63. Graph Coloring and Recombination

Rhyd Lewis

It is widely acknowledged that some of the most powerful algorithms for graph coloring involve the combination of evolutionary-based methods with exploitative local search-based techniques. This chapter conducts a review and discussion of such methods, principally focussing on the role that recombination plays in this process. In particular we observe that, while in some cases recombination seems to be usefully combining substructures inherited from parents, in other cases it is merely acting as a macro perturbation operator, helping to reinvigorate the search from time to time.

63.1 Graph Coloring

Graph coloring is a well-known NP-hard combinatorial optimization problem that involves using a minimal number of colors to paint all vertices in a graph such that all adjacent vertices are allocated different colors. The problem is more formally stated as follows: given an undirected simple graph $G=(V,E)$, with vertex set V and edge set E, our task is to assign each vertex $v \in V$ an integer $c(v) \in \{1,2,\ldots,k\}$ so that:

- $c(v) \neq c(u) \forall \{v,u\} \in E$
- k is minimal.

Though essentially a theoretical problem, graph coloring is seen to underpin a wide variety of seemingly unrelated operational research problems, including satellite scheduling [63.1], educational timetabling [63.2, 3], sports league scheduling [63.4], frequency assignment problems [63.5, 6], map coloring [63.7], airline crew scheduling [63.8], and compiler register allocation [63.9]. The design of effective algorithms for graph coloring thus has positive implications for a large range of real-world problems.

Some common terms used with graph coloring are as follows:

- A coloring of a graph is called *complete* if all vertices $v \in V$ are assigned a color $c(v) \in \{1,\ldots,k\}$; else it is considered *partial*.
- A *clash* describes a situation where a pair of adjacent vertices $u, v \in V$ are assigned the same color (that is, $\{u,v\} \in E$ and $c(v)=c(u)$). If a coloring contains no clashes, then it is considered *proper*; else it is *improper*.
- A coloring is *feasible* if and only if it is both complete and proper.
- The *chromatic number* of a graph G, denoted $\chi(G)$, is the minimal number of colors required in a feasible coloring. If a feasible coloring uses $\chi(G)$ colors, it is considered *optimal*.
- An *independent set* is a subset of vertices $I \subseteq V$ that are mutually non-adjacent. That is, $\forall u,v \in I$, $\{u,v\} \notin E$. Similarly, a *clique* is a subset of vertices $C \subseteq V$ that are mutually adjacent: $\forall u,v \in C$, $\{u,v\} \in E$.

Given these definitions, we might also view graph coloring as a type of partitioning/grouping problem where the aim is to split the vertices into a set of subsets $\mathcal{U}=\{U_1,\ldots,U_k\}$ such that $U_i \cap U_j = \emptyset (1 \leq i < j \leq k)$.

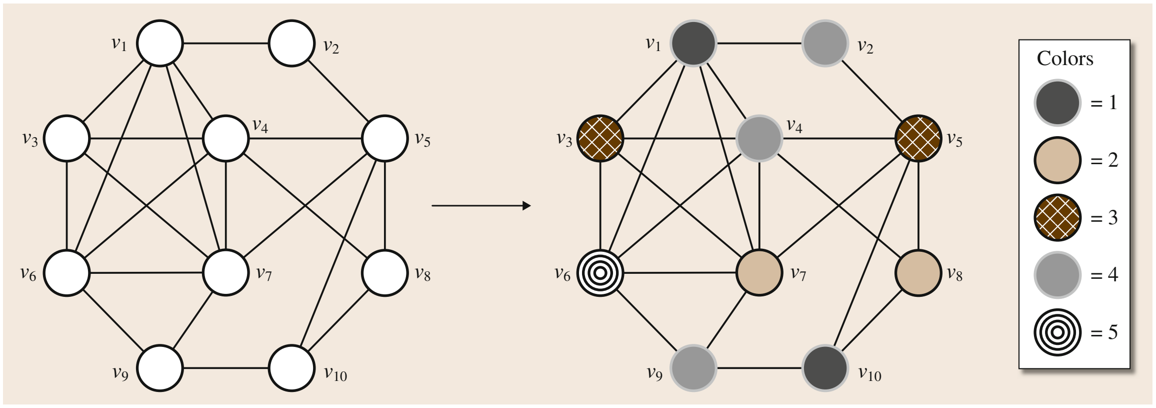


Fig. 63.1 A simple graph (*left*) and a feasible five-coloring (*right*)

If $\bigcup_{i=1}^{k} U_i = V$, then the partition represents a complete coloring. Moreover, if all subsets $U_1, \dots, U_k$ are independent sets, the coloring is also feasible.

To exemplify these concepts, Fig. 63.1 shows an example graph with ten vertices, together with a corresponding coloring. In this case the presented coloring is both complete and proper, and therefore feasible. It is also optimal because it uses just five colors, which happens to be the chromatic number in this case. The graph also contains one clique of size 5 (vertices v_1, v_3, v_4, v_6, and v_7), and numerous independent sets, such as vertices v_2, v_3, v_8, and v_9. As a partition, this coloring is represented $\mathcal{U} = \{\{v_1, v_{10}\}, \{v_7, v_8\}, \{v_3, v_5\}, \{v_2, v_4, v_9\}, \{v_6\}\}$.

It should be noted that various subsidiary problems related to the graph coloring problem are also known to be NP-hard. These include computing the chromatic number itself, identifying the size of the largest clique, and determining the size of the largest independent set in a graph [63.10, 11]. In addition, the decision variant of the graph coloring problem, which asks: *given a fixed positive integer k, is there a feasible k-coloring of the vertices?* is NP-complete.

63.2 Algorithms for Graph Coloring

Graph coloring has been studied as an algorithmic problem since the late 1960s and, as a result, an abundance of methods have been proposed. Loosely speaking, these methods might be grouped into two main classes: *constructive methods*, which build solutions step-by-step, perhaps using various heuristic and backtracking operators; and *stochastic search-based methods*, which attempt to navigate their way through a space of candidate solutions while optimizing a particular objective function.

The earliest proposed algorithms for graph coloring generally belong to the class of constructive methods. Perhaps the simplest of these is the *first-fit* (or *greedy*) algorithm. This operates by taking each vertex in turn in a specified order and assigning it to the lowest indexed color where no clash is induced, creating new colors when necessary [63.12]. A development on this method is the DSATUR algorithm [63.13, 14] in which the ordering of the vertices is determined dynamically – specifically, by choosing at each step the uncolored vertex that currently has the largest number of different colors assigned to adjacent vertices, breaking ties by taking the vertex with the largest degree. Other constructive methods have included backtracking strategies, such as those of *Brown* [63.15] and *Korman* [63.16], which may ultimately perform complete enumerations of the solution space given excess time. A survey of backtracking approaches was presented by *Kubale* and *Jackowski* [63.17] in 1985.

Many of the more recent methods for graph coloring have followed the second approach mentioned above, which is to search a space of candidate solutions and attempt to identify members that optimize a specific objective function. Such methods can be further classified according to the composition of their search spaces, which can comprise:

(a) The set of all feasible solutions (using an undefined number of colors)
(b) The set of complete colorings (proper and improper) for a fixed number of colors k
(c) The set of proper solutions (partial and complete), also for a fixed number of colors k.

Algorithms following scheme (a) have been considered by, among others, *Culberson* and *Luo* [63.18], *Mumford* [63.19], *Erben* [63.20], and *Lewis* [63.21]. Typically, these methods consider different permutations of the vertices, which are then fed into a constructive method (such as first-fit) to form feasible solutions. An intuitive cost function in such cases is simply the number of colors used in a solution, though other more fine-grained functions have been suggested, such as the following due to *Erben* [63.20]

$$f_1 = \frac{\sum_{U_i \in \mathcal{U}} \left[\sum_{v \in U_i} \deg(v)\right]^2}{|\mathcal{U}|} . \qquad (63.1)$$

Here, the term $(\sum_{v \in U_i} \deg(v))$ gives the sum of the degrees of all vertices assigned to a color class U_i. The aim is to maximize f_1 by making increases to the numerator (by forming large color classes that contain high-degree vertices), and decreases to the denominator (by reducing the number of color classes).

On the other hand, algorithms following scheme (b) operate by first proposing a fixed number of colors k. At the start of a run, each vertex will be assigned to one of the k colors using heuristics, or randomly. However, this may involve the introduction of one or more clashes, resulting in a complete, improper k-coloring. The cost of such a solution might then be evaluated using the following cost function, which is simply a count on the number of clashes

$$f_2 = \sum_{\forall \{v,u\} \in E} g(v,u) \quad \text{where}$$
$$g(v,u) = \begin{cases} 1 & \text{if } c(v) = c(u) \\ 0 & \text{otherwise} . \end{cases} \qquad (63.2)$$

The strategy in such approaches is to make alterations to a solution such that the number of clashes is reduced to zero. If this is achieved k can be reduced; alternatively if all clashes cannot be eliminated, k can be increased. This strategy has been quite popular in the literature, involving the use of various stochastic search methodologies, including simulated annealing [63.22, 23], tabu search [63.24], greedy randomized adaptive search procedure (GRASP) methods [63.25], iterated local search [63.26, 27], variable neighborhood search [63.28], ant colony optimization [63.29], and evolutionary algorithms (EA) [63.30–35].

