Two Novel Evolutionary Formulations of the Graph Coloring Problem

Valmir C. Barbosa[*]
Carlos A. G. Assis[*]
Josina O. do Nascimento[*†]

[*] Universidade Federal do Rio de Janeiro
Programa de Engenharia de Sistemas e Computação, COPPE
Caixa Postal 68511, 21945-970 Rio de Janeiro - RJ, Brazil

[†] Observatório Nacional
Rua General José Cristino, 77, 20921-400 Rio de Janeiro - RJ, Brazil

Abstract

We introduce two novel evolutionary formulations of the problem of coloring the nodes of a graph. The first formulation is based on the relationship that exists between a graph's chromatic number and its acyclic orientations. It views such orientations as individuals and evolves them with the aid of evolutionary operators that are very heavily based on the structure of the graph and its acyclic orientations. The second formulation, unlike the first one, does not tackle one graph at a time, but rather aims at evolving a "program" to color all graphs belonging to a class whose members all have the same number of nodes and other common attributes. The heuristics that result from these formulations have been tested on some of the Second DIMACS Implementation Challenge benchmark graphs, and have been found to be competitive when compared to the several other heuristics that have also been tested on those graphs.

Keywords: Graph coloring, evolutionary algorithms, genetic algorithms, genetic programming.

Address for correspondence and proofs:

Valmir C. Barbosa, `valmir@cos.ufrj.br`
Programa de Sistemas, COPPE/UFRJ
Caixa Postal 68511
21945-970 Rio de Janeiro - RJ
Brazil

1. Introduction

We consider a graph $G = (N, E)$, where N is the graph's set of nodes and E its set of edges, each edge being an unordered pair of nodes, called *neighbors* of each other. Graph G is said to be an *undirected* graph, since the unordered nature of its edges assigns no "directionality" to them. We let $n = |N|$ and $m = |E|$.

A *coloring* of G is an assignment of *colors* (positive integers) to the nodes in such a way that every node is assigned exactly one color and, furthermore, no two neighbors are assigned the same color. The *graph coloring problem* is the problem of providing G with a coloring that employs the least possible number of colors. This number is called the *chromatic number* of G, denoted by $\chi(G)$.

The reasons why the graph coloring problem is important are twofold. First, there are several areas of practical interest in which the ability to color an undirected graph with as small a number of colors as possible has direct influence on how efficiently a certain target problem can be solved. Such areas include time tabling and scheduling [12, 29], frequency assignment for use of the electromagnetic spectrum [19], register allocation in compilers [9], printed circuit board testing [21], and the solution of sparse linear systems originating from finite-element meshes [35].

The other reason is that the graph coloring problem has been shown to be computationally hard at a variety of levels: not only is its decision-problem variant NP-complete [20, 27], but also it is NP-hard to solve it approximately within $n^{1/7-\epsilon}$ of the optimum for any $\epsilon > 0$ [1, 2, 6]. What this means is that, given the current expectation of how likely it is for an algorithm to be found to solve an NP-hard problem efficiently, coloring the nodes of G with at most $n^{1/7-\epsilon}\chi(G)$ colors is an intractable problem for any $\epsilon > 0$. There is a sense, then, in which graph coloring stands among the hardest of the hard combinatorial optimization problems.

These two reasons are important enough to justify the quest for heuristics to solve the graph coloring problem, and to place it together with the favorite problems on which new meta-heuristics are tested. Many such approaches have been proposed (e.g., [25] and the references therein), but we single one out which, although it misses the graph's chromatic number by rather a wide margin in several cases, has an important inspirational role in part of this paper. This heuristic is based on the nodes' *degrees* (numbers of neighbors) and on the known property that $\chi(G) \leq \Delta + 1$, where Δ is the maximum degree over all nodes [7]. The heuristic visits nodes in nonincreasing order of the so-called *saturation degree* (number of colors already used to color neighbors) and assigns the current node the smallest color that is not already taken by one of its neighbors [8]. Symmetry is broken by using a nonincreasing order of degrees to which only neighbors that have not yet been colored contribute, and after that randomly. This heuristic is known as the *DSatur* heuristic.

Despite the problem's importance and the many approaches that have been tried to tackle it, it was not until relatively recently that a concentrated effort was undertaken to test several heuristics on a common set of graphs of various sizes and characteristics. Such an effort was part of the Second DIMACS Implementation Challenge [26], which remains to date the most comprehensive project that we know of to have addressed a systematic experimental evaluation of heuristics for the graph coloring problem.

Comprehensive though this effort was, only one of the approaches to make it to the final meeting was of an evolutionary nature [16]. This approach involved the combination of evolutionary and other search techniques, and was based on previous work by the same authors [15]. In fact, to our knowledge the situation remains that no purely evolutionary approach seems to have had success on the graph coloring problem, although there have been evolutionary approaches for restricted versions of the problem (e.g., [14]) and also other hybrid approaches with evolutionary ingredients (e.g., [18] and the references therein). In our

view, the central difficulty is the difficulty of formulating the problem adequately so that individuals and the appropriate evolutionary operators can be properly identified. Suppose, for example, that we choose to approach the formulation in the obvious, straightforward manner, and say that an individual is simply a particular coloring. That is sound enough, but how does one go about, say, generating a new individual from two parent colorings? Coloring some nodes according to one parent and the others according to the other parent might work, but not without some additional rule to handle the cases in which the same color ends up assigned to neighbors, or else allowing such infeasible color assignments while somehow penalizing them through appropriately low fitness values.

As we perceive it, the problem with this naïve first approach and others closely related to it is the same that prevents massive parallelism from working satisfactorily on the graph coloring problem under certain other meta-heuristics [5]: the problem speaks of a global property of graphs (how many colors overall?), and it is a nontrivial matter to express this global property in terms of more localized indicators, e.g., the chromatic number of *subgraphs* of G (a subgraph of G is a graph whose node set is a subset of N, with the corresponding subset of E for edge set). In particular, if G_1 and G_2 are distinct subgraphs of G, then it does not follow that $\chi(G) = \chi(G_1) + \chi(G_2)$.

Notwithstanding such difficulties, graph coloring is a research area in which much of the underlying structure has already been uncovered and considerable heuristic experience has been accumulated. In this paper, we take advantage of some key aspects of this structure and knowledge to introduce two new evolutionary formulations of the graph coloring problem. The first formulation belongs in the class of techniques we normally think of as genetic algorithms [24, 33], while the second one can be thought of as an instance of genetic programming [3, 28].

Our first approach, described in Section 2, is based on a view of the colorings of G that relates them to the *acyclic orientations* of G. An acyclic orientation of G is any of the possible ways of assigning directions to the (undirected) edges of G in such a way that no *directed cycle* is formed (that is, in such a way that, by traversing edges only according to the assigned directions it is impossible to reach the same node more than once). In general, an orientation of G can be regarded as a function $\omega : E \to N$ such that, for $e = (i, j) \in E$, $\omega(e) = j$ if and only if edge e is directed from node i to node j.

Let a *directed path* in G according to some acyclic orientation be a sequence of nodes, each of which is a neighbor of its predecessor, if it has one, such that the last node can be reached from the first by traversing the edges that connect consecutive nodes to each other along the directions assigned to them by the orientation. The essence of our first approach is that, to any coloring that uses c colors overall there corresponds an acyclic orientation whose longest directed path has no more than c nodes; and that, conversely, to any acyclic orientation whose longest directed path has l nodes there corresponds a coloring by no more than l colors. If we let $\Omega(G)$ denote the set of all acyclic orientations of G, then this gives rise to the following relation between $\chi(G)$ and $\Omega(G)$. For $\omega \in \Omega(G)$, we let $P(\omega)$ denote the set of all directed paths in G according to ω. For $p \in P(\omega)$, let $l(p)$ be the number of nodes in p. Then, from [13], we have

$$\chi(G) = \min_{\omega \in \Omega(G)} \max_{p \in P(\omega)} l(p). \tag{1}$$

Our first approach regards $\Omega(G)$ as the search space out of which populations are formed, that is, each acyclic orientation of G is an individual. The fittest possible individual is the one whose longest directed path is shortest among all possible individuals.

