Comput Optim Appl (2013) 54:741–770
DOI 10.1007/s10589-012-9496-5

Graph 3-coloring with a hybrid self-adaptive evolutionary algorithm

Iztok Fister · Marjan Mernik · Bogdan Filipič

Received: 19 March 2011 / Published online: 28 June 2012

Abstract This paper proposes a hybrid self-adaptive evolutionary algorithm for graph coloring that is hybridized with the following novel elements: heuristic genotype-phenotype mapping, a swap local search heuristic, and a neutral survivor selection operator. This algorithm was compared with the evolutionary algorithm with the SAW method of Eiben et al., the Tabucol algorithm of Hertz and de Werra, and the hybrid evolutionary algorithm of Galinier and Hao. The performance of these algorithms were tested on a test suite consisting of randomly generated 3-colorable graphs of various structural features, such as graph size, type, edge density, and variability in sizes of color classes. Furthermore, the test graphs were generated including the phase transition where the graphs are hard to color. The purpose of the extensive experimental work was threefold: to investigate the behavior of the tested algorithms in the phase transition, to identify what impact hybridization with the DSatur traditional heuristic has on the evolutionary algorithm, and to show how graph structural features influence the performance of the graph-coloring algorithms. The results indicate that the performance of the hybrid self-adaptive evolutionary algorithm is comparable with, or better than, the performance of the hybrid evolutionary algorithm which is one of the best graph-coloring algorithms today. Moreover, the fact that all the considered algorithms performed poorly on flat graphs confirms that graphs of this type are really the hardest to color.

I. Fister (✉) · M. Mernik
Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ulica 17, 2000 Maribor, Slovenia
e-mail: iztok.fister@uni-mb.si

M. Mernik
e-mail: marjan.mernik@uni-mb.si

B. Filipič
Department of Intelligent Systems, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
e-mail: bogdan.filipic@ijs.si

Keywords Graph 3-coloring · Constraint satisfaction · Local search · Evolutionary algorithms · Self-adaptation

1 Introduction

The graph-coloring problem (GCP) is a well-known combinatorial optimization problem that has many practical applications. For example, it can be applied in register allocation in compilers [19, 23], timetabling [12, 22], frequency assignment in cellular networks [37, 55, 62], scheduling [36, 53], printed circuit board testing [39], and manufacturing [40]. In addition to its practical relevance, the problem represents a test bed for new algorithms primarily because of its simple definition.

The problem can be defined informally as follows: In a given undirected graph $G = (V, E)$, where V denotes a finite set of vertices and E a finite set of unordered pairs of vertices called edges, the vertices have to be colored using k colors such that no two vertices connected with an edge are of the same color. Such a coloring, if it exists, is named the *proper* k-coloring. The minimum number of colors k necessary to color the graph G is called the *chromatic number* χ [52]. The graph is k-colorable if it has a k-coloring. Searching for a proper graph k-coloring is denoted as k-GCP. The decision form of this problem, where the question is whether a particular graph is k-colorable, is *NP*-complete [38], while the problem of finding the chromatic number of graph G is *NP*-hard [9].

The first algorithms for graph coloring tried to solve this problem exactly [11], i.e., by enumerating all possible orderings of the vertices. However, these algorithms were too time consuming and infeasible for large graphs, therefore, many approaches that tackled the problem heuristically were proposed. They tried to find an approximate solution in reasonable time. The most natural approach to coloring graph vertices heuristically was the greedy approach [9]. The best known heuristics of this type are the *largest saturation degree* heuristic DSatur [10], and the *recursive largest first* heuristic RLF [53]. Before coloring, DSatur orders graph vertices according to the *saturation degree* ρ_v that is defined as the number of distinctly colored vertices adjacent to vertex v [52]. On the other hand, RLF divides the uncolored graph into color classes that contain vertices colored with the same color. The algorithm colors the vertices one color class at a time. The vertices from the uncolored subgraph are added to the current color class in turn so that the number of edges left in the uncolored subgraph remains as few as possible.

Some of the most popular algorithms for solving k-GCP today are metaheuristics based on local search [6, 7]. One of the first metaheuristics was developed by Hertz and de Werra [44] under the name Tabucol. This was the first application of the tabu search [41] to graph coloring. Tabucol first generates an initial random k-coloring, which typically contains a large number of conflicting edges. Then, the heuristic iteratively looks for a single vertex that most decreases the number of conflicting edges when it is recolored with another color, i.e., moved to another color class. A tabu list prevents the moves from cycling. Proper k-coloring may be obtained after a finite number of iterations. Later, Tabucol was improved to more sophisticated graph-coloring algorithms [25, 33, 60].

On the other hand, other local search heuristics include simulated annealing [13, 49], iterative local search [15, 17], reactive partial tabu search [4, 5, 56], variable neighborhood search [2], adaptive memory [35], variable search space [45], and population-based methods [25, 30, 33, 54]. One of the best population-based algorithms for k-GCP, the Hybrid Evolutionary Algorithm (HEA) developed by Galiner and Hao [33] combines local search with the partition-based crossover operator. The Tabucol metaheuristic is used as a local search operator. For a comprehensive survey of the main methods, see, e.g., [34, 57].

In general, evolutionary algorithms [59] that operate on a population of solutions are good optimizers, but suffer from a lack of constraint handling abilities. This lack arises from the fact that the variation operators, i.e., crossover and mutation, tend to violate the constraints [27]. As a result, offspring generated from parents by these operators can be infeasible. Indirect constraint handling can be used to overcome this problem. In this case, constraints are transformed into optimization objectives that can be expressed by an objective function. This function penalizes the candidate solutions violating the constraints. A solution to the problem is found when all constraints are satisfied. The value of the objective function in this case decreases to zero.

GCP is a constraint optimization problem where two vertices connected with an edge cannot be colored with the same color. However, the graph vertices have different numbers of edges. The number of edges incident to vertex v is called the *vertex degree* $\deg_G(v)$ [9]. Regarding the vertex degrees, it may be harder to color vertices with higher rather than lower degree. This dependence can be expressed by an objective function defined as the sum of the weights assigned to the vertices violating the constraints. The higher the weight of a vertex, the more difficult the vertex is to color. The manner in which the weights are changed has a great influence on the performance of a graph-coloring algorithm. Weights can be changed deterministically, adaptively or self-adaptively [26].

This paper focuses on graph 3-coloring (3-GCP) that is a special case of k-GCP, where $k = 3$. It proposes a self-adaptive evolutionary algorithm for 3-GCP, hybridized with:

- heuristic genotype-phenotype mapping (construction of solutions),
- a local search heuristic,
- a neutral survivor selection operator.

3-GCP is often the subject of research because of the lowest k where k-GCP makes any sense. This work was motivated by the paper of Eiben et al. [28], which was our starting point for 3-GCP. In their paper, Eiben et al. describe an evolutionary algorithm with Stepwise Adaptation of Weights (SAW), denoted as SAW-EA. Solutions in this evolutionary algorithm are represented as permutations of vertices and corresponding weights that denote the hardness of the vertices to be colored. This hardness is expressed by the penalty reward reflected in the objective function. In order to minimize this objective function, the graph-coloring algorithm (greedy algorithm) directs itself to first color the vertices with higher values of weights. Note that here the weights are modified adaptively. A slightly different approach to graph 3-coloring with differential evolution and DSatur graph-coloring algorithm can be found in Fister et al. [29].

The results of the proposed self-adaptive evolutionary algorithm were compared with the results of three graph-coloring algorithms: SAW-EA [66], Tabucol [16] and HEA [16]. Extensive experiments using these algorithms were conducted on a collection of random 3-colorable graphs generated by the Culbersone graph generator [20]. The test suite used in the experiments was the same as in [28] because the DIMACS challenge suite [50], which was designed for testing graph k-coloring algorithms, does not contain 3-colorable graphs. Our test suite satisfies two conditions:

– it comprises the instances of the random graphs in the phase transition [65], where the graphs are really hard to color,
– it enables the determination of how various structural features of the random graphs, e.g., the graph size and type, the edge density, and the variability in sizes of color classes, influence the performance of the tested graph-coloring algorithms.

In summary, the hybrid self-adaptive evolutionary algorithm incorporates three novelties: adaptation of weights representing the coefficients of the objective function in the DSatur algorithm, a new local search heuristic, and a new operator of the neutral survivor selection. Although this approach recalls SAW-EA, at least two differences can be exposed. The SAW method adapts the coefficients of the objective function in the greedy coloring heuristic (in contrast to the DSatur heuristic) and is adaptive (in contrast to self-adaptation). Furthermore, while Igel & Taussaint [48] emphasize the significance of neutral mutation in evolution strategies, the proposed approach exploits this finding explicitly by creating the neutral survivor selection operator. The contribution of this evolutionary algorithm to the original DSatur algorithm was evaluated during the experimental work.

The structure of the rest of this paper is as follows. In Sect. 2 the problem of graph 3-coloring is formally defined. Section 3 describes the structure of the hybrid self-adaptive evolutionary algorithm in detail. Section 4 presents experiments and results on various classes of random graphs. Section 5 concludes the paper by summarizing the work, the experimental results, and ideas for further work.

2 Graph 3-coloring

Let us assume an undirected graph $G = (V, E)$, where V represents the finite set of vertices $v_i \in V$ and E the finite set of unordered pairs $e = \{v_i, v_j\}$ named edges for $i = 1, \ldots, n \wedge j = 1, \ldots, n \wedge i \neq j$, where, $n = |V|$ denotes the number of vertices. A graph 3-coloring is a mapping $c : V \to C$, where $C = \{1, 2, 3\}$ denotes a set of 3-colors. In other words, one of the three colors is assigned to each vertex of the graph G. A proper 3-coloring is obtained if no two adjacent vertices are assigned the same color [9].

3-GCP is a well-known *constraint satisfaction problem* (CSP) [27] convenient for solving with evolutionary algorithms. CSP can be represented as a pair $\langle S, \phi \rangle$, where $S = C^n$ with $C = \{1, 2, 3\}$ denoting a *search space*, in which all solutions $\mathbf{s} = \langle s_1, \ldots, s_n \rangle \in S$ are *feasible* and ϕ a *feasibility condition*, i.e., a Boolean function on S, that is composed of constraints belonging to edges. That is, for each edge $e \in E$ the corresponding constraint b_e is defined by $b_e(\langle s_1, \ldots, s_n \rangle) = \mathit{true}$ if and only if $e = \{v_i, v_j\}$ and $s_i \neq s_j$. Let $B^i = \{b_e | e = \{v_i, v_j\} \wedge j = 1, \ldots, m\}$ be the set of constraints involving variable v_i (edges connecting to node i). Then the feasibility condition is the conjunction of all constraints $\phi(\mathbf{s}) = \wedge_{v \in V} B^i(\mathbf{s})$. Note that the feasibility condition divides the search space into *feasible* and *infeasible regions*.

