A GRASP for Coloring Sparse Graphs

MANUEL LAGUNA
Graduate School of Business, University of Colorado, Boulder, CO 80309-0419, USA
Manuel.Laguna@Colorado.Edu

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
Rafael.Marti@uv.es

Latest revision: November 17, 1999.

ABSTRACT — We first present a literature review of heuristics and metaheuristics developed for the problem of coloring graphs. We then present a Greedy Randomized Adaptive Search Procedure (GRASP) for coloring sparse graphs. The procedure is tested on graphs of known chromatic number, as well as random graphs with edge probability 0.1 having from 50 to 500 vertices. Empirical results indicate that the proposed GRASP implementation compares favorably to classical heuristics and implementations of simulated annealing and tabu search. GRASP is also found to be competitive with a genetic algorithm that is considered one of the best currently available for graph coloring.

kewords: graph coloring, metaheuristics, GRASP

1. Introduction

Consider a graph $G = (V, E)$ with vertex set V and edge set E. A k-coloring of G is a partition of V into k sets $V_1, \ldots, V_k$, such that no two vertices in the same set are adjacent, i.e., if $v, w \in V_i$, $1 \leq i \leq k$ then $(v, w) \notin E$. The sets $V_1, \ldots, V_k$ are referred to as *color classes* or *colors*. The *chromatic number*, $\chi(G)$, is defined as the minimum k for which G is k-colorable. The coloring problem can be stated as follows. Given a graph G, find $\chi(G)$ and the corresponding coloring. Graph coloring is a well-known NP-hard problem (Garey and Johnson, 1979).

Practical applications of graph coloring are numerous. They include exam scheduling (Wood, 1969; Brelez, 1979; Leighton, 1979), the design and operation of Flexible Manufacturing Systems (Stecke, 1985), and the computation of sparse Jacobian elements by finite differencing in mathematical programming (Coleman and More, 1983). In all of these applications the related graph is usually sparse, that is, the number of edges is the same order as the number of vertices.

Since graph coloring is NP-hard, all existing exact procedures require exponential time, precluding their use as problem size increases. As a result of this intractability, a variety of heuristic and metaheuristic approaches have been proposed to produce good colorings in a reasonable amount of time.

2. Heuristics and Metaheuristics for Graph Coloring

The simplest heuristic methods are the *sequential coloring* approaches. First, the vertices are sorted, and the top vertex is put in color class number one. The remaining vertices are considered in order, and each is placed in the first color class for which it has no adjacencies with the vertices already assigned to the class. If no such class exists, then a new class is created. Several different schemes have been used for the initial ordering. The *Largest First* (LF) approach of Welsh and Powell (1967) sorts the vertices by decreasing degree. The *Smallest Last* (SL) ordering of Matula, Marble and Isaacson (1972) lists the smallest degree vertices in reverse order in the following fashion. A vertex of minimum degree is placed last in the list. Assume that $\{v_{k+1}, \ldots, v_n\}$ have been ordered. Choose v_k such that the degree of v_k in the graph induced by $V - \{v_{k+1}, \ldots, v_n\}$ is minimum. Note that both LF and SL tend to color the higher degree nodes first. Although these methods are easy to implement and fast, they often produce colorings which are far from optimal.

The colorings produced by the sequential coloring heuristics can be improved by performing interchanges. At each stage, a previously color vertex is switched to another class, if performing this interchange allows the current vertex to be colored without the addition of a new color. Largest First with Interchange (LFI) and Smallest Last with Interchange (SLI) are the resulting procedures (Matula, Marble and Isaacson, 1972). While the interchanges are more time consuming, they usually improve the performance of the basic routines.

Other techniques have been developed which attempt to reorder the nodes at each stage. DATSUR (Brelez, 1979) chooses the next vertex to color based on its *saturation degree* — the number of color classes for which the vertex is adjacent to at least one vertex in that class. A vertex with the maximum saturation degree is selected and placed in the first legal color class found.