This first approach works on fixed G, that is, every new graph that comes along to be colored triggers the evolution of acyclic orientations in order to find one of high fitness. The second approach, which we describe

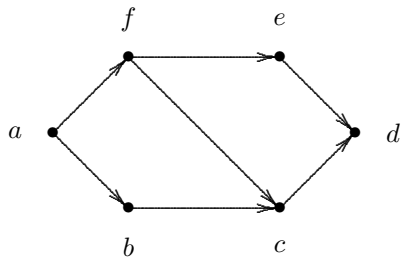

Figure 1. G oriented acyclically

in Section 3, is by contrast targeted at an entire class $\mathcal{G}$ of undirected graphs sharing certain characteristics, as for example the value of n. It is inspired by the DSatur heuristic discussed above, in the sense that it approaches the coloring of G's nodes as the search for a degree-based ordering of those nodes that will yield a good coloring if nodes are colored as dictated by that ordering.

More specifically, suppose for the moment that the number of nodes is the sole characteristic shared by the members of $\mathcal{G}$, and let $\mathrm{K}(n)$ stand for the set of all $n!$ permutations of the sequence $\langle 1, \ldots, n \rangle$. A member $\kappa = \langle k_1, \ldots, k_n \rangle$ of $\mathrm{K}(n)$ can be interpreted as the following sequence of instructions to assign colors to the nodes of any graph $G \in \mathcal{G}$: first assign color 1 to the node having the k_1th largest degree, then assign to the node having the k_2th largest degree the smallest color that does not conflict with the colors possibly already assigned to its neighbors, and so on. If we for now disregard the details that come from the existence of nodes having the same degree, then $\mathrm{K}(n)$ is the search space that contains the populations of our second evolutionary approach. Fitness is in this case an average taken over a pre-selected subclass of $\mathcal{G}$. An individual fitter than another will require a smaller average number of colors over all members of that subclass than the other.

An illustration of the essentials of the two approaches is given in Figure 1, where G is shown with $n = 6$ and directions assigned to the undirected edges in such a way that the graph's orientation is acyclic. The acyclic orientation is meaningful in the context of our first approach only, and implies that G can be colored by no more than four colors, since this is the number of nodes in any longest directed path (e.g., the one from a to d through b and c). In fact, assigning color 1 to node d, color 2 to nodes c and e, color 3 to nodes b and f, and color 4 to node a is a legitimate coloring of the nodes of G, although it is possible to do better easily by employing two colors only. In the latter case, what follows from (1) is that another acyclic orientation of G exists for which the longest directed path has two nodes. Readily, by reversing the directions of all edges directed away from nodes b and f in the figure, one gets such an orientation.

Figure 1 can also be used to illustrate the second approach, but disregarding the directions assigned to edges altogether. Suppose that we have $\kappa = \langle 3, 4, 5, 6, 1, 2 \rangle$; that is, κ instructs us to assign the smallest color that does not cause conflicts to nodes a, b, d, and e first (this is a possible arrangement of the nodes having the third through sixth largest degrees), for example, then to nodes c and f (a possible arrangement of the nodes having the first and second largest degrees), again for example. Clearly, four colors would be needed overall. Once again, it is simple to see that an alternative member of $\mathrm{K}(6)$, for example $\kappa' = \langle 1, 2, 3, 4, 5, 6 \rangle$, would lead to a coloring of the nodes of G by a total of two colors regardless of how the nodes having the same degree were arranged with respect to one another.

In addition to Sections 2 and 3, the remainder of the paper comprises three additional sections. In Section 4, a brief discussion on how the two strategies' evolutionary operators relate to their state spaces is presented. Section 5 contains a description of the graphs that we use in our experiments, which are

drawn from the set of test graphs employed in the DIMACS implementation challenge. Experimental results are given in Section 6. They indicate that our two new evolutionary formulations yield algorithms that are competitive when compared to several others, including those developed in response to the DIMACS challenge. Section 7 contains concluding remarks.

2. The first formulation

Our first formulation is based on the interplay between acyclic orientations of G and colorings of its nodes. If a coloring exists that uses c colors overall, then consider the orientation obtained by directing every edge from the node with the highest color to the one with the lowest. The resulting orientation is clearly acyclic, and furthermore no directed path exists having more than c nodes. In the same vein, consider an acyclic orientation whose longest directed path has l nodes, and consider the nodes that are *sinks* according to this orientation (a sink is a node whose incident edges are all directed inward by the orientation). Now assign colors to nodes as follows. First assign color 1 to all sinks. Then remove all sinks from G (together with all edges incident to them) and assign color 2 to all the new sinks that are thus formed. This removal of sinks can be repeated exactly l times, and each time it is repeated the resulting sinks are assigned their color, which conceivably can be some of the colors used previously. What results is then a coloring that employs no more than l colors.

An important consequence of this interplay is (1), which relates the chromatic number of G to its acyclic orientations. Our first formulation regards acyclic orientations as individuals and aims at evolving an acyclic orientation whose longest directed path is as short as possible. Once this individual is identified, the corresponding coloring of the nodes of G can be found as indicated previously.

Before we give the details of this formulation, we pause momentarily to comment on the cardinality of $\Omega(G)$, the set of all acyclic orientations of G. While for some graphs such cardinality is known as a function of n or m that can be computed easily (2^{n-1} for trees, $n!$ for complete graphs, $2^n - 2$ for rings), the general case requires the evaluation of the so-called *chromatic polynomial* of G. For $c > 0$, this polynomial, denoted by $C_G(c)$, yields the number of distinct ways in which the nodes of G can be colored by a total of at most c colors [7]. Clearly, the smallest such c for which $C_G(c) > 0$ is equal to $\chi(G)$.

Once the chromatic polynomial of G has been introduced, the cardinality of $\Omega(G)$ can be proven to be given by

$$\left|\Omega(G)\right| = (-1)^n C_G(-1),$$

as demonstrated in [37]. We think this is a remarkable property, and felt it should be presented even though it bears no further relationship to the topic of the paper than clarifying how to assess $\left|\Omega(G)\right|$, at least in principle, in the general case, and perhaps providing additional evidence that colorings of G's nodes and acyclic orientations of G are indeed deeply related to one another.

The remainder of this section contains a detailed description of the key elements of our first approach. These are the assessment of an acyclic orientation's fitness, and the functioning of the two evolutionary operators we employ, namely crossover and mutation. We use the following additional notation. For $\omega \in \Omega(G)$, we let $S^+(\omega)$ be the set of sinks in G according to ω. Likewise, $S^-(\omega)$ denotes the set of *sources* in G according to ω (these are nodes whose incident edges are all directed outward by ω). For $i \in N$, $N_i^-(\omega)$ is the set of neighbors j of i such that edge (i, j) is directed outward from i by ω.

Fitness evaluation. The fitness of an individual $\omega \in \Omega(G)$ is the nonnegative integer $f_1(\omega)$, given by

$$f_1(\omega) = n - \max_{p \in P(\omega)} l(p), \tag{2}$$

where we recall that $P(\omega)$ is the set of all directed paths in G according to ω and $l(p)$ is the number of nodes in directed path p. It follows from (2) that an individual has higher fitness than another if its lengthiest directed path is shorter than the other's. By our preceding discussion, an individual ω yields a coloring of G's nodes by no more than $n - f_1(\omega)$ colors.

Clearly, an equivalent reformulation of $f_1(\omega)$ is

$$f_1(\omega) = n - \max_{i \in S^-(\omega)} l_i, \tag{3}$$

where l_i is the number of nodes on the longest directed path in G according to ω that starts at node i. The expression for $f_1(\omega)$ in (3) is more useful because the value of l_i for all $i \in N$ can be computed efficiently by straightforward depth-first search [10], as in procedure $dfs(i, \omega)$, given next.

For all $i \in N$, let $reached_i$ be a Boolean variable used to indicate whether node i has been reached by the depth-first search (it is set to **false** initially). We then have:

procedure $dfs(i, \omega)$
  if $reached_i =$ **false**
    $reached_i :=$ **true**
    if $i \in S^+(\omega)$
      $l_i := 1$
    else
      $l_i := 1 + \max_{j \in N_i^-(\omega)} dfs(j, \omega)$
  Return l_i

This procedure allows us to rewrite $f_1(\omega)$ yet again as

$$f_1(\omega) = n - \max_{i \in S^-(\omega)} dfs(i, \omega), \tag{4}$$

which indicates directly how the fitness of an individual is to be evaluated in practice, requiring for such $O(m)$ time.

