Genetic and hybrid algorithms for graph coloring

Charles Fleurent and Jacques A. Ferland

Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 2900 Édouard-Montpetit, C.P. 6128 Succ. A, Montréal, Québec, Canada H3C 3J7

Some genetic algorithms are considered for the graph coloring problem. As is the case for other combinatorial optimization problems, pure genetic algorithms are outperformed by neighborhood search heuristic procedures such as tabu search. Nevertheless, we examine the performance of several hybrid schemes that can obtain solutions of excellent quality. For some graphs, we illustrate that genetic operators can fulfill long-term strategic functions for a tabu search implementation that is chiefly founded on short-term memory strategies.

1. Introduction

Let $G = (V, E)$ be an undirected graph. A k-coloring of G is a partition of V into k subsets C_i, $i = 1, \ldots, k$, such that no adjacent nodes belong to the same subset. The graph coloring problem is to find a k-coloring of G with k as small as possible. This smallest k corresponds to the chromatic number $\chi(G)$ of graph G. It is well known that this problem is NP-hard [14], and consequently, heuristic methods must be used for large graphs.

There exist many heuristic approaches for the graph coloring problem. Some of these are simple schemes in which nodes are colored sequentially according to some criteria [4,27]. These methods are relatively easy to analyze and usually very fast. When a better solution is needed, other techniques requiring more computational effort can be used. For instance, simulated annealing [6,24] and tabu search [22] have been adapted for graph coloring. To our knowledge, there has been only one attempt to adapt a genetic algorithm to color graphs [7]. One of the goals of this paper is to further examine possible ways in which the graph coloring problem can be solved using genetic algorithms.

It is now widely established that pure genetic algorithms are not well suited to fine-tuned search in highly combinatorial spaces [5,20,21,39]. Because of this, a number of hybrid schemes have recently been proposed and used to improve the results obtained with genetic algorithms for various combinatorial optimization problems [7,11,32]. Moreover, the addition of recombination operators to other heuristic procedures has also been previously proposed in a more general setting [15,16]. Accordingly, we investigate the use of local search and short-term-based tabu search procedures as supplementary tools that can enhance the performance of standard genetic algorithms. In the case of tabu search, the resulting hybrids can in some cases be interpreted as a way to fulfill a long-term beneficial effect on tabu search, such as other strategies proposed in [2,8,16].

In the next section, a generic steady-state genetic algorithm is presented, as well as several ways of adapting it for the graph coloring problem. In section 3, local search and tabu search implementations for graph coloring are described in detail. In section 4, the relative performance of some hybrid schemes is discussed. Then, section 5 gives some numerical results for random graphs, as well as for Leighton graphs [27] for which the actual chromatic number is known. Finally, in section 6, we discuss these results and indicate some directions for further research.

2. Genetic algorithms

As introduced by Holland [23], genetic algorithms (GA) form a class of optimization procedures in which populations of individual solutions evolve in a manner inspired by evolution and natural selection. A review of their theoretical and practical aspects can be found in [7,20].

A genetic algorithm can be implemented in several different ways. Our implementation uses a steady-state population where only a few individuals are changed at each generation, whereas in classical genetic algorithms, the complete population is usually replaced at each generation. The merits of the steady-state model are discussed in [7,34]. In particular, it exhibits strong "elitism" and thus may be better suited for hybrid schemes. Our genetic algorithm is summarized in figure 1. N is the size of the population. During each of the *nb_gen* generations, *nb_per_gen* offsprings are generated through the crossover of parents selected from the population. A mutation operator is then applied to each offspring with probability p_m. Several *crossover*( ) and *mutate*( ) functions will be introduced in the next sections. After each generation, the size of the population is brought back to N using the operator *Cull*(P, *nb_gen*) that simply removes the *nb_per_gen* worst individuals from P. In our sequential implementation, the value of *nb_per_gen* is set to 2 (see [7,35], for instance).

Some elements of this general framework must be adapted to the graph coloring problem, and these will be discussed in the remainder of this section. Other operators are common to most genetic algorithms and may be implemented in many ways. For instance, the selection mechanisms for parents should allow for better solutions to reproduce more often. In our implementation, we use a rank-based mechanism proposed in [39]. The solutions are ordered according to their values,

```
Genetic Algorithm
$P := \emptyset$
For $i := 1$ to $N$ do
    Generate a random solution $s$
    With probability $p_m$
        mutate $s$
    Evaluate $s$ and add it to $P$
Sort $P$
For $i := 1$ to $nb\_gen$ do
    For $j := 1$ to $nb\_per\_gen$ do
        Select two parents $p_1$ and $p_2$ from $P$
        $offspring := crossover(p_1, p_2)$
        With probability $p_m$
            mutate($offspring$)
        Evaluate $offspring$ and add it to $P$
    Sort $P$
    $Cull(P, nb\_per\_gen)$
Return the best solution from $P$
```

Figure 1. Genetic algorithm.

the best solutions coming first. To select a parent, a random number $u \in [0, N^{1/r})$ is generated (where the parameter r is a real number in the interval $[1, 2]$). The solution in position $\lceil u^r \rceil$ is then selected as a parent. It is easy to see that the probability of selecting better parents increases with the value of the selection parameter r. In our case, we initially set $r = 2$ and periodically adjust it to $r = 1 + D$, where $D \in [0, 1]$ is the population's diversity measure (see section 2.2). Several other selection schemes are discussed in [7,20].

2.1. ORDER-BASED ENCODING AND OPERATORS

In the early days, most genetic algorithms used binary encodings and well-studied crossover and mutation operators [20,23]. More recently, alternate encoding schemes have been considered in order to account for problem-specific information [21]. In [7], Davis proposed an hybrid genetic algorithm for the graph coloring problem in which individuals are represented as permutations of the nodes in V. This type of encoding is referred to as **order-based encoding**. In order to evaluate an individual, a greedy algorithm is applied, which sequentially colors the nodes in the order defined by the permutation.

Our implementation of the greedy algorithm is slightly modified so as to color a graph with a fixed number of colors k. The greedy procedure simply scans the nodes in the order of the permutations, giving each node the first color still available, given previous assignments. If none of the k available colors can legally be given to a node, the greedy algorithm picks the color that is the least often assigned among the adjacent colored nodes. The greedy algorithm thus generates a partition $C_1, C_2, \ldots, C_k$, which can be evaluated by

$$f(C_1, C_2, \ldots, C_k) = \sum_{i=1}^{k} |E(C_i)|, \tag{1}$$

where $E(C_i) = \{(x, y) \in E : x, y \in C_i\}$ and $|E(S)|$ denotes the cardinality of set S. The objective is to generate a partition with $f(C_1, C_2, \ldots, C_k) = 0$.

In Davis' scheme, the genetic algorithm searches the space of permutations, and the value of each permutation (its fitness) is evaluated according to (1), once the modified greedy algorithm has been applied. Davis also defines the crossover and mutation operators illustrated in tables 1 and 2. To generate an offspring permutation from $parent_1$ and $parent_2$ with the uniform order-based crossover, a

Table 1

Uniform order-based crossover.

$parent_1$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$parent_2$	2	5	8	1	9	3	7	4	6
str (step 1)	0	0	1	0	1	1	1	0	1
offspring (step 2)	–	–	**3**	–	**5**	**6**	**7**	–	**9**
offspring (step 3)	2	8	**3**	1	**5**	**6**	**7**	4	**9**

Table 2

Scramble sublist mutation.

before mutation	2	8	3	1	5	6	7	4	9
random positions			↑			↑			
after mutation	2	8	**6**	**1**	**3**	**5**	7	4	9

random bit-string *str* of size $|V|$ is generated (step 1 in table 1). Then, for positions i where $str[i]$ is equal to 1, we make the assignments $offspring[i] = parent_1[i]$ (step 2 in table 1). The remaining positions of the offspring are filled using the unassigned nodes in the order in which they appear in $parent_2$. The mutation operator (see table 2) randomly selects two positions in the permutation. The elements in and between these positions are then "scrambled" randomly. The relative performance of Davis' GA will be discussed in section 4.

