitness value) is assigned. The fitness value represents the quality of the candidate solution encoded by the chromosome. Assigning fitness values to chromosomes is called evaluation.

A selection procedure simulates the "survival of the fittest" paradigm from nature. Better-fitted chromosomes have higher chances of surviving to the next generation. The number of chromosomes per generation is constant.

As in natural life, offspring chromosomes are obtained from parent chromosomes. One possibility is for two parents to exchange encoded information and thus creating two new offspring; this is called crossover. Another possibility is to alter the encoded information in a chromosome obtaining a slightly different new chromosome; this is called mutation. Some other chromosomes simply survive unaltered, while others die off. Mutation and crossover are referred to as genetic operators.

3.1 Representation

A chromosome represents an ordering of the vertices of the graph. We encode the ordering by means of a permutation.

If the graph has n vertices, the chromosome will be a vector

$$chrom = (v_1, v_2, ..., v_n),\ where\ v_i \in \{1, 2, ..., n\},\ v_i \neq v_j, i \neq j \qquad (3)$$

The candidate solution represented by the chromosome is obtained in a straightforward manner: the position of vertex v_i in the ordering is i.

3.2 Evaluation

The purpose of evaluation is to point out how good the candidate solution encoded by a chromosome is. In order to provide such a measurement, a coloring is constructed from each ordering (chromosome). The number of colors of the coloring (or, alternatively, the number of bad edges) is taken as a measure of an individual's quality. This is consistent with the minimization of the number of colors.

Typically, a heuristic for graph coloring consists of two parts: in the first part, a suitable ordering of the vertices is fixed and in the second part, the actual coloring algorithm is applied using the fixed order of vertices. A very basic coloring procedure is the *sequential algorithm* (sometimes called *greedy algorithm*), which proceeds as follows. Assume the vertices of the graph are given in the order $v_1, v_2, ..., v_n$; they will be processed in this order. Assign color 1 to v_1. For each of the remaining vertices v_i, assign to v_i the minimum color number available, that is, the smallest color number that, so far, has not been assigned to any vertex adjacent to v_i. Though the local action of the sequential algorithm appears to be quite reasonable, globally it may fail miserably for unfavorable

vertex orderings. It is not difficult to construct a sequence $G_3, ..., G_m, ...$ of graphs such that each G_m is 2-colorable, the size of G_m is linear in m, and yet the number of colors used by the sequential algorithm on input G_m is at least m for at least one vertex ordering.

Similar results can be obtained for other graph coloring heuristics, most of which apply the sequential coloring algorithm to various vertex orderings that are obtained by seemingly reasonable procedures.

Furthermore, these computational results conducted us to a novel approach to the above sequential procedure (we call it *LexBF*). Clearly, the sequential coloring can be equivalently implemented by finding successive lexicographic-first maximal stable sets with respect to a previously fixed ordering $v_1, v_2, ..., v_n$, until all vertices are considered. If the order of the remaining vertices is changed after each construction of a color class it seems that this adaptive coloring algorithm proceeds better. A good candidate for a dynamic changing of the ordering can be deduced from the following interesting observation.

Proposition 1. *Let $G = (V, E)$ be a graph and $w_1, w_2, ..., w_n$ a vertex ordering generated by a breadth first traversal of G. If S is the lexicographic-first (with respect to the BFS ordering $w_1, w_2, ..., w_n$) maximal stable set of G, then the bipartite graph obtained from G by deleting all edges with both extremities in $V - S$, has the same connected components as the graph G.*

Indeed, we can suppose that G is connected and it is not difficult to show that any vertex v of G can be reached from the vertex w_1 on a path that uses edges with only one extremity in $V - S$ (by induction on the distance in G from v to w_1). The maximal stable set S has, by the above proposition, the property that it is incident to many edges (at least $n-1$, if G is connected) in a somewhat uniform way. We note that there are nondeterministic choices to be made whenever there are more neighbors not yet visited at any point in the BF traversal of G. These ties will be solved by using an initial ordering $v_1, v_2, ..., v_n$. Our new graph coloring heuristic can be described as:

```
Use a particular ordering v_1, v_2, ..., v_n to direct all the BF traversals

While G is not empty do
   Apply a BF traversal to G and find the first
   (in the lexicographic BF ordering obtained) maximal stable set S;
   Let S be the new color class;
   G:=G-S
```

Experimental results show that the LexBF heuristic performs better than the sequential algorithm (table 1). For this reason we have used LexBF as fitness function for the genetic algorithm.