Crossover. The crossover of two parent individuals $\omega_1, \omega_2 \in \Omega(G)$ to yield the two offspring $\omega_1', \omega_2' \in \Omega(G)$ is better understood if we adopt a linear representation for the acyclic orientations of G. For $\omega \in \Omega(G)$, say that a node j is *reachable* from node i according to ω if a directed path exists from i to j. Now let the sequence of nodes

$$L(\omega) = \langle i_1, \ldots, i_n \rangle$$

be such that, for $1 \leq x, y \leq n$, $x < y$ if i_y is reachable from i_x.

If we regard ω as a partial order of the nodes of G, then clearly $L(\omega)$ is a linear extension of that partial order and has, as such, the following characteristic. For $i_x \in N$, every other node i_y such that $x < y$ (that is, to the right of i_x in $L(\omega)$) is either reachable from i_x or neither node is reachable from the other. Note that the placement of i_x and i_y in $L(\omega)$ is somewhat arbitrary when no directed path exists between the two nodes, so some symmetry-breaking rule is necessary to construct the linear representation. When this happens, our choice is to let $x < y$ if and only if $i_x < i_y$, where, as in sections to come, we assume for simplicity that the set of nodes N can be regarded as the set of natural numbers $\{1, \ldots, n\}$, so that the comparison "$i_x < i_y$" is meaningful. In Section 6, we comment briefly on the possible effects of using randomness to break symmetry.

We describe the crossover of ω_1 and ω_2 in terms of their linear representations, namely

$$L(\omega_1) = \langle i_1, \ldots, i_n \rangle$$

and

$$L(\omega_2) = \langle j_1, \ldots, j_n \rangle.$$

If z such that $1 \leq z < n$ is the crossover point and we let

$$L(\omega_1') = \langle i_1', \ldots, i_n' \rangle$$

and

$$L(\omega_2') = \langle j_1', \ldots, j_n' \rangle,$$

then $\langle i_1', \ldots, i_z' \rangle = \langle i_1, \ldots, i_z \rangle$ and $\langle i_{z+1}', \ldots, i_n' \rangle$ is the subsequence of $\langle j_1, \ldots, j_n \rangle$ comprising all the nodes that are not already in $\langle i_1, \ldots, i_z \rangle$. The construction of $L(\omega_2')$ is entirely analogous: $\langle j_1', \ldots, j_z' \rangle = \langle j_1, \ldots, j_z \rangle$ and $\langle j_{z+1}', \ldots, j_n' \rangle$ is the subsequence of $\langle i_1, \ldots, i_n \rangle$ comprising all the nodes that are not already in $\langle j_1, \ldots, j_z \rangle$.

In order to give an interpretation of this crossover operation that is meaningful in the context of the acyclic orientations of G, we must first recognize that the linear representation we adopted is unambiguous. In fact, any sequence L comprising all nodes from N can be seen to give rise to a unique acyclic orientation, as follows. For $x < y$, if i_x and i_y are neighbors in G, then direct the edge between them from i_x to i_y. Any directed cycle in the resulting orientation would imply the reflexivity of $<$, which does not hold. Consequently, we are justified in having written $L(\omega_1')$ and $L(\omega_2')$, because the unique orientations that result from those sequences are indeed acyclic.

Not only this, but we can now see clearly what is inherited by ω_1' and ω_2' from ω_1 and ω_2. Orientation ω_1' inherits from ω_1 the directions assigned to all edges joining nodes in the set $\{i_1', \ldots, i_z'\}$ to any other nodes, and from ω_2 those assigned to the edges whose end nodes all lie in $\{i_{z+1}', \ldots, i_n'\}$. As for ω_2', it inherits from ω_2 the directions assigned to all edges joining nodes in the set $\{j_1', \ldots, j_z'\}$ to any other nodes, and from ω_1 those assigned to the edges whose end nodes are all members of $\{j_{z+1}', \ldots, j_n'\}$.

Mutation. There are several possibilities for mutation on an acyclic orientation of G, but we have decided to opt for what seems to be the most elementary one while guaranteeing the generation of another acyclic orientation. Given an acyclic orientation ω and the mutation point $i \in N$, the operation consists of turning node i into a source to yield a new orientation ω'. That ω' is also acyclic has been argued elsewhere (e.g., [4]), and the argument goes as follows. If ω' is not acyclic, then the directed cycle that exists in it must involve node i, which is where the only change was effected on ω to yield ω'. But this is clearly impossible, since according to ω' node i is a source.

3. The second formulation

Our second evolutionary formulation of the graph coloring problem is conceptually simpler than the first. The individuals that it evolves are permutations of the sequence $\langle 1, \ldots, n \rangle$, in which each number is used as an index into a certain reference sequence of the nodes of G. We let $D = \langle i_1, \ldots, i_n \rangle$ be such a sequence, which is arranged in nonincreasing order of node degrees and, like in Section 2, regards nodes as natural numbers for the purpose of breaking symmetry when more than one node with the same degree exist. In D,

i_x comes to the left of i_y if either the degree of i_x is larger than the degree of i_y or the two degrees are the same but $i_x < i_y$.

Thus, recalling that $\mathrm{K}(n)$ stands for the set of all permutations of $\langle 1, \ldots, n\rangle$, a permutation $\kappa = \langle k_1, \ldots, k_n\rangle \in \mathrm{K}(n)$ is to be interpreted as a sequence of n steps to assign colors to the nodes of G. If we let $node(k)$ denote the kth node in sequence D (i.e., $node(k) = i_k$), then the zth step in κ assigns to $node(k_z)$ the smallest color that has not been assigned to any of its neighbors in G. Of course, the outcome of such a sequence of steps depends on our choice on how to break symmetry; in Section 6, we comment on the possible effects of randomness and of choosing an altogether different reference sequence.

Each $\kappa \in \mathrm{K}(n)$ can in principle be regarded as a *program* to provide any graph on n nodes with a coloring (hence our earlier indication that this second formulation relates to genetic programming, even though individuals are in our case considerably less complex in structural terms). The goal is to evolve one such program that, on average, can use as small a number of colors as possible. However, the number of nonisomorphic graphs on n nodes grows so fast with n (over 165 billion graphs for n as small as 12 [31]) that it is not reasonable to expect acceptable performance over such a wide class of graphs of a program evolved within tolerable time bounds. On the other hand, it also seems improper to proceed as with the first formulation and evolve a program that is specific to a single graph, since now the very nature of our individuals calls for greater generality.

Our choice has been to evolve programs that are specific to graphs belonging to a certain class of graphs, all having in common not only the number of nodes but also the *density*, which is defined to be the ratio of the graph's number of edges to the maximum possible number of edges when there can only be at most one edge between any two nodes. We let p denote such density, which is then such that

$$p = \frac{2m}{n(n-1)}.$$

The class of graphs having n nodes and density p such that $p^- \leq p \leq p^+$ for $0 \leq p^- \leq p^+ \leq 1$ will be denoted by $\mathcal{G}(n, p^-, p^+)$.

We now describe how to assess the fitness of an individual, and how to perform crossover and do mutation and inversion.

Fitness evaluation. Let $\mathcal{T}$, henceforth referred to as the *training set*, be a randomly selected subset of $\mathcal{G}(n, p^-, p^+)$. The fitness of an individual $\kappa \in \mathrm{K}(n)$ is based on the average number of colors that individual requires over all members of $\mathcal{T}$. If for program κ we let $colors(\kappa, G)$ denote the number of colors required for κ to assign colors to the nodes of graph G, then the fitness of κ is the nonnegative rational $f_2(\kappa)$, given by

$$f_2(\kappa) = n - \frac{1}{|\mathcal{T}|} \sum_{G \in \mathcal{T}} colors(\kappa, G), \tag{5}$$

which can be computed in $O\big(\sum_{G\in\mathcal{T}} m_G\big)$ time, where m_G denotes the number of edges in $G \in \mathcal{T}$. Program κ is capable of coloring the nodes of all graphs in $\mathcal{T}$ with, on average, $n - f_2(\kappa)$ colors.