2.2. STRING-BASED ENCODING AND OPERATORS

We now examine another genetic approach for graph coloring in which a string-based encoding is used for solutions. In the preceding section, Davis' order-based ordering was well suited to be combined (or hybridized) with the greedy algorithm that operates on sequences of nodes. The string-based encoding allows for hybrid schemes with neighborhood search methods such as local search and tabu search.

Let $P = \{p_1, p_2, \ldots, p_N\}$ be a population of partitions of V into k subsets. Each partition $p \in P$ can be denoted by $p = \{C_1, C_2, \ldots, C_k\}$, which can be encoded as a string of length $|V|$ where $s[i] = j$ if node $i \in C_j$. Using such an encoding, it is easy to derive crossovers based on classical bit-string crossovers [7]. Tables 3, 4 and 5 illustrate natural extensions of 1-point, 2-point, and uniform bit-string crossovers.

Table 3

String-based 1-point crossover.

parent$_1$	**1**	**2**	**3**	**2**	**1**	**3**	**2**	**1**	**1**
parent$_2$	3	1	3	2	2	1	3	1	2
random position				↑					
offspring$_1$	**1**	**2**	**3**	2	2	1	3	1	2
offspring$_2$	3	1	3	**2**	**1**	**3**	**2**	**1**	**1**

Table 4

String-based 2-point crossover.

parent$_1$	**1**	**2**	**3**	**2**	**1**	**3**	**2**	**1**	**1**
parent$_2$	3	1	3	2	2	1	3	1	2
random positions			↑			↑			
offspring$_1$	**1**	**2**	3	2	2	1	**2**	**1**	**1**
offspring$_2$	3	1	**3**	**2**	**1**	**3**	3	1	2

Table 5

String-based uniform crossover.

parent$_1$	**1**	**2**	**3**	**2**	**1**	**3**	**2**	**1**	**1**
parent$_2$	3	1	3	2	2	1	3	1	2
random bit string	0	1	1	0	1	0	1	0	0
offspring$_1$	**1**	1	3	**2**	2	**3**	3	**1**	**1**
offspring$_2$	3	**2**	**3**	2	**1**	1	**2**	1	2

The 1-point crossover operator randomly selects a cut point in the parents, and generates offsprings by combining the genetic material of one parent on the left-hand side of the cut point with that of the other parent for the positions on the right-hand side (and including the cut point). 2-point crossover is very similar, except that the genetic material is swapped between two random cut points instead of one. The uniform crossover operator generates a random bit string of size $|V|$ that specifies which parent determines the color of each node. A simple mutation operator can be defined so that each node has a fixed probability of being assigned a new random color. This mutation operator is the natural extension of the classical operator which randomly flips some bits in the case of binary strings.

Recently, crossover operators have been proposed in order to account for problem-specific information. For the traveling salesman problem, for instance [21], Grefenstette reports better results when using a crossover operator that takes into account the distances between the cities. Since then, many other crossover operators have been proposed for the TSP [12,32]. For graph coloring, it is also possible to design crossover operators which use the graph structure. These are particularly interesting for hybrid schemes where most of the computational effort is put into applying long mutation operators to the offspring. The extra effort required by adapted crossover operators then becomes negligible. It should also be pointed out that the term "crossover" should preferably be replaced by the "recombination operator" appellation, in order to establish some relation between these operators and other methods, such as the scatter search approach [15,16]. This choice of terminology may be very important to avoid restricting future designs of recombination operators to the usual templates that are commonly used with classical genetic algorithm implementations.

The graph-adapted recombination operator is described in figure 2. For solution s, the set of conflicting nodes is denoted by $CN(s) = \{x \in V : \exists (x, y) \in E$ such that $s[x] = s[y]\}$. An offspring from parents $parent_1$ and $parent_2$ is generated by coloring conflicting nodes in one parent using the color supplied by the other parent, if it is conflict-free for the latter. When a node is in conflict in both parents, we select the color that is used the least often among adjacent nodes in either parent. Each of the remaining nodes is colored using a color in one of the parents, selected randomly, as in the case of uniform crossover.

In any genetic algorithm, it is very useful to have a way of evaluating the diversity of a population of solutions. Hence, we consider the population entropy specified by (2) and (3), where n_{ij} is the number of individuals in P for which node i is assigned color j:

$$D_i = \frac{-\sum_{j=1, n_{ij} \neq 0}^{k} (n_{ij} / |P|) \log (n_{ij} / |P|)}{\log k}, \tag{2}$$

$$D = \frac{\sum_{i=1}^{|V|} D_i}{|V|}. \tag{3}$$

```
For $x \in CN(parent_1)$ do
  If $(x \notin CN(parent_2))$
    $offspring[x] = parent_2[x]$
For $x \in CN(parent_2)$ do
  If $(x \notin CN(parent_1))$
    $offspring[x] = parent_1[x]$
For $x \in (CN(parent_1) \cap CN(parent_2))$ do
  $offspring[x] = \underset{l \in \{1,\dots,k\}}{\operatorname{argmin}} \left\{ \sum_{y \in N(x)} (I_l(\{parent_1[y], parent_2[y]\})) \right\}$
For $x \in (V - (CN(parent_1) \cup CN(parent_2)))$ do
  If $(DU[0,1] = 1)$
    $offspring[x] = parent_1[x]$
  Else
    $offspring[x] = parent_2[x]$
```

Figure 2. Graph-adapted recombination operator. $DU[0, 1]$ is a function that returns 0 or 1 with equal probability; $\arg\min_{l \in S}\{a_l\}$ return the index l^* such that $a_{l^*} \le a_l$, $\forall l \in S$; $I_l(S)$ is the indicator variable of set S.

Thus, D is a normalized value in the interval [0, 1], indicating the diversity of the population. A population which consists entirely of identical individuals has entropy 0; a population in which the colors of the nodes are uniformly distributed has an entropy value near 1. A measure based on entropy has several interesting properties that are discussed in [40]. For instance, the entropy of a probability distribution is maximal when all possible outcomes are equiprobable. Also, among uniform distributions, the entropy is higher when more outcomes are possible. This diversity measure is used to design stopping criteria, and to influence parent selection, as indicated in section 2.

3. Local search and tabu search for graph coloring

Local search and tabu search [17] are procedures that explore a space in which solutions are related by a neighborhood structure. While local search moves from solutions to improving neighbors until a local optimum is reached, tabu search allows the process to continue by admitting non-improving moves. In order to avoid cycling, the neighborhood structure is dynamically modified so as to forbid recent moves for a number of iterations. Tabu search procedures can also be enhanced to provide for additional diversification or intensification objectives; these are discussed in more detail in [19].

For graph coloring, an efficient neighborhood structure was proposed in [22]. In TABUCOL, solutions are encoded as partitions of the nodes into k subsets and are evaluated according to equation (1). Let the current solution be the partition of V denoted by $s = (C_1, C_2, \dots, C_k)$. For solution s, the set of conflicting nodes is denoted by $CN(s) = \{x \in V : \exists (x, y) \in E \text{ such that } s[x] = s[y]\}$. A neighborhood solution s' is obtained by coloring a conflicting node $x \in CN(s)$ with a color j that is different from its current color i. The corresponding neighboring solution is then defined as $s' = (C_1, \dots, C_i', \dots, C_j', \dots, C_k)$, where $C_j' = C_j \cup \{x\}$ and $C_i' = C_i - \{x\}$. Using this neighborhood definition, we can define the natural local search procedure that selects at each iteration the move that induces the greatest decrease in the objective function.