3.3 Genetic Operators

The role of the genetic operators ([12], [17]) is to explore the search space and to exploit the information acquired during the evolution process. Each operator has a rate, which results in the number of chromosomes to which that operator is applied. These rates are parameters of the genetic algorithm; they are tuned in order to balance exploration and exploitation [2].

Guiding the search of the genetic algorithm towards promising regions in the search space requires additional knowledge about the problem to be incorporated. Valuable information is provided by the fitness function because color classes are constructed in a deterministic way. Early choices influence subsequent options. This means that preserving the relative order of some vertices may preserve the structure of a particular color class. Also, changing the relative order of vertices may induce changes in the color classes.

We define one crossover operator and three mutation operators. They are designed to either preserve a relative ordering of some vertices (crossover), or modify it (mutations).

Crossover The crossover is designed to propagate and exchange information regarding color classes throughout evolution. Since color classes are supposedly built from successive nodes in parent chromosomes, there are blocks of nodes which are inherited by offspring. There is no a priori reason for such blocks to have the same size, which leads to the idea of using two possibly different cutpoints, one for each parent.

Let $\text{parent}^1 = (v_1^1, v_2^1, ..., v_n^1)$ and $\text{parent}^2 = (v_1^2, v_2^2, ..., v_n^2)$ be two parent chromosomes. We consider two randomly generated cutpoints $c^1, c^2 \in \{1, 2, ..., n-1\}$ for parent^1 and parent^2 respectively. One offspring is obtained by keeping unaltered the genetic information from parent^1 before c^1; the vertices after c^1 from the first parent are rearranged using the ordering defined by the second parent. The second offspring is constructed in a similar way (figure 1).

Random Swap Mutation Any mutation operator is aimed at sampling the search space in a neighborhood of a chromosome. For random swap mutation this is achieved by changing the order of some (few) vertices.

Let $\text{parent} = (v_1, v_2, ..., v_n)$ be a parent chromosome. The offspring differs from the parent by a randomly generated number l of interchanges between vertices. The interval for the values of l is empirically adjusted before the genetic algorithm is run. Small values for l may lead to a too focused exploration, while large values may alter dramatically the genetic heritage. The interval is scaled to the number of vertices in the graph. This way, the dependence between the performance of the genetic algorithm and the instance of the problem is diminished. An example is illustrated in figure 2.

Block Move Mutation This type of mutation performs translations of blocks of consecutive vertices in the ordering. Let $\text{parent} = (v_1, v_2, \ldots, v_i, \ldots, v_j, \ldots, v_n)$

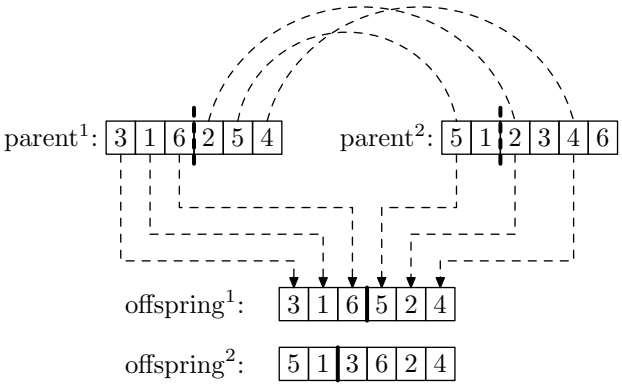

Fig. 1. Example of crossover for a graph with 6 vertices. The parent chromosomes are parent^1 and parent^2. If we consider the cutpoints $c^1 = 3$ and $c^2 = 2$, the offspring obtained are offspring^1 and offspring^2