Crossover. The crossover of two parent programs $\kappa_1, \kappa_2 \in \mathrm{K}(n)$ to yield the two offspring $\kappa_1', \kappa_2' \in \mathrm{K}(n)$, given the crossover point z such that $1 \leq z < n$, works as follows. Let

$$\kappa_1 = \langle k_1, \ldots, k_n\rangle,$$

$$\kappa_2 = \langle \ell_1, \ldots, \ell_n\rangle,$$

$$\kappa'_1 = \langle k'_1, \ldots, k'_n \rangle,$$

and

$$\kappa'_2 = \langle \ell'_1, \ldots, \ell'_n \rangle.$$

Then $\langle k'_1, \ldots, k'_z \rangle = \langle k_1, \ldots, k_z \rangle$ and $\langle k'_{z+1}, \ldots, k'_n \rangle$ is the subsequence of $\langle \ell_1, \ldots, \ell_n \rangle$ comprising all the indices that are not already in $\langle k_1, \ldots, k_z \rangle$. As for κ'_2, $\langle \ell'_1, \ldots, \ell'_z \rangle = \langle \ell_1, \ldots, \ell_z \rangle$ and $\langle \ell'_{z+1}, \ldots, \ell'_n \rangle$ is the subsequence of $\langle k_1, \ldots, k_n \rangle$ comprising all the indices that are not already in $\langle \ell_1, \ldots, \ell_z \rangle$.

Inheritance in this case is easier to identify than in the formulation of Section 2. Program κ'_1 visits $node(k'_1), \ldots, node(k'_z)$ for coloring in the same relative order as program κ_1 and $node(k'_{z+1}), \ldots, node(k'_n)$ in the same relative order as κ_2. Likewise, program κ'_2 visits $node(\ell'_1), \ldots, node(\ell'_z)$ for coloring in the same relative order as program κ_2 and $node(\ell'_{z+1}), \ldots, node(\ell'_n)$ in the same relative order as κ_1.

Mutation. Single-locus mutations are meaningless on a program $\kappa \in \mathrm{K}(n)$, so what we call a mutation in this second formulation is really a swap of two of the indices in κ, which then acquire each other's positions in the sequence. For mutation points z and z', the effect of this operation on $\kappa = \langle k_1, \ldots, k_n \rangle$ is to yield the program $\kappa' = \langle k'_1, \ldots, k'_n \rangle$ with $k'_z = k_{z'}$ and $k'_{z'} = k_z$.

Inversion. For program $\kappa = \langle k_1, \ldots, k_n \rangle$ in $\mathrm{K}(n)$, the effect of inversion on κ to yield another program $\kappa' = \langle k'_1, \ldots, k'_n \rangle$ is to set $k'_z = k_{n-z+1}$ for $z = 1, \ldots, n$.

4. On the evolutionary operators

The evolutionary operators described in Sections 2 and 3 for use respectively in our first and second formulations of the graph coloring problem were selected with a variety of goals in mind, including meaningfulness in the context of the formulation at hand, parsimony in the final number of operators, and expected effectiveness (as far as this can be inferred from a few initial experiments on reasonably-sized instances of the problem). While it is possible to see relatively easily, as we argued in those sections, that the crossover operator is indeed in both formulations responsible for the transmission to offspring of structurally meaningful information from the parents, for the remaining operators (mutation and, in the case of the second formulation, inversion as well) it may seem that they only fulfill the elusive role of occasionally helping the search escape local optima [17, 32]. In this section, we argue that mutation, in both formulations, can be ascribed the more precise role of allowing the search to lead from any one individual to any other with nonzero probability (possibly with the help of inversion, in the case of the second formulation). Understanding this property, which holds trivially in the case of bit-string representations under single-locus mutations, for example, requires a little elaboration in the case of our two formulations, especially so for the first one.

We first discuss mutation in the context of Section 2. In this case, mutation on an acyclic orientation is the turning of an arbitrary node (the mutation point) into a source, which yields another acyclic orientation. For arbitrary $\omega, \omega' \in \Omega(G)$, the question that we answer affirmatively in this section is whether there exists a finite sequence of mutations that turns ω into ω'. The argument requires a little additional notation, as follows. Let $M \subseteq E$ be the set of edges that ω and ω' direct differently, and let $U \subseteq N$ be the set of end nodes of the edges in M. Likewise, let $M^+ \subseteq E$ be the set of all edges lying on directed paths according to ω that lead to nodes in U, and let $U^+ \subseteq N$ be the set of end nodes of the edges in M^+. Clearly, $U \subseteq U^+$ and $M \subseteq M^+$. The desired transformation of ω into ω' must reverse the directions assigned by ω to all edges in M while keeping all the remaining edges' directions untouched.

In order to describe a finite sequence of mutations that achieves this we need still more notation. For $k \in N$, let t_k be a nonnegative integer indicating the number of times node k must be turned into a source in such a sequence. We show that there exist an assignment of values to such integers and a corresponding sequence of mutations that always guarantee the transformation of ω into ω'. In this assignment, we first let $t_k = 0$ for every k such that $k \in N \setminus U^+$ (here, and henceforth, $\setminus$ is used to denote set difference). As for $k \in U^+$, we let values be assigned in such a way that

$$\begin{aligned} t_i &= t_j, \quad \text{if } (i,j) \notin M, \\ t_i &< t_j, \quad \text{if } (i,j) \in M \end{aligned} \tag{6}$$

for $(i,j) \in M^+$ directed by ω from i to j. What (6) is doing is to assign values that are strictly increasing as one traverses edges of M along the directions given by ω, but remain constant over the traversal of edges of $M^+ \setminus M$ also along the directions given by ω.

Note that such an assignment is always possible, owing to the fact that ω is acyclic, which implies that the strict inequalities appearing in (6) for members of M can always be satisfied. In particular, because values are nondecreasing along directed paths, it is a trivial matter to choose them in such a way that $t_k < |U^+|$ for all $k \in U^+$.

If $\{t_k; k \in U^+\}$ is a set of values that obeys (6), then the corresponding sequence of mutations that leads from ω to ω' is obtained by repeating the following step until $t_k = 0$ for all $k \in U^+$. If $\ell \in U^+$ is not currently a source and t_ℓ is nonzero and greatest over all of U^+, then turn ℓ into a source and set t_ℓ to $t_\ell - 1$; when two neighbors are tied, pick the one that is pointed to by the edge between them according to the current orientation. Note that, if every t_k is assigned the smallest possible value, then this sequence of mutations comprises $O\big(|U^+|^2\big)$ mutations overall.

Now let $e = (i,j)$ be any edge, and let the direction assigned to it by ω be from i to j. If e is not an edge of M^+, then clearly the process we just described does not affect it. If $e \in M^+ \setminus M$, then t_i and t_j have initially the same value, and therefore i and j get turned into sources the same number of times and alternately with respect to each other. Also, because e is directed toward j by ω, j is the first of the two nodes to be turned into a source and i is the last, so at the end the edge's direction settles as initially, that is, from i to j. For $e \in M$, initially it holds that $t_i < t_j$, so j is turned into a source strictly more times than i is. Two scenarios can be envisaged. In the first, the inequality $t_i < t_j$ is maintained throughout the process, so j is the last of the two nodes to be turned into a source. In the second scenario, t_i and t_j become equal before the end of the process, which must happen upon one of the turnings of j into a source, therefore leaving the edge directed toward i. From then on the two nodes become sources the same number of times and alternately, i being the first and j the last. In either scenario, edge e ends up directed from j to i. It follows from this that ω' is the orientation of the graph at the end of the sequence of mutations.

The case of the formulation of Section 3 is considerably simpler. For two programs $\kappa = \langle k_1, \ldots, k_n \rangle$ and $\kappa' = \langle k'_1, \ldots, k'_n \rangle$, the following is a sequence of mutations that leads from κ to κ'. For $z = 1, \ldots, n$, check whether $k'_z = k_z$; if not, then let $z' > z$ be such that $k'_z = k_{z'}$ and swap k_z with $k_{z'}$. Clearly, the resulting κ is the same as κ'. Also, fast though this transformation of κ into κ' is (no more than $n-1$ swaps are ever needed), it is easy to conceive of cases in which the use of inversion can make it even faster when applied as a first step.

5. The experimental test set

This section contains a brief description of the graphs to be used in Section 6 as benchmark graphs for

implementations of our first and second formulations. They have all been extracted from the DIMACS challenge suite [38]. In what follows, G is a graph of the type being described.

`DSJC`n`.`q [25]. These are random graphs on n nodes. They are constructed by independently joining any unordered pair of nodes by an edge with probability $q/10$. We use these graphs with $n \in \{125, 250, 500\}$ and $q = 5$.