In TABUCOL, a random sample of the neighborhood is generated at each iteration, and the best non-tabu move is selected. After a transition is made by changing the color of node x from i to j, a tabu restriction is imposed on the node–color pair (x, i) for 7 iterations. Our implementation differs in that the whole neighborhood is considered at each iteration, and in that the tabu list size is randomly varied during the search. This practice was introduced by Taillard [36,37], and provides an easy way to diversify the search. In our case, the tabu list size is periodically changed around a base value T_b. The actual tabu list size T_a is randomly selected in the interval $[t_{min}, t_{max}]$, where $t_{min} = \lfloor 0.9 T_b \rfloor$ and $t_{max} = \lceil 1.1 T_b \rceil$. Once a move has been accepted, its tabu status stands for the next T_a iterations. The current tabu list size T_a is modified every $2t_{max}$ iterations. Since the number of neighbors is proportional to the number of conflicts in the solution, the tabu list base size also has to be updated accordingly. Even if the most appropriate value may depend on the particular graph to color, empirical experiments have shown that a robust tabu list base size is:

$$T_b = \sqrt{2f(s) \times k}. \tag{4}$$

Our tabu search implementation is chiefly based on short-term memory strategies. Other schemes [2] make it possible to dynamically modify the tabu list size by using information gathered throughout the search. However, as the results of section 5 indicate, our implementation remains very efficient when compared to earlier tabu search adaptations for graph coloring [22].

In any neighborhood-based search method, it is crucial to evaluate quickly the effect of moving to adjacent solutions. In our case, it is possible to use a $|V|$-by-k matrix Δ, in which entry $\Delta(x, j)$ represents the effect of changing the current color of node x from i to j. Each entry can be initialized in $O(|V|)$ operations as follows:

$$\Delta(x, j) = \sum_{y \in N(x)} (I_{C_j}(y) - I_{C_i}(y)), \tag{5}$$

where for $x \in V$, $N(x) = \{ y \in V \mid (y, x) \in E \}$; and I_A is the indicator variable of set A, defined as

$$I_A(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{otherwise.} \end{cases}$$

```
For $y \in N(x)$ do {
  If $(y \notin C_i)$
    $\Delta(y,i) = \Delta(y,i) - 1$
  If $(y \notin C_j)$
    $\Delta(y,j) = \Delta(y,j) + 1$
  If $(y \in C_j)$
    For $c = 1$ to $k$ do {
      If $(c \neq j)$
        $\Delta(y,c) = \Delta(y,c) - 1$
      $\Delta(x,c) = \Delta(x,c) - 1$
    }
  If $(y \in C_i)$
    For $c = 1$ to $k$ do {
      If $(c \neq i)$
        $\Delta(y,c) = \Delta(y,c) + 1$
      $\Delta(x,c) = \Delta(x,c) + 1$
    }
}
```

Figure 3. Procedure for updating Δ. Invoked before node x moves from C_i to C_j.

Hence, using (5), Δ can be initialized in $O(k|V|^2)$ operations. Before the search procedure moves to a neighbor by changing the color of node x from i to j, Δ can be updated using the procedure shown in figure 3, requiring $O(|V|k)$ operations in the worst case (that of a complete graph).

4. Some hybrid schemes

In general, it is well known that pure genetic algorithms do not give very good results for combinatorial optimization problems [5,39,21,12]. Because of this, it is now common to hybridize genetic algorithms with heuristic methods that already perform well in specific contexts (see [7]). Such hybrid schemes were first introduced by Muhlenbein [31] and Davis [7], where a local search is used as the mutation operator. Similar approaches have also been introduced in a more general context by Glover in [15] and [16].

In this section, we will study several variants of the generic GA described in figure 1, using different encoding schemes, crossover and mutation operators. Each algorithm will be identified by a short label: $GA(|P| \in \mathcal{N}, E \in \{P, S\}, CR \in \{1, 2, U, A\}, MUT \in \{LS, TSn, p_m\})$, which describes the parameters used. The first parameter gives the size of the population and the second parameter indicates the encoding scheme (permutation or string). The third parameter indicates the recombination operator that is used (1-point, 2-point or uniform crossover, or graph-adapted recombination operator). The last parameter indicates which mutation operator is used: local search or tabu search (with the number of performed iterations) or the mutation rate p_m for pure GA schemes (which use the mutation operator associated with the encoding scheme). When local search or tabu search mutation operators are used, they are applied to each individual.

Two classes of graphs are used to compare the efficiency of the procedures: random graphs and Leighton graphs. A $G_{n,p}$ random graph denotes a graph with n nodes, where p is the probability that there exists an edge between any pair of nodes. Leighton graphs have a special structure, described in [27], such that their chromatic numbers are known.

4.1. ORDER-BASED GA

To illustrate the behavior of Davis' algorithm, consider a $G_{100,0.5}$ random graph having an expected chromatic number of 16 (see [25]). Figure 4 shows the average performance over three runs of Davis' genetic algorithm with a population of size 100 and mutation rate $p_m = 1\%$ ($GA(|P| = 100, E = P, CR = U, MUT = 1\%)$).

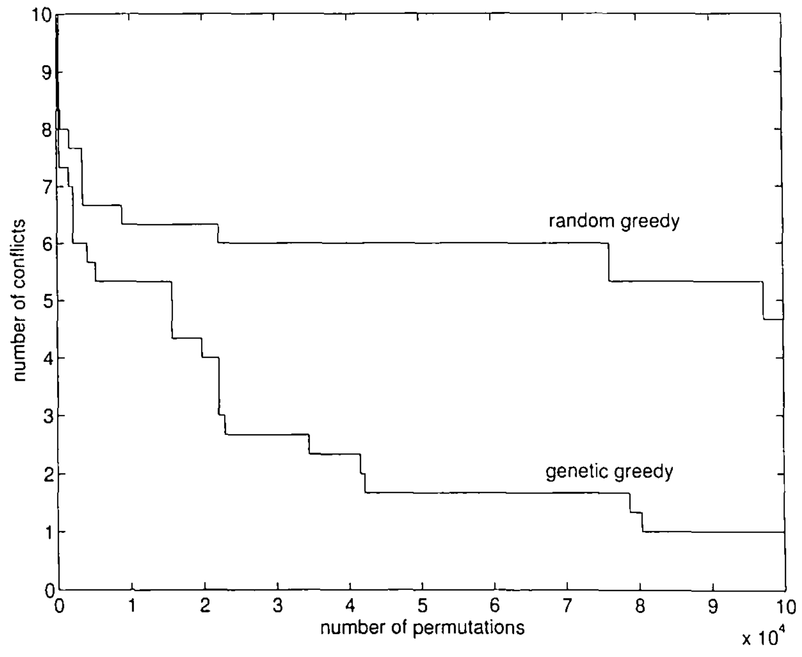

Figure 4. Effect of a genetic component on a greedy algorithm searching for a 16-coloring in a $G_{100,0.5}$ random graph. $GA(|P| = 100, E = P, CR = U, MUT = 1\%)$. Average of three runs.

Also plotted is the average behavior of the greedy algorithm when applied to as many random permutations as are evaluated using the genetic algorithm. The x-axis in figure 4 represents the number of generated permutations, and the y-axis indicates the value of the best solution found. The improvement introduced by the genetic component is obvious.

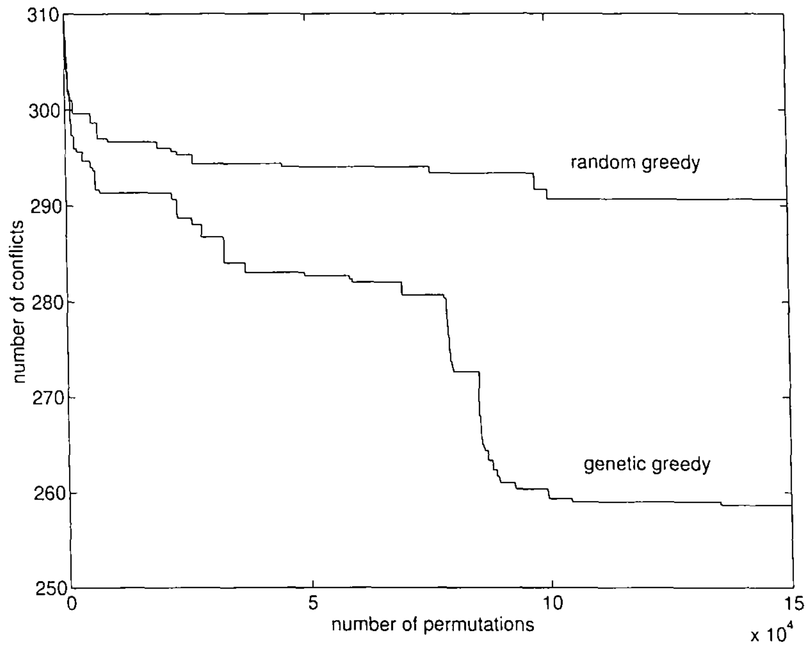

Figure 5. Effect of a genetic component on a greedy algorithm applied to a Leighton graph with chromatic number 15. $GA(|P| = 50, E = P, CR = U, MUT = 1\%)$. Average of three runs.