Constraints are usually handled in evolutionary algorithms indirectly through a penalty function that transforms the CSP into the *free optimization problem* (FOP) [27]. Thus, those infeasible solutions that are far away from the feasible region are punished with higher penalties. The penalty function for 3-GCP that can also be used as an objective function is

$$f(\mathbf{s}) = \sum_{i=1}^{n} \psi(\mathbf{s}, B^i), \tag{1}$$

where the function $\psi(\mathbf{s}, B^i)$ is defined as

$$\psi(\mathbf{s}, B^i) = \begin{cases} 1 & \text{if } \mathbf{s} \text{ violates at least one } b_e \in B^i, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The penalty function in Eq. (1) transforms the constraint satisfaction problem into a free optimization problem such that for each $\mathbf{s} \in S$ we have $\phi(\mathbf{s}) = true$ if and only if $f(\mathbf{s}) = 0$. Equation (2) represents the feasibility condition and estimates the quality of candidate solution $\mathbf{s}$. In fact, Eq. (1) counts the number of vertices that violate the constraints as expressed by Eq. (2).

3 The hybrid self-adaptive evolutionary algorithm

The hybrid self-adaptive evolutionary algorithm (HSA-EA) for graph 3-coloring integrates concepts from various problem solving methods. As a base, the self-adaptive evolution strategy [3] is used and then hybridized with heuristic genotype-phenotype mapping, a local search heuristic, and the neutral survivor selection.

3.1 The algorithms outline

HSA-EA involves the following components and features:

- representation of individuals,
- evaluation of objective function,
- population model,
- parent selection,
- mutation operator,
- survivor selection,
- initialization procedure and
- termination condition.

In this section these components and features are presented in detail.

In HSA-EA, an individual $\mathbf{y}^{(\mathbf{t})}$ consists of problem variables $y_1^{(t)}, \ldots, y_n^{(t)}$ encoding the values of weights that determine the initial order in which the vertices are colored, and the control variables $q_1^{(t)}, \dots, q_n^{(t)}$ denoting the mutation strengths that are used by the operator of normally distributed mutation [3]. The number of variables n is equivalent to the number of graph vertices. The problem variables can take the values from the domain $y_i^{(t)} \in Y$, where $Y \in [0.1, 1]$. Note that weights cannot reach the value of zero because this value would imply that no constraints exist for the coloring of the corresponding vertex. The control variables can take the values $q_i^{(t)} \in [\epsilon_0, 1]$, where the constant ϵ_0 is a predefined minimum value of $q_i^{(t)}$.

The evaluation of the objective function value is based on Eq. (1) that counts the number of uncolored vertices. Note that this function admits an occurrence of neutral solutions as well [51]. HSA-EA for graph 3-coloring uses the generational population model (μ, λ), where the whole population is replaced in each generation. Specifically, μ selected parents produce λ offspring and the ratio $\mu/\lambda = 7$ is used, as suggested in [27]. An additional individual that represents the best solution found so far is added to the population of μ members. This solution is called the reference solution $\mathbf{y}^*$. The tournament selection is used as the parent selection operator and relies on the ordering relation that can rank any k individuals [27]. Here, k represents the tournament size.

In self-adaptation, most frequently a normally distributed mutation [3], is used that changes the weights as follows:

$$q_i^{(t+1)} = q_i^{(t)} \cdot \exp\bigl(\tau' \cdot N(0,1) + \tau \cdot N_i(0,1)\bigr), \tag{3}$$

and

$$y_i^{(t+1)} = y_i^{(t)} + q_i^{(t+1)} \cdot N_i(0,1). \tag{4}$$

The mutation described in (3) and (4) is also called uncorrelated mutation with n step sizes [27]. In addition to the weights $y_i^{(t)}$ of dimension n, this kind of mutation requires the mutation strengths $q_i^{(t)}$ of the same dimension. The mutation strengths $q_i^{(t)}$ determine an area around the particular weight $y_i^{(t)}$ in which the search process could progress. The parameters $\tau \propto 1/\sqrt{2 \cdot \sqrt{n}}$ and $\tau' \propto 1/\sqrt{2 \cdot n}$ designate the learning rate [3]. The values for mutation strengths can be reduced to zero by the multiplication process in (4). In that case, the evolutionary process stagnates. The following condition needs to be considered to prevent this event:

$$q_i^{(t+1)} < \epsilon_0 \Rightarrow q_i^{(t+1)} = \epsilon_0. \tag{5}$$

Uncorrelated normally distributed mutation with n step sizes can be described as a multiplicative process that decreases mutation strengths over generations. The mutation strengths determine how big change of weights can be made. The proper setting of the strengths has a great influence on the exploration of the search space. On the other hand, the parameter value depends on the number of generations. The higher the number of generations, the bigger the required initial mutation strengths.

The neutral survivor selection operator was developed that represents the hybridization of this evolutionary algorithm and is presented in Sect. 3.4. The solutions within the population are initialized randomly except for the first solution. For initialization of this solution, a heuristic initialization procedure is used that is explained latter in this section. The algorithm terminates when the maximum number of objective function evaluations is reached or a proper graph 3-coloring is found.

3.2 Hybrid genotype-phenotype mapping

Usually, the original problem context in evolutionary algorithm is distinguished from the search space, where the evolutionary process takes place [27]. Candidate solutions in the original problem space are referred to as phenotypes, while their encodings form the genotypes and represent points in the search space. That is, candidate solutions need to be decoded from their representation, i.e., mapped to the phenotype before they are evaluated. This mapping is also known as a genotype-phenotype mapping. Note that genotype-phenotype mapping is not injective because several genotypes can be mapped into the same phenotype. In line with this, a lot of neutral solutions can appear. Actually, the phenotypes only depend on problem variables, while control parameters determine the manner in which the genotype space is explored. It can be said that the control parameters describe the strategy for exploring the genotype space and can direct the evolutionary search to new undiscovered regions of the search space.

This genotype-phenotype mapping consists of two steps: transformation of weights to the permutation of vertices and decoding of the solution by the DSatur algorithm. In the former step, from the encoded values of weights, i.e., tuple $\mathbf{y_i^{(t)}} = \langle y_{i,1}^{(t)}, \ldots, y_{i,n}^{(t)}, q_{i,1}^{(t)}, \ldots, q_{i,n}^{(t)} \rangle$, $i = 1, \ldots, \mu$, where μ denotes the population size, the initial permutation of the vertices is built $\mathbf{v_i^{(t)}} = \{v_{i,j}^{(t)}\}$ for $j = 1, \ldots, n$. The permutation determines the ordering in which the vertices are colored. That is, the vertices are ordered according to the descending values of the corresponding weights. Note that the mutation steps are not used in this transformation. In the latter step, the DSatur algorithm is taken as the construction heuristic by HSA-EA that from the permutation of vertices decodes a coloring $\mathbf{s_i^{(t)}} = \{s_{i,j}^{(t)}\}$, where $s_{i,j}^{(t)} \in \{1, 2, 3\}$. The order of coloring is determined by the DSatur heuristic, as follows:

1. The heuristic selects the vertex with the highest saturation, and colors it with the lowest of the three colors.
2. In the case of a tie, the heuristic selects the vertex with the maximal weight.
3. In the case of a tie, the heuristic selects a vertex randomly.

The main difference between this heuristic and the original DSatur algorithm is in the second step where the heuristic selects the vertices according to the weights instead of degrees. Note that the genotype space determined by permutation of the vertices is huge, i.e., $n!$. Therefore, $(n-1)!$ solutions can have equal values of the objective function because only n values of the function exist according to Eq. (1). That is, many solutions with the same fitness value can arise that represent neutral networks [58] in the fitness landscape. On the other hand, the phenotype space is much smaller than the genotype space, namely $|C| = 3^n$.

Initially, the original DSatur algorithm orders the vertices $v_{i,j} \in V$, $j = 1, \ldots, n$, of a given graph G descending according to the vertex degrees $\deg_G(v_{i,j})$ that count the number of edges incident with vertex $v_{i,j}$ [9]. To simulate the behavior of the original DSatur algorithm [10], the first solution in the population is initialized as follows:

$$y_{i,j}^{(0)} = \frac{\deg_G(v_{i,j})}{\max_{j=1,\ldots,n} \deg_G(v_{i,j})}, \quad j = 1, \ldots, n. \tag{6}$$

Because the genotype is mapped into a permutation of weights, the same ordering is obtained as by the original DSatur, where the solution can be found in one step.

3.3 Local search heuristic

Experiments with self-adaptive evolutionary algorithms prove that a search can rapidly converge towards good areas of the search space (exploration) [63]. During the exploitation phase, these algorithms are less convenient because of the stochastic nature of the variation operators. In this phase, improvement heuristics are useful that may improve current solutions with a more systematic search in their vicinity. Local search [46] is the most often used kind of improvement heuristic that can incorporate problem-specific knowledge into the evolutionary algorithm.

Local search can be described as an iterative heuristic that explores a set of candidate solutions around the current solution (also known as neighborhood) and can replace the current solution with a better one, if it is found. A neighborhood of the current solution $\mathbf{y}$ is a mapping $\mathcal{N} : Y \mapsto 2^Y$, which for each solution $\mathbf{y} \in Y$ defines a set $\mathcal{N}(\mathbf{y}) \subseteq Y$ of solutions that can be reached using a unary operator [1]. In fact, each solution in the neighborhood $\mathbf{y}' \in \mathcal{N}(\mathbf{y})$ can be reached from the current solution $\mathbf{y}$ in k steps. Therefore, this neighborhood is also called the k-opt neighborhood of the current solution $\mathbf{y}$.

Although various unary operators have been developed to be used with HSA-EA, experiments have shown that the best performance can be achieved by the unary operator termed the *hybrid swap*. Therefore, this operator was used in this work. The functioning of this operator is illustrated in Fig. 1, which presents a solution to a graph G with ten vertices. This solution is composed of a permutation of vertices $\mathbf{v}$, corresponding coloring $\mathbf{s}$, weights $\mathbf{y}$, and saturation degrees $\boldsymbol{\rho}$. The hybrid swap unary operator takes the first uncolored vertex in a solution and orders the predecessors according to the saturation degree, in descending order. The uncolored vertex is swapped with the vertex that has the highest saturation degree. In the case of a tie, the operator randomly selects a vertex among the vertices with higher saturation degrees. Thus, the best neighbor of the current solution is determined by an exchange of two vertices (2-opt neighborhood).