The methods discussed thus far proceed by first choosing a vertex and then assigning an appropriate color. A different approach completes each color before introducing a new one. One of the most successful deterministic heuristics, Recursive Largest First (RLF) due to Leighton (1979), is based on this idea. The method selects vertices one color class at a time. Assume V_1, ..., V_{i-1} have already been constructed, so that V_i is the next class to be completed. Let $\hat{V} = V - \bigcup_{j=1}^{i-1} V_j$ be the set of currently admissible uncolored vertices. Let U be an initially empty set of uncolored vertices that are inadmissible because they are adjacent to at least one vertex in V_i. The method constructs V_i as follows:

1. Choose $v_0 \in \hat{V}$ to be a vertex with *maximum* degree in the graph induced by $\hat{V}$. Place v_0 in V_i and move all $v \in \hat{V}$ that are adjacent to v_0 from $\hat{V}$ to U.

2. While $\hat{V} \neq \emptyset$ repeat:

   Choose $v \in \hat{V}$ of maximum degree in U (where the degree of v is calculated as the number of vertices in U that are adjacent to v).
   Add v to V_i and move all $w \in \hat{V}$ that are adjacent to v from $\hat{V}$ to U.

Steps 1 and 2 are repeated until all nodes are colored. The strategy of this procedure is to make $|V_i|$ large and the graph induced by the remaining vertices sparse.

Randomization has been used in a number of settings as an attempt to improve the performance of simple heuristics. Following this idea, Johnson, et al. (1991) developed the XRLF method. For each color, several candidate classes are generated and the one inducing the fewest edges in the remaining graph is chosen. Each candidate class is constructed using a modified RLF procedure as follows. At each stage, a fixed number of uncolored vertices are selected at random from $\hat{V}$. The one with the largest degree in U is chosen as the next vertex to be colored. When the number of vertices left in the set $\hat{V}$ becomes less than a user specified number SETLIM, the candidate class is made as large as possible by an exhaustive search procedure.

Metaheuristics, such as simulated annealing (Kirkpatrick, et al. 1983), tabu search (Glover and Laguna, 1997), genetic algorithms (Holland, 1975) and neural networks (Hopfield and Tank, 1985), have also been applied to graph coloring. Metaheuristics may or may not use local search but all have the goal of avoiding being trapped in local optima by permitting moves that deteriorate the value of the objective function. Chams, Hertz and de Werra (1987) present results from applying simulated annealing to graph coloring. Both pure simulated annealing and an approach that combines XRLF and simulated annealing are discussed. The hybrid method, which we will denote SA proceeds as follows. XRLF is used to construct color classes until the number of vertices in the graph is below a specified level. Simulated annealing is then applied to color the remaining vertices. Johnson, et al. (1991) report on extensive experiments using several variants of simulated annealing on random graphs of varying size and density.

Hertz and de Werra (1988) describe a tabu search implementation for graph coloring (TABUCOL), which gives good solutions and outperforms simulated annealing on

several random dense graphs tested. Their method attempts to find a legal coloring with k colors, where the value of k can be changed to find improved coloring schemes. A solution $s = \{V_1, \ldots, V_k\}$ is a partition of the set of vertices V into k subsets and $E(V_i) = \{ (v, w) \in E \mid v \text{ and } w \in V_i \}$. The quality of a solution s, is then given by the following objective function:

$$f(s) = \sum_{i=1}^{k} |E(V_i)|$$

The procedure stops when $f(s) = 0$ (or after a pre-specified number of iterations are reached without finding a legal coloring for a given k). A neighbor s' of s is found by first randomly choosing $v' = v$ or w, such that $(v, w) \in E(V_1) \cup \ldots \cup E(V_k)$, and then, assuming that $v' \in V_i$, randomly choosing a color $j \neq i$. The new solution $s' = \{V'_1, \ldots, V'_k\}$ is obtained by:

$$\begin{aligned} V'_j &= V_j \cup \{ v' \} \\ V'_i &= V_i \setminus \{ v' \} \\ V'_r &= V_r \qquad \text{for } r = 1, \ldots, k \text{ and } r \neq i, j. \end{aligned}$$

The procedure selects the best neighbor s' (as measured by the objective function value) from a collection of randomly generated neighbors. The size of the neighborhood is a controlled search parameter. After a move of vertex v from V_i to V_j, a tabu activation rule forbids the move that returns v to V_i for a number of iterations. The number of iterations that the move remains tabu-active is the so-called *tabu tenure* (which becomes another search parameter). Hertz and De Werra (1987) also experimented with hybrid approaches that used TABUCOL within a procedure that sequentially finds independent sets that are as large as possible.