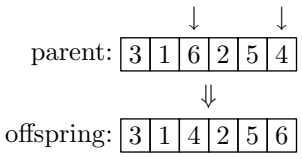

Fig. 2. Example of random swap for a graph with 6 vertices when $l = 1$. The arrows indicate vertices that will interchange their position in the ordering.

be a parent chromosome, i the position where the bloc starts, k the size of the bloc and $j \in \{1, 2, \ldots, i-2, i-1, i+k, i+k+1, \ldots n\}$ a new position in the ordering. The bloc defined by i and k contains the vertices $v_i, v_{i+1}, \ldots, v_{i+k-1}$. Values for i and j are randomly generated, while k is a parameter of the GA for which lower and upper bounds are specified.

- If $j < i$, j represents the new start position for the bloc, and the vertices $v_j, v_{j+1}, \ldots, v_{i-1}$ are shifted k positions to the right (figure 3 a).
- If $j \geq i + k$, j represents the new end position for the bloc, and the vertices $v_{i+k}, v_{i+k+1}, \ldots, v_j$ are shifted k positions to the left(figure 3 b).

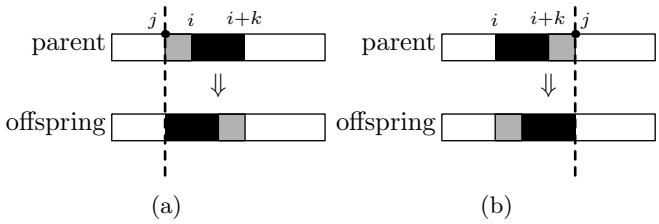

Fig. 3. Illustration of block move mutation.

Neighbors Swap Mutation This kind of mutation randomly selects a vertex v_i and rearranges the neighbors of v_i. To this end, pairs of neighbors of v_i, if exist, are randomly chosen. Each pair of selected neighbors interchange their position in the ordering. Vertices that are not neighbors of v_i are not affected by this operator (their position remain unchanged).

3.4 Selection

The selection we use is rank-based. The chromosomes are ordered according to their fitness values. Chromosomes having the same fitness value are grouped into one rank (there are as many ranks as there are different fitness values). We build a probability field which indicates what are the chances for a particular rank to be selected. Once a rank is selected, a chromosome from that rank is randomly picked.

4 Discussion of experimental results

We have used graphs from the DIMACS Computational Challenge suite [1]. The benchmark program had the following output: `(sys) 0 (user) 56 (Real) 56`.

Table 1 summarizes the experimental results we obtained. For each coloring method the number of colors obtained and the computational time is recorded.

The time is measured in seconds. A zero indicates a value less than a millisecond. Due to their nature, sequential algorithm, LexBF heuristic and GA may give different results in multiple runs. In table 1 we have recorded the best coloring obtained and its associated time.

The ICA is described in section 5. The time recorded is the number of seconds between the start of the algorithm and the moment when the last reduction of the number of colors is achieved.

In order to compare the sequential algorithm and the LexBF heuristic for a given graph, we have generated a set of orderings. From each ordering, a coloring was constructed accordingly. The LexBF heuristic outperformed the sequential algorithm on most graphs, while the computational times were similar. Also, LexBF proved to be less sensitive to variations in initial orderings.

Table 1 provides the best results obtained by the LexBF and sequential heuristics over 200 randomly generated orderings. We have also tested the heuristics on sets of orderings up to 100000. Apart from increasing the computational time, only few minor improvements in the colorings were obtained. This suggests that the number of favorable orderings is very small and that random search has little chance of leading towards an optimal solution [7].