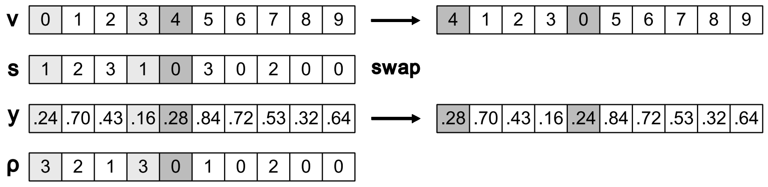


Fig. 1 The hybrid swap unary operator

In Fig. 1, an element of the solution corresponding to the first uncolored vertex 4 is in dark gray and the vertices 0 and 3 with the highest saturation degree are in light gray. From vertices 0 and 3, hybrid swap randomly selects vertex 0 and swaps it with vertex 4 (the right-hand side of Fig. 1).

3.4 Neutral survivor selection

Genotype diversity is one of the main prerequisites for efficient self-adaptation. The smaller genotypic diversity causes the population to be crowded within the search space. As a result, the search space is exploited. On the other hand, the larger genotypic diversity causes the population to be better distributed over the search space and therefore, the search space is better explored [3]. The genotype diversity is explicitly maintained using the proposed neutral survivor selection that is inspired by the neutral theory of molecular evolution [51].

However, a measure is needed to determine the distance between solutions. An Euclidian distance is the most appropriate when the solutions are represented as real-coded vectors. The Euclidian distance between two vectors $\mathbf{y_1}$ and $\mathbf{y_2}$ is expressed as follows:

$$d_E(\mathbf{y_1},\mathbf{y_2}) = \sqrt{\frac{1}{n}\cdot\sum_{j=1}^{n}(y_{1,j}-y_{2,j})^2}, \tag{7}$$

A reference solution is needed to determine how the solutions are dissipated over the search space. For this reason, the current best solution $\mathbf{y}^*$ in the population is used, as defined in Sect. 3.1.

The operation of the neutral survivor selection is divided into two phases. During the first phase, in the population of λ offspring the evolutionary algorithm finds the set of the best neutral solutions $N_S = \{\mathbf{y_1},\dots,\mathbf{y_k}\}$. If the neutral solutions are better than, or equal, to the reference solution, i.e., $f(\mathbf{y_i}) \le f(\mathbf{y}^*)$ for $i = 1,\dots,k$, the reference solution $\mathbf{y}^*$ is replaced by the neutral solution $\mathbf{y_i} \in N_S$ that is most distant from the reference solution according to Eq. (7). In line with this, it is expected that the evolutionary search is directed towards new, undiscovered regions of the search space. Conversely, if the neutral solutions are worse than the reference solution, the reference solution remains unchanged.

During the second phase, the reference solution $\mathbf{y}^*$ is used for determining the next population of survivors. For this purpose, the offspring are ordered according to the ordering relation $\prec$ (read: is better than) as follows:

$$f(\mathbf{y_1}) \prec \cdots \prec f(\mathbf{y_i}) \prec f(\mathbf{y_{i+1}}) \prec \cdots \prec f(\mathbf{y_\lambda}), \tag{8}$$

where the ordering relation $\prec$ is defined as

$$f(\mathbf{y_i}) \prec f(\mathbf{y_{i+1}}) \Rightarrow \begin{cases} f(\mathbf{y_i}) < f(\mathbf{y_{i+1}}), \\ f(\mathbf{y_i}) = f(\mathbf{y_{i+1}}) \wedge (d(\mathbf{y_i},\mathbf{y}^*) > d(\mathbf{y_{i+1}},\mathbf{y}^*)). \end{cases} \tag{9}$$

Finally, for the next generation the evolutionary algorithm selects the best μ offspring according to Eq. (8). These individuals take random positions in the next generation. Like the neutral theory of molecular evolution, neutral survivor selection offers three possible outcomes to offspring, as follows. First, the best offspring survive.

Additionally, the neutral solution that is the most distant from the reference solution becomes the new reference solution. Second, the low-fitness offspring are usually eliminated from the population. Third, all other solutions, that could be neutral as well, can survive if they take the first μ positions according to Eq. (8).

4 Experiments and results

The goal of the experiments performed in this study was to compare the results of different graph 3-coloring algorithms and show that the performance of the proposed HSA-EA for graph 3-coloring is comparable with, or better than, the performance of other algorithms for solving this problem. Here, the following algorithms were used:

- SAW-EA [66],
- Tabucol [16],
- HEA [16],
- HSA-EA.

An implementation of SAW-EA can be found in [66] and is described in [28]. The Tabucol and HEA algorithms are among the best known algorithms for solving k-GCP [22, 33]. In addition, implementations of both algorithms are publicly available for facilitating comparisons between newly developed algorithms [18]. Although the implementation of Tabucol, as presented in [42], gained slightly better results than in [16], the former implementation is not publicly available and, therefore, was not used in our study.

The characteristics of HSA-EA used in the experiments were as follows. The fitness function according to Eq. (1) was considered by this algorithm. Note that the fitness value cannot exceed the number of vertices since the fitness function (Eq. (1)) counts the number of vertices violating the constraints. This value, however, cannot be higher than the number of all vertices. HSA-EA employed the selection scheme $(15, 100)$. That is, 100 candidate solutions were generated from the population of 15 parents, from which 15 fittest offspring were selected by the neutral survivor selection. Although other selection schemes were also examined, the experiments showed that this selection scheme was the most effective. Typically, a larger population requires higher initial mutation strengths. Furthermore, a larger population can maintain higher diversity, but a smaller population converges faster than the larger one. In our opinion, this is due to the fact that the used predetermined number of evaluations is insufficient for the larger population to converge. Experiments varying the tournament size showed that on average the best results are obtained using a tournament size of 3. However, a detailed analysis of the experimental results proved that a tournament size of 3 is more appropriate for smaller populations (such as in our case). As the population size grows, choosing a tournament size of 5 is more suitable. Determination of initial mutation strengths was crucial for the performance of HSA-EA. After extensive experimentation, the initial mutation strength $q_i^{(0)} = 0.03$ and $\epsilon_0 = 0.001$ produced the best results. As the termination condition, the number of function evaluations was limited to 300,000, while the number of independent runs was fixed at 25. The last two settings were the same as in [28]. This experimental setup was used in all of the tested algorithms to make the comparison as fair as possible.

The algorithms were compared according to two performance measures:

– success rate (SR) and
– average number of objective function evaluations to solution (AES).

The former reflects the stochastic nature of HSA-EA and is defined as the ratio of successfully terminated runs, while the latter expresses the efficiency of a particular algorithm, and is defined as the number of objective function evaluations needed to find a solution.

4.1 Test problems

Heuristics for solving the k-GCP are commonly compared on a set of graphs as proposed in the DIMACS challenge suite [50]. However, this set does not permit the statistical study of relations between algorithm performance and the structural features of graphs [18]. The structural features of graphs may be induced by:

– generating a graph with a given number of color classes and edge densities,
– influencing the variability in the sizes of the color classes.

Both presumptions can be satisfied using the Culberson graph generator [20]. Although this graph generator can generate various types of k-colorable graphs, we focus on three specific types: uniform, equi-partite, and flat graphs. The type of graphs imposes an edge distribution by the graph generator as follows. Uniform 3-colorable graphs are random graphs, where the vertices are randomly assigned to one of the three color classes uniformly and independently. The main property of equi-partite 3-colorable graphs is that the three color classes are as equal in size as possible. Graphs of this type are less difficult to 3-color but difficult enough for many existing algorithms [21]. Flat graphs seem the hardest to 3-color because besides the minimum variation in number of vertices in the three color classes the variation in degree for each node is also kept to minimum.

Random graphs are created using the parameter $\Delta \in \{0, 1, 2\}$ that controls the variability in sizes of the color classes. Uniform 3-colorable graphs are generated when $\Delta = 0$ as mentioned previously. Variable 3-colorable graphs are obtained when $\Delta > 0$. Graphs of this type are generated as follows. For each vertex in turn, the generator randomly selects an integer r from the interval $[0, \Delta]$. Then, this vertex is assigned to the color class i that is randomly selected from the interval $[r, 2]$. In general, variable 3-colorable graphs are easier to color, as reported by Turner [65].

The following parameters were used when generating the random graphs: graph type, number of vertices, edge density controlled by the parameter p determining the probability that two vertices v_i and v_j are connected with an edge (v_i, v_j), and the seed of the random graph generator. In general, these graphs can be denoted as $G_{t,n,p,s}$, where t denotes the graph type (i.e., *uni* for uniform ($\Delta = 0$), *eq* for equi-partite, *flat* for flat), n the number of vertices, p the probability controlling the edge density, and s the seed of the random graph generator. Three types of uniform graphs were generated in order to investigate the influence of variability in sizes of the color classes on the algorithm results. When $\Delta = 0$, the generated graphs have no variability in the sizes of the color classes (the classes tend to be nearly equal in size, like in the equi-partite graphs), while when $\Delta = 1$ or $\Delta = 2$, the sizes tend to vary considerably. Flat graphs can be generated using different flatness when determining variations in the degrees of the vertices. If the flatness is zero, the generated graphs are fairly uniform and these graphs are the hardest to color. This type of flat graph was used in our experiments.

Graphs with the number of vertices $n = 500$ and $n = 1{,}000$, were considered during the experiments. These graphs are denoted as medium-scale and large-scale graphs in this paper. Note that the graphs with the number of vertices $n = 200$ as also treated in [28] were not considered in this study because these graphs are too simple for the tested graph 3-coloring algorithms. The random graph generator seed parameter was varied from 1 to 10 with a step of one in order to test the statistical significance of the results. Additionally, the seeds of the random number generator for all the used algorithms were different in each run.

Most combinatorial optimization problems are sensitive to the phase transition, which refers to the regions where the problem passes from the state of "solvable" to the state "unsolvable", and vice versa [65]. Typically, these regions are determined by a problem-specific parameter. For 3-GCP this parameter is the edge density determined by probability that two vertices are connected, p. Many authors have identified various critical values for this parameter when determining the phase transition region. For example, Petford and Welsh [61] stated that this phenomenon occurs when $2pn/3 \approx 16/3$, Cheeseman et al. [14] $2m/n \approx 5.4$, Eiben et al. [28] $7/n \leq p \leq 8/n$, and Hayes [43] $m/n \approx 2.35$. The parameter m in the above formulas denotes the number of edges.