Finally, scheme (c) also involves using a fixed number of colors k; however in this case, rather than allowing clashes to occur in a solution, vertices that cannot be feasibly assigned to a color are placed into a set of uncolored vertices S. The aim is, therefore, to make changes to a solution so that these vertices can eventually be feasibly colored, resulting in $S = \emptyset$. This approach has generally been less popular in the literature than scheme (b), though some prominent examples include the simulated annealing approach of *Morgenstern* [63.36], the tabu search method of *Blochliger* and *Zufferey* [63.37], and the EA of *Malaguti* et al. [63.38]. More recently, *Hertz* et al. [63.39] also suggested an algorithm that searches different solution spaces during different stages of a run. The idea is that when the search is deemed to have stagnated in one space, a procedure is used to alter the current solution so that it becomes a member of another space (e.g., clashing vertices are *uncolored* by transferring them to S). Once this has been done, the search can then be continued in this new space where further improvements might be made.

63.2.1 EAs for Graph Coloring

In this section we now examine the ways in which EAs have been applied to the graph coloring problem, particularly looking at issues surrounding the recombination of solutions.

Assignment-Based Operators

Perhaps the most intuitive way of applying EAs to the graph coloring problem is to view the task as one of assignment. In this case, a candidate solution can be viewed as a mapping of vertices to colors $c : V \to \{1, \ldots, k\}$, and a natural chromosome representation is a vector $(c(v_1), c(v_2), \ldots, c(v_{|V|}))$, where $c(v_i)$ gives the color of vertex v_i (the solution given in Fig. 63.1 would be represented by $(1, 4, 3, 4, 3, 5, 2, 2, 4, 1)$ under this scheme). However, it has long been argued that this sort of approach brings disadvantages, not least because it contradicts a fundamental design principle of EAs: the principle of minimum redundancy [63.40], which states that each member of the search space should be represented by as few distinct chromosomes as possible. To expand upon this point, we observe that under this *assignment-based* representation, if we are given a solution using $l \leq k$ colors, the number of different chromosomes representing this solution will be ${}^k\mathrm{P}_l$ due to the arbitrary way in which colors are allocated labels. (For example, swapping the labels of colors 2 and 4 in Fig. 63.1's solution would give a new chromosome (1,2,3,2,3,5,4,4,2,1), but the same solution.) Of course, this implies a search space that is far larger than necessary.

Furthermore, authors such as *Falkenauer* [63.41] and *Coll* et al. [63.42] have also argued that traditional recombination schemes such as 1, 2, and n-point crossover with this representation have a tendency to recklessly break up building-blocks that we might want promoted in a population. As an example, consider a recombination of the two example chromosomes given in the previous paragraph using two-point crossover: (1,4,3,4,3,**5**,**2**,**2**,4,1) crossed with (**1**,**2**,**3**,**2**,**3**,5,4,4,**2**,**1**) would give (1,4,3,4,3,5,4,4,4,1) as one of the offspring. Here, despite the fact that the two parent chromosomes actually represent the same feasible solution, the resultant offspring seems to have little in common with its parents, having lost one of its colors, and seen a number of clashes having been introduced. Thus, it is concluded by these authors that such operations actually constitute more of a random perturbation operator, rather than a mechanism for combining meaningful substructures from existing solutions. Nevertheless, recent algorithms following this scheme are still reported in the literature [63.43].

In recognition of the proposed disadvantages of the assignment-based representation, *Coll* et al. [63.42] proposed a procedure for relabeling the colors of one of the parent chromosomes before applying crossover. Consider two (not necessarily feasible) parent solutions represented as partitions: $\mathcal{U}_1 = \{U_{1,1}, \dots, U_{1,k}\}$, and $\mathcal{U}_2 = \{U_{2,1}, \dots, U_{2,k}\}$. Now, using $\mathcal{U}_1$ and $\mathcal{U}_2$, a complete bipartite graph $K_{k,k}$ is formed. This bipartite graph has k vertices in each partition, and the weights between two vertices i, j from different partitions are defined as $w_{i,j} = |U_{1,i} \cap U_{2,j}|$. Given $K_{k,k}$, a maximum weighted matching can then be determined using any suitable algorithm (e.g., the Hungarian algorithm [63.44] or auction algorithm [63.45]), and this matching can be used to re-label the colors in one of the chromosomes.

Figure 63.2 gives an example of this procedure and shows how the second parent can be altered so that its color labelings maximally match those of parent 1. In this case, we note that the color classes $\{v_1, v_{10}\}$, $\{v_3, v_5\}$, and $\{v_6\}$ occur in both parents and will be preserved in any offspring produced via a traditional crossover operator. However, this will not always be the case and will depend very much on the best matching that is available in each case.

A further scheme for color relabeling that also addresses the issue of redundancy was proposed by *Tucker* et al. [63.46]. This method involves representing solutions using the assignment-based scheme, but under the following restriction

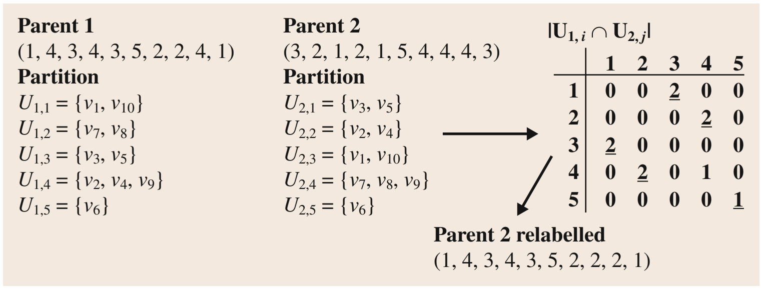
$$c(v_1) = 1\,, \tag{63.3}$$

$$c(v_{i+1}) \le \max\{c(v_1), \dots, c(v_i)\} + 1\,. \tag{63.4}$$

Chromosomes obeying these labeling criteria might, therefore, be considered as being in their canonical form such that, by definition, vertex v_1 is always colored with color 1, v_2 is always colored with color 1 or 2, and so on. (The solution given in Fig. 63.1 would be represented by (1,2,3,2,3,4,5,5,2,1) under this scheme.) However, although this ensures a one-to-one correspondence between the set of chromosomes and the set of vertex partitions (thereby removing any redundancy), research by *Lewis* and *Pullin* [63.47] demonstrated that this scheme is not particularly useful for graph coloring, not least because minor changes to a chromosome (such as the recoloring a single vertex) can lead to major changes to the way colors are labeled, making the propagation of useful solution substructures more difficult to achieve when applying traditional crossover operators.

Partition-Based Operators

Given the proposed issues with the assignment-based approach, the last 15 years or so have also seen a num-



Parent 1
(1, 4, 3, 4, 3, 5, 2, 2, 4, 1)
Partition
$U_{1,1} = \{v_1, v_{10}\}$
$U_{1,2} = \{v_7, v_8\}$
$U_{1,3} = \{v_3, v_5\}$
$U_{1,4} = \{v_2, v_4, v_9\}$
$U_{1,5} = \{v_6\}$

Parent 2
(3, 2, 1, 2, 1, 5, 4, 4, 4, 3)
Partition
$U_{2,1} = \{v_3, v_5\}$
$U_{2,2} = \{v_2, v_4\}$
$U_{2,3} = \{v_1, v_{10}\}$
$U_{2,4} = \{v_7, v_8, v_9\}$
$U_{2,5} = \{v_6\}$

$|U_{1,i} \cap U_{2,j}|$

	1	2	3	4	5
1	0	0	2	0	0
2	0	0	0	2	0
3	2	0	0	0	0
4	0	2	0	1	0
5	0	0	0	0	1

Parent 2 relabelled
(1, 4, 3, 4, 3, 5, 2, 2, 2, 1)

Fig. 63.2 Example of the relabeling procedure proposed by *Coll* et al. [63.42]. Here, parent 2 is relabeled as $1 \to 3$, $2 \to 4$, $3 \to 1$, $4 \to 2$, and $5 \to 5$

ber of articles presenting recombination operators focussed on the partition (or grouping) interpretation of graph coloring. The philosophy behind this approach is that it is actually the color classes (and the vertices that are assigned to them) that represent the underlying building blocks of the graph coloring problem. In other words, it is not the color of individual vertices per se, but the *way in which vertices are grouped* that form the meaningful substructures. Consequently, the focus should be on the design of operators that are successfully able to combine and promote these within a population.