The genetic algorithm was build on top of the `even` genetic algorithms library [11]. The GA was run using various parameter settings. The purpose was to study the influence of each genetic operator on the evolution. The crossover, the block moves and neighbors swap proved to have a favorable effect on the quality of the solution, and also on the convergence speed. Random swap, however, had a destructive effect: whenever it was used it lead to poorer results.

The results for GA presented in table 1 were obtained using a population of 100 individuals. For genetic operators, the following rates were used: crossover 0.5, block move 0.1 and neighbors swap 0.2. If for 30 generations no better coloring is found, the genetic algorithm stops.

The GA is able to find a better coloring than the LexBF and sequential heuristics, at the expense of a higher computational cost.

Table 1. Experimental results

Seq= Sequential heuristic
LexBF= LexBF heuristic
ICA= *Intriguing* coloring algorithm, described in section 5
GA= Genetic algorithm

graph	n	m	density	LB	colors				run time (seconds)			
					Seq	LexBF	GA	ICA	Seq	LexBF	GA	ICA
DSJC125.1.col	125	736	9 %	5	7	7	7	5	0.040	0.080	1.782	6.41
DSJC125.5.col	125	3891	50 %	12	23	23	22	18	0.090	0.110	3.586	17.19
DSJC125.9.col	125	6961	90 %	30	52	52	49	44	0.160	0.171	6.91	22.12
DSJC250.1.col	250	3218	10 %	8	12	11	11	9	0.241	0.320	8.131	0.95
DSJC250.5.col	250	15668	50 %	13	40	40	38	32	0.531	0.571	16.604	120.11
DSJC250.9.col	250	27897	90 %	35	93	92	88	77	1.101	1.222	52.085	183.6
DSJC500.1.col	500	12458	10 %	6	18	18	18	14	1.052	1.352	39.286	14.75
DSJC500.5.col	500	62624	50 %	16	70	70	68	59	4.937	5.218	173.049	129.4
DSJC500.9.col	500	224874	90 %	42	171	169	167	153	12.488	12.959	766.732	222.26
DSJR500.1.col	500	3555	3 %	12	13	12	12	12	0.781	2.013	47.619	0.88
DSJR500.1c.col	500	121275	97 %	63	101	101	94	89	6.630	6.990	361.28	86.36
DSJR500.5.col	500	58862	47 %	26	140	135	132	130	9.353	7.351	364.144	273.4
DSJC1000.1.col	1000	49629	10 %	6	30	30	29	25	8.682	10.175	514.250	31.39
DSJC1000.5.col	1000	249826	50 %	17	123	123	120	111	44.353	48.550	2174.21	283.38
DSJC1000.9.col	1000	449449	90 %	54	310	312	306	290	103.990	105.972	4394.71	203.59
latin_square_10.col	900	307350	76 %		144	145	138	121	31.455	32.617	1840.05	272.21
le450_15a.col	450	8168	8 %	15	20	20	19	16	0.931	1.091	25.767	7.99
le450_15b.col	450	8169	8 %	15	20	19	19	15	0.911	0.951	26.057	251.02
le450_15c.col	450	16680	17 %	15	28	28	28	23	1.142	1.262	37.815	16.61
le450_15d.col	450	16750	17 %	15	29	28	27	22	1.202	1.402	39.386	295.36
le450_25a.col	450	8260	8 %	25	26	26	26	25	1.101	1.032	28.16	0.67
le450_25b.col	450	8263	8 %	25	26	26	25	25	0.941	0.941	32.207	0.63
le450_25c.col	450	17343	17 %	25	34	34	33	28	1.292	1.272	46.687	16.87
le450_25d.col	450	17425	17 %	25	34	34	32	28	1.292	1.392	60.928	14.59
le450_5a.col	450	5714	6 %	5	12	12	11	6	0.751	1.282	26.277	9.74
le450_5b.col	450	5734	6 %	5	12	11	11	6	0.641	0.901	24.806	10.87
le450_5c.col	450	9803	10 %	5	14	12	10	5	0.772	0.971	36.112	1.61
le450_5d.col	450	9757	10 %	5	15	13	12	5	0.771	0.992	54.268	3.36
school1_nsh.col	352	14612	24 %	14	35	29	23	14	0.861	0.981	50.633	10.13