In order to capture the phase transition, the parameter p needs to be varied appropriately. Preliminary experiments were conducted to determine suitable values of the value p. The following experimental setup was applied to show that the phase transition was located correctly. Medium- and large-scale graphs were generated with edge density $p \in [0.004, 0.696]$ with a step of 0.004. As a result, 174 instances were obtained for each type of graphs. Specifically, random graphs were generated with different variabilities in size, i.e., with $\Delta \in \{0, 1, 2\}$. Graphs of each type were generated varying random graph generator seed from 1 to 10. Then, the four algorithms, i.e., SAW-EA, Tabucol, HEA, and HSA-EA, run the graph 3-coloring and the results on random graphs with different seeds were accumulated over 25 runs. The results of this experiment showed that none of the tested algorithms had problems with the 3-coloring of medium-scale graphs with $p > 0.028$ and large-scale graphs with $p > 0.014$. For this reason, our further experiments were conducted with these two values as upper bounds of the observed edge densities.

Essentially, the experiments were divided into three parts. In the first part, the behavior of the mentioned graph-coloring algorithms was observed in the phase transition region. This part is comparable to the study of Eiben et al. in [28]. The influence of the evolutionary algorithm on the traditional DSatur heuristic was observed during the second part. The structural features of graphs were studied during the third part. This part considered the novel experimental approach to graph coloring, as proposed by Chiarandini and Stützle [18].

4.2 Behavior of graph-coloring algorithms in the phase transition

These experiments were executed on medium- and large-scale graphs. The results for both graph sizes are presented in the next sections. In both cases, however, the phenomenon of the phase transition was captured.

4.2.1 Medium-scale graphs

Medium-scale graphs ($n = 500$) were generated with edge densities varied from $p = 0.008$ to $p = 0.028$ with a step of 0.001. As a result, 21 instances of randomly created graphs were obtained. Note that the phase transition occurs at $p = 0.014$ according to Hayes [43], $p = 0.016$ according to Cheeseman [14], and Petford and Welsh [61], and $p \in [0.014, 0.016]$ according to Eiben et al. [28].

The averaged results of the tested algorithms on medium-scale graphs with different seeds are illustrated in Figs. 2–7. These figures represent six diagrams for three graph types according to two measures. The best results on medium-scale graphs were obtained by HSA-EA (Fig. 2). The average results of Tabucol and HEA were

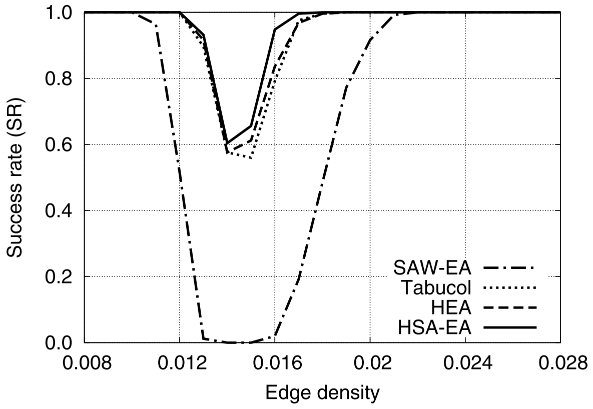


Fig. 2 SR on uniform medium-scale graphs

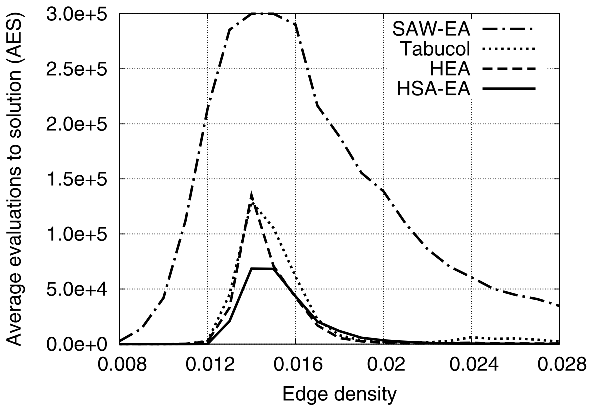


Fig. 3 AES on uniform medium-scale graphs

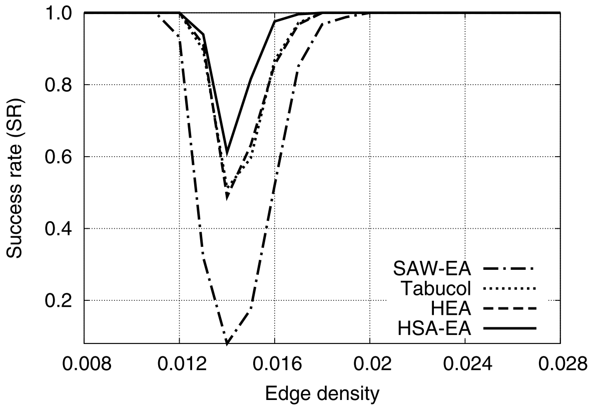


Fig. 4 SR on equi-partite medium-scale graphs

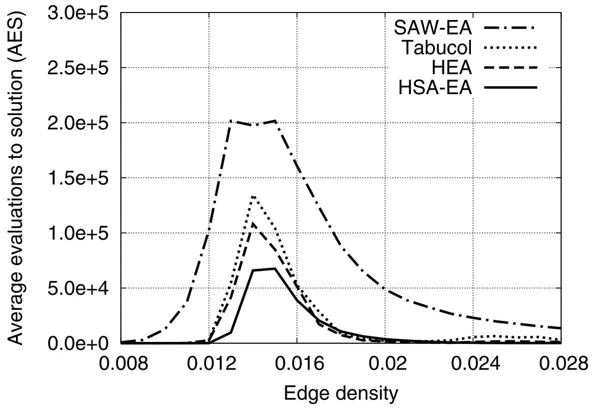


Fig. 5 AES on equi-partite medium-scale graphs

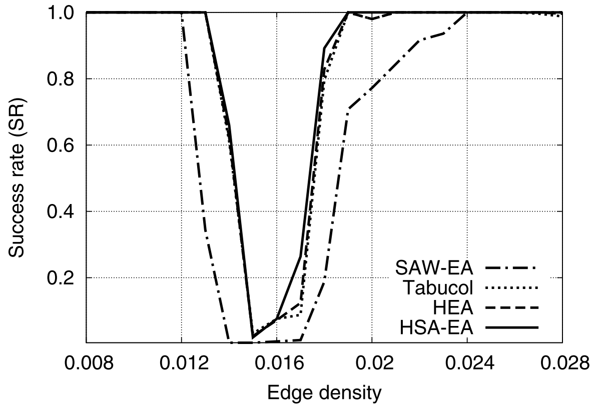
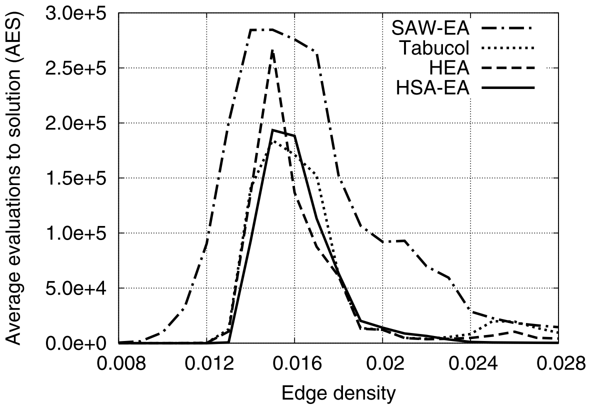


Fig. 6 SR on flat medium-scale graphs

similar, while SAW-EA gained the worst results according to SR. In fact, this algorithm was sensitive to the phase transition as proposed by Hayes ($p = 0.014$). Results on uniform graphs according to AES (Fig. 3) confirmed that graphs of this type can

Fig. 7 AES on flat medium-scale graphs



be colored by all tested algorithms, except SAW-EA. For this reason, the AES plots did not reach the maximum number of evaluations to solution.

As illustrated in Fig. 4, the equi-partite graphs were also best 3-colored by HSA-EA. Note that all the mentioned algorithms were sensitive to the phase transition determined by $p = 0.014$. In this case, all algorithms 3-colored the equi-partite graphs with $\mathrm{SR} > 0$, as seen in the AES plots in Fig. 5. Note that SAW-EA achieved $\mathrm{SR} = 0.08$ at $p = 0.014$.

Experiments on flat graphs confirmed that these graphs were the hardest to color. Interestingly, the SR plots for these graphs showed single valleys (Fig. 6) and the AES plots single peaks (Fig. 7). Note that for SAW-EA, SR did not reach zero on average, i.e., $\mathrm{SR} = 0.004$ at $p = 0.014$, as can be seen in the corresponding AES plot. As a matter of fact, the peaks (valleys) were located as suggested by Hayes [43], Cheeseman [14], Petford and Welsh [61], and Eiben et al. [28], i.e., in the interval $p \in [0.14, 0.16]$.

4.2.2 Large-scale graphs

Large-scale graphs ($n = 1{,}000$) were generated with edge densities varying from $p = 0.004$ to $p = 0.014$ with a step of 0.0005. As a result, 21 instances of randomly generated graphs were obtained for each graph type. The phase transition occurs at $p = 0.007$ according to Hayes [43], $p = 0.008$ according to Cheeseman [14], and Petford and Welsh [61], and $p \in [0.007, 0.008]$ according to Eiben et al. [28].

The results of graph 3-coloring using the tested algorithms are illustrated in Figs. 8–13. These figures show the results of coloring graphs of three types according to two measures.