`R`n`.`q. These are random graphs on n nodes. They are constructed by randomly placing the nodes inside a unit square and then connecting by an edge any two nodes that are no farther apart from each other than $q/10$. We use these graphs with $n \in \{125, 250\}$ and $q \in \{1, 5\}$.

`R`n`.`q`c`. These graphs are the *complements* of the `R`n`.`q graphs, that is, each one of them has exactly the edges that are absent from its counterpart. We use these graphs with $n \in \{125, 250\}$ and $q = 1$.

`DSJR`n`.`q [25]. The same as `R`n`.`q. We use $n = 500$ and $q = 1$.

`DSJR`n`.`q`c` [25]. The same as `R`n`.`q`c`. We use $n = 500$ and $q = 1$.

`flat`n`_`k`_`f [11]. These are *k-partite* random graphs on n nodes with density $p \approx 0.5$. Being k-partite means that the node set can be partitioned into k *independent sets*, that is, sets whose members are not neighbors; consequently, $\chi(G) \leq k$. By construction, all independent sets have as close to the same number of nodes as possible, and furthermore edges are distributed among pairs of independent sets as equitably as possible. In addition, given any two of the independent sets, no node in one of them is allowed more edges joining it to the other set than any other node in its own set (except for a number f of edges, known as a "flatness" parameter). We use these graphs with $n = 300$, $k \in \{20, 26, 28\}$, and $f = 0$.

`le`n`_`k`x` [29]. These are random graphs on n nodes with $\chi(G) = k$. They have numbers of edges such that p, the density, is such that $p \leq 0.25$, and are constructed by the creation of cliques of varying sizes in such a way that the pre-specified value k of $\chi(G)$ is not violated. We use these graphs with $n = 450$ and $k = 15$; `x` is one of `a`, `b`, `c`, or `d`, and is used to differentiate among instances.

`mulsol.i.1` [30]. This graph results from an instance of the problem of register allocation in compilers.

`school1-nsh` [30]. This graph results from the problem of constructing class schedules in such a way as to avoid conflicts. Because it comes from an actual class schedule, it is known that $\chi(G) \leq 14$.

We show in Table 1 all the graphs we employ in our experiments. For each graph G, the table displays the values of n, m, and p, as well as $\chi(G)$ (or an upper bound on it), when known from design characteristics. The table also contains the graph class of which it is a member that is targeted in our experiments by the implementation of our second formulation.

6. Experimental results

Henceforth, we let Evolve_AO and Evolve_P denote the algorithms resulting, respectively, from the evolutionary formulations of Sections 2 and 3. Evolve_AO works on a fixed graph G and seeks the acyclic orientation of G whose lengthiest directed path is shortest over $\Omega(G)$. Evolve_P, by contrast, searches for the program (a member of $\mathrm{K}(n)$) that colors each of the graphs in $\mathcal{G}(n, p^-, p^+)$ with as few colors as possible.

Table 1. Benchmark graphs

G	n	m	p	$\chi(G)$	Target class
R125.1	125	209	0.027		$\mathcal{G}(125, 0.02, 0.09)$
R125.5	125	3,838	0.495		$\mathcal{G}(125, 0.40, 0.99)$
DSJC125.5	125	3,891	0.502		$\mathcal{G}(125, 0.40, 0.99)$
R125.1c	125	7,501	0.968		$\mathcal{G}(125, 0.40, 0.99)$
mulsol.i.1	197	3,925	0.203		$\mathcal{G}(197, 0.10, 0.39)$
R250.1	250	867	0.028		$\mathcal{G}(250, 0.02, 0.09)$
R250.5	250	14,849	0.477		$\mathcal{G}(250, 0.40, 0.99)$
DSJC250.5	250	15,668	0.503		$\mathcal{G}(250, 0.40, 0.99)$
R250.1c	250	30,227	0.971		$\mathcal{G}(250, 0.40, 0.99)$
flat300_20_0	300	21,375	0.477	≤ 20	$\mathcal{G}(300, 0.40, 0.99)$
flat300_26_0	300	21,633	0.482	≤ 26	$\mathcal{G}(300, 0.40, 0.99)$
flat300_28_0	300	21,695	0.484	≤ 28	$\mathcal{G}(300, 0.40, 0.99)$
school1-nsh	352	14,612	0.237	≤ 14	$\mathcal{G}(352, 0.10, 0.39)$
le450_15a	450	8,168	0.081	15	$\mathcal{G}(450, 0.02, 0.09)$
le450_15b	450	8,169	0.081	15	$\mathcal{G}(450, 0.02, 0.09)$
le450_15c	450	16,680	0.165	15	$\mathcal{G}(450, 0.10, 0.39)$
le450_15d	450	16,750	0.166	15	$\mathcal{G}(450, 0.10, 0.39)$
DSJR500.1	500	3,555	0.028		$\mathcal{G}(500, 0.02, 0.09)$
DSJC500.5	500	62,624	0.502		$\mathcal{G}(500, 0.40, 0.99)$
DSJR500.1c	500	121,275	0.972		$\mathcal{G}(500, 0.40, 0.99)$

Both EVOLVE_AO and EVOLVE_P iterate for g generations, each one characterized by a population of fixed size s. Upon generating the last population, each algorithm outputs the best individual found during all the evolutionary search, denoted respectively by ω^* and κ^*. For $k > 1$, the kth population is generated from the $k-1$st population by first transferring the fs fittest individuals to the new population (this is an elitist first step, with $0 \leq f < 1$), and then repeatedly selecting individuals for application of the evolutionary operators. The resulting individuals are then added to the new population until it is filled. The first population is generated randomly, which can be achieved in a similar fashion by both algorithms by randomly creating s sequences of n integers. In each of these sequences, every element of $\{1, \ldots, n\}$ must appear exactly once. For EVOLVE_AO, one such sequence is identified with the linear representation $L(\omega)$ of some acyclic orientation ω; for EVOLVE_P, it is identified with a program directly.

As with the choice of evolutionary operators for both algorithms, choosing an appropriate selection method has relied on the outcome of some preliminary experiments on reasonably-sized graphs. For EVOLVE_AO, this has resulted in selecting individuals proportionally to its linearly normalized fitness in the current population. In other words, if for $k \in \{1, \ldots, s\}$ a certain individual ω has the kth smallest value of $f_1(\omega)$ over the entire population for $f_1(\omega)$ as in (4), then it is selected with probability proportional to

$$g_1(\omega) = 2k. \tag{7}$$

Ties in the order of fitness within the population are broken by the order in which individuals were added to the population in the first place.

The case of EVOLVE_P is similar, but because fitnesses are now averages taken over the members of a subclass of graphs $\mathcal{T}$ of $\mathcal{G}(n, p^-, p^+)$, there is a markedly increased tendency for individuals to be clustered very nearly one another according to their fitness values. The way we do selection in this case is then proportionally to a function that very steeply increases with $f_2(\kappa)$ for κ in the current population, $f_2(\kappa)$ being given as in (5). Specifically, we let κ be selected with probability proportional to

$$g_2(\kappa) = e^{f_2(\kappa)}. \tag{8}$$

The decision as to how to manipulate the individuals selected according to either (7) or (8) before inclusion in the new population is also reached probabilistically. The various probabilities involved are denoted as follows.

p_c: the probability of performing the crossover of two selected individuals;

p_m: the probability of performing mutation on a selected individual;

p_i: the probability of performing inversion on a selected individual.

In this section, we summarize the results of our experiments with EVOLVE_AO and EVOLVE_P on the benchmark graphs described in Section 5.[1] The summary we present is far from exhaustive with respect to the possible combinations of the parameters involved, but rather contains the combinations that yielded the best results we have obtained.

We present two types of performance figures. Figures of the first type reflect how well a graph or class of graphs is colored by a given individual. For EVOLVE_AO, which operates on fixed G, this performance figure is the number of colors needed to provide G with a coloring. During the evolutionary search, this number is given as $n - f_1(\omega^+)$, where ω^+ is the best individual of the current generation; likewise, once ω^* has been identified as the best individual found during the whole search, then it is given as $n - f_1(\omega^*)$. For EVOLVE_P, which aims at coloring all the graphs belonging to the class $\mathcal{G}(n, p^-, p^+)$ by the program κ^* that on average performs best on the training set $\mathcal{T} \subset \mathcal{G}(n, p^-, p^+)$, this first performance figure is presented under two guises. During the evolutionary search, it is presented as the average number of colors needed by the graphs in $\mathcal{T}$ under the best program, say κ^+, of the current generation, that is, $n - f_2(\kappa^+)$. After κ^* has been identified, then the performance figure is presented as $colors(\kappa^*, G)$ for each graph $G \in \mathcal{G}(n, p^-, p^+)$ that is in the benchmark set as introduced in Section 5.