graph	n	m	density	LB	colors				run time (seconds)			
					Seq	LexBF	GA	ICA	Seq	LexBF	GA	ICA
queen8_12.col	96	1368	30 %	12	13	14	13	12	0.050	0.070	1.863	0.56
queen8_8.col	64	728	36 %	9	11	11	11	9	0.030	0.030	0.721	0.66
queen9_9.col	81	2112	65 %	10	12	13	12	10	0.040	0.050	1.132	3.1
queen10_10.col	100	2940	59 %		14	14	13	11	0.060	0.070	2.384	209.27
queen11_11.col	121	3960	55 %	11	15	15	15	12	0.090	0.100	2.664	273.33
queen12_12.col	144	5192	50 %		16	17	16	14	0.130	0.151	5.899	3.41
queen13_13.col	169	6656	47 %	13	18	19	18	15	0.190	0.210	5.528	3.19
queen14_14.col	196	8372	44 %		19	19	19	16	0.261	0.270	8.312	7.43
queen15_15.col	225	10360	41 %		21	21	20	17	0.390	0.411	16.734	39.37
queen16_16.col	256	12640	39 %		22	23	22	18	0.480	0.491	15.802	73.75
myciel5.col	47	236	22 %	6	6	6	6	6	0.010	0.010	0.261	0
myciel6.col	95	755	17 %	7	7	7	7	7	0.030	0.040	1.011	0.01
myciel7.col	191	2360	13 %	8	8	8	8	8	0.130	0.171	4.356	0.06
mugg100_25.col	100	166	3 %	4	4	4	4	4	0.02	0.11	1.102	0.01
ash331GPIA.col	662	4185	2 %		6	7	6	4	1.908	6.064	139.931	1.94
will199GPIA.col	701	6772	3 %		9	8	7	7	3.574	8.825	149.215	2.17
1-Insertions_4.col	67	232	10 %	4	5	5	5	5	0.020	0.020	0.541	0
1-Insertions_5.col	202	1227	6 %	4	6	6	6	6	0.150	0.260	5.237	0.06
1-Insertions_6.col	607	6337	3 %		7	7	7	7	1.392	2.884	57.632	1.4
2-Insertions_4.col	149	541	5 %	4	5	5	5	5	0.060	0.130	2.924	0.03
2-Insertions_5.col	597	3936	2 %	4	6	6	6	6	1.172	2.804	57.873	1.33
3-Insertions_4.col	281	1046	3 %	3	5	5	5	5	0.201	0.510	11.717	0.15
3-Insertions_5.col	1406	9695	1 %		7	6	6	6	6.439	16.804	381.268	17.03
4-Insertions_3.col	79	156	5 %	3	4	4	4	7	0.010	0.040	0.741	0.02
4-Insertions_4.col	475	1795	2 %	3	5	5	5	8	0.551	1.472	34.890	2.04
1-FullIns_3.col	30	100	23 %	4	4	4	4	4	0.000	0.010	0.160	0
1-FullIns_4.col	93	593	14 %	5	5	5	5	5	0.030	0.060	1.181	0.01
1-FullIns_5.col	282	3247	8 %	6	6	6	6	6	0.311	0.590	12.908	0.16
2-FullIns_3.col	52	201	15 %	5	5	5	5	5	0.010	0.010	0.361	0
2-FullIns_4.col	212	1621	7 %	5	6	6	6	6	0.160	0.341	6.299	0.07
2-FullIns_5.col	852	12201	3 %	6	8	7	7	7	3.054	7.130	153.331	3.87
3-FullIns_3.col	80	346	11 %	5	6	6	6	6	0.020	0.050	0.842	0.01
3-FullIns_4.col	405	3524	4 %	6	7	7	7	7	0.911	1.212	25.566	0.43
3-FullIns_5.col	2030	33751	2 %	6	9	8	8	8	20.460	39.026	866.185	50.87
4-FullIns_3.col	114	541	8 %	7	7	7	7	7	0.040	0.080	1.602	0.02
4-FullIns_4.col	690	6650	3 %	7	8	8	8	8	1.783	3.455	70.411	2.04
4-FullIns_5.col	4146	77305	1 %		10	9	9	9	77.502	169.574	4573.420	463
5-FullIns_3.col	154	792	7 %	8	8	8	8	8	0.080	0.150	2.854	0.03
5-FullIns_4.col	1085	11395	2 %		9	9	9	9	4.556	8.843	188.922	7.87