HSA-EA produced the best results for $p < 0.007$ when coloring the uniform graphs (Fig. 8). On average, graph instances with $p = 0.007$ were a hard nut to crack for all the tackled algorithms. However, HSA-EA obtained $\mathrm{SR} = 0.16$, while the other algorithms performed slightly worse. Note that SAW-EA did not find any solution. HEA improved its own results at $p = 0.0075$, and, thus, reached the results of HSA-EA. When the edge density was increased, the best results were obtained by HEA. Tabucol and HSA-EA produced similar results, while SAW-EA obtained the worst

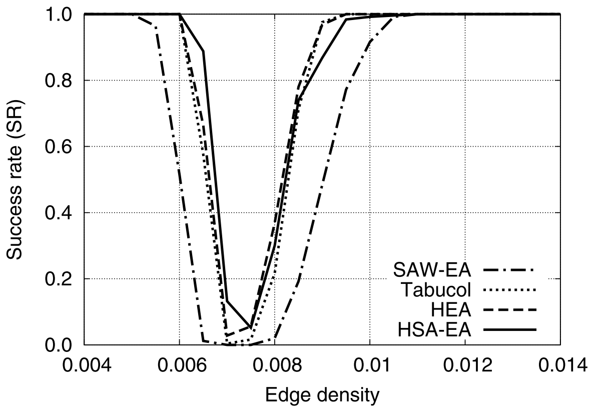


Fig. 8 SR on uniform large-scale graphs

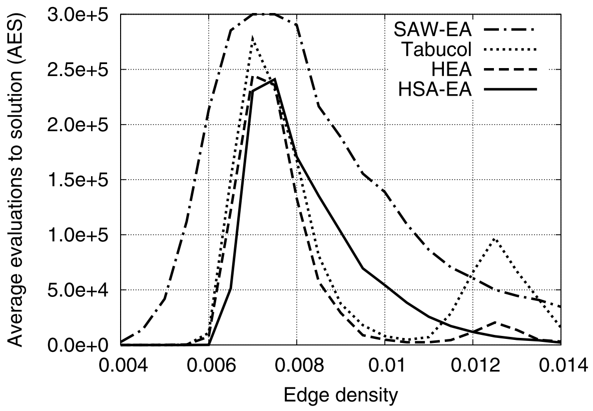


Fig. 9 AES on uniform large-scale graphs

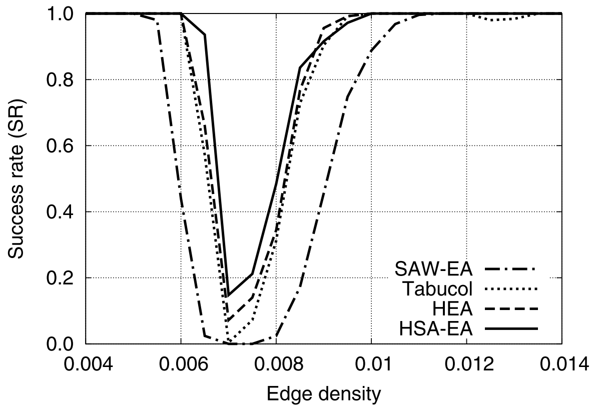


Fig. 10 SR on equi-partite large-scale graphs

results. According to AES (Fig. 9), the best results were produced by HEA that spent the minimum number of evaluations on the graph instances with edge density away from the phase transition. Similar results were gained by Tabucol except for the graph instance with $p = 0.013$, where the AES measure increased significantly.

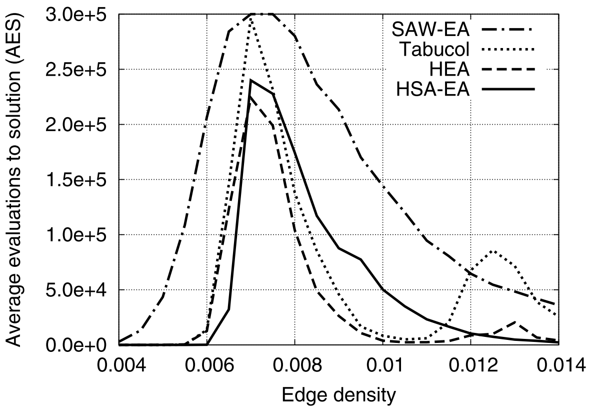


Fig. 11 AES on equi-partite large-scale graphs

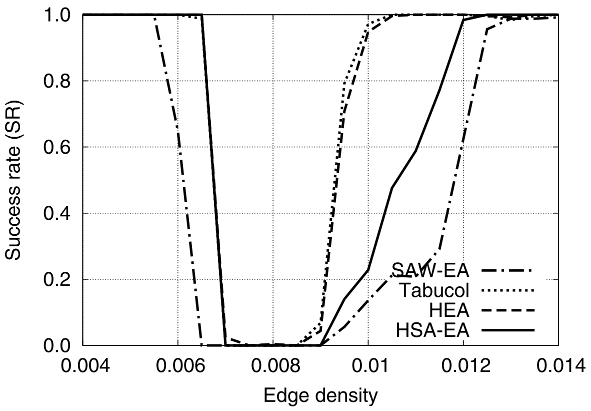


Fig. 12 SR on flat large-scale graphs

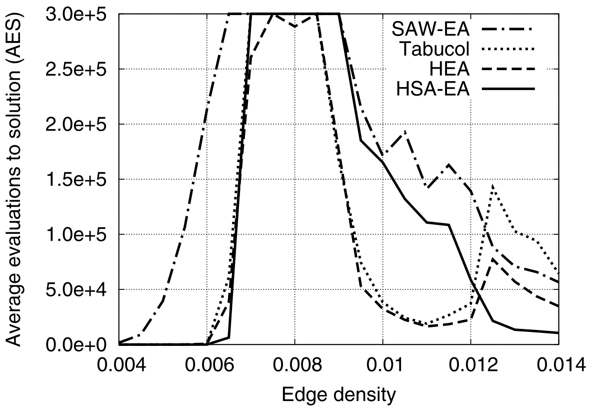


Fig. 13 AES on flat large-scale graphs

Equi-partite graphs were most successfully colored by HSA-EA (Fig. 10). In general, the SR plot describing the behavior of a particular algorithm in the phase transition was the narrowest for HSA-EA although HEA obtains slightly better results according to the AES measure (Fig. 11). However, the HEA's and HSA-EA's plots did not reach the maximum number of evaluations during the observed interval of the edge density. Note that Tabucol did not completely solve all of the equi-partite graph instances in the vicinity of $p = 0.013$. This behavior of Tabucol seemed to be related to the occurrence of the second phase transition [8].

The results on flat graphs (Fig. 12) indicate that graphs of this type are the hardest to color. Interestingly, the best coloring algorithms, like HEA and Tabucol, also did not color graph instances in the phase transition. Moreover, the region where these algorithms do not find any solution was widened to $p \in [0.007, 0.0085]$. This region was broader for HSA-EA and even broader for SAW-EA. Note that Tabucol and HEA were also sensitive to the second phase transition. According to AES (Fig. 13), the worst results in the phase transition region were obtained by SAW-EA. Slightly better results were produced by HSA-EA which, on the other hand, outperformed the other algorithms in the second phase transition region.

4.3 Comparing HSA-EA with the DSatur traditional algorithm

To see the contribution on HSA-EA with regard to the original DSatur algorithm, the two algorithms were compared. Note that HSA-EA affects the selection of the first vertex on DSatur heuristic. In order to show how important this decision is, a modified variant of the DSatur algorithm [21] was built, where only one run is performed similarly to the fitness calculation phase in HSA-EA. As for the first vertex, each of the n vertices in the permutation is selected sequentially. As a result, n different runs of this algorithm (denoted as ModDSat) were performed. Additionally, the results were also compared with the backtracking variant of DSatur [64] (denoted as BkDSat). Thanks to tie breaking, this algorithm is also stochastic because different random number generator seed values during the initialization of the algorithm may lead to different results.

Like in Sect. 4.2, the experiments were conducted on medium-scale and large-scale graphs. Furthermore, the phase transition was also included.

4.3.1 Medium-scale graphs

The medium-scale graphs were generated as described in Sect. 4.2.1. The ModDSat algorithm was executed 500 times by varying the first vertex from 1 to 500. The BkDSat algorithm terminated when the solution was found or the number of backtracking steps to the first vertex that could be colored with another color reached 300,000. 300,000 objective function evaluations were also considered as the termination condition for HSA-EA. All three algorithms were run on graph instances created with random seeds from 1 to 10. Each algorithm was executed 25 times on each graph instance. The results are summarized in Figs. 14, 15, and 16. Note that the results are compared with regard to the SR measure only.

The behavior of both DSatur variants is similar on the graphs of all types. While BkDSat obtained $\mathrm{SR} > 0$ on uniform (Fig. 14) and equi-partite (Fig. 15) graphs, and $\mathrm{SR} = 0$ on flat graphs (Fig. 16) in the phase transition ($p \in [0.014, 0.016]$), this was not the case with ModDSat that could only successfully solve the instances with $p < 0.012$.

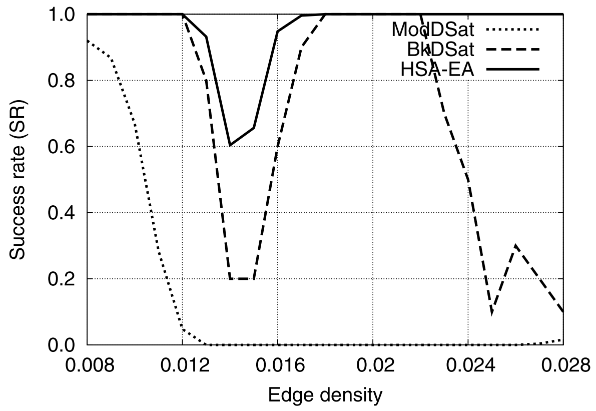


Fig. 14 SR on uniform medium-scale graphs

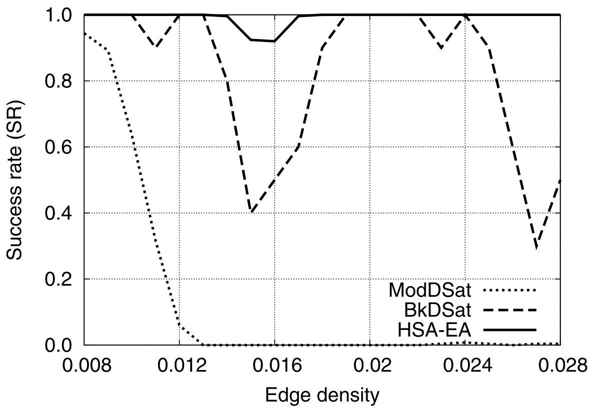


Fig. 15 SR on equi-partite medium-scale graphs

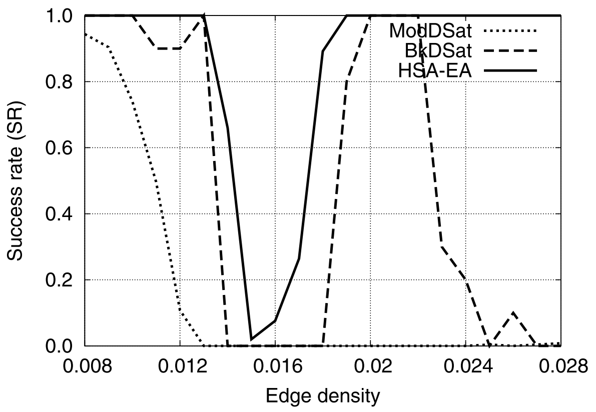


Fig. 16 SR on flat medium-scale graphs

4.3.2 Large-scale graphs

Large-scale graphs were generated according to the scenario described in Sect. 4.2.2. However, ModDSat was executed 1,000 times. BkDSat terminated when the number of backtracking steps to the first vertex that could be colored with another color reached 300,000, while for HSA-EA when the number of objective function evaluations reached the same value. All three algorithms run graph 3-coloring created with random seeds from 1 to 10 and repeated the graph 3-coloring 25 times using different seeds. The results were accumulated for each graph instance. They are presented in Figs. 17, 18 and 19.