Our second performance figure is related to how the quality of the best individual output by the search is affected by the number g of generations elapsed before completion and the number s of individuals of each generation. This figure is given directly by the combined values of g and s, and is to be thought of as a platform-independent assessment of each algorithm's efficiency.

This section also contains a comparative assessment of EVOLVE_AO and EVOLVE_P with respect to seven other approaches that have also been tested on the DIMACS benchmark graphs. These are the six approaches that appeared in the challenge's proceedings volume [26] and the one in [18], which appeared later. Because nearly all nine approaches are based on widely differing strategies and have been implemented

[1] All experiments were performed either on a Sun UltraSparc station running at 167 MHz or on machines based on Intel processors running at 450 MHz. The time elapsed during each of the reported runs of either algorithm varies greatly, depending on the machine used and of course on the graph, or graph class, under consideration. Typical figures range from a few minutes to several hours.

on an equally diverse set of platforms, our comparison is restricted to performance figures that indicate how well the benchmark graphs are colored. The following is a brief description of those seven approaches.

I_Greedy [11]. This is an iterated version of the greedy procedure that assigns colors to nodes following a certain order and choosing, for each node, the smallest color that still has not been assigned to any of its neighbors. At each iteration, a different order is chosen according to a criterion that involves the latest color assignment and also degree-related information, as in DSatur, among other indicators.

T_B&B [23]. This is a branch-and-bound algorithm that employs elements inspired in tabu search [22] for control. For each subproblem, it requires a *clique* (a subgraph whose nodes are all joined to one another by edges) of large size (number of nodes) to be found, its size being then used as a lower bound. The algorithm is good for either exact or heuristic graph coloring, depending on whether the large clique that is found can be guaranteed to be maximum (have maximum size), since it can be easily shown that $\chi(G) \geq \omega(G)$, where $\omega(G)$ is the size of the maximum clique of G [7]. Providing such a guarantee is of course as hard as coloring the graph [20, 27], but seems to become only tolerably burdensome as the graphs become sparser.

Dist [34]. This algorithm employs multiple sequential computers to work concurrently on several partial colorings of the graph (that is, assignments of colors to subsets of N). One of the computers maintains a pool of the best partial colorings so far output by the others, and continually submits such colorings to those computers to be extended by neighborhood search and possibly improved.

Par [30]. This is a combination of Dist with exhaustive search by parallel branch-and-bound. The two computations are started concurrently on a parallel machine, each one taking a certain number of processors. The processor that in Dist is responsible for maintaining a pool of partial colorings is now also fed by the exhaustive-search computation, and relays the best partial colorings found by one computation to the other.

E_B&B [36]. This is an exact branch-and-bound algorithm. Its key ingredients include finding a maximum clique to work as lower bound (cf. our earlier description of T_B&B) and a novel branching rule that, like DSatur, uses the nodes' saturation degrees in the decision.

T_Gen_1 [16]. This is a hybrid approach that employs tabu search inside a genetic algorithm. In this approach, an individual is a partition of the graph's nodes into color classes, that is, subsets of nodes all of which are to be assigned the same color. The generation of new individuals by crossover does not in principle guarantee the absence of edges joining nodes in the same color class, which is where tabu search comes in looking for the re-arrangement into color classes that requires the least number of classes.

T_Gen_2 [18]. This approach is entirely analogous to T_Gen_1, from which it differs mainly on how crossover is performed (by "partitioning" color classes, as opposed to "assigning" colors to nodes).

The plots in Figure 2 show the behavior of Evolve_AO, each plot corresponding to a run of the algorithm on one of the graphs in Table 1. They have all been obtained with $p_c = 0.60$, $p_m = 0.02$, $f = 0.70$, $g = 5{,}000$, and $s = 100$. For clarity of exposition, the algorithm's behavior prior to the hundredth generation is not shown (at the first generation, and shortly onward, the number of colors implied by the best individual tends to be inordinately high). Analogously, Figure 3 contains one plot for each of the eleven distinct target classes appearing in Table 1, each plot corresponding to a run of Evolve_P on randomly selected members of its target class. These plots have been obtained with parameters valued as in Table 2.

The plots in Figures 2 and 3 indicate that both EVOLVE_AO and EVOLVE_P exhibit the fitness-improvement behavior one expects of well formulated evolutionary computations, often very markedly so in the earliest generations. While for EVOLVE_AO this happens smoothly over time, for EVOLVE_P some plots are a little jagged, perhaps reflecting the greater difficulty of concomitantly improving the colorings of all graphs in the training set $\mathcal{T}$.

We show in Table 3 the best results obtained by EVOLVE_AO and EVOLVE_P side by side with those obtained by the competing algorithms we outlined earlier. For each of the graphs in Table 1, what Table 3 shows is the value of its chromatic number (or an upper bound on it), when known by design, and the colorings obtained by each of the algorithms to which the graph was presented. For some algorithms this is given as an average over a certain number of runs (10 for I_GREEDY, 5 for PAR, between 3 and 10 for T_GEN_1, and 5 for EVOLVE_AO), for some others as the result of a single run (T_B&B, E_B&B, and EVOLVE_P). Yet for other algorithms (DIST and T_GEN_2), what appears in Table 3 is the lowest value of k for which a coloring by k colors was obtained for the corresponding graph, since those algorithms were tested by successively decreasing the value of k from some initial value. The entries in Table 3 corresponding to all algorithms but our own are either from [26] or, in the case of T_GEN_2, from [18].

While none of the heuristics appearing in Table 3 can be proclaimed strictly best over all graphs, when we examine each graph individually many of the heuristics can be said to do as well as the one that does best on that graph. That is also the case with EVOLVE_AO and EVOLVE_P, which on several graphs (those for which performance figures are given in bold typeface) performed at least as well as the best performers on those same graphs, occasionally even better. For the other graphs, often EVOLVE_AO and EVOLVE_P miss the best performers very narrowly.

7. Concluding remarks

We have in this paper introduced two novel evolutionary formulations of the graph coloring problem. The first formulation views the problem of finding a graph's chromatic number as the problem of finding an acyclic orientation of the graph according to which the longest directed path is shortest among all acyclic orientations. Viewing the problem from this perspective immediately provides the essential foundation for the evolutionary formulation, since each acyclic orientation can be viewed as an individual whose fitness can be said to be higher as its longest directed path is shorter. Despite this initial simplicity, the design of appropriate evolutionary operators has relied on sophisticated graph-theoretic arguments.

Our second formulation takes an entirely different route, and seeks to find a program that will color any graph having a pre-specified number of nodes and density within a pre-specified interval. This program is a permutation of indices into a sequence of nodes that is previously agreed upon. The sequence we have used contains nodes in nonincreasing order of degrees, having been loosely inspired by the DSatur heuristic. But being only a reference sequence, any other sequence could have been used as well. A program is then something like "first color the node whose degree is the third highest with the lowest available color, then the node whose degree is the seventh highest, etc.," for example. In this formulation, each individual is a program, its fitness being higher as the average number of colors it requires to color all the graphs in a randomly selected subset of graphs having that number of nodes and density in that interval is smaller.

We have presented experimental results on some of the DIMACS benchmark graphs and compared our algorithms' performances with those of the other heuristics that we know to have been tested on those graphs as well. Our results indicate generally competitive performance, sometimes reaching the best results obtained by the other heuristics, occasionally better.