5 Why reducing the number of colors is difficult

What the genetic algorithm actually tries to do is to construct a $k-1$-coloring from a k-coloring it has already constructed. This approach to coloring is shared by a wider range of heuristics. We show that while this approach has a theoretical justification, it suffers from provable computational difficulties.

Perhaps the most natural approach to reduce the number of colors in a coloring is to pick a color class and then redistribute its nodes into the other color classes. Obviously, if for some node in the picked color class there is another color class containing no neighbors of that node, then the node can be moved into the neighbor-free color class. If the property holds further for all nodes in the chosen color class, the number of colors can be reduced by one. However, this is seldom the case when non-trivial coloring problems are tackled. More often than not, some node will have neighbors in all other color classes, preventing the node to be assigned to another color. In this situation, most heuristics would just quit.

The first question is: "can more be done"? Since the neighbors of a node prevent it from having its color reassigned, shouldn't we try to reassign the colors of the neighbors to make room for that node? The second question is: "would this work at all"? Is there any guarantee that this approach would eventually lead us to a coloring with less colors (if any)?

We show that the answer is "yes" to both questions. Based on that, a graph coloring algorithm can be devised.

For a k-coloring c we define restrictions by disallowing certain nodes to have a specific color. We will show how these restrictions can be used to guide the process of reducing the number of color classes.

Definition 2. *A denial list for a k-coloring is a function $D : \{1, 2, \ldots, k\} \to \mathcal{P}(\mathcal{V})$. We say that a k-coloring c respects the denial list D if $c^{-1}(i) \cap D(i) = \emptyset$, for all $i \in \{1, 2, \ldots, k\}$.*

Let be $I \subseteq V$ an arbitrary set of "intriguing" nodes. The intriguing nodes corresponds to the nodes in the above discussion that hinders the removal of one color class. We will focus on these kind of nodes with the goal to recolor them with less distinct colors.

Definition 3. *A $k^{\langle l,I,D\rangle}$-coloring is a k-coloring c that respects D so that the intriguing nodes are colored with l distinct colors: $|c(I)| = l$.*

Theorem 2. *Let $G = (V, E)$ be a graph and c a $k^{\langle k,I,D\rangle}$-coloring. G admits a $k^{\langle k-1,I,D\rangle}$-coloring if and only if there is a color class $i \in c(V)$ such that, if $v_1, v_2, \ldots, v_t$ is an arbitrary ordering of $\mathrm{DS}_i = I \cap c^{-1}(i)$*

$$
\begin{array}{llll}
\exists\, c_1 \text{ a } k^{\langle k-1,N(v_1),D'\rangle}\text{-coloring for} & G', & \text{so that} & \mathrm{mayAdd}(v_1, c_1, D') \\
\exists\, c_2 \text{ a } k^{\langle k-1,N(v_2),D'\rangle}\text{-coloring for} & G' \cup \{v_1\}, & \text{so that} & \mathrm{mayAdd}(v_2, c_2, D') \\
 & \ldots & & \\
\exists\, c_t \text{ a } k^{\langle k-1,N(v_t),D'\rangle}\text{-coloring for} & G' \cup \{v_1 \ldots v_{t-1}\}, & \text{so that} & \mathrm{mayAdd}(v_t, c_t, D')
\end{array}
$$