Figure 5 illustrates the average performance of the genetic greedy: $GA(|P| = 50, E = P, CR = U, MUT = 1\%)$, and the random greedy algorithms for a Leighton graph with 450 nodes and chromatic number 15. Again, we can see a clear improvement when genetic operators are used. Although interesting, these results must be viewed in perspective; while the approach illustrated in figure 4 allows us to find 16-colorings for a $G_{100,0.5}$ graph in about 30 minutes using a SPARC 10 processor, other heuristics, such as tabu search described in section 3, can accomplish the same task in about a second. In figure 5, each run takes more than 12 hours on a SPARC 10 and generates a solution which is far from optimal. Even if it might be possible to improve the performance of Davis' scheme by increasing the size of the populations, using alternate crossover operators [3,12,41], or fine-tuning some parameters, it is far more promising to consider another encoding that will allow hybridization with procedures that are more powerful than sequential greedy algorithms.

4.2. STRING-BASED GA

Again, we use a $G_{300,0.5}$ random graph (figure 6) and the Leighton graph introduced in the preceding section (figure 7) to illustrate the behavior of string-based genetic algorithms with different crossover operators. Again, we can see a clear improvement over random search, which simply generates and evaluates solutions. For the $G_{300,0.5}$ graph, a 34-coloring is sought using parameters $GA(|P|$

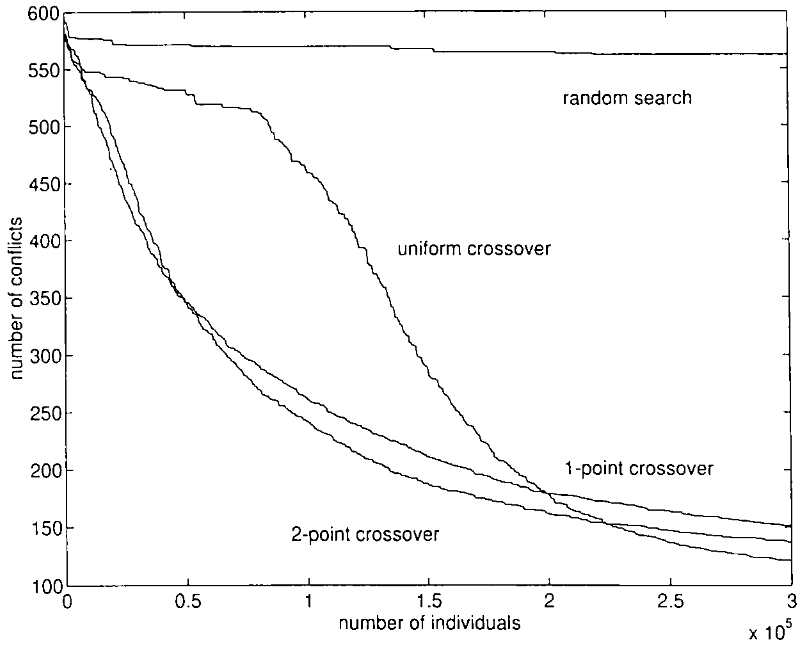

Figure 6. Effect of different crossovers on a $G_{300,0.5}$ graph. $GA(|P| = 1000, E = S, CR = \{1, 2, U\}, MUT = 1\%)$, seaching for a 34-coloring. Average of three runs.

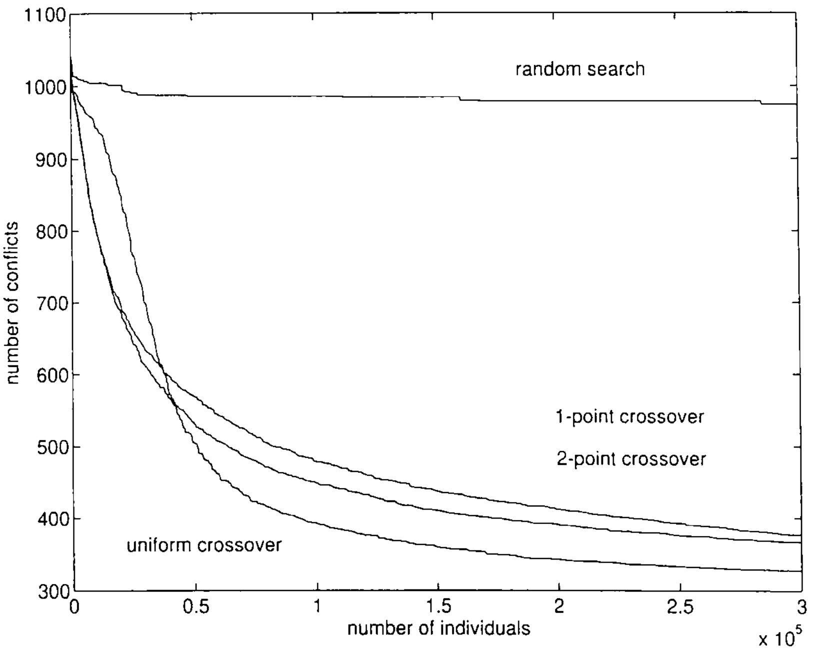

Figure 7. Effect of different crossovers on a 450-node Leighton graph with chromatic number 15. $GA(|P| = 400, E = S, CR = \{1, 2, U\}, MUT = 1\%)$. Average of three runs.

= 1000, $E = S$, $CR = \{1, 2, U\}$, $MUT = 1\%$). Using a SPARC 10 processor, it takes more than 8 hours to generate the 300000 individuals of figure 6. For the Leighton graphs, we use $GA(|P| = 400$, $E = S$, $CR = \{1, 2, U\}$, $MUT = 1\%$), and it takes 8 hours to generate the 300000 individuals of figure 7. These results may not be very impressive when compared with those obtained using other heuristic procedures, but nevertheless they clearly indicate that genetic operators can be helpful in searching for good colorings.

In a genetic algorithm, we say that a population has converged whenever all individuals are the same, or very similar. The entropy diversity measure can be used to monitor the convergence of a population. For instance, it provides interesting information on the behavior of genetic algorithms using different crossover operators. Figure 8 illustrates the evolution of the populations' entropies for the runs shown in figure 7. We note that the uniform crossover operator converges more slowly, but ultimately gives better results.

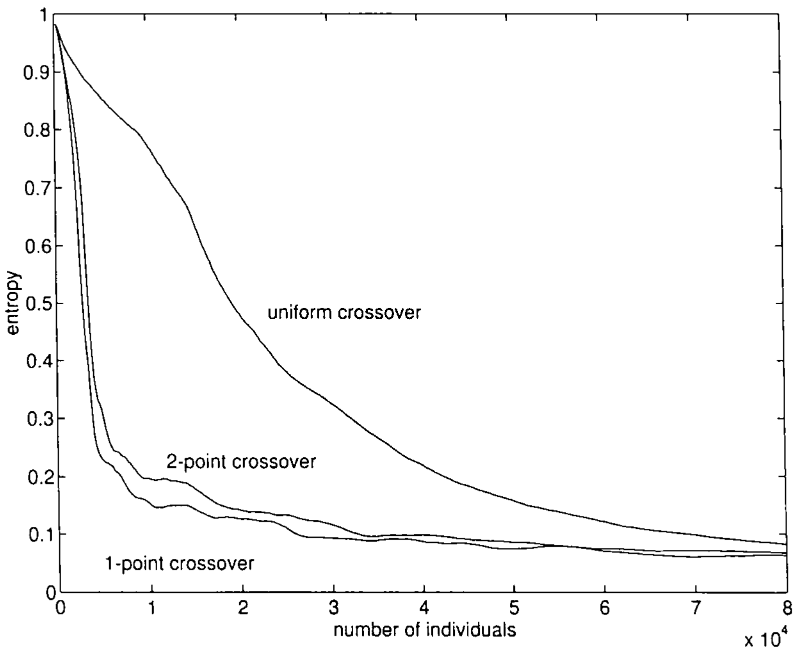

Figure 8. Effect of different crossovers on the entropy of a population. 450-node Leighton graph with chromatic number 15. $GA(|P| = 400$, $E = S$, $CR = \{1, 2, U\}$, $MUT = 1\%$). Average of three runs.