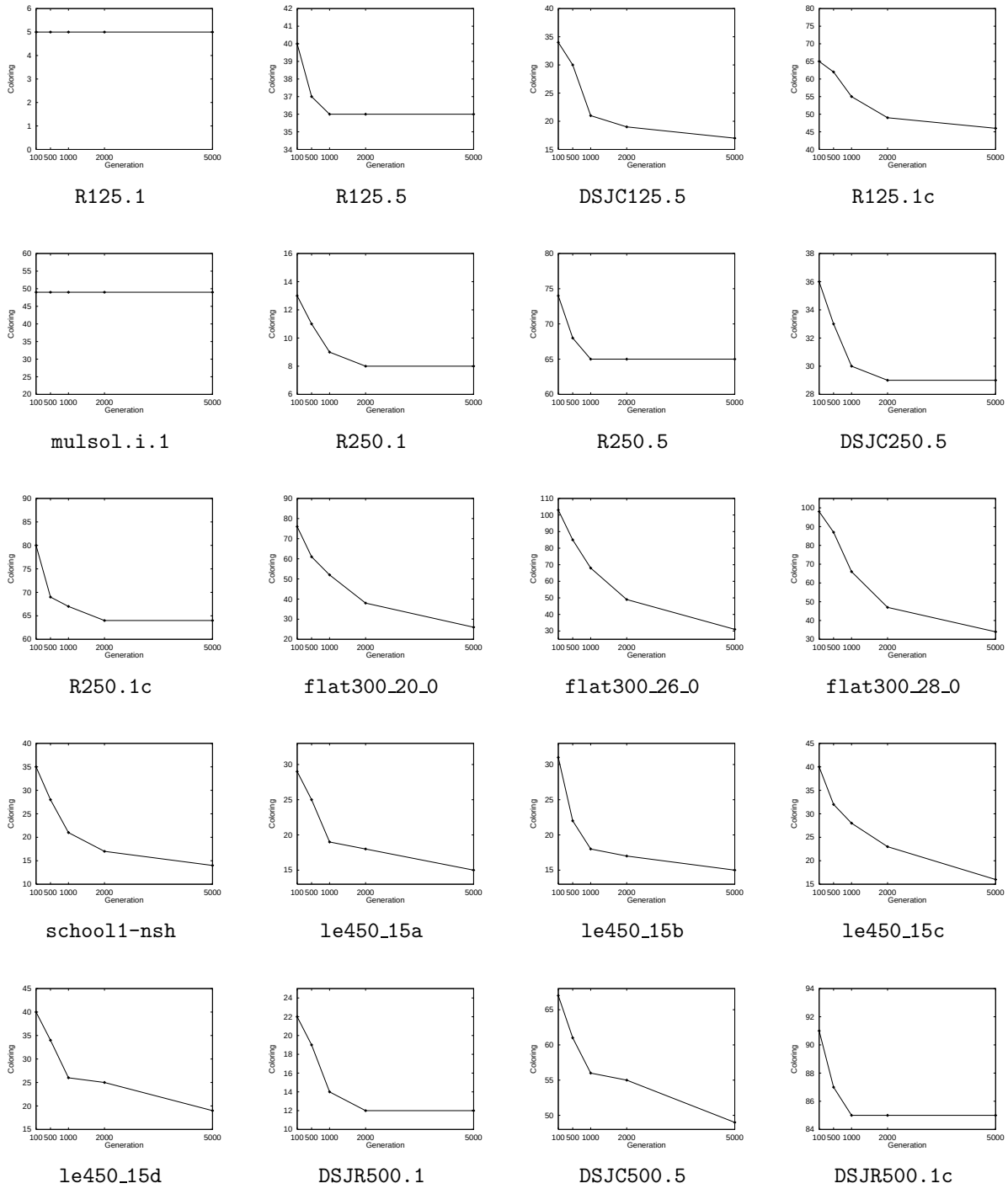

Figure 2. Convergence of EVOLVE_AO on the graphs of Table 1

Notwithstanding these positive results, there is room, at least in principle, for improvements to be attempted. For example, in more than one occasion we have resorted to deterministic decisions to break symmetry, so it may be worth checking whether introducing randomness at these points can have any noticeable effect. Also, our two formulations are both purely evolutionary, in the sense that at no point do

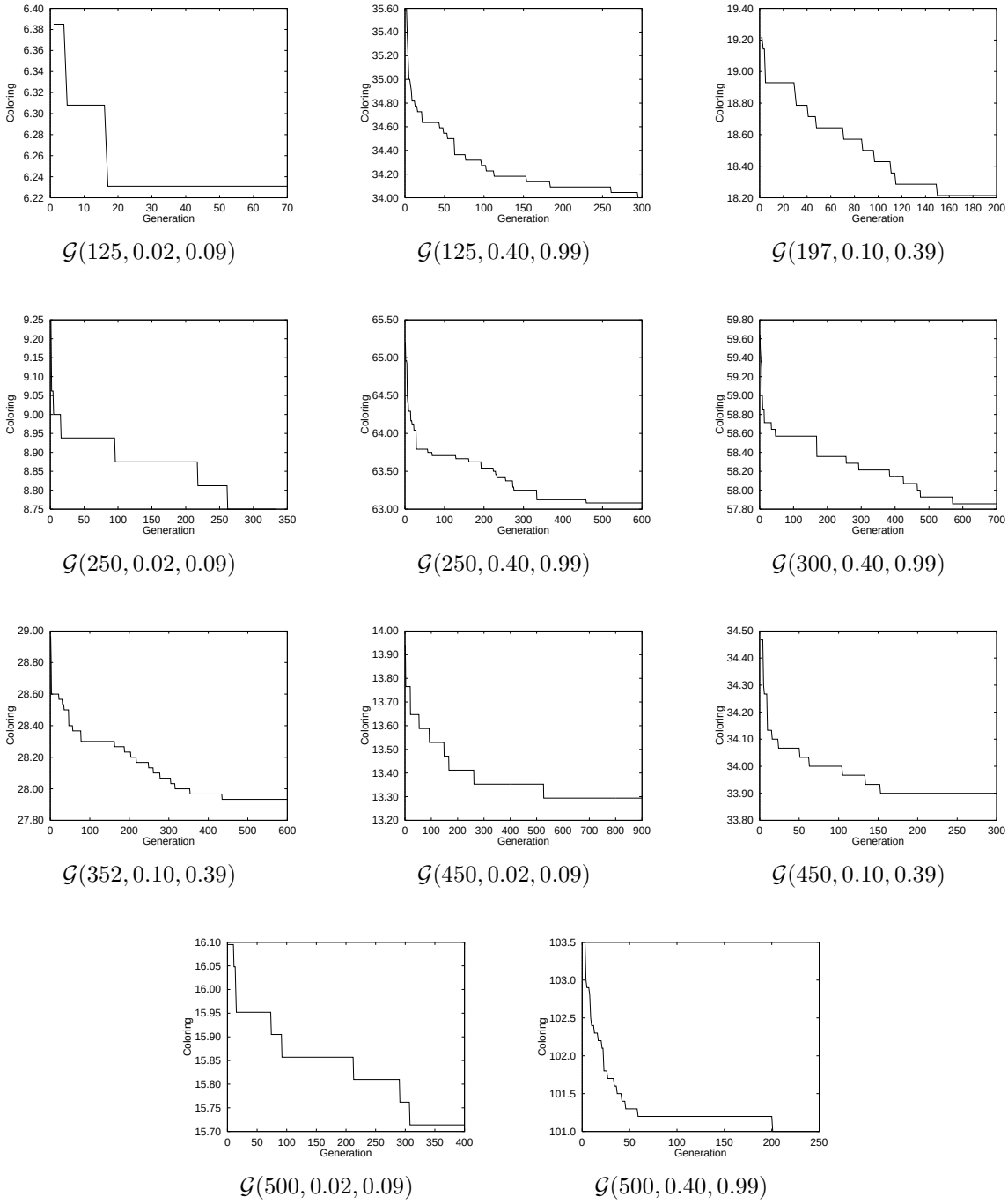

Figure 3. Convergence of EVOLVE_P on the target classes of Table 1

they employ auxiliary heuristics, like for example some form of local search. We think it may be worthwhile to investigate such hybrid alternatives.