where

- DS_i *is called the dwindling source because these nodes are removed from the graph and then reinserted (so that they obey additional constraints) until none remains*
- $G' = G - \mathrm{DS}_i$
- D' *is the same as* D *but also denies intriguing nodes into color* i*:* $D'(i) = D(i) \cup I$
- $\mathrm{mayAdd}(v, c, D)$ *holds by definition if there is a color class* j *in the* k*-coloring* c *which contains no neighbors of* v *and, moreover,* v *is not denied by* D *into* j*:* $\exists j \in \{1, 2, \ldots, k\} - c(N(v))$ *such that* $v \notin D(j)$*.*

Proof. $\Longrightarrow$ Suppose G admits a $k^{\langle k-1,I,D\rangle}$-coloring, c'. This means that in c' all the intriguing nodes $v \in I$ are colored with at most $k-1$ distinct colors. Since in c the intriguing nodes were colored with k colors, it means that at least one of these colors is I-free in c'. Let i be one of these I-free colors in c'.

Let us consider an arbitrary node $v_1 \in \mathrm{DS}_i$, as colored by c'.

Obviously, $c'(N(v_1)) \leq k-1$, since none of v_1's neighbors has the same color as v_1. This means that coloring c' restricted to $G - \mathrm{DS}_i$ (which is denoted by c_1) is a $k^{\langle k-1,N(v_1),D'\rangle}$-coloring, where $D'(i) = D(i) \cup I$ (since no nodes in I have color i).

Let j be the color of v_1 in c', $j = c'(v_1)$. Since c_1 is a restriction of c' and furthermore respect D, it means that in coloring c_1 there is one color class j that contains no neighbors of v (otherwise it couldn't have been colored in c'). Node v_1 is not colored with color i (v_1 is intriguing) and furthermore is allowed by D' into color class j. This means, by definition, that $mayAdd(v_1, c_1, D')$ holds.

A similar argument works for the remaining nodes in DS_i.

$\Longleftarrow$ It suffice to see that if c_t exists, we can construct c a $k^{\langle k-1,I,D\rangle}$-coloring for G. Coloring c will be the same as c_t on $G-v$: $c(v) = c_t(v)$ for all $v \in G - \{v_t\}$.

Since c_t respects D', it means that there are no intriguing nodes in I colored with color i: $I \cap c^{-1}(i) = \emptyset$. Moreover, some color class contains no neighbors of c_t, since c_t is a $k^{\langle k-1,N(v_t),D'\rangle}$-coloring. Since $\mathrm{mayAdd}(v_t, c_t, D')$ holds, it means that there is a color j so that v_t is allowed by D' into j. However, this color cannot be i, since v_t is an intriguing node. Therefore, we can set $c(v_t) = j$ and obtain a $k^{\langle k-1,I,D\rangle}$-coloring with color class i containing no intriguing nodes. □

Remark 1. If we consider $I = V$ (all nodes are intriguing) and empty denial list, the theorem gives a characterization for the existence of a $k-1$ coloring for a graph G once a k coloring has been constructed. The characterization has a recursive nature, the subproblems consisting a coloring problem for a subgraph of G.

Note that the theorem states the existence of a color class with the specified property, but it does not indicate which one is it. This means that a practical implementation has to either guess a start color class or exhaustively try all of them. The problem replicates itself recursively on a subgraph.

It is interesting to make a comparison between the approach based on orderings and the approach based on "intriguing" nodes. Finding a suitable ordering will require in the worst case enumerating over all possible orderings of the nodes. On the other hand, constructing a $k-1$ coloring from a k coloring will require the exploration of a decision tree for which each node has a number of children roughly equal to the number of colors. The two approaches can be seen as complementary. Experiments have shown that the ordering based genetic algorithm constructs reasonable colorings; once such a coloring with relatively fewer colors is constructed, one may find more advantageous to improve it iteratively by exploring (the first levels of) the decision tree.