Perhaps the first major work in this area was due to *Falkenauer* [63.48] in 1994 (and later [63.41]) who argued in favor of the partition interpretation in the justification of his grouping genetic algorithm (GGA) – an EA methodology specifically designed for use with partitioning problems. *Falkenauer* applied this GGA to two important operational research problems: the bin-packing problem and bin-balancing problem, with strong results being reported. In subsequent work, *Erben* [63.20] also tailored the GGA for graph coloring. Erben's approach operates in the space of feasible colorings and allows the number of colors in a solution to vary. Solutions are then stored as partitions, and evaluated using (63.1). In this approach, recombination operates by taking two parent solutions and randomly selecting a subset of color classes from the second. These color classes are then copied into the first parent, and all color classes coming from the first parent containing duplicate vertices are deleted. This operation results in an offspring solution that is proper, but most likely partial. Thus uncolored vertices are then reinserted into the solution, in this case using the first-fit algorithm. A number of other recombination operators for use in the space of feasible solutions have also been suggested by *Mumford* [63.19]. These operate on permutations of vertices, which are again decoded into solutions using the first-fit algorithm.

Another recombination operator that focusses on the partition interpretation of graph coloring is due to *Galinier* and *Hao*, who in 1999 proposed an EA that, at the date of writing, is still understood to be one of the best performing algorithms for graph coloring [63.33, 38, 49, 50]. Using a fixed number of colors k, Galinier and Hao's method operates in the space of complete (proper and improper) k-colorings using cost function f_2 (63.2). A population of candidate solutions is then evolved using local search (based on tabu search) together with a specialized recombination operator called greedy partition crossover (GPX). The latter is used as a global operator and is intended to guide the search over the long term, gently directing it towards favorable regions of the search space (exploration), while the local search element is used to identify high quality solutions within these regions (exploitation).

The idea behind GPX is to construct offspring using large color classes inherited from the parent solutions. A demonstration of how this is done is given in Fig. 63.3. As is shown, the largest (not necessarily proper) color class in the parents is first selected and copied into the offspring. Then, in order to avoid dupli-

		Parent 1	Parent 2	Offspring	
a)	$U_1 =$	$\{v_1, v_2, v_3\}$	$\{v_3, v_4, v_5, v_7\}$	$\{\}$	Select the color with most vertices and copy to the
	$U_2 =$	$\{v_4, v_5, v_6, v_7\}$	$\{v_1, v_6, v_9\}$	$\{\}$	child (U_2 from parent 1 here).
	$U_3 =$	$\{v_8, v_9, v_{10}\}$	$\{v_2, v_8, v_{10}\}$	$\{\}$	Delete copied vertices from both parents.
b)	$U_1 =$	$\{v_1, v_2, v_3\}$	$\{v_3\}$	$\{v_4, v_5, v_6, v_7\}$	Select the color with most vertices in parent 2 and
	$U_2 =$	$\{\}$	$\{v_1, v_9\}$	$\{\}$	copy to child.
	$U_3 =$	$\{v_8, v_9, v_{10}\}$	$\{v_2, v_8, v_{10}\}$	$\{\}$	Delete copied vertices from both parents.
c)	$U_1 =$	$\{v_1, v_3\}$	$\{v_3\}$	$\{v_4, v_5, v_6, v_7\}$	Select the color with most vertices in parent 1 and
	$U_2 =$	$\{\}$	$\{v_1, v_9\}$	$\{v_2, v_8, v_{10}\}$	copy to the child.
	$U_3 =$	$\{v_9\}$	$\{\}$	$\{\}$	Delete copied vertices from both parents.
d)	$U_1 =$	$\{\}$	$\{\}$	$\{v_4, v_5, v_6, v_7\}$	Having formed k colors, assign any missing vertices
	$U_2 =$	$\{\}$	$\{v_9\}$	$\{v_2, v_8, v_{10}\}$	to random colors.
	$U_3 =$	$\{v_9\}$	$\{\}$	$\{v_1, v_3\}$	
e)	$U_1 =$	$\{\}$	$\{v_9\}$	$\{v_4, v_5, v_6, v_7\}$	A complete (though not necessarily proper) solution
	$U_2 =$	$\{v_9\}$	$\{\}$	$\{v_2, v_8, v_{10}, v_9\}$	results.
	$U_3 =$	$\{\}$	$\{\}$	$\{v_1, v_3\}$	

Fig. 63.3 Demonstration of the GPX operator using $k = 3$

cate vertices occurring in the offspring at a later stage, these copied vertices are removed from both parents. To form the next color, the other (modified) parent is then considered and, again, the largest color class is selected and copied into the offspring, before again removing these vertices from both parents. This process is continued by alternating between the parents until the offspring's k color classes have been formed. At this point, each color class in the offspring will be a subset of a color class existing in one or both of the parents. That is

$$\forall U_i \in \mathcal{U}_c\ \exists U_j \in (\mathcal{U}_1 \cup \mathcal{U}_2)\ :\ U_i \subseteq U_j\,, \tag{63.5}$$

where $\mathcal{U}_c$, $\mathcal{U}_1$, and $\mathcal{U}_2$ represent the offspring, and parents 1 and 2, respectively.

One feature of the GPX operator is that on production of an offspring's k color classes, some vertices may be missing (this occurs with vertex v_9 in Fig. 63.3). *Galinier* and *Hao* [63.33] suggest assigning these uncolored vertices to random classes, which of course could introduce further clashes. This element of the procedure might, therefore, be viewed as a type of perturbation (mutation) operator in which the number of random assignments (the size of the perturbation) is determined by the construction stages of GPX. However, *Glass* and *Prugel-Bennett* [63.49] observe that GPX's strategy of inheriting the largest available color class at each step (as opposed to a random color class) generally reduces the number of uncolored vertices. This means that the amount of information inherited directly from the parents is increased, reducing the potential for disruption. Once a complete offspring is formed, it is then modified and improved via a local search procedure before being inserted into the population.

Since the proposal of GPX by *Galinier* and *Hao* [63.33], further recombination schemes based on this method have also been suggested, differing primarily in the criteria used for selecting the color classes that are inherited by the offspring. *Lü* and *Hao* [63.34], for example, extended the GPX operator to allow more than two parents to play a part in producing a single offspring (Sect. 63.5). On the other hand, *Porumbel* et al. [63.35] suggest that instead of choosing the largest available color class at each stage of construction, classes with the least number of clashes should be prioritized, with class size (and information regarding the degrees of the vertices) then being used to break ties. *Malaguti* et al. [63.38] also use a modified version of GPX with an EA that navigates the space of partial, proper solutions. In all of these cases the authors combined their recombination operators with a local search procedure in the same manner as *Galinier* and *Hao* [63.33] and, with the problem instances considered, the reported results are generally claimed to be competitive with the state of the art.

Assessing the Effectiveness of EAs for Graph Coloring

In recent work carried out by the author of this chapter [63.50], a comparison of six different graph coloring algorithms was presented. This study was quite broad and used over 5000 different problem instances. Its conclusions were also rather complex, with each method outperforming all others on at least one class of problems. However, a salient observation was that the GPX-based EA of *Galinier* and *Hao* [63.33] was by far the most consistent and high-performing algorithm across the comparison.

In the remainder of this chapter we pursue this matter further, particularly focussing on the role that GPX plays in this performance. Under a common EA framework, described in Sect. 63.3, we first evaluate the performance of GPX by comparing it to two other recombination operators (Sect. 63.4). Using information gained from these experiments, Sect. 63.5 then looks at how the performance of the GPX-based EA might be enhanced, particularly by looking at ways in which population diversity might be prolonged during a run. Finally, conclusions and a further discussion surrounding the virtues of recombination in this problem domain are presented in Sect. 63.6.

63.3 Setup

The EA used in the following experiments operates in the same manner as *Galinier* and *Hao's* [63.33]. To form an initial population, a modified version of the DSATUR algorithm is used. Specifically, each individual is formed by taking the vertices in turn according to the DSATUR heuristic and then assigning it to the lowest indexed color $i \in \{1, \dots, k\}$ where no clash occurs. Vertices for which no clash-free color exists are assigned to random colors at the end of this process. Ties in the DSATUR heuristic are broken randomly, providing di-

Table 63.1 Details of the five problem instances used in our analysis

#: Name	$\lvert V\rvert$	Density	Vertex degree min; med; max	Mean	SD	Best known (colors)
1: Random	1000	0.499	450; 499; 555	499.4	16.1	83
2: Flat(10)	500	0.103	36; 52; 61	51.7	4.4	10
3: Flat(100)	500	0.841	393; 421; 445	420.7	7.6	100
4: TT(A)	682	0.128	0; 77; 472	87.4	62.0	27
5: TT(B)	2419	0.029	0; 47; 857	71.3	92.3	32

versity in the initial population. Each individual is then improved by the local search routine.