The results on large-scale graphs were similar to the results on medium-scale graphs. That is, ModDSat and BkDSat successfully 3-colored the graphs below the phase transition ($p < 0.007$). BkDSat 3-colored some instances of the uniform (Fig. 17) and equi-partite graphs (Fig. 18) in the interval $p \in [0.0095, 0.012]$, while the results are poor outside this interval. On flat graphs no results were obtained for $p > 0.012$. The behavior of ModDSat was even worse because it only found solutions for the graph instances with $p < 0.0065$.

4.4 Impact of the graph structural features

The graph size has the major impact on the performance of the graph-coloring algorithms. This is expressed as the number of vertices. Obviously, the more vertices in the graph, the harder the graph to color. However, given a fixed graph size, additional variables determine the hardness of the graph to color, as follows:

- the graph type,
- the edge density and
- the variability in sizes of the color classes.

These variables determine the structural features of graphs and are also referred to as stratification variables.

An additional quality measure, *Error rate* (ER), was defined to facilitate the analysis of the structural features. ER reflects the average number of unsuccessful runs. Obviously, the best ER is zero. This measure is derived from the success rate as follows:

$$\mathrm{ER} = 1 - \overline{\mathrm{SR}}, \tag{10}$$

where $\overline{\mathrm{SR}}$ determines the average success rate according to the observed stratification variable. While SR is defined as the average number of successful runs in coloring one graph instance, $\overline{\mathrm{SR}}$ also considers the average success rate over a number of instances. The next subsections present the influence of stratification variables on the behavior of the graph-coloring algorithms.

4.4.1 Influence of the graph size

The impact of graph size on the performance of graph-coloring algorithms was investigated for two graph sizes, i.e., $n = 500$ (medium-scale graphs) and $n = 1{,}000$

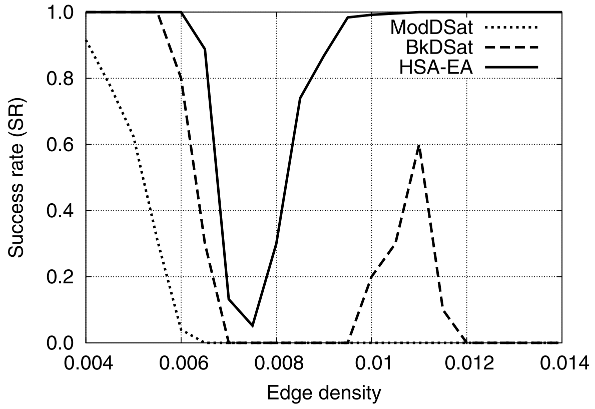


Fig. 17 SR on uniform large-scale graphs

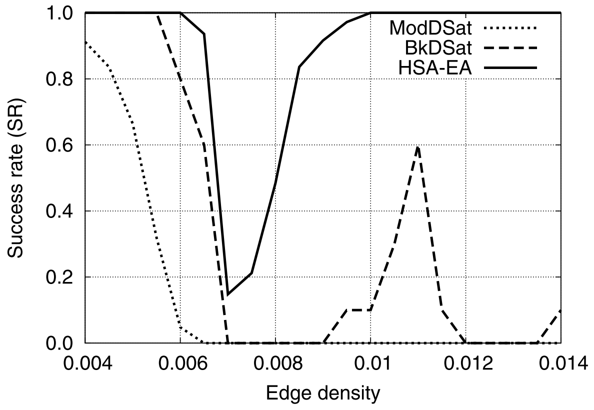


Fig. 18 SR on equi-partite large-scale graphs

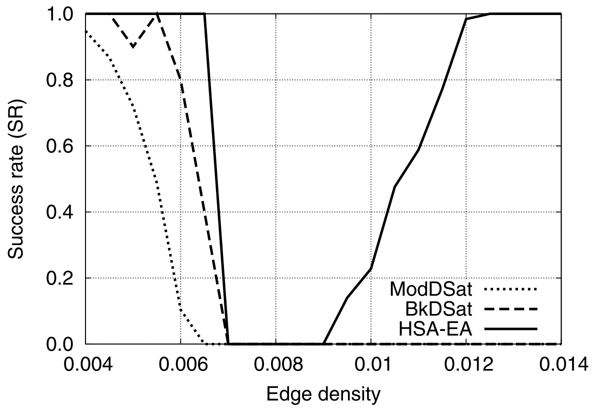


Fig. 19 SR on flat large-scale graphs

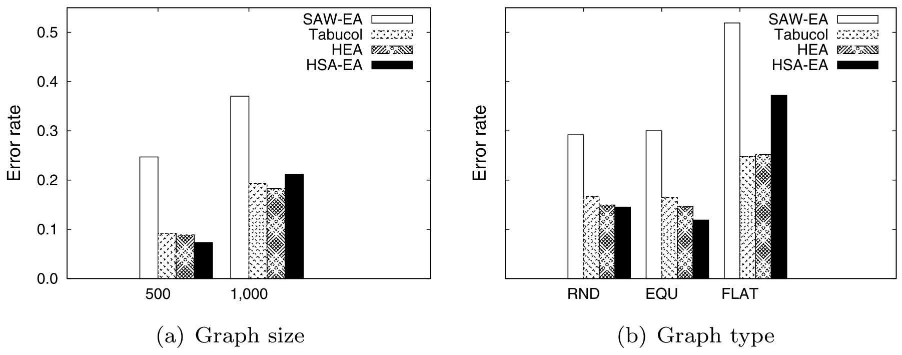


Fig. 20 Influence of graph size and type on the performance of graph-coloring algorithms

(large-scale graphs). Here, $\overline{\text{SR}}$ from Eq. (10) is defined as the average SR for various types of medium-scale graphs varying $p \in [0.008, 0.028]$ with a step of 0.001, and large-scale graphs varying $p \in [0.004, 0.014]$ with a step of 0.0005. These intervals were selected to cover the respective phase transition regions. The SRs were then aggregated for each tested algorithm.

As seen from Fig. 20(a), the graph size has a big influence on the algorithm performance. The ERs for all tested algorithms increased (and SR declined) with increasing graph size. The best results on medium-scale graphs were gained by HSA-EA (ER = 0.07), while large-scale graphs were best solved by HEA (ER = 0.18). In both cases, SAW-EA achieved the worst results. Specifically, the improvement of the results as obtained by HEA is more gradual than by HSA-EA. That is, HEA is less sensitive to increasing graph size than HSA-EA.

4.4.2 *Influence of the graph type*

The performance of the graph-coloring algorithms was analyzed with regard to the graph type. The following types of graphs were taken into consideration: uniform, equi-partite, and flat. Note that here, only the large-scale graphs were considered. In this experiment, the $\overline{\text{SR}}$ from (10) was defined as the average SR achieved when coloring various types of large-scale graphs varying $p \in [0.004, 0.014]$ with a step of 0.0005.

As seen from Fig. 20(b), the best result (ER = 0.12) was achieved by HSA-EA on equi-partite graphs. In general, there existed only a small difference between coloring of uniform and equi-partite graphs when comparing particular algorithms, while the flat graphs were the hardest to all tested algorithms. When comparing the performance of individual algorithms, it could be seen that the best results were obtained by HSA-EA, which outperformed the HEA on the uniform and equi-partite graphs. Tabucol was slightly worse, but SAW-EA did not reach the performance of the other algorithms. The flat graphs were best colored by Tabucol and HEA, slightly worse by HSA-EA, and worst by SAW-EA.

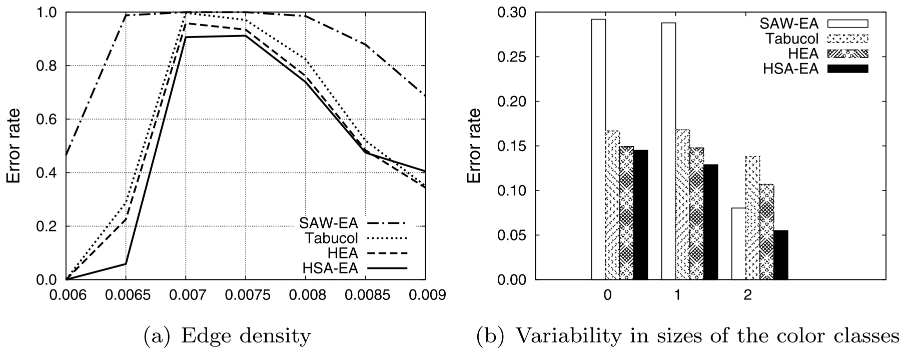


Fig. 21 Influence of the edge density and the variability in sizes of the color classes on the performance of graph-coloring algorithms

4.4.3 *Influence of the edge density*

The influence of the edge density when coloring the large-scale graphs has already been discussed in Sect. 4.2.2, where the results were evaluated according to SR. Here, the interest is to investigate the influence of edge density on the performance of the graph-coloring algorithms. Therefore, the ER results from coloring the three types of large-scale graphs varying $p \in [0.006, 0.009]$ with a step of 0.0005 were aggregated for each coloring algorithm. Note that the observed interval of the edge density precisely corresponded to the phase transition region.

As illustrated in Fig. 21(a), the best results were achieved by HSA-EA that colored all instances of the observed graphs. HEA and Tabucol were also close to this result. Actually, graph 3-coloring in the hardest instance with $p = 0.007$ was best performed by HSA-EA, followed by HEA and Tabucol, while graph instances in the phase transition, i.e., at $p = 0.007$ and at $p = 0.0075$, remained a challenge for SAW-EA.

4.4.4 *Influence of the variability in sizes of the color classes*

This influence was studied to confirm the assertion of Turner [65] that the variable 3-colorable graphs are easier to color. The results of the tested graph-coloring algorithms were compared in 3-coloring the large-scale graphs varying $p \in [0.004, 0.014]$ with a step of 0.0005.

As shown in Fig. 21(b), the results confirmed that the assertion by Turner holds for all graph-coloring algorithms. The variability in sizes of the color classes had the highest impact on the performance of SAW-EA that 3-colored the graphs with $\Delta = 2$ even better than HEA and Tabucol. Unfortunately, the results of this algorithm were much worse when compared with the results of the other algorithms on graphs with variabilities $\Delta = 0$ and $\Delta = 1$. According to this parameter, HSA-EA generally outperformed all other algorithms.