5.1 Notes on algorithm engineering

Theorem 2 can be used in a straightforward manner to devise a recursive graph coloring algorithm which we call *ICA* (intriguing coloring algorithm). There are some challenges that must be addressed.

An important problem is that, quite early in the coloring process a very deep level in the decision tree may be reached. If the top choices are bad (would not lead to a solution, if it exists at all), then a lot of computational power is wasted making useless tries. To address this issue we have explored the decision tree up to a certain level and if no solution was found, we would increase the maximal depth, until some time limit is reached. Compared to the variant that does not restrict the depth, this approach works well, as table 1 shows.

We have also attempted to run the ICA starting from a coloring obtained by means of another heuristic, to see if this would make any difference with regard to the number of colors which can be reached. The answer was generally no, but we have found that this approach may speed up by little margins the coloring process. Also, ICA is capable in most situations to improve the colorings offered by other heuristics employed here (sequential, LexBF, GA).

It has been previously noted [18] that for DSJC125.5 no heuristic manages to go below 18 colors. ICA may give an explanation as why this happens. We have explored exhaustively the decision tree for 18-colorings of DSJC125.5 up to 7 levels deep and found that it was not possible to reduce the number of colors. (We were indeed able to create a 17-coloring for this graph but using a modified version of ICA and exploiting a very lucky seed.) This accounts for the complexity of the problem at hand, since for only 7 levels approximately 18^7 choices must be explored. The presence of this type of intricate constraints may well be used to explain why most heuristics stop at 18.

6 Conclusions

We have presented a novel approach to the graph coloring problem, suitable for evolutionary implementations. We implemented a genetic algorithm based on this approach. Our experiments prove that a heuristic-hybridized GA performs better than stand-alone heuristics.

We have also evinced a pattern for all algorithms that attempt to iteratively reduce the number of color classes of an existing coloring. We have shown why this type of approach to graph coloring faces inherent difficulties.

References

1. DIMACS benchmark graphs. `http://mat.gsia.cmu.edu/COLORING02/index.html`.
2. Thomas Bäck, David Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. IOP Publishing Ltd., 1997.
3. Valmir C. Barbosa, Carlos A. G. Assis, and Josina O. do Nascimento. Two novel evolutionary formulations of the graph coloring problem. Available electronically.
4. Mihir Bellare and Madhu Sudan. Improved non-approximability results. volume STOC, pages 184–193, 1994.
5. D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–265, 1979.
6. J. Culberson and F. Luo. Exploring the k-colorable landscape with iterated greedy. *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, RI:AMS*, pages 245–284, 1996.
7. Joseph C. Culberson. On the futility of blind search. Technical Report 96–18, Edmonton, Alberta, Canada, 1996.
8. R.W. Deming. Acyclic orientations of a graph and chromatic and independence numbers. *Journal of Combinatorial Theory B*, (26):101–110, 1979.
9. P. Galinier and J.K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorics 3(4)*, pages 379–397, 1999.
10. M.R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, New York, 1979.
11. Ovidiu Gheorghieş. `even`, A C++ genetic algorithms library. `http://students.infoiasi.ro/~cevol`, 2000.
12. D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.
13. A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing 39(4)*, pages 345–351, 1987.
14. D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and member partitioning. *Operations Research 39*, pages 378–406, 1991.
15. R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
16. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 3 edition, 1996.
17. H. Muhlenbein. How genetic algorithms really work. Part I: Mutation and hill-climbing. In R Manner and B Manderick, editors, *Parallel Problem Solving from Nature, 1*, Amsterdam, 1992. Elsevier-North-Holland.
18. Michael Trick. CP2002, Summary of Symposium. `http://mat.gsia.cmu.edu/COLOR02/summary.ppt`, 2002.
19. David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe, NM, 1995.
20. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.