The EA evolves the population using recombination, local search, and replacement pressure. In each iteration two parent solutions are selected at random, and the selected recombination operator is used to produce one offspring. This offspring is then improved via local search and inserted into the population by replacing the weaker of its two parents.

The local search element of this EA makes use of tabu search – specifically the TabuCol algorithm of *Hertz* and *de Werra* [63.24], run for a fixed number of iterations. In this method, moves in the search space are achieved by selecting a vertex v whose assignment to color i is currently causing a clash, and moving it to a new color $j \neq i$. The inverse of this move is then marked as tabu for the next t steps of the algorithm (meaning that v cannot be re-assigned to color i until at least t further moves have been performed). In each iteration, the complete neighborhood is considered, and the non-tabu move that is seen to invoke the largest decrease in cost (or failing that, the smallest increase) is performed. Ties are broken randomly, and tabu moves are also carried out if they are seen to improve on the best solution observed so far in the process. The tabu search routine terminates when the iteration limit is reached (at which case the best solution found during the process is taken), or when a zero cost solution is achieved. Further descriptions of this method, including implementation details, can be found in [63.51].

In terms of parameter settings, in all cases we use a population size of 20 (as in [63.34, 35]) and set the tabu search iteration limit to $16|V|$, which approximates the settings used in the best reported runs in [63.33]. As with other algorithms that use this local search technique [63.29, 33, 37], the tabu tenure t is made proportional to the current solution cost: specifically, $t = \lceil 0.6 f_2 \rceil + r$, where r is an integer uniformly selected from the range 0–9 inclusive.

Finally, because this algorithm operates in the space of complete k-colorings (proper and improper), values for k must be specified. In our case, initial values are determined by executing DSatur on each instance and setting k to the number of colors used in the resultant solution. During runs, k is then decremented by 1 as soon as a feasible k-coloring is found, and the algorithm is restarted. Computational effort is measured by counting the number of constraint checks carried out by the algorithm, which occur when the algorithm requests information about a problem instance, including checking whether two vertices are adjacent (by accessing an adjacency list or matrix), and referencing the degree of a vertex. In all trials a cut-off point of 5×10^{11} checks is imposed, which is roughly double the length of the longest run performed in [63.33]. In our case, this led to run times of ≈ 1 h on our machines (algorithms were coded in C++ and executed on a PC under Windows XP using a 3.0 GHz processor with 3.18 GB of RAM).

63.3.1 Problem Instances

For our trials a set of five problem instances is considered. Though this set is quite small, its members should be considered as case studies that have been deliberately chosen to cover a wide range of graph structure – a factor that we have found to be very important in influencing the relative performance of graph coloring algorithms [63.50]. The first three graphs are generated using the publicly available software of *Culberson* [63.52], while the remaining two are taken from a collection of real-world timetabling problems compiled by *Carter* et al. [63.53]. Names and descriptions of these graphs now follow. Further details are also given in Table 63.1:

#1: *Random*. This graph features $|V| = 1000$ and is generated such that each of the $\binom{|V|}{2}$ pairs of vertices is linked by an edge with probability 0.5. Graphs of this nature are nearly always considered in comparisons of coloring algorithms.

#2: *Flat(10)*. Flat graphs are generated by partitioning the vertices into K equi-sized groups and then adding edges between vertices in different groups with probability p. This is done such that the variance in vertex degrees is kept to a minimum. It is well known that feasible K-colored solutions to such graphs are generally easy to achieve except in cases where p is within a specific range of values, which results in problems that are notoriously difficult. Such ranges are commonly termed *phase transition regions* [63.54]. This particular instance is generated so that it features a relatively small number of large color classes (using $V = 500$ and $K = 10$, implying ≈ 50 vertices per color). A value of $p = 0.115$ is used, which has been observed to provide very difficult instances for a range of different graph coloring algorithms [63.50].

#3 *Flat(100)*. This graph is generated in the same manner as the previous one, using $|V| = 500, K = 100$, and $p = 0.85$. Solutions thus feature a relatively large number of small color classes (≈ 5 vertices per color).

#4: *TT(A)*. This graph is named *car_s_91* in the original dataset of *Carter* et al. [63.53]. It is chosen because it is quite large and, unlike the previous three graphs, the variance in vertex degrees is quite high. This problem's structure is also much less regular than the previous three graphs, which are generated in a fairly regimented manner.

#5: *TT(B)*. This graph, originally named *pur_s_93*, is the largest problem in Carter's dataset, with $|V| = 2419$. It is also quite sparse compared to the previous graph, though it still features a high variance in vertex degrees (Table 63.1).

The rightmost column of Table 63.1 also gives information on the best solutions known for each graph. These values were determined via extended runs of our algorithms, or due to information provided by the problem generator.

63.4 Experiment 1

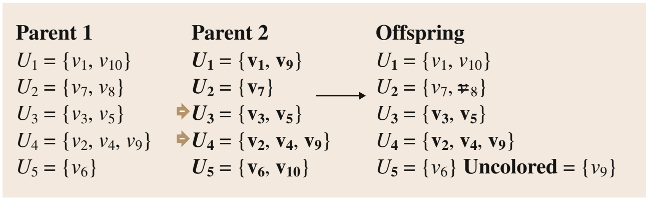
Our first set of experiments looks at the performance of GPX by comparing it to two additional recombination operators. To gauge the advantages of using a global operator (recombination in this case), we also consider the performance of TABUCOL on its own, which iterates on a single solution until the run cut-off point is met.

Our first additional recombination operator follows the assignment-based scheme discussed in Sect. 63.2.1 and, in each application, utilizes the procedure of *Coll* et al. [63.42] (Fig. 63.2) to relabel the second parent. Offspring are then formed using the classical n-point crossover, with each gene being inherited from either parent with probability 0.5.

Our second recombination operator is based on the grouping genetic algorithm (GGA) methodology (Sect. 63.2.1), adapted for use in the space of k-colorings. An example is given in Fig. 63.4. Given two parents, the color classes in the second parent are first relabeled using Coll et al.'s procedure. Using the partition-based representations of these solutions, a subset of colors in parent 2 is then chosen randomly, and these replace the corresponding colors in a copy of parent 1. Duplicate vertices are then removed from color classes originating from parent 1 and uncolored vertices are assigned to random color classes. Note that like GPX, before uncolored vertices are assigned, the property defined by (63.5) is satisfied by this operator; however, unlike GPX there is no requirement to inherit larger color classes or to inherit half of its color classes from each parent.

A summary of the results achieved by the three recombination operators (together with TABUCOL) is given in Table 63.2. For each instance the same set of 20 initial populations was used with the EAs, and entries in bold signify samples that are significantly different to the non-bold EA entries according to a Wilcoxon signed-rank test at the 0.01 significance level. For graph #1 we see that GPX has clearly produced the best results – indeed, even its worst result features two fewer colors than the next best solution. However, for graphs #2 and #5, no significant difference between the EAs is observed, while for #3 and #4, better results are produced by the GGA and the n-point crossover.

Figure 63.5 shows run profiles for two example graphs. We see that in both cases TABUCOL provides



Parent 1		**Parent 2**		**Offspring**
$U_1 = \{v_1, v_{10}\}$		$\boldsymbol{U_1 = \{v_1, v_9\}}$		$U_1 = \{v_1, v_{10}\}$
$U_2 = \{v_7, v_8\}$		$\boldsymbol{U_2 = \{v_7\}}$	$\longrightarrow$	$U_2 = \{v_7, \cancel{v_8}\}$
$U_3 = \{v_3, v_5\}$	➔	$\boldsymbol{U_3 = \{v_3, v_5\}}$		$U_3 = \{\boldsymbol{v_3, v_5}\}$
$U_4 = \{v_2, v_4, v_9\}$	➔	$\boldsymbol{U_4 = \{v_2, v_4, v_9\}}$		$U_4 = \{\boldsymbol{v_2, v_4, v_9}\}$
$U_5 = \{v_6\}$		$\boldsymbol{U_5 = \{v_6, v_{10}\}}$		$U_5 = \{v_6\}$ **Uncolored** $= \{v_9\}$

Fig. 63.4 Demonstration of the GGA recombination operator. Here, color classes in parent 2 are labeled to maximally match those of parent 1