In [34], Syswerda studies 1-point, 2-point, and uniform crossovers for binary strings in relation to the survival rate and the combination rate of schemata which are similarity templates of solutions [20]. For binary strings, Syswerda shows that 1-point and 2-point crossovers generally have better survival rates than uniform crossover, but that uniform crossover is better at combining schemata. His empirical analysis indicates that uniform crossover consistently outperforms 1-point and 2-point crossovers on a wide variety of test problems. This observation is also verified for our string-based uniform crossover.

These results may indicate that a better combination rate is more important than a better survival rate in the overall performance of a genetic algorithm. This observation is important for the genetic hybrid schemes introduced in the next section, since their survival rates will be even lower due to the mutation effect of local search or tabu search. However, the elitism provided by a steady-state model somewhat makes up for this effect since the best individuals are surviving in successive generations. A discussion along these lines can be found in [10,33].

4.3. STRING-BASED GA HYBRIDIZED WITH LOCAL SEARCH

The main drawback of local search procedures is that they have to stop whenever they reach a local optimum, i.e., when no neighbor is better than the current solution. Historically, the first approach to bypass this problem was to restart local search procedures from several random solutions [28,29]. We will now examine the effect of hybridizing a string-based genetic algorithm with the local search algorithm. To do so, the mutation operator of figure 1 corresponds to a local search procedure applied to each individual.

Figures 9 and 10 illustrate the results obtained with genetic local search hybrids and with the multi-start algorithm in which a local search is applied to several random solutions. In figure 9, we are looking for a 34-coloring in a $G_{300,0.5}$ random graph with a $GA(|P| = 300, E = S, CR = \{1, 2, U, A\}, MUT = LS)$. For all crossover operators, the genetic hybrids clearly outperform the multi-start procedure. Furthermore, we note that the uniform operator still dominates 1-point and 2-point crossovers. The graph-adapted recombination operator also gives very good results. In figure 9, each run takes an average of 14 hours.

As mentioned earlier, increasing the size of the population generally slows down the convergence of genetic algorithms and generates better solutions. In figure 10, we try to color optimally the same Leighton graph used in figures 5 and 7. Figure 10 illustrates the behavior of different population sizes with a local search-genetic hybrid algorithm using the uniform crossover operator. We note that a population of size 300 is sufficient to generate an optimal solution in about 6 hours. Populations of size 400 also lead to optimal solutions, but the convergence is slower.

4.4. STRING-BASED GA HYBRIDIZED WITH TABU SEARCH

Tabu search is a powerful heuristic that gives excellent results for graph coloring. For instance, the $G_{100,0.5}$ graph of figure 4 is colored with 16 colors in less than a second, and with 15 colors within 10 seconds using our procedure described

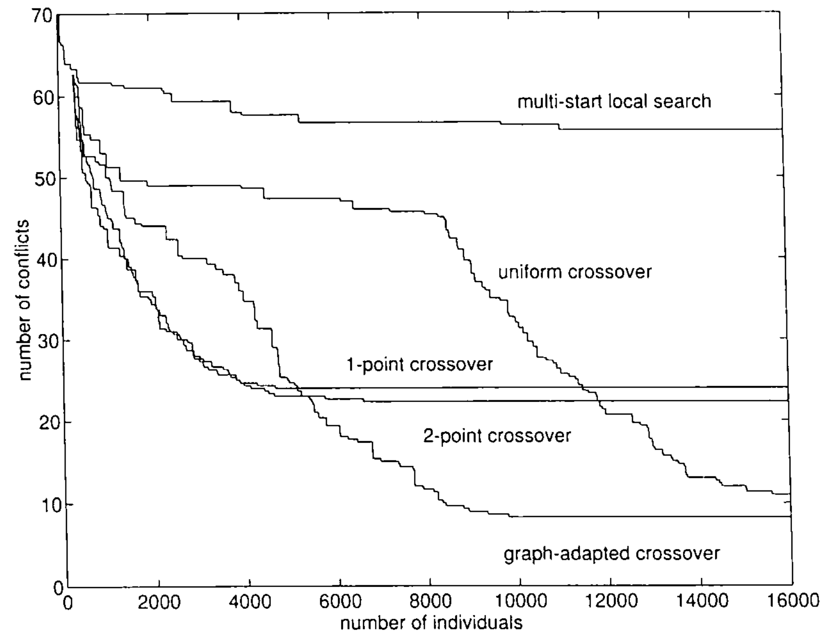

Figure 9. Effect of different crossover operators on local search. Searching for a 34-coloring in a $G_{300,0.5}$ graph. $GA(|P| = 300, E = S, CR = \{1, 2, U, A\}, MUT = LS)$. Average of three runs.

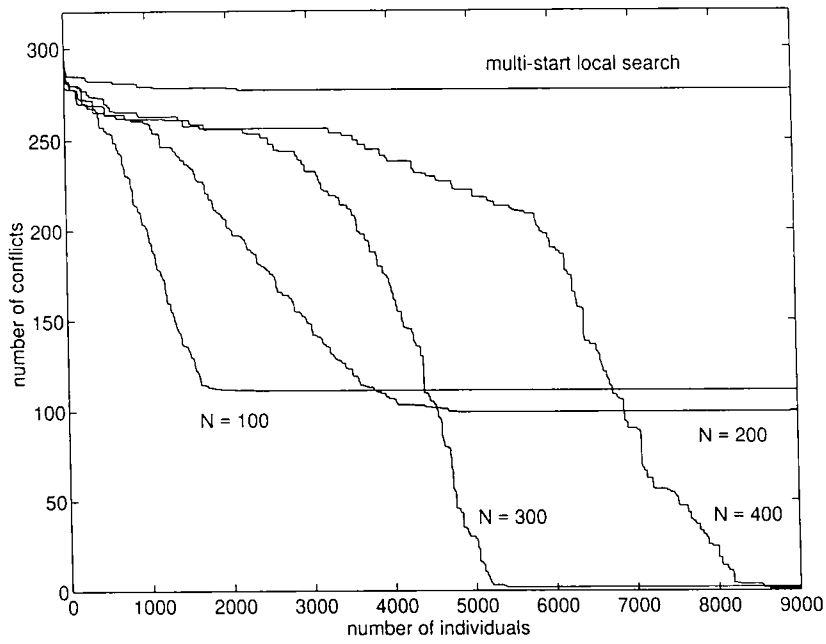

Figure 10. Effect of different population sizes on local search-genetic hybrids with uniform crossover. Average of three runs on a 450-node Leighton graph with chromatic number 15. $GA(|P| = \{100, 200, 300, 400\}, E = S, CR = U, MUT = LS)$.

in section 3. For larger and more difficult graphs, diversification and intensification of the search become issues. Our implementation already includes variable list sizes which have been found to improve the search in other problems [36,37]. However, it must be pointed out that our implementation primarily uses short-term memory strategies. Other sophisticated long-term diversification mechanisms are discussed in, for instance, [2,19]. It is our objective to illustrate that using genetic operators may also be viewed as another way of providing a long-term strategy to a basic tabu search implementation. Similar ideas have already been proposed by Glover in the form of "path relinking" strategies, which are further explained in [18].