It is, however, in the context of EVOLVE_P, the algorithm that implements our second formulation, that we believe the possibilities for extension and improvement are the most challenging and interesting. For example, of all the graphs that were presented to EVOLVE_P for coloring, the ones that prompted the least

Table 2. Parameters for the experiments with EVOLVE_P

Target class	$\lvert\mathcal{T}\rvert$	p_c	p_m	p_i	f	g	s
$\mathcal{G}(125, 0.02, 0.09)$	40	0.71	0.01	0.28	0.02	70	200
$\mathcal{G}(125, 0.40, 0.99)$	60	0.71	0.01	0.28	0.02	300	100
$\mathcal{G}(197, 0.10, 0.39)$	60	0.71	0.01	0.28	0.01	200	100
$\mathcal{G}(250, 0.02, 0.09)$	60	0.71	0.01	0.28	0.01	350	100
$\mathcal{G}(250, 0.40, 0.99)$	60	0.71	0.01	0.28	0.02	600	100
$\mathcal{G}(300, 0.40, 0.99)$	60	0.71	0.01	0.28	0.01	700	100
$\mathcal{G}(352, 0.10, 0.39)$	60	0.71	0.01	0.28	0.01	600	100
$\mathcal{G}(450, 0.02, 0.09)$	100	0.71	0.01	0.28	0.02	900	100
$\mathcal{G}(450, 0.10, 0.39)$	100	0.70	0.01	0.29	0.02	300	100
$\mathcal{G}(500, 0.02, 0.09)$	40	0.70	0.01	0.29	0.02	400	100
$\mathcal{G}(500, 0.40, 0.99)$	40	0.71	0.01	0.28	0.02	250	200

Table 3. Comparative performance on the benchmark graphs

G	$\chi(G)$	I_GREEDY	T_B&B	DIST	PAR	E_B&B
R125.1		5.0	5	5	5.0	5
R125.5		36.9	36	36	37.0	36
DSJC125.5		18.9	20	17	17.0	
R125.1c		46.0	46	46	46.0	46
mulsol.i.1		49.0	49	49	49.0	49
R250.1		8.0	8	8	8.0	8
R250.5		68.4	66	65	66.0	65
DSJC250.5		32.8	35	28	29.2	
R250.1c		64.0	65	64	64.0	64
flat300_20_0	≤ 20	20.2	39	20	20.0	
flat300_26_0	≤ 26	37.1	41	26	32.4	
flat300_28_0	≤ 28	37.0	41	31	33.0	
school1-nsh	≤ 14	14.1	26	20	14.0	
le450_15a	15	17.9	16	15	15.0	
le450_15b	15	17.9	15	15	15.0	15
le450_15c	15	25.6	23	15	16.6	
le450_15d	15	25.8	23	15	16.8	
DSJR500.1		12.0	12	12	12.0	12
DSJC500.5		58.6	65	49	53.0	
DSJR500.1c		85.0	87	85	85.3	

satisfactory results were those whose nodes have degrees very closely concentrated near the average, that is, graphs with very little variance of node degree (this cannot be inferred from the data we have presented, but

Table 3 (continued)

G	$\chi(G)$	T_Gen_1	T_Gen_2	Evolve_AO	Evolve_P
R125.1		5.0		**5.0**	**5**
R125.5		35.6		36.2	36
DSJC125.5		17.0		17.2	20
R125.1c		46.0		**46.0**	**46**
mulsol.i.1		49.0		**49.0**	**49**
R250.1		8.0		**8.0**	**8**
R250.5		69.0		65.2	**65**
DSJC250.5		29.0	28	29.1	29
R250.1c		64.0		**64.0**	**64**
flat300_20_0	≤ 20	20.0		26.0	23
flat300_26_0	≤ 26	26.0		31.0	28
flat300_28_0	≤ 28	33.0	31	33.0	**29**
school1-nsh	≤ 14	14.0		**14.0**	20
le450_15a	15	15.0		**15.0**	17
le450_15b	15	15.0		**15.0**	17
le450_15c	15	16.0	15	16.0	25
le450_15d	15	16.0		19.0	25
DSJR500.1		12.0		**12.0**	**12**
DSJC500.5		51.0	48	52.5	59
DSJR500.1c		85.3		**85.0**	**85**

is clear from a closer examination of the DIMACS files). We think this may be due to the fact that perhaps this quantity was poorly represented by the graphs of the training set. Because Evolve_P is strongly based on degrees and how they relate to one another inside the graph, it is possible that targeting the evolution not just at a certain number of nodes and a certain density interval, but also at a certain interval of node-degree variance, might yield significant improvements in some cases. This would, of course, call for a generator of random graphs capable of controlling such variances in addition to densities, which is to our knowledge also a topic for research.

Acknowledgments

The authors acknowledge partial support from CNPq, CAPES, the PRONEX initiative of Brazil's MCT under contract 41.96.0857.00, and a FAPERJ BBP grant.

References

1. S. Arora and C. Lund, "Hardness of approximations," in D. S. Hochbaum (Ed.), *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, Boston, MA, 1997, 399–446.

2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and Approximation*, Springer-Verlag, Berlin, Germany, 1999.

3. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann Publishers, San Francisco, CA, 1998.

4. V. C. Barbosa, *An Atlas of Edge-Reversal Dynamics*, Chapman & Hall/CRC, London, UK, 2000.

5. V. C. Barbosa and E. Gafni, "A distributed implementation of simulated annealing," *J. of Parallel and Distributed Computing* **6** (1989), 411–434.

6. M. Bellare, O. Goldreich, and M. Sudan, "Free bits, PCPs, and nonapproximability—towards tight results," *SIAM J. on Computing* **27** (1998), 804–915.

7. J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland, New York, NY, 1976.

8. D. Brélaz, "New methods to color vertices of a graph," *Comm. of the ACM* **22** (1979), 251–256.

9. F. C. Chow and J. L. Hennessy, "The priority-based coloring approach to register allocation," *ACM Trans. on Programming Languages and Systems* **12** (1990), 501–536.

10. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*, The MIT Press, Cambridge, MA, 2001.

11. J. C. Culberson and F. Luo, "Exploring the k-colorable landscape with Iterated Greedy," in [26], 245–284.

12. D. de Werra, "An introduction to timetabling," *European J. of Operational Research* **19** (1985), 151–162.

13. R. W. Deming, "Acyclic orientations of a graph and chromatic and independence numbers," *J. of Combinatorial Theory B* **26** (1979), 101–110.

14. A. E. Eiben, J. K. van der Hauw, and J. I. van Hemert, "Graph coloring with adaptive evolutionary algorithms," *J. of Heuristics* **4** (1998), 25–46.

15. C. Fleurent and J. Ferland, "Genetic and hybrid algorithms for graph coloring," *Annals of Operations Research* **63** (1996), 437–461.

16. C. Fleurent and J. A. Ferland, "Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability," in [26], 619–652.

17. D. B. Fogel, *Evolutionary Computation*, IEEE Press, New York, NY, 1995.

18. P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *J. of Combinatorial Optimization* **3** (1999), 379–397.

19. A. Gamst, "Some lower bounds for a class of frequency assignment problems," *IEEE Trans. on Vehicular Technology* **35** (1986), 8–14.

20. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, NY, 1979.

21. M. R. Garey, D. S. Johnson, and H. C. So, "An application of graph coloring to printed circuit testing," *IEEE Trans. on Circuits and Systems* **CAS-23** (1976), 591–599.

22. F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, MA, 1997.

23. F. Glover, M. Parker, and J. Ryan, "Coloring by tabu branch and bound," in [26], 285–307.

24. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Publishing Company, Reading, MA, 1989.

25. D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning," *Operations Research* **39** (1991), 378–406.

26. D. S. Johnson and M. A. Trick (Eds.), *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, American Mathematical Society, Providence, RI, 1996.

27. R. M. Karp, "Reducibility among combinatorial problems," in R. E. Miller and J. W. Thatcher (Eds.), *Complexity of Computer Computations*, Plenum Press, New York, NY, 1972, 85–103.

28. J. R. Koza, *Genetic Programming*, The MIT Press, Cambridge, MA, 1992.

29. F. T. Leighton, "A graph coloring algorithm for large scheduling problems," *J. of Research of the National Bureau of Standards* **84** (1979), 489–505.

30. G. Lewandowski and A. Condon, "Experiments with parallel graph coloring heuristics and applications of graph coloring," in [26], 309–334.

31. B. D. McKay, "Isomorph-free exhaustive generation," *J. of Algorithms* **26** (1998), 306–324.

32. Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, Berlin, Germany, 2000.

33. M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, Cambridge, MA, 1996.

34. C. Morgenstern, "Distributed coloration neighborhood search," in [26], 335–357.

35. Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, Boston, MA, 1996.

36. E. C. Sewell, "An improved algorithm for exact graph coloring," in [26], 359–373.

37. R. P. Stanley, "Acyclic orientations of graphs," *Discrete Mathematics* **5** (1973), 171–178.

38. M. A. Trick, "Appendix: Second DIMACS Challenge test problems," in [26], 653–657.