Evolutionary algorithms have also been adapted in the context of graph coloring. Evolutionary methods operate on a population of solutions and seek improved outcomes by a sequence of *cycles* consisting of a *cooperation* step and a *self-adaptation* step. In the cooperation step, solutions in the current population exchange information with the goal of producing new solutions that inherit good attributes. In the self-adaptation step, solutions modify their internal structure without interacting with other solutions in the population. In the context of genetic algorithms, cycles are referred to as *generations*, the so-called *crossover operators* achieve the cooperation step and the self-adaptation step consists of what is known as *mutation*.

Costa, Hertz and Dubuis (1995) describe a procedure that combines a simple descent method to achieve self-adaptation within the general framework of a genetic algorithm. The descent method is based on moving from a solution s to a neighbor solution s' as defined by Hertz and De Werra (1987). The objective function $f(s)$ is modified to assigned weights to edges. The weights are changed from one generation to the next to avoid always manipulating the same conflicting edges (i.e., those edges in $E(V_1) \cup \ldots \cup E(V_k)$). The mutation operator consists of replacing, with a given probability, a solution s by a randomly chosen neighbor s'. The mutation probability is changed during the search, using a systematic scheme. A *union crossover*, originally designed by Costa (1995) for a scheduling application, was adapted for graph coloring to implement the cooperation step.

Fleurent and Ferland (1996) proposed another evolutionary method for graph coloring. This implementation uses a graph-adapted recombination operator for the cooperation step. It also employs an entropy measure to evaluate the diversification of the solutions in the population. If the entropy is zero, then all the solutions in the population are identical. This measure is used both to design stopping rules and to influence parent selection. Another important feature is that members of the population are subject to local search. A variant of the TABUCOL scheme is used as one of the self-adaptation steps. Unlike the original TABUCOL implementation that randomly samples the neighborhood of a solution s, the Fleurent and Ferland's approach considers the entire neighborhood when searching for the best move. They also use a dynamic tabu tenure, which randomly changes within specified bounds. A number of variants are tested in this study, which are the result of choosing (1) a population size, (2) an encoding scheme (permutation or string), (3) a cooperation step (1- or 2-point crossover or graph-adapted recombination), and (4) a self-adaptation step (pure local search, tabu search, or a mutation rate). For graphs with 500 and 1000 vertices, Fleurent and Ferland follow the common approach of finding independent sets first and then applying their procedure to the remaining vertices in the graph.

In another evolutionary approach, Eiben, et al. (1997) employ the 3-coloring problem to test several variants of an asexual evolutionary algorithm. The algorithm uses an order-based representation and an adaptation mechanism. The focus of their research is to explore the applicability of their implementation to constraint satisfaction problems. However, the procedure was tested in the context of graph coloring, where nodes can be viewed as variables while edges can be viewed as constraints. It is important to note that the algorithm does not use context-specific information (e.g., the special structure of graph coloring problems), because the authors are interested in exploring the applicability of their design to other constraint satisfaction problems.

Neural networks have also been applied to the graph coloring problem, this effort was started more than 10 years ago by Dahl (1987) and more recently continued by Jagota (1996). Neural network applications in this context are based on mapping the k-coloring problem to a Hopfield network. This process is accomplished by first considering a reduction to the maximum independent set (MIS) problem, followed by a mapping of MIS onto a Hopfield network. Jagota (1996) follows the approach of choosing an initial k value and gradually decreasing it in an attempt to find improved feasible colorings. If the algorithm fails to find a feasible coloring in one phase, the current k value is increased and the process continues. Since k is initially set to a sufficiently large value, the procedure is guaranteed to yield a proper coloring. The procedure was tested on a set of 30 graphs associated with the Second DIMACS Challenge (Trick, 1993). The results were compared with the parallel procedure *Hybrid* of Lewandowski and Condon, 1993. Jagota's implementation is outperformed by *Hybrid* in all but 6 instances. These instances correspond to the Modified k-partite set, for which the optimal solution is known to be k colors (where k is the number of partitions used to construct each instance). It is interesting to note that after the DIMACS Challenge, the creator of the Modified k-partite set admitted that due to an omission these instances could in fact be optimally solved by a greedy heuristic. However, none of the algorithms in the DIMACS workshop (including *Hybrid*) seemed to be able to exploit this particular feature (Jagota, 1996).

The computational studies in all the procedures reviewed above employ graphs for which the probability that an edge exists is 0.5 or more. In specific, random graphs $G_{n,p}$ are employed, where n is the number of vertices and p (for $p \geq 0.5$) is the probability

that an edge exists between any pair of edges (independently from the existence of any other edge).