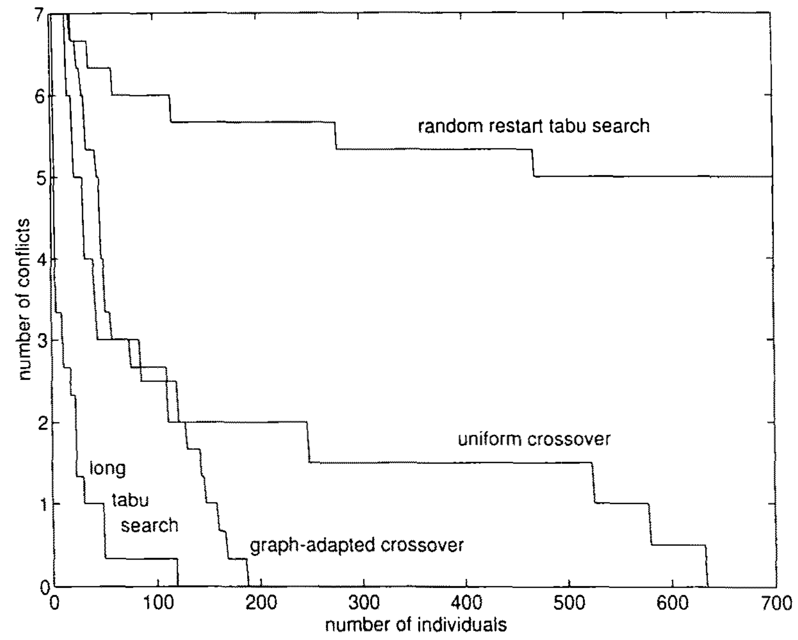

Figure 11. Effect of different genetic hybridization on tabu search. Average of three runs on a $G_{300,0.5}$ random graph when searching for a 34-coloring. $GA(|P| = 10, E = S, CR = \{U, A\}, MUT = TS2000)$.

Figure 11 illustrates the average behavior of some methods dealing with a $G_{300,0.5}$ random graph to be colored with 34 colors: a long tabu search (more than 2000000 iterations), multiple tabu searches from random starting points (700 searches of 2000 iterations each), and a tabu search-genetic hybrid $GA(|P| = 10, E = S, CR = \{U, A\}, MUT = TS2000)$. We see that the long tabu searches are the most efficient, reaching a 34-coloring in an average of 343 seconds. The genetic hybrids also lead to 34-coloring, but they take more time.

For the Leighton graph studied in the earlier sections, longer searches are applied to each individual. In figure 12, we used $GA(|P| = 50, E = S, CR = \{U, A\}, MUT = TS10000)$. For both the uniform and graph-adapted recombination operators,

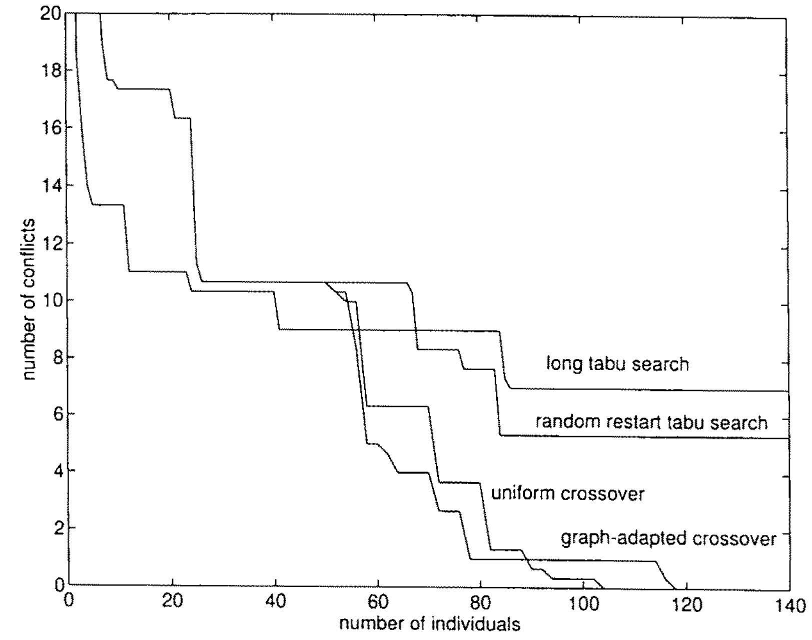

Figure 12. Effect of different genetic hybridizations on tabu search. Average of three runs on a 450-node Leighton graph with chromatic number 15. $GA(|P| = 50,\ E = S,\ CR = \{U, A\},\ MUT = TS10000)$.

few generations are needed to obtain an optimal solution (in less than 2 hours). Furthermore, in this case the genetic hybrids outperform our basic tabu search procedure. In fact, starting from five random starting solutions, the tabu search procedure described in section 3 was unable to reach an optimal solution, even after 20000000 iterations for each starting point.

5. Results

5.1. $G_{n,0.5}$ GRAPHS

Because they are easy to generate, $G_{n,0.5}$ random graphs have traditionally been a popular class of benchmark problems for graph coloring. In addition, a probabilistic estimate $\bar{\chi}(G)$ of the chromatic number of these graphs can be obtained easily [25]. In [22], a sample of random graphs is generated to test the TABUCOL algorithm. This sample includes twenty $G_{100,0.5}$ graphs, ten $G_{300,0.5}$ graphs, five $G_{500,0.5}$ graphs, and two $G_{10000,0.5}$ graphs. In order to generate a similar sample, we used the MLCG random number generator from the GNU C++ library. This generator implements a combination of Multiplicative Linear Congruential Generators, which is described in detail in [26]. This generator needs two seeds. To generate each class of graphs, the first seed varies from 1 to the number of graphs generated, while the second seed value is set at 15. This random number generator is currently considered among the best available. In [24], a 500-node and a 1000-node graph are used for testing various algorithms. These two graphs were included in our sample by replacing the last graph that we would have generated for this class. All graphs in this sample, as well as the others used in this paper, are available via anonymous ftp at the ftp.iro.umontreal.ca site in the directory/pub/optim/fleurent/graphs.

We were able to improve the results reported in [22] for every class of graphs. Table 6 summarizes results reported for TABUCOL, as well as those obtained with our hybrid algorithms. The value k^* denotes the smallest number of colors for which all graphs of the same size can be colored without a failure. $\bar{k}$ denotes the average number of colors needed to color the graphs in the sample, if we try to use k^* or less colors.

Table 6

Results on $G_{n,0.5}$ graphs.

Number of nodes	Number of graphs	$\tilde{\chi}(G)$	k^* from [22]	New k^*	Average time for new k^*	$\bar{k}$
100	20	16	16	15	9.5 sec	14.95
300	10	35	35	34	353 sec	33.5
500	5	50	50	49	5.66 hours	49.0
1000	2	85	87	84	41.35 hours	84.0

In order to obtain these results, we use different hybrid schemes. For the 100-node and 300-node graphs, the figures in table 6 are obtained with our tabu search procedure. Although our tabu search implementation does not use long-term intensification/diversification tools, the improvement over TABUCOL can be explained by the combined effect of using a variable tabu list size depending on the value of the current solution, and of using the complete neighborhood instead of a random sample of it.

It is also possible to obtain similar colorings with tabu search-genetic hybrids, but it generally takes much more time. For instance, we applied some tabu search-genetic hybrids on all ten $G_{300,0.5}$ random graphs in our sample. Table 7 gives the population size, recombination operators, number of tabu iterations performed on each individual, average number of generations (2 offsprings per generation), and average time needed to obtain 34-colorings. When longer tabu search mutations are used, a small population is sufficient to converge to a coloring. When using shorter searches, larger populations are required, and more time is needed to converge. In a parallel implementation, the number of available processors would influence the choice of many parameters, such as the size of the population and the number of tabu iterations performed on each individual.

Table 7

Parameters that can be used to color $G_{300,0.5}$ random graphs with 34 colors using tabu search-genetic hybrid algorithms.

$\|P\|$	Crossover	Mutation	Generations	Time
10	uniform	2000	191.8	3294.9 sec
10	graph-adapted	2000	234.2	4023.3 sec
50	uniform	500	3656.7	9.54 hours
50	graph-adapted	500	2787.5	7.39 hours

For the graphs having 500 and 1000 nodes, tabu search alone is not sufficient to obtain the results reported in table 6. Even for tabu search-genetic hybrids, the populations do not converge in a reasonable amount of time (more than two days of computation). Instead, we use a hybrid approach that first colors part of the graph. Then, tabu search or tabu search-genetic hybrids are applied on smaller residual graphs. Such an approach was also used in [22,24]. The basic idea is to first find large independent sets in the graph.