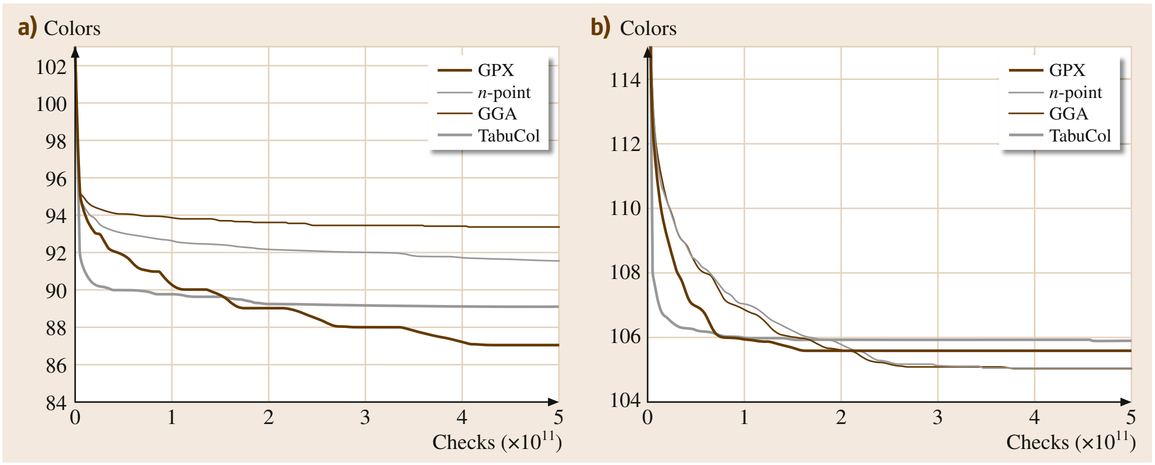


Fig. 63.5a,b Run profiles for the instances (mean of 20 runs): **(a)** #1 (random). **(b)** #3 (Flat 100)

the fastest rates of improvement, though it is eventually overtaken by at least one of the EAs. Table 63.2, however, also reveals that TabuCol performs very poorly with graphs #4 and #5. This seems due to the high degree variance in these cases, which we observe makes the cost of neighboring solutions in the search space vary more widely. This suggests a more *spiky* cost landscape in which the use of local search in isolation exhibits a susceptibility for becoming trapped at local optima (see also [63.50]).

An important factor behind the differing performances of these EAs is the effect that recombination has on the population diversity. To examine this, we first define a metric for measuring the distance between two solutions: Given a solution $\mathcal{U}$, let $P_{\mathcal{U}} = \{\{u, v\} : c(u) = c(v)\}$, for $\forall u, v \in V$, $u \neq v$. The *distance* between two solutions $\mathcal{U}_1$ and $\mathcal{U}_2$ can then be defined,

$$D(\mathcal{U}_1, \mathcal{U}_2) = \frac{|P_{\mathcal{U}_1} \cup P_{\mathcal{U}_2}| - |P_{\mathcal{U}_1} \cap P_{\mathcal{U}_2}|}{|P_{\mathcal{U}_1} \cup P_{\mathcal{U}_2}|} \,. \quad (63.6)$$

This measure gives the proportion of vertex pairings (assigned to the same color) that exist in just one of the two solutions. Consequently, if $\mathcal{U}_1$ and $\mathcal{U}_2$ are identical, then $P_{\mathcal{U}_1} \cup P_{\mathcal{U}_2} = P_{\mathcal{U}_1} \cap P_{\mathcal{U}_2}$, giving $D(\mathcal{U}_1, \mathcal{U}_2) = 0$. Conversely, if no vertex pair is assigned the same color, $P_{\mathcal{U}_1} \cap P_{\mathcal{U}_2} = \emptyset$, implying $D(\mathcal{U}_1, \mathcal{U}_2) = 1$. Population diversity can also be defined as the mean distance between each pair of solutions in the population. That is, given a set of m individuals $\mathbf{U} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_m\}$

$$\text{Diversity}(\mathbf{U}) = \frac{1}{\binom{m}{2}} \sum_{\forall \mathcal{U}_i, \mathcal{U}_j \in \mathbf{U} : i<j} D(\mathcal{U}_i, \mathcal{U}_j) \,. \quad (63.7)$$

Considering our results, the two scatter plots of Fig. 63.6 demonstrate the positive correlation that exists between parental distance and the number of uncolored vertices that result in applications of the GPX and GGA operators. This data was derived from graph #4, though similar patterns were observed for the other instances. Note that the correlation is weaker for GGA due to two reasons. First, unlike GPX, which requires half of the color classes to be inherited from each parent, with GGA this proportion can vary. Thus if the majority of color classes are inherited from just one par-

Table 63.2 Number of colors in the best feasible solution achieved at the cut-off point (mean (min; median; max) of 20 runs)

	GPX		*n*-point		GGA		TabuCol	
#1	**87.00**	**(87; 87; 87)**	93.35	(93; 93; 94)	91.55	(91; 92; 92)	89.10	(89; 89; 90)
#2	12.95	(12; 13; 13)	13.00	(13; 13; 13)	13.00	(13; 13; 13)	13.00	(13; 13; 13)
#3	105.60	(105; 106; 106)	**105.05**	**(105; 105; 106)**	**105.05**	**(105; 105; 106)**	105.90	(105; 106; 106)
#4	29.05	(28; 29; 30)	**28.00**	**(28; 28; 28)**	**27.90**	**(27; 28; 29)**	38.20	(32; 37.5; 46)
#5	33.30	(33; 33; 34)	33.15	(32; 33; 34)	33.10	(32; 33; 34)	52.05	(47; 52; 56)

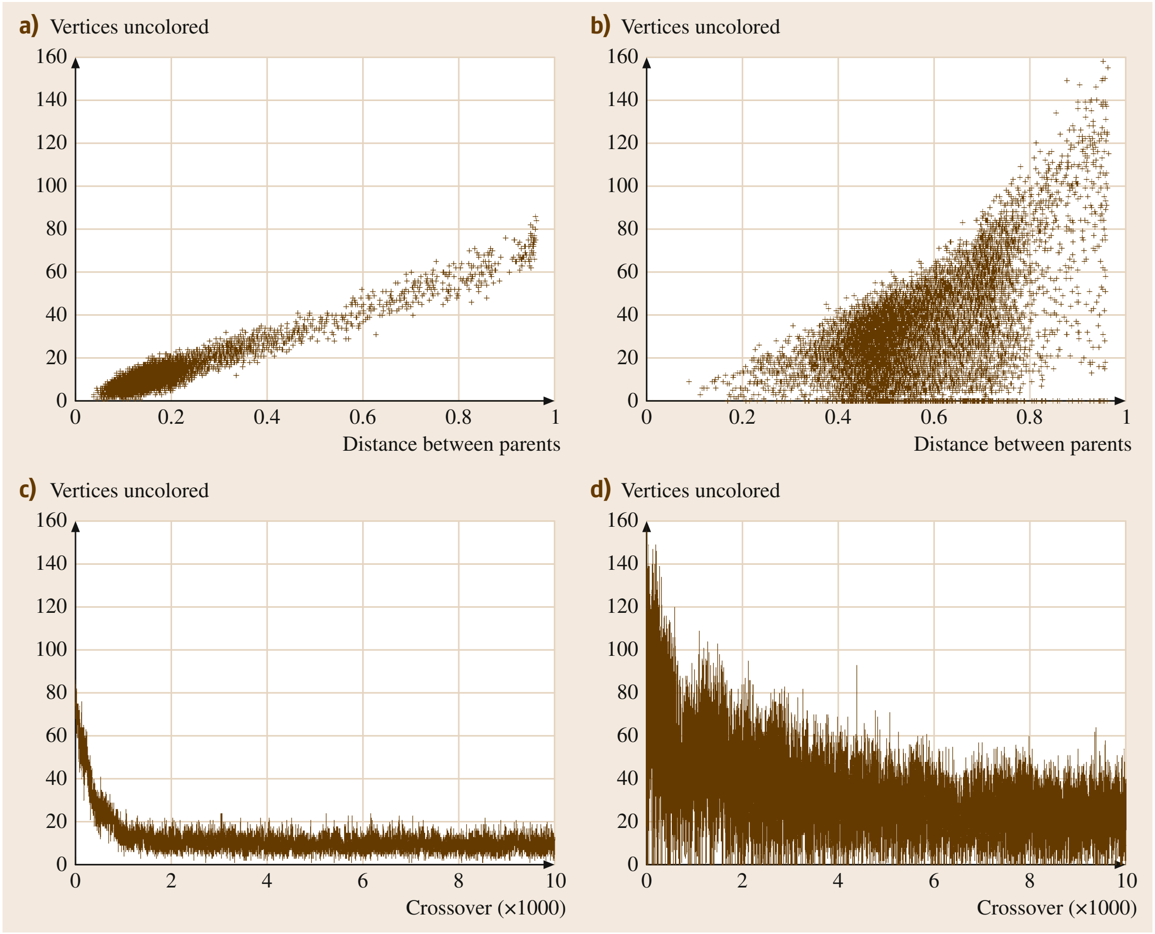


Fig. 63.6a–d Relationship between parental distance and number of uncolored vertices with the GPX **(a)** and GGA **(b)** operators. Also shown is the number of uncolored vertices in the first 10 000 applications of GPX **(c)** and GGA **(d)**

ent, it is possible to have two very different parents, but only a small number of uncolored vertices. Second, as mentioned earlier GGA shows no bias towards inheriting larger color classes, meaning that the number of uncolored vertices can also be higher than GPX, particularly when inheriting around half of the color classes from each parent. An effect of these patterns is shown in the lower graphs of Fig. 63.6, where throughout the evolutionary process, the number of uncolored vertices occurring during recombination is fewer and less varied with GPX. In comparison to GGA, this behavior leads to a more rapid loss of diversity, as is demonstrated in Fig. 63.7 for two example graphs.