4.4.5 *Computing time and scalability*

Computing time of HSA-EA is comparable with the computing time of other tested algorithms on problem instances with up to 500 vertices. Here, the solution is found quickly and, in line with this, a small number of function evaluations is used. When the number of vertices is increased, the computing time also increases. In summary, Tabucol and HEA are the most efficient algorithms, while SAW-EA and HSA-EA are worse in this respect. The main reason for longer computing time of HSA-EA is in the DSatur construction heuristic that uses dynamic vertex ordering. Although this ordering is based on the Heapsort algorithm [47] with time complexity $O(n\log(n))$, this increases especially in runs where the solution is not found.

In the first place, the scalability of HSA-EA depends on the algorithm design. Because HSA-EA is a representative of evolution strategies, it very much depends on the way of selecting the initial mutation strength $q_i^{(0)}$. In general, the progress of evolution process in evolution strategies can only occur in a narrow band of step sizes, i.e. *evolution window*. The initial mutation strength $q_i^{(0)}$ determines the size of the search space exploration. Mutation strength decreases as the number of generations increases and, therefore, the size of the explored search space depends on the maximum number of fitness evaluations. For example, for graph instances with $n \leq 1{,}000$, an appropriate value of this parameter $q_i^{(0)} = 0.03$ was detected through extensive experiments. In summary, the scalability of HSA-EA requires proper determination of the evolution window. However, this demands additional experiments to check if the ranking amongst the tested algorithms still holds when the graph size increases.

4.5 Statistical analysis of results

To evaluate the quality of the graph-coloring algorithms, a non-parametric analysis of their results was carried out, as suggested by Demšar [24]. As a basis, the Friedman non-parametric test was considered [31, 32]. This test compares the average ranks of algorithms. A null-hypothesis states that two algorithms are equivalent and, therefore, their ranks should be equal. If the null-hypothesis is rejected, i.e., the performance of the algorithms is statistically different, the Bonferroni-Dunn test [24] is performed that calculates the critical difference between the average ranks of those two algorithms. When the statistical difference is higher than the critical difference, the algorithms are significantly different. The equation for the calculation of critical difference can be found in [24].

The Friedman non-parametric tests referred to the large-scale graphs only. Two Friedman non-parametric tests were performed. In the first test, the behavior of algorithms was observed when coloring specific instances of different graph types in the phase transition. For this purpose, 30 uniform graphs, 10 equi-partite, and 10 flat graphs in the phase transition were taken into account. In the second test, the average results of the graph 3-coloring algorithms were compared in the phase transition. In this case, the edge densities $p \in [0.006, 0.009]$ with a step of 0.0005 were treated, obtaining 7 graph instances. Moreover, the average results on all uniform graphs with different variabilities in sizes of the color classes $\Delta \in \{0, 1, 2\}$ were taken during this analysis and, in line with this, the number of graph instances increased to 350, i.e., $5 \times 10 \times 7$.

The results of the first Friedman non-parametric test are presented in Fig. 22 being divided into nine diagrams that show the ranks and confidence intervals (critical differences) for the algorithms under consideration. The diagrams are organized according to the graph types and the edge densities. Two algorithms are significantly

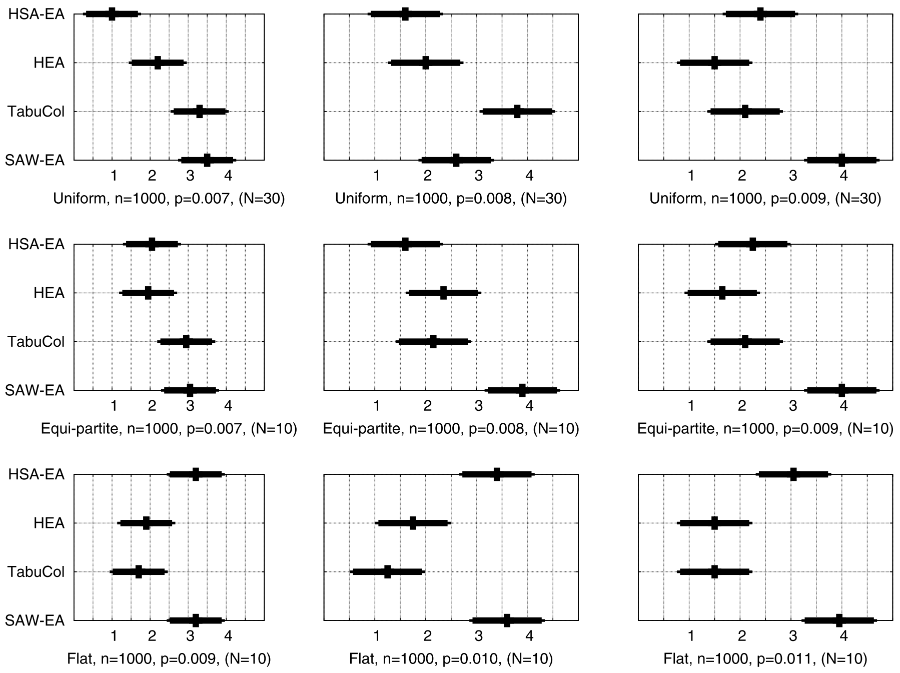


Fig. 22 Results of the Friedman non-parametric test on the specific large-scale graph instances

different if their intervals in Fig. 22 do not overlap. Note that the uniform and equi-partite graphs with edge densities $p = \{0.007, 0.008, 0.009\}$ are considered, while the flat graphs are taken with edge densities $p = \{0.009, 0.010, 0.011\}$ because the tested algorithms did not find any solution coloring the graphs of this type with edge densities $p = \{0.007, 0.008\}$.

The following conclusions can be derived from Fig. 22:

- On uniform graphs, HSA-EA significantly improves the results of Tabucol and SAW-EA on instances with $p = 0.007$, while HSA-EA and HEA are significantly better than Tabucol on instances with $p = 0.008$. In the case of instances with $p = 0.009$, all other algorithms are significantly better than SAW-EA.
- On equi-partite graphs, no significant difference can be detected on graphs with $p = 0.007$, while all other algorithms color graphs with $p = 0.008$ and $p = 0.009$ significantly better than SAW-EA.
- On flat graphs, no significant difference can be detected on graphs with $p = 0.007$, while coloring of other graph instances is performed significantly better by HEA and Tabucol than HSA-EA and SAW-EA.

The results of the second Friedman non-parametric test are presented in Fig. 23 showing the average ranks and confidence intervals for the algorithms under consideration. Here, besides the algorithms mentioned in the first test, both variants of the DSatur algorithm were also observed. The figure consists of three diagrams that cor-

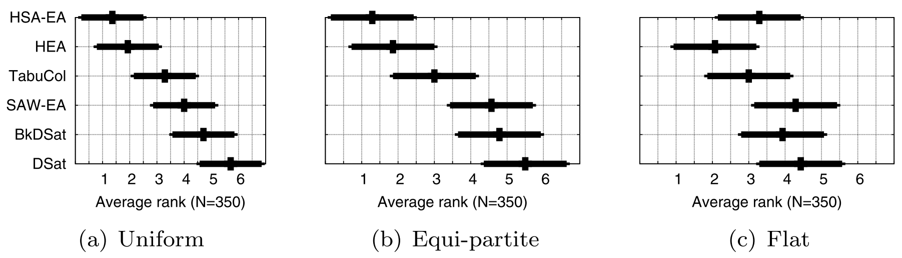


Fig. 23 Averaged results of the Friedman non-parametric test on large-scale graphs

respond to three graph types, i.e., uniform, equi-partite, and flat. Two algorithms are significantly different if their intervals in Fig. 23 do not overlap.

The following conclusions can be inferred from the results of the Friedman tests:

- On uniform graphs, HSA-EA and HEA significantly improve the results of BkDSat and ModDSat. The results of HSA-EA are slightly better than the results of HEA.
- On equi-partite graphs, the results of HSA-EA and HEA significantly improve the results of SAW-EA, BkDSat, and ModDSat. Essentially, HSA-EA improves the results of HEA, but the difference is not significant.
- On flat graphs, the results of the tested algorithms are comparable.

It should to be noted that all the compared algorithms are tailored to the problem in a specific manner. For example, the DSatur algorithm is a traditional heuristic for graph coloring that sequentially colors the graph vertices according to the saturation degrees. This algorithm can be improved with backtracking that exploits the tie breaking. Moreover, HSA-EA represents a hybridization of the evolutionary algorithm with the original DSatur algorithm. Furthermore, Tabucol is a well known traditional heuristic for graph coloring. As matter of fact, the application of this heuristic to the evolutionary algorithm has led to HEA that improved the results of the original Tabucol. Finally, the SAW mechanism tries to improve the behavior of the classical evolutionary algorithm in graph coloring.

By analyzing how successfully domain-specific knowledge can be incorporated into the particular evolutionary algorithm, it can be concluded that hybridization of the evolutionary algorithm with the DSatur heuristic (HSA-EA) exhibited better results on uniform and equi-partite graphs, while coloring flat graphs was better performed by the evolutionary algorithm hybridized with the Tabucol heuristic (HEA). On the other hand, applying the SAW mechanism improves the results of the original evolutionary algorithm, but this improvement does not outperform HSA-EA and HEA.

5 Conclusions

In this paper, we have proposed the HSA-EA for graph 3-coloring, which is hybridized with hybrid genotype-phenotype mapping, a swap local search heuristic, and the neutral survivor selection operator. In addition, a heuristic initialization procedure is applied by HSA-EA. This evolutionary algorithm was compared with SAW-EA [28], Tabucol [44], and HEA [33].

The objective of our experiments was threefold: first, to investigate the behavior of the tested graph-coloring algorithms in the phase transition, where the graph instances are hard to color, second, to indicate the impact of hybridizing the evolutionary algorithm with the DSatur traditional heuristic and third, to analyze how various structural features of randomly generated graphs influence the performance of the graph-coloring algorithms.

To satisfy the first objective, the graphs were generated varying the probability $p \in [0.008, 0.028]$ with a step of 0.001 for medium-scale graphs ($n = 500$) and $p \in [0.004, 0.014]$ with a step of 0.0005 for large-scale graphs ($n = 1{,}000$). In both cases, 21 instances of random 3-colorable graphs of three types were obtained, i.e., uniform, equi-partite and flat. However, all the tested algorithms encountered troubles when coloring graph instances in the phase transition region.