An independent set of graph $G = (V, E)$ is a subset $S \subset V$, such that $u, v \in S \Rightarrow (u, v) \notin E$. Once a large independent set is identified, we can assign the same color to all of its elements, remove the corresponding nodes from the graph, and try to color the residual graph. To determine an independent set of size l, we use a tabu search algorithm which is a variant of STABULUS [13]. Solutions are encoded as partitions $s = (S, \bar{S})$ of V, where $|S| = l$. Solutions are then evaluated according to the number of edges connecting the nodes in S. A tabu search procedure moves from solution s to s' by swapping a node in S with one in $\bar{S}$.

It also seems likely to be advantageous to include a secondary objective of finding an independent set that is connected to as many nodes as possible in the residual graph. This idea was proposed in [24] (algorithm XRLF), where the authors use a different heuristic to find the independent sets. In our variant, STABULUS is modified so that each solution is evaluated according to

$$f(s) = \sum_{x \in S} \sum_{y \in S - \{x\}} M \cdot I_{(x,y)}(E) - \sum_{x \in S} \sum_{y \in \bar{S}} I_{(x,y)}(E), \tag{6}$$

where $I_{(x,y)}(E)$ is the indicator variable of set E and $M = |S||V|$, for instance. Hence, a legitimate solution s^* has an objective value $f(s^*) < 0$. This objective function allows us to find independent sets "bringing" as many edges as possible with them. Hence, the residual graph tends to be easier to color.

For the 500-node and 1000-node graphs, the modified STABULUS procedure is used to identify independent sets until no such sets of size 10 (500-node graphs) and 11 (1000-node graphs) can be found. The first phase generates residual graphs having 100 to 200 nodes. We found that using larger residual graphs did not improve the final colorings, but did increase the computation time.

Given the moderate sizes of the residual graphs, tabu search-genetic hybrids can be used, and were found to be more robust than tabu search alone. For instance, table 8 illustrates results obtained when trying to color a residual graph of a $G_{1000,0.5}$ random graph. The residual graph (also available via anonymous ftp) has 200 nodes and has to be colored with 23 colors so that the original $G_{1000,0.5}$ graph is colored

Table 8

Results of ten runs of tabu search and tabu search-genetic hybrids on a residual graph of 200 nodes when searching for a 23-coloring.

Method	Number of failures ($\overline{t_f}$)	Number of successes ($\overline{t_s}$)
Tabu search	5 (39.52 hours)	5 (19.26 hours)
Tabu search-genetic hybrid	1 (22.8 hours)	9 (5.3 hours)

with 84 colors. In table 8, we summarize results when a 100000000-iteration tabu search procedure is applied from 10 different random points, and when a tabu search-genetic hybrid with parameters $GA(|P| = 40, E = S, CR = A, MUT = TS1000)$ is applied ten times. $\overline{t_f}$ is the average elapsed time before the algorithm is stopped after a failure, and $\overline{t_s}$ is the average elapsed time when a solution is found. Table 8 thus indicates that for some graphs, tabu search-genetic hybrids may induce a beneficial long-term diversification effect.

As stated earlier, the results in table 6 indicate that the results obtained by our methods are better than those reported in [22]. Recall that our sample also includes the $G_{500,0.5}$ and $G_{1000,0.5}$ graphs used in [24]. For these graphs, Johnson et al. have obtained solutions with 49 and 86 colors, respectively. To our knowledge, only one other method [30] can color $G_{1000,0.5}$ graphs with 84 colors. There are two main differences between our implementation and the XRLF algorithm in [24]. We use a different method to determine the independent sets, and we color the residual graph with tabu search-genetic hybrids. In the XRLF implementation, independent sets are sought until the residual graph can be colored by a branch-and-bound algorithm (70 nodes).

5.2. 450-NODE LEIGHTON GRAPHS

In [27], Leighton introduces a procedure to generate graphs with known chromatic numbers. These graphs are also available from the authors and the

DIMACS ftp site [9]. Table 9 summarizes the results obtained by tabu search and some tabu search-genetic hybrids for the twelve Leighton graphs with 450 nodes. In table 9, we specify the size of the graphs, their chromatic number, the seed that identified the graphs in [27], and the number of colors k^* that were required to color the graphs. For the tabu search-genetic hybrids, we give population sizes $|P|$, the number of generations (2 offsprings/generation), the number of tabu iterations applied to each individual, and the average running time in seconds necessary to find a coloring. For all graphs, we used the graph-adapted recombination operator. For the tabu search procedure alone, we give the average number of iterations and running time in seconds needed to find the coloring. Average running times are computed over five runs.

Table 9

Results on Leighton graphs.

Graph information					Tabu search-genetic hybrid				Tabu search	
$\|V\|$	$\|E\|$	$\chi(G)$	seed X_0	k^*	$\|P\|$	gen.	mutation	time	iterations	time
450	5714	5	0	5	10	14.6	1000	49.2	333543	303.0
450	5734	5	41794	5	10	13.6	1000	45.3	251805	228.4
450	9803	5	35428	5	5	0	1000	15.2	2443.4	3.3
450	9757	5	47927	5	5	1.4	1000	39.0	2549.6	3.3
450	8168	15	36276	15	10	19.0	25000	1920	165498	220.6
450	8169	15	213549	15	10	25.8	25000	2464	149205	191.8
450	16680	15	161712	15	50	33.2	10000	4828	–	–
450	16750	15	160056	15	50	40.2	10000	5036	–	–
450	8260	25	192625	25	1	0	1000	4.0	528.6	4.0
450	8263	25	358531	25	1	0	1000	3.9	455.6	3.9
450	17343	25	247337	26	10	4.0	100000	3089	1518124	2756
450	17425	25	274995	26	10	19.4	100000	3322	2310773	4258

Some of these graphs are much more difficult to color than others. For instance, graphs having a chromatic number of 5 are rather easy to color. However, in two instances out of four, the tabu search-genetic hybrids outperform tabu search. The graphs with chromatic number 15 proved to be much more difficult. For the first two graphs of this class, an optimal coloring is found in less time with the tabu search procedure. However, the tabu search procedure is unable to color optimally the other two graphs (even after 24 hours of CPU time), while the tabu search-genetic hybrid can. Among the Leighton graphs with chromatic number 25, two are very easy and two are very difficult. An excellent challenge would be to try to color optimally the last two graphs of table 9, which we could not achieve with either approach (we had to use 26 colors). In addition to knowing their chromatic numbers, Leighton graphs are characterized by a known number of cliques of different sizes (the size of the largest clique being equal to the chromatic number). It would be interesting to study how these parameters influence the relative difficulty of coloring such graphs, something we were unable to do.

5.3. CHOICE OF PARAMETER VALUES

In general, it is a difficult task to select appropriate parameter values for a genetic algorithm. In [7], Davis describes four different approaches that can be adopted in this regard: carrying out hand optimization, using a genetic algorithm, carrying out brute force search, and using adapting parameter settings. In the case of genetic hybrids where even more parameters have to be set (e.g., number of tabu iterations), the problem becomes even more difficult.

Relying on the results presented in this paper, we can however suggest a few guidelines. When a local search mutation is used, the problem is to find an appropriate population size that will converge to good solutions in a prescribed amount of time. For instance, the results in figure 10 suggest that a population size of 300 is appropriate for this graph. If one can identify a good set of parameters for a particular graph, it may then be reasonable to use those for graphs of the same type.

When tabu search-genetic hybrids are used, the best choice of parameter values depends also on the structure of the graphs. As the results in table 7 indicate, different parameter values can have a big influence on the time required to obtain a coloring. In the case of a distributed implementation, it may be better to use smaller populations and longer searches for each individual if few processors are available, and large populations with smaller searches in a massively parallel context. For the results in table 9, we had to resort to a "trial and error" process to find appropriate values. Further research is needed in order to design more elaborate approaches to set the different parameter values. It is our feeling that a parallel implementation will be required to accomplish this task.