Whether sustained diversity is a help or hindrance with these EAs thus seems to depend on the type of graph being tackled. As can be seen in Fig. 63.7, for graph #1 GPX is the only recombination operator that leads to any sort of population convergence, and it is also the algorithm that produces the best solutions given sufficient time, suggesting that is suitably *homing in* on high-quality regions of the search space. On the other hand, for graphs #3 and #4, GGA's more sustained diversity (caused and perpetuated by the greater number of uncolored vertices that occur during recombination) causes the operator to be more disruptive. However, in these cases this factor also seems to provide a useful diversification mechanism, allowing the algorithm to sample wider areas of the search space, leading to better results. An extreme case of diversity loss occurs with graph #5, which we recall has a low density and high degree variance. In this case, when using GPX large color classes of low-degree vertices that are formed in

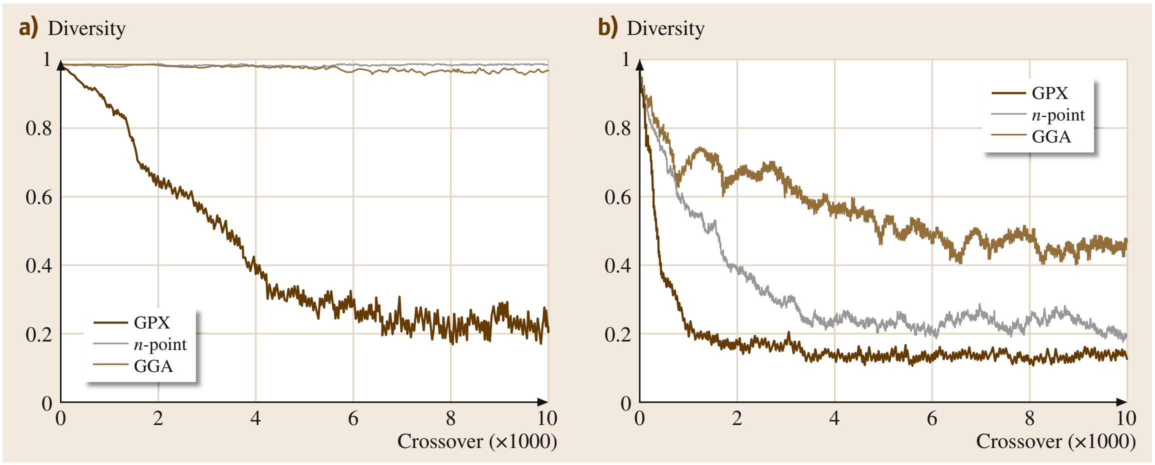


Fig. 63.7a,b Population diversity during the first 10 000 recombinations with **(a)** the random (#1) and **(b)** TT(A) (#4) instances

early stages of the algorithm quickly come to dominate the population limiting the exploration that then takes place – indeed, in many runs the algorithm was actually unable to improve on costs achieved in the initial population.

Figure 63.7 also shows that *n*-point crossover tends to maintain diversity for longer periods than GPX in this case, allowing it to produce superior results for graphs #3 and #4. However, the sustained diversity is not due to uncolored vertices (which do not occur with this operator); rather, it seems due to the naturally occurring disruption that results from the color labeling issues mentioned in Sect. 63.2.1.

Finally, we also mention that during our runs with these EA's, the local search element was observed to be by far the most expensive part of the algorithm, with none of the recombination operators consuming more than 1.8% of the available run time.

63.5 Experiment 2

In this section we now consider ways in which the results of the GPX operator might be improved, particularly looking at how we might encourage diversity to be sustained in the population.

As mentioned in Sect. 63.2.1, *Lü* and *Hao* [63.34] previously proposed extending the GPX operator to allow offspring to be produced using $m \geq 2$ parents. In this operator, which we call MULTIX, offspring are constructed in the same manner as GPX, except that at each stage the largest color class from *multiple parents* is chosen to be copied into the offspring. The intention behind this increased choice is that larger color classes will be identified, resulting in fewer uncolored vertices once the k color classes have been constructed. In order to prohibit too many colors being inherited from one particular parent, Lü and Hao also make use of a parameter q, specifying that if the i-th color class in an offspring is copied from a particular parent, then this parent should not be considered for further q colors. In our application of MULTIX we follow the recommendations of the Lü and Hao, choosing m randomly from the set $\{2, \dots, 6\}$ in each application and using $q = \lfloor m/2 \rfloor$. Note also that GPX is simply an application of MULTIX using $m = 2$ and $q = 1$.

Though having the potential to produce good results [63.34], an issue with MULTIX is that it could result in diversity being lost even more rapidly than with GPX, particularly if fewer vertices need to be randomly recolored at the end of each application. In [63.34], *Lü* and *Hao* attempt to deal with this using a mechanism whereby offspring are only inserted into the population if they are seen to be sufficiently different or better than existing members. However, in our case, we suggest two alternative methods.

The first of these involves altering the MULTIX operator so that it works exclusively with proper col-

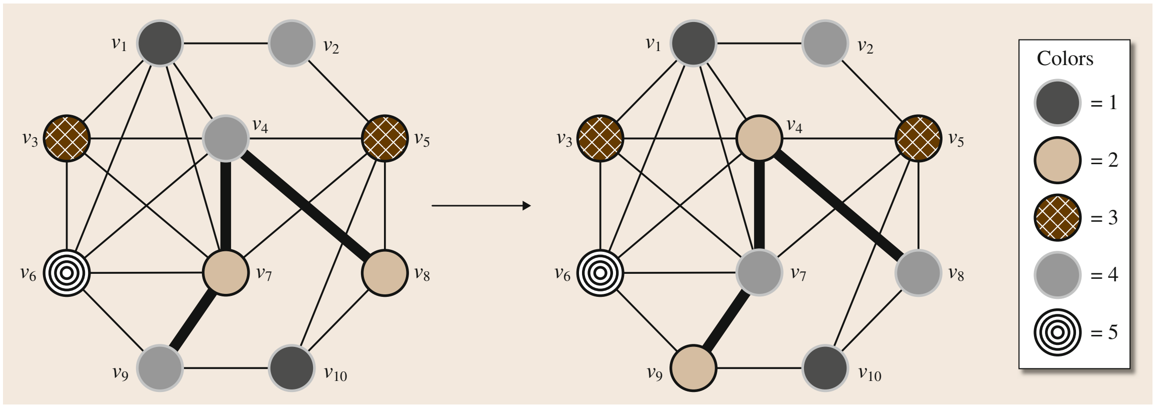


Fig. 63.8 Example Kempe chain involving, e.g., vertex v_7 and color 4 (*left*), and the resultant coloring due to a color interchange (*right*)

orings. As noted, GPX and MULTIX currently operate on colorings in which clashes are permitted; however, this could in theory result in large color classes that feature many clashes being unduly promoted in the population, when perhaps the real emphasis should be on the promotion of large color classes that are *independent sets*. The ISETS approach thus operates by first iteratively removing clashing vertices from each parent (in a random order, until proper colorings are achieved), and then using the MULTIX operator to produce an offspring as before. This implies that, before recoloring missing vertices, offspring will also be proper, since subsets of independent sets are themselves independent sets. A further effect is that a greater number of vertices might need to be recolored, since vertices originally removed from the parents could also be missing in the resultant offspring.

Our second proposal for prolonging diversity is to make changes directly to an offspring to try to increase its distance from its parents before reinsertion into the population. One way of doing this would be to increase the iteration limit of the local search procedure, as demonstrated by *Galinier* and *Hao* [63.33]. However, we find that such an approach can slow the algorithm unnecessarily, particularly because as the procedure progresses, movements in the search space (due to improving or sideways moves) become less frequent. An alternative in this case is to exploit the structure of the graph coloring problem via the use of a Kempe chain interchange operator. Kempe chains define connected sub-graphs that involve exactly two colors, and can be generated by taking an arbitrary vertex v and color i, such that $c(v) \neq i$. An example is given in Fig. 63.8. Note that when interchanging the colors of vertices in a Kempe chain, if the original coloring is proper, then so is the new coloring. Thus we have the opportunity to quickly alter colorings without compromising their quality.