We have provided a partial review of the graph coloring literature, focusing on metaheuristic approaches. For more information on the graph coloring problem and a more comprehensive bibliography, we refer the reader to Michael Trick's "Network Resources for Coloring a Graph" (http://mat.gsia.cmu.edu/COLOR/color.html), Joe Culberson's "Graph Coloring Page" (http://web.cs.ualberta.ca/~joe/Coloring/index.html), Tommy Jensen and Bjarne Toft's "Graph Coloring Problems" (http://www.imada.ou.dk/Research/Graphcol/), and Pardalos, et. al (1999) survey.

Our paper presents the results of applying a *greedy randomized adaptive search procedure* (GRASP) to the problem of coloring sparse graphs. GRASP (Feo and Resende, 1989 and 1995) consists of two phases: construction phase and improvement phase. For the construction phase, our GRASP implementation uses a randomized version of RLF to generate initial colorings. The improvement step consists of a local search that finds improved neighbor solutions. Data structures and sorting routines that exploit sparsity are employed to increase efficiency. The proposed procedure is tested in a set of instances for which $\chi(G)$ is known (Trick, 1993) and large sparse random graphs (i.e., $G_{n,0.1}$).

The next section presents a detailed description of the proposed procedure. This description includes pseudo-code for both the randomized RLF adaptation for the construction phase, and the local search for the improvement phase. The data structures used to exploit sparsity are also discussed together with the computational complexity of the method. Section 3 presents the empirical results and concluding remarks are given in section 4.

3. Proposed Procedure

Figure 1 gives a pseudo-code for our GRASP implementation, denoted by **GraspColor**. In addition to $G = (V, E)$, **GraspColor** takes as user defined input *GIter*, the number of GRASP iterations, *CIter*, the number of iterations per color, and *CSize*, the size of the candidate list. The output from **GraspColor** is k^*, the minimum number of colors used and $V^* = \{V_1, \ldots, V_{k^*}\}$ the corresponding color classes.

The outer **for** loop (lines 2 to 32) performs the GRASP iterations consisting of the construction and improvement phases. The number of color classes i and the vertex set V^* are initialized for the current iteration in line 3. The construction phase is executed by the outer **while** loop (lines 4 to 27). It constructs the next coloring, one color at a time. The inner **for** loop (lines 6 to 25) generates the *CIter* candidate color classes from which the best is chosen. Each class is constructed by the inner **while** loop in lines 9 to 20. Lines 10 to 14 shows the construction of the candidate list (*CL*) of vertices. The list is constructed selecting *CSize* vertices from $\hat{V}$. Note that for the first vertex selected in each class, overall degree is used (i.e., the degree with respect to V is used). Randomization occurs in line 15, where a vertex is randomly chosen from the candidate list. This vertex is then added to the current color class in line 16. The chosen vertex and its neighbors are removed from the set of admissible uncolored vertices $\hat{V}$, and the neighbors are added to the inadmissible uncolored set U. This constitutes the adaptive component of our GRASP implementation.

Figure 1. GRASP for Graph Coloring.

```
GraspColor(V, E, GIter, CIter, CSize, k*, V*)
1.      k* = |V|;
// Outer loop, GRASP iterations
2.      for iter = 1 to GIter
3.          i := 0; Vt = V;
4.          while Vt ≠ ∅
5.              i := i + 1;
6.              ecount := ∞;
// Start construction phase
7.              for j = 1 to CIter
8.                  V̂ := Vt; U := ∅; C := ∅;
9.                  while V̂ ≠ ∅
// Build candidate list
10.                     if U = ∅ then
11.                         CL := { CSize vertices of max degree in V̂ };
12.                     else
13.                         CL := { CSize vertices in V̂ of max degree in U };
14.                     end if
15.                     Select v ∈ CL at random;
16.                     C := C ∪ { v };
17.                     N(v) := {w | (v, w) ∈ E};
// Update V̂ and U
18.                     V̂ := V̂ - { v } – N(v);
19.                     U := U ∪ N(v);
20.                 end while
//Determine set of edges in Vt - C
21.                 E* = {(u, v) ∈ E | u, v ∈ Vt - C };
// Update best color class
22.                 if | E* | < ecount then
23.                     Vi := C; ecount := | E* |;
24.                 end if
25.             end for
// Update set of uncolored vertices
26.             Vt = Vt - Vi;
27.         end while
28.         ImprovePhase(V, E, i, { V1, …, Vi });
// Update best coloring
29.         if i < k* then
30.             V* := { V1, …, Vi }; k* := i;
31.         end if
32.     end for
end GraspColor
```