To satisfy the second objective, two modified versions of the DSatur algorithm were taken into account. The first one (ModDSat) was inspired by the sequential coloring of the original DSatur algorithm that seems to be dependent on the selection of the first vertex. The second one (BkDSat) was the standard backtracking DSatur algorithm [64]. Both DSatur versions showed poor performance when coloring the medium- and large-scale graphs, not only in the phase transition but also outside this region, i.e., at $p > 0.028$ for medium-scale graphs and $p > 0.016$ for large-scale graphs. On the other hand, the results of ModDSat indicate that selection of the first vertex in DSatur is not so crucial.

To satisfy the third objective, the influence of the structural features of graphs, the performance of the considered graph-coloring algorithms was compared depending on the graph size, type, edge density and variability in sizes of the color classes. Regarding the graph size, we can conclude that increasing the graph size decreases the performance of the graph-coloring algorithms. Actually, HEA was the least sensitive to the increasing graph size. Flat graphs represented the hardest graph type to color for all considered graph-coloring algorithms. In regard to the edge density, HSA-EA was the most successful as it solved all instances of the observed graphs. The variability in sizes of the color classes most influenced SAW-EA. In general, on uniform graphs HEA significantly improved the results of SAW-EA, while on equi-partite graphs HSA-EA was better than HEA. However, both algorithms are significantly better than SAW-EA. Unfortunately, all considered algorithms performed poorly on the flat graphs.

Our future work in graph coloring using evolutionary algorithms will focus on k-GCP. A key to improved performance of the evolutionary algorithm might be to use direct representation of solutions in the form of vertex color vectors instead of vertex permutations as used now. Furthermore, a new local search heuristic is needed that would better improve the solutions. It is our intention to further enhance the results on the hardest graph instances of all three classes, and a large number of nodes near the phase transition.

References

1. Aarts, E., Lenstra, J.K.: Local Search in Combinatorial Optimization. Princeton University Press, Princeton (1997)
2. Avanthay, C., Hertz, A., Zufferey, N.: A variable neighborhood search for graph coloring. Eur. J. Oper. Res. **151**, 379–388 (2003)
3. Bäck, T.: Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press, Oxford (1996)
4. Blöchliger, I., Zufferey, N.: A reactive tabu search using partial solutions for the graph coloring problem. In: Kral, D., Sgall, J. (eds.) Coloring Graphs from Lists with Bounded Size of their Union: Result from Dagstuhl Seminar 03391. ITI-Series, vol. 156. Department of Applied Mathematics and Institute for Theoretical Computer Science, Prague (2003)
5. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput. Oper. Res. **35**(3), 960–975 (2008)
6. Blum, C., Puchinger, J., Raidl, G.A., Roli, A.: Hybrid metaheuristics in combinatorial optimization: a survey. Appl. Soft Comput. **11**(6), 4135–4151 (2011)
7. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)
8. Boettcher, S., Percus, A.G.: Extremal optimization at the phase transition of the three-coloring problem. Phys. Rev. E **69**(6), 66–73 (2004)
9. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer, Berlin (2008)
10. Brelaz, D.: New methods to color vertices of a graph. Commun. ACM **22**, 251–256 (1979)
11. Brown, R.: Chromatic scheduling and the chromatic number problem. Manag. Sci. **19**(4), 456–463 (1972)
12. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for timetabling problems. Eur. J. Oper. Res. **176**(1), 177–192 (2007)
13. Chams, M., Hertz, A., de Werra, D.: Some experiments with simulated annealing for coloring graphs. Eur. J. Oper. Res. **32**, 260–266 (1987)
14. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. In: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 1, pp. 331–337. Morgan Kaufmann, San Mateo (1991)
15. Chiarandini, M., Dumitrescu, I., Stützle, T.: Stochastic local search algorithms for the graph colouring problem. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics. Computer & Information Science Series, vol. 63, pp. 1–17. Chapman & Hall/CRC, Boca Raton (2007). Preliminary version available as Tech. Rep. AIDA-05-03 at Intellectics Group, Computer Science Department, Darmstadt University of Technology, Darmstadt, Germany
16. Chiarandini, M., Stützle, T.: Online compendium to the article: an analysis of heuristics for vertex colouring. http://www.imada.sdu.dk/~marco/gcp-study/. Accessed 20 December 2010
17. Chiarandini, M., Stützle, T.: An application of iterated local search to graph coloring. In: Johnson, D.S., Mehrotra, A., Trick, M. (eds.) Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, NY, USA, September 2002, pp. 112–125 (2002)
18. Chiarandini, M., Stützle, T.: An analysis of heuristics for vertex colouring. In: Festa, P. (ed.) Proceedings of the 9th International Symposium. Lecture Notes in Computer Science, vol. 6049, pp. 326–337. Springer, Berlin (2010)
19. Chow, F.C., Hennessy, J.L.: The priority-based coloring approach to register allocation. ACM Trans. Program. Lang. Syst. **12**(4), 501–536 (1990)
20. Culberson, J.: Graph Coloring Page. http://web.cs.ualberta.ca/~joe/Coloring/. Accessed 20 December 2010
21. Culberson, J., Luo, F.: Exploring the k-colorable landscape with iterated greedy. In: Johnson, D.S., Trick, M.A. (eds.) Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, pp. 245–284. American Mathematical Society, Rhode Island (1996)
22. de Werra, D.: An introduction to timetabling. Eur. J. Oper. Res. **19**(2), 151–162 (1985)
23. de Werra, D., Eisenbeis, C., Lelait, S., Marmol, B.: On a graph-theoretical model for cyclic register allocation. Discrete Appl. Math. **93**(2–3), 191–203 (1999)
24. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
25. Dorne, R., Hao, J.K.: A new genetic local search algorithm for graph coloring. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.P. (eds.) Parallel Problem Solving from Nature—PPSN V, 5th International Conference. Lecture Notes in Computer Science, vol. 1498, pp. 745–754. Springer, Berlin (1998)

26. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**, 124–141 (1999)
27. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Berlin (2003)
28. Eiben, A.E., Van Der Hauw, J.K., Van Hemert, J.I.: Graph coloring with adaptive evolutionary algorithms. J. Heuristics **4**(1), 25–46 (1998)
29. Fister, I., Brest, J.: Using differential evolution for the graph coloring. In: Proceedings of IEEE SSCI2011 Symposium Series on Computational Intelligence, Piscataway, pp. 150–156 (2011)
30. Fleurent, C., Ferland, J.: Genetic and hybrid algorithms for graph coloring. Ann. Oper. Res. **63**, 437–464 (1996)
31. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J. Am. Stat. Assoc. **32**, 675–701 (1937)
32. Friedman, M.: A comparison of alternative tests of significance for the problem of m rankings. Ann. Math. Stat. **11**, 86–92 (1940)
33. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. **3**(4), 379–397 (1999)
34. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Comput. Oper. Res. **33**, 2547–2562 (2006)
35. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the k-coloring problem. Discrete Appl. Math. **156**(2), 267–279 (2008)
36. Gamache, M., Hertz, A., Ouellet, J.O.: A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding. Comput. Oper. Res. **34**(8), 2384–2395 (2007)
37. Gamst, A.: Some lower bounds for a class of frequency assignment problems. IEEE Trans. Veh. Technol. **35**, 8–14 (1986)
38. Garey, M.R., Johnson, D.S.: Computers and Intractability: a Guide to the Theory of NP-Completeness. Freeman, New York (1979)
39. Garey, M.R., Johnson, D.S., So, H.C.: An application of graph coloring to printed circuit testing. IEEE Trans. Circuits Syst. **23**, 591–599 (1976)
40. Glass, C.: Bag rationalization for a food manufacturer. J. Oper. Res. Soc. **53**, 544–551 (2002)
41. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**(5), 533–549 (1986)
42. Hamiez, J.P., Hao, J.K., Glover, F.: A study of tabu search for coloring random 3-colorable graphs around the phase transition. Int. J. Appl. Metaheuristic Comput. **1**(4), 1–24 (2010)
43. Hayes, B.: On the threshold. Am. Sci. **91**, 12–17 (2003)
44. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**, 345–351 (1987)
45. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. Discrete Appl. Math. **156**(13), 2551–2560 (2008)
46. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, San Francisco (2005)
47. Horowitz, E., Sahni, S.: Fundamentals of Computer Algorithms. Potomac, Maryland (1978)
48. Igel, C., Toussaint, M.: Neutrality and self-adaptation. Nat. Comput. **2**, 117–132 (2003)
49. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation, Part II; Graph coloring and number partitioning. Oper. Res. **39**(3), 378–406 (1991)
50. Johnson, D.S., Trick, M.A.: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge vol. 26. American Mathematical Society, Providence (1996)
51. Kimura, M.: Evolutionary rate at the molecular level. Nature **217**, 624–626 (1968)
52. Kubale, M.: Graph Colorings. American Mathematical Society, Rhode Island (2004)
53. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. J. Res. Natl. Bur. Stand. **84**(6), 489–506 (1979)
54. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. Eur. J. Oper. Res. **203**, 241–250 (2010)
55. Mabed, H., Caminada, A., Hao, J.K.: Genetic tabu search for robust fixed channel assignment under dynamic traffic data. Comput. Optim. Appl. (2010). doi:10.1007/s10589-010-9376-9
56. Malaguti, E., Monaci, M., Toth, P.: A metaheuristic approach for the vertex coloring problem. INFORMS J. Comput. **20**, 302–316 (2008)
57. Malaguti, E., Toth, P.: A survey on vertex coloring problems. In: International Transactions in Operational Research, pp. 1–34 (2009)
58. Merz, P., Freisleben, B.: Fitness landscapes and memetic algorithm design. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 245–260. McGraw-Hill, Cambridge (1999)

59. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin (1992)
60. Palubeckis, G.: A multistart tabu search approach for graph coloring. Inf. Technol. Control **4**(21), 7–15 (2001)
61. Petford, A.D., Welsh, D.J.A.: A randomized 3-coloring algorithm. Discrete Math. **74**, 253–261 (1989)
62. Smith, D.H., Hurley, S., Thiel, S.U.: Improving heuristics for the frequency assignment problem. Eur. J. Oper. Res. **107**(1), 76–86 (1998)
63. Stadler, P.: Towards a theory of landscapes. In: Lopez-Pena, R. (ed.) Complex Systems and Binary Networks. Lecture Notes in Physics, vol. 461, pp. 77–163. Springer, Berlin (1995)
64. Trick, M.: Network resources for coloring a graph. http://mat.gsia.cmu.edu/COLOR/color.html. Accessed 20 December 2010
65. Turner, J.S.: Almost all k-colorable graphs are easy to color. J. Algorithms **9**, 63–82 (1988)
66. Van, J.I.: Hemert. Jano's Homepage. http://www.vanhemert.co.uk/csp-ea.html. Accessed 20 December 2010