Our KEMPE approach operates in the same manner as ISETS, except that before reassigning uncolored vertices, a series of randomly selected Kempe chain interchanges are performed on the existing proper coloring. In our case, $2k$ such moves are applied.

The results achieved by our three modifications are summarized in Table 63.3, where bold entries signify samples that are significantly different to GPX at significance level 0.01. We see that improvements over GPX were only obtained on graph #1, where all three variants were successful, and graph #4 using the KEMPE variant. In practice, we found that MULTIX causes diversity to be lost more quickly than GPX with these graphs – however, the ISETS mechanism did not seem to alter this behavior a great deal, usually because the number of clashing vertices needing to be removed was quite small (less than 10).

Surprisingly, we also found that the KEMPE variant was only able to maintain higher levels of diversity with instances #4 and #5. For graphs #1, #2, and #3, it turns out that when using a suitably low number of colors k, the bipartite graphs induced by most pairs of color classes in a solution are connected. In these cases, all of the vertices belonging to the two color classes are included in the Kempe chain, meaning that a color interchange does not alter the structure of the solution, but merely produces a relabeling of the two color classes. (An example of such a Kempe chain would occur in Fig. 63.8 using vertex v_3 and color 2.) This is not the case for the less structured graphs #4 and #5, where we

Table 63.3 Number of colors in the best feasible coloring achieved at the cut-off point (mean (min; median; max) from 20 runs)

	GPX		MULTIX		ISETS		KEMPE	
#1	87.00	(87; 87; 87)	**85.00**	**(85; 85; 85)**	**85.05**	**(85; 85; 86)**	**85.15**	**(85; 85; 86)**
#2	12.95	(12; 13; 13)	13.00	(13; 13; 13)	13.00	(13; 13; 13)	12.90	(12; 13; 13)
#3	105.60	(105; 106; 106)	105.55	(105; 106; 106)	105.85	(105; 106; 106)	105.30	(105; 105; 106)
#4	29.05	(28; 29; 30)	29.10	(29; 29; 30)	29.00	(28; 29; 30)	**28.00**	**(28; 28; 28)**
#5	33.30	(33; 33; 34)	33.30	(33; 33; 34)	33.30	(33; 33; 34)	33.30	(33; 33; 34)

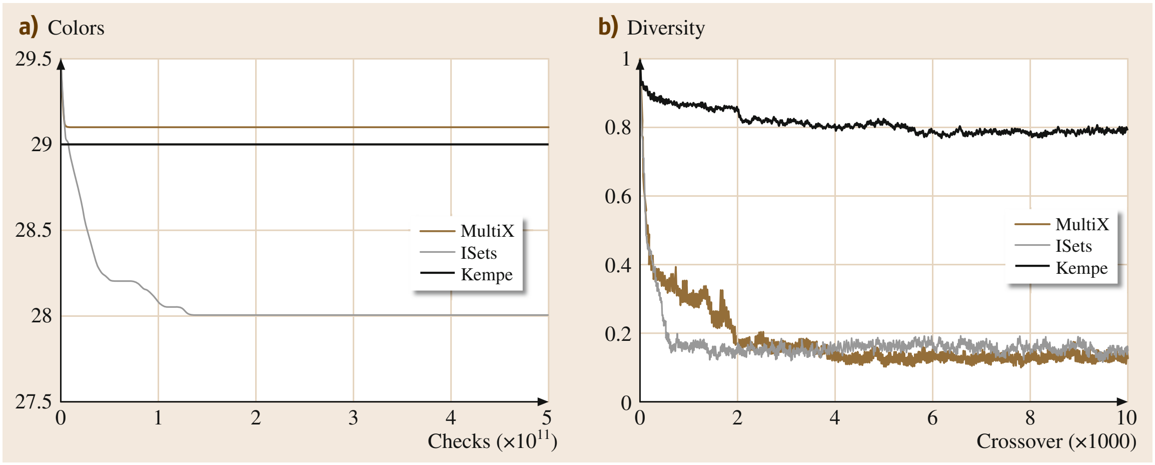


Fig. 63.9 **(a)** Run profile for TT(A) (graph #4, left), and **(b)** its diversity over the first 10 000 recombinations

found that diversity could be maintained for longer periods. However, this only led to significant improvements in the results for graph #4, whose run profiles are shown in Fig. 63.9. Also note that these enhanced results still fail to beat those of the GGA and *n*-point operators, as shown in Table 63.2.

63.6 Conclusions and Discussion

In this chapter we have examined the relative performance of a number of different graph coloring recombination operators. Using a common evolutionary framework, we have seen that this performance varies, particularly due to the underlying structures of the graphs being tackled.

A desirable property of recombination is that it should be able to combine meaningful substructures of existing candidate solutions (parents) in the production of new, hopefully fitter, offspring. However, does that process actually occur with any of these operators? Or, by involving the random reassignment of some vertices, do the operators simply provide a mechanism by which large random perturbations are periodically applied to a solution, helping to re-invigorate the search process?

Again, the answer to such a question seems to depend on the problem instance at hand. In Table 63.4 we compare the costs of solutions achieved by the best available recombination operator for each instance, together with those produced by a corresponding random perturbation operator. Specifically, for each graph we identified the best run from the EA's sample of 20 and recorded the number of uncolored vertices that resulted in each application of recombination. We then used these figures, together with the same *k*-value, to specify the number of vertices that would be randomly selected and reassigned in each corresponding application of our random perturbation operator. In each iteration this algorithm then operated by selecting two parents, making a copy of parent 1, randomly perturbing this copy, ap-

Table 63.4 Comparison of the best EA and corresponding random perturbation operator. (Cost of best solutions using f_2 (63.2); (mean, (min; median; max) from 20 runs), and proportion of runs where $f_2 = 0$ (feasibility) was achieved)

		EA				Random		
	k	Type	Cost		Feas.	Cost		Feas.
#1	85	MULTIX	**0.00**	**(0; 0; 0)**	**1.00**	16.80	(4; 17.5; 31)	0.00
#2	12	GPX	**2.40**	**(0; 2; 4)**	0.05	7.60	(5; 8; 10)	0.00
#3	105	GPX	0.90	(0; 1; 2)	0.40	1.75	(0; 2; 3)	0.15
#4	27	GGA	1.10	(0; 1; 2)	0.15	1.35	(0; 1; 2)	0.05
#5	32	GGA	1.75	(0; 2; 3)	0.05	1.50	(0; 1.5; 3)	0.15

plying local search, and finally replacing the weaker of the two parents.

The results in Table 63.4 indicate that, for graph #1, recombination is clearly doing more than just randomly perturbing solutions since all runs have resulted in feasible 85-colorings. However, although recombination has achieved significantly lower *costs* with graph #2, the proportion of runs where feasibility has been achieved shows no significant difference for any of the graphs #2 to #5 (according to McNemar's test at significance level 0.01). We find this observation compelling as it might suggest that better results might ultimately be achieved using schemes that make more informed decisions about the size and frequency of perturbations. Indeed, currently the size of random perturbations tends to fall as the run progresses (Fig. 63.6); however, it may be useful to allow this trend to be reversed, particularly if improvements are not achieved for a lengthy period of time. In addition, the *way* in which vertices are chosen for random reassignment might also influence performance – for example, we might target those belonging to a specific color, those that are causing clashes, those that have been assigned to a particular color for the longest, and so on. This requires further research.

An interesting point regarding the structure of solutions was raised previously by *Porumbel* et al. [63.35], who considered the sizes of the color classes. Specifically, they propose that when solutions involve a small number of large color classes (such as graph #2 in our case), good quality colorings tend to result through the identification of large independent sets. On the other hand, if a solution involves many small color classes, quality is determined more by the *productive interaction* between classes. In other words, the proposal is that small independent sets in isolation do not constitute good features in these cases; rather, quality results from appropriate combinations of these sets. Such an observation might provide evidence as to why the GGA recombination has outperformed GPX with graph #3 because, unlike GPX, it does not require half of the color classes to be inherited from each parent, thus potentially allowing more class-combinations to be considered. However, this argument is countered by the fact that, according to Table 63.4, GGA has not outperformed the random perturbation operator, suggesting that it is actually this mechanism that influences the search. Clearly, further research in this area is also required.

Given such observations, another important avenue of future research will be to increase our understanding of the links between a graph's structure and the best algorithms that can then be used to color it. This might, for example, be derived by increasing our understanding of the behavior, strengths, and weaknesses of the various algorithmic operators available for graph coloring, and also via more empirical means such as data mining, as discussed by *Smith-Miles* and *Lopes* [63.55].