Once a color is maximal, the number of edges remaining in the graph induced by the uncolored vertices are counted in line 21. If this number is less than the current best, *ecount*, then the color class is saved as the incumbent in line 23. After *CIter* classes

have been sampled, the best class is removed from the graph in line 26. Once a coloring is completed, it is passed to the improvement phase (**ImprovePhase**, line 28). If the solution returned by **ImprovePhase** is the smallest found so far, then it is saved in V^* as the incumbent in line 30. After completing *GIter* GRASP iterations the current incumbent is returned as the solution.

There are several important differences between **GraspColor** and XRLF. XRLF uses the greedy function to select the best candidate from a set of randomly chosen vertices, while **GraspColor** uses the objective function to rank all candidate vertices and then randomly chooses among the *CSize* best. Furthermore, XRLF is executed once, while **GraspColor** produces many different sample solutions, and selects the best.

The improvement phase **ImprovePhase** is given in Figure 2. Its input is the graph $G = (V, E)$ and a valid i-coloring $\{V_1, \ldots, V_i\}$. The output is the k-coloring $\{V_1, \ldots, V_k\}$ with $k \leq i$. The procedure attempts to reduce the size of the current coloring as follows:

1. Let V_{s1} and V_{s2} be the smallest and second smallest cardinality color classes, respectively. Combine the two classes into one, leaving $k = i-1$ classes. Let $s = \{\tilde{V}_1, \ldots, \tilde{V}_k\}$ denote these classes.

2. Let $f(s) = \sum_{j=1}^{k} \left| E(\tilde{V}_j) \right|$, where $E(\tilde{V}_j)$ is the set of edges with both endpoints in $\tilde{V}_j$.

Figure 2. Improvement phase of GRASP.

```
ImprovePhase(V, E, i, {V1, ..., Vi})
1.      do
2.          Sort V such that |V1| ≥ |V2| ≥ ... ≥ |Vi-1| ≥ |Vi|;
3.          k := i – 1;
4.          for j = 1 to k
5.              Ṽj := Vj;
6.          end for
7.          s := { Ṽ1, ..., Ṽk ∪ Vi };
8.          LocalSearch(k, s);
9.          if f(s) = 0 then
10.             for j = 1 to k
11.                 Vj := Ṽj ;
12.             end for
13.             i := k;
14.         end if
15.     while f(s) = 0;
end ImprovePhase
```

3. Apply a local search to minimize $f(s)$. The local search operates as follows:

3.1 *NoImprove* := 0;
3.2 **while** ($f(s) > 0$ and *NoImprove* < *NoImpIter*)
3.3 Randomly choose an illegal vertex (i.e., one that is colored with the same color as an adjacent vertex).
3.4 Make all possible attempts to switch v to a different color to improve the current value of $f(s)$.
3.5 **if** step 3.4 is successful **then** *NoImprove* := 0 **else** *NoImprove* := *NoImprove* + 1.
3.6 **end while**

4. If a solution s with $f(s) = 0$ is encountered, then s is a valid coloring. Save such coloring by setting $i = k$ and $V_j = \tilde{V}_j$ for $1 \le j \le k$. Return to step 1 and restart the improvement phase.

5. If the local search terminates with $f(s) > 0$, then no improved coloring was found.

The computationally expensive operations in procedure **GraspColor** are ranking the set of uncolored vertices in $\hat{V}$, updating the sets $\hat{V}$ and U when a new vertex is colored, and executing **ImprovePhase**. We keep the graph in a dynamic list of vertices, where each vertex can be accessed either directly or through an adjacent vertex. The dynamic nature of the list allows us to handle the memory more efficiently. Specifically, each vertex is stored in a structure that contains all the relevant information about the vertex, including a list of pointers to the structures where adjacent vertices are stored.

The candidate list starts with *CSize* uncolored vertices and we maintain a pointer to the vertex with the smallest degree. As we examine the rest of the vertices, we compare their degree with the lowest degree vertex currently in *CL* and update the list and the pointer if necessary. Each vertex is examined