6. Conclusion

In this paper, we examine different ways in which genetic algorithms can be adapted to solve the graph coloring problem. We find that classical implementations can not generate results as good as those generated with other heuristic search procedures. On the other hand, experiments indicate that genetic operators can be used to improve the performance of existing heuristics.

For $G_{n,0.5}$ random graphs, we improve the results obtained with an earlier tabu search implementation [22]. For $G_{n,0.5}$ random graphs of up to 300 nodes, the populations can be evolved to find excellent solutions using hybrid genetic algorithms.

However, similar results can be obtained in much less time with the variant of tabu search presented in this paper. For larger random graphs, running times become prohibitive for genetic hybrid algorithms. For $G_{500,0.5}$ and $G_{1000,0.5}$ graphs, we had to resort to a combined approach in which tabu search-genetic hybrids were instrumental in solving the coloring problem for the residual graphs.

For Leighton graphs, tabu search-genetic hybrids were able to color optimally some instances that could not be solved by our tabu search procedure. In this case, hybridizing genetic operators and tabu search procedures can be seen as a diversification strategy for tabu search, like those proposed in [2,8,19,37]. It would be interesting to further investigate if comparable results could be obtained by using other tabu search long-term strategies. Such issues are considered for the quadratic assignment problem in [38]. Furthermore, it would also be interesting to perform a parallel implementation of the genetic hybrids (as in [31], for instance) and analyze how it compares with parallel tabu search adaptations for which quasi-optimal speedups have also been reported [36,1].

One of the main hindrances to deeply understand and analyze genetic hybrid algorithms is the large amount of computational effort they require. Aside from parallel implementation, other strategies could be considered to accelerate the convergence to good solutions. One of these is the improvement of specialized recombination operators such as the one presented in this paper. Also, the basic implementation of genetic algorithms could be modified according to other approaches, such as the scatter search method suggested by Glover in [16].

Acknowledgements

The authors wish to thank the anonymous referees for their detailed comments that led to this improved version of the paper.

References

[1] R. Battiti and G. Tecchiolli, Parallel biased search for combinatorial optimization: Genetic algorithms and tabu, Microproc. Microsyst. 16(1992)351–367.

[2] R. Battiti and G. Tecchiolli, The reactive tabu search, ORSA J. Comp. 6(1994)126–140.

[3] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, A comparison of genetic sequencing operators, *Proc. 4th Int. Conf. on Genetic Algorithms*, ed. R.K. Belew and L.B. Booker (Morgan Kaufmann, San Mateo, CA, 1991) pp. 69–76.

[4] D. Brelaz, New methods to color vertices of a graph, Commun. ACM 22(1979)251–256.

[5] B. Carter and K. Park, How good are genetic algorithms at finding large cliques: An experimental study, Technical Report BU-CS-93-015, Boston University (1994).

[6] M. Chams, A. Hertz and D. de Werra, Some experiments with simulated annealing for coloring graphs, Euro. J. Oper. Res. 32(1987)260–266.

[7] L. Davis, *Handbook of Genetic Algorithms* (Van Nostrand Reinhold, New York, 1991).

[8] F. Dammeyer and S. Voss, Dynamic tabu list management using the reverse elimination method, Ann. Oper. Res. 41(1993)31–46.

[9] DIMACS, Discrete Mathematics and Theoretical Computer Science anonymous ftp site at dimacs.rutgers.edu.

[10] L.J. Eshelman, The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, in: *Foundations of Genetic Algorithms*, ed. G.J.E. Rawlings (Morgan Kaufmann, San Mateo, CA, 1992) pp. 265–283.

[11] C. Fleurent and J.A. Ferland, Genetic hybrids for the quadratic assignment problem, DIMACS Series, Discr. Math. Theor. Comp. Sci. 16(1994)173–187

[12] B.R. Fox and M.B. McMahon, Genetic operators for sequencing problems, in: *Foundations of Genetic Algorithms*, ed. G.J.E. Rawlings (Morgan Kaufmann, San Mateo, CA, 1992) pp. 284–300.

[13] C, Friden, A. Hertz and D. de Werra, STABULUS: A technique for finding stable sets in large graphs with tabu search, Computing 42(1989)35–44.

[14] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[15] F. Glover, Heuristics for integer programming using surrogate constraints, Dec. Sci. 8(1977) 156–166.

[16] F. Glover, Genetic algorithms and scatter search: Unsuspected potentials, Technical Report, University of Colorado at Boulder (1993).

[17] F. Glover, Future paths for integer programming and links to artificial intelligence, Comp. Oper. Res. 13(1986)533–549.

[18] F. Glover, Tabu search for nonlinear and parametric optimization (with links to genetic algorithms), to appear in Discr. Appl. Math.

[19] F. Glover and M. Laguna, Tabu search, in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R. Reeves (Blackwell Scientific, Oxford, 1993) pp. 70–141.

[20] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison–Wesley, 1989).

[21] J.J. Grefenstette, Incorporating problem specific knowledge into genetic algorithms, in: *Genetic Algorithms and Simulated Annealing* (Morgan Kaufmann, 1987) pp. 42–60.

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

[23] J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, 1975).

[24] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by simulated annealing: An experimental evaluation, Part II; Graph coloring and number partitioning, Oper. Res. 39(1991) 378–406.

[25] A. Johri and D.W. Matula, Probabilistic bounds and heuristic algorithms for coloring large random graphs, Technical Report, Southern Methodist University, Dallas, TX (1982).

[26] P. L'Écuyer, Efficient and portable combined random number generators, Commun. ACM 31 (1988)742–774.

[27] F. Leighton, A graph coloring algorithm for large scheduling problems, J. Res. Nat. Bureau of Standards 84(1979)489–505.

[28] S. Lin, Computer solutions of the traveling salesman problem, Bell. Syst. Tech. J. 44(1965) 2245–2269.

[29] S. Lin and B.W. Kernighan, An effective heuristic for the traveling salesman problem, Oper. Res. 21(1973)498–516.

[30] C. Morgenstern, Distributed coloration neighborhood search, presented at the *15th Int. Symp. on Mathematical Programming*, Ann Arbor (1994).

[31] H. Muhlenbein, M. Gorges-Schleuter and O. Kramer, Evolution algorithms in combinatorial optimization, Parallel Comp. 7(1988)65–88.

[32] H. Muhlenbein, Evolution in time and space – the parallel genetic algorithm, in: *Foundations of Genetic Algorithms*, ed. G.J.E. Rawlings (Morgan Kaufmann, San Mateo, CA, 1992) pp. 316–335.

[33] J.D. Schaffer, L.J. Eshelman and D. Offutt, Spurious correlations and premature convergence in genetic algorithms, in: *Foundations of Genetic Algorithms*, ed. G.J.E. Rawlings (Morgan Kaufmann, San Mateo, CA, 1992) pp. 102–112.
[34] G. Syswerda, Uniform crossover in genetic algorithms, *Proc. 3rd Int. Conf. on Genetic Algorithms* (Morgan Kaufmann, 1989) pp. 2–9.
[35] G. Syswerda, A study of reproduction in generational and steady-state genetic algorithms, in: *Foundations of Genetic Algorithms*, ed. G.J.E. Rawlings (Morgan Kaufmann, San Mateo, CA, 1992) pp. 94–101.
[36] E. Taillard, Robust tabu search for the quadratic assignment problem, Parallel Comp. 17(1991) 443–455.
[37] E. Taillard, Recherches itératives dirigées parallèles, Ph.D. Thesis, École Polytechnique Fédérale de Lausanne (1993).
[38] E. Taillard, Comparison of iteratives searches for the quadratic assignment problem, Technical Report ORWP94/04, DMA, École Polytechnique Fédérale de Lausanne (1994).
[39] D.E. Tate and A.E. Smith, A genetic approach to the quadratic assignment problem, Comp. Oper. Res. 22(1995)73–83.
[40] D. Welsh, *Codes and Cryptography* (Oxford University Press, New York, 1988).
[41] D. Whitley, T. Starkweather and D. Shaner, The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination, in: *Handbook of Genetic Algorithms* (Van Nostrand Reinhold, New York, 1991) pp. 350–372.