References

63.1 N. Zufferey, P. Amstutz, O. Giaccari: Graph colouring approaches for a satellite range scheduling problem, J. Sched. **11**(4), 263–277 (2008)

63.2 M. Carter: A survey of practical applications of examination timetabling algorithms, Oper. Res. **34**(2), 193–202 (1986)

63.3 R. Lewis: A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum **30**(1), 167–190 (2008)

63.4 R. Lewis, J. Thompson: On the application of graph colouring techniques in round-robin sports scheduling, Comput. Oper. Res. **38**(1), 190–204 (2010)

63.5 K. Aardel, S. van Hoesel, A. Koster, C. Mannino, A. Sassano: Models and solution techniques for the frequency assignment problems, 4OR: Q. J. Belg. Fr. Ital. Oper. Res. Soc. **1**(4), 1–40 (2002)

63.6 C.M. Valenzuela: A study of permutation operators for minimum span frequency assignment using an order based representation, J. Heuristics **7**, 5–21 (2001)

63.7 K. Appel, W. Haken: Solution of the four color map problem, Sci. Am. **4**, 108121 (1977)

63.8 M. Gamache, A. Hertz, J. Ouellet: A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding, Comput. Oper. Res. **34**, 2384–2395 (2007)

63.9 G. Chaitin: Register allocation and spilling via graph coloring, ACM SIGPLAN Notices **39**(4), 66–74 (2004)

63.10 M.R. Garey, D.D. Johnson: *Computers and Intractability – A guide to NP-completeness*, 1st edn. (W. H. Freeman, San Francisco 1979)

63.11 M. Karp: Reducibility among combinatorial problems. In: *Complexity of Computer Computations*, The IBM Research Symposia Series, Vol. 1972, ed. by R.E. Miller, J.W. Thatcher, J.D. Bohlinger (Plenum Press, New York 1972) pp. 85–103

63.12 D. Welsh, M. Powell: An upper bound for the chromatic number of a graph and its application to timetabling problems, Comput. J. **12**, 317–322 (1967)

63.13 D. Brélaz: New methods to color the vertices of a graph, Commun. ACM **22**(4), 251–256 (1979)

63.14 P. Spinrad, G. Vijayan: Worse case analysis of a graph colouring algorithm, Discrete Appl. Math. **12**, 89–92 (1984)

63.15 R. Brown: Chromatic scheduling and the chromatic number problem, Manag. Sci. **19**(4), 451–463 (1972)

63.16 S. Korman: The graph-colouring problem. In: *Combinatorial Optimization*, ed. by N. Christofides, A. Mingozzi, P. Toth, C. Sandi (Wiley, New York 1979) pp. 211–235

63.17 M. Kubale, B. Jackowski: A generalized implicit enumeration algorithm for graph colouring, Communications ACM **28**(28), 412–418 (1985)

63.18 J. Culberson, F. Luo: Exploring the k-colorable landscape with iterated greedy, Proc. 2nd DIMACS Implement. Chall. (1996), pp. 245–284

63.19 C. Mumford: New order-based crossovers for the graph coloring problem, Lect. Notes Comput. Sci. **4193**, 880–889 (2006)

63.20 E. Erben: A grouping genetic algorithm for graph colouring and exam timetabling, Lect. Notes Comput. Sci. **2079**, 132–158 (2001)

63.21 R. Lewis: A general-purpose hill-climbing method for order independent minimum grouping problems: A case study in graph colouring and bin packing, Comput. Oper. Res. **36**(7), 2295–2310 (2009)

63.22 M. Chams, A. Hertz, O. Dubuis: Some experiments with simulated annealing for coloring graphs, Eur. J. Oper. Res. **32**, 260–266 (1987)

63.23 D. Johnson, C. Aragon, L. McGeoch, C. Schevon: Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning, Oper. Res. **39**, 378–406 (1991)

63.24 A. Hertz, D. de Werra: Using tabu search techniques for graph coloring, Computing **39**(4), 345–351 (1987)

63.25 M. Laguna, R. Marti: A GRASP for coloring sparse graphs, Comput. Optim. Appl. **19**, 165–178 (2001)

63.26 M. Chiarandini, T. Stützle: An application of iterated local search to graph coloring, Proc. Comput. Symp. Graph Color. Gen. (2002) pp. 112–125

63.27 L. Paquete, T. Stützle: An experimental investigation of iterated local search for coloring graphs, applications of evolutionary computing, Lect. Notes Comput. Sci. **2279**, 121–130 (2002)

63.28 C. Avanthay, A. Hertz, N. Zufferey: A variable neighborhood search for graph coloring, Eur. J. Oper. Res. **151**, 379–388 (2003)

63.29 J. Thompson, K. Dowsland: An improved ant colony optimisation heuristic for graph colouring, Discrete Appl. Math. **156**, 313–324 (2008)

63.30 R. Dorne, J.-K. Hao: A new genetic local search algorithm for graph coloring, Lect. Notes Comput. Sci. **1498**, 745–754 (1998)

63.31 A.E. Eiben, J.K. van der Hauw, J.I. van Hemert: Graph coloring with adaptive evolutionary algorithms, J. Heuristics **4**(1), 25–46 (1998)

63.32 C. Fleurent, J. Ferland: Genetic and hybrid algorithms for graph colouring, Ann. Oper. Res. **63**, 437–461 (1996)

63.33 P. Galinier, J.-K. Hao: Hybrid evolutionary algorithms for graph coloring, J. Comb. Optim. **3**, 379–397 (1999)

63.34 Z. Lü, J.-K. Hao: A memetic algorithm for graph coloring, Eur. J. Oper. Res. **203**(1), 241–250 (2010)

63.35 D. Porumbel, J.-K. Hao, P. Kuntz: An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring, Comput. Oper. Res. **37**, 1822–1832 (2010)

63.36 C. Morgenstern: Distributed coloration neighborhood search, Discrete Math. Theor. Comput. Sci. **26**, 335–358 (1996)

63.37 I. Blochliger, N. Zufferey: A graph coloring heuristic using partial solutions and a reactive tabu scheme, Comput. Oper. Res. **35**, 960–975 (2008)

63.38 E. Malaguti, M. Monaci, P. Toth: A metaheuristic approach for the vertex coloring problem, INFORMS J. Comput. **20**(2), 302–316 (2008)

63.39 A. Hertz, M. Plumettaz, N. Zufferey: Variable space search for graph coloring, Discrete Appl. Math. **156**(13), 2551–2560 (2008)

63.40 N.J. Radcliffe: Forma analysis and random respectful recombination, Proc. 4th Int. Conf. Genet. Algorithms (1991) pp. 222–229

63.41 E. Falkenauer: *Genetic Algorithms and Grouping Problems*, 1st edn. (Wiley, New York 1998)

63.42 E. Coll, G. Duran, P. Moscato: A discussion on some design principles for efficient crossover operators for graph coloring problems, An. XXVII Simp. Bras. Pesqui. Oper. (1995)

63.43 R. Abbasian, M. Mouhoub, A. Jula: Solving graph coloring problems using cultural algorithms, Proc. 24th Florida Artif. Intell. Res. Soc. Conf. (2011)

63.44 J. Munkres: Algorithms for the assignment and transportation problems, J. Soc. Ind. Appl. Math. **5**(1), 32–38 (1957)

63.45 D. Bertsekas: Auction algorithms for network flow problems: A tutorial introduction, Comput. Optim. Appl. **1**, 7–66 (1992)

63.46 A. Tucker, J. Crampton, S. Swift: RGFGA: An efficient representation and crossover for grouping genetic algorithms, Evol. Comput. **13**(4), 477–499 (2005)

63.47 R. Lewis, E. Pullin: Revisiting the restricted growth function genetic algorithm for grouping problems, Evol. Comput. **19**(4), 693–704 (2011)

63.48 E. Falkenauer: A new representation and operators for genetic algorithms applied to grouping problems, Evol. Comput. **2**(2), 123–144 (1994)

63.49 C. Glass, A. Prugel-Bennett: Genetic algorithms for graph coloring: Exploration of Galnier and Hao's algorithm, J. Comb. Optim. **7**, 229–236 (2003)

63.50 R. Lewis, J. Thompson, C. Mumford, J. Gillard: A wide-ranging computational comparison of high-performance graph colouring algorithms, Comput. Oper. Res. **39**(9), 1933–1950 (2012)

63.51 P. Galinier, A. Hertz: A survey of local search algorithms for graph coloring, Comput. Oper. Res. **33**, 2547–2562 (2006)

63.52 J. Culberson: Graph coloring page, http://web.cs.ualberta.ca/~joe/Coloring/ (2010)

63.53 M. Carter, G. Laporte, S.Y. Lee: Examination timetabling: Algorithmic strategies and applications, J. Oper. Res. Soc. **47**, 373–383 (1996)

63.54 T. Hogg, B. Huberman, C. Williams: Refining the phase transition in combinatorial search, Artif. Intell. **81**(1/2), 127–154 (1996)

63.55 K. Smith-Miles, L. Lopes: Measuring instance difficulty for combinatorial optimization problems, Comput. Oper. Res. **39**(5), 875–889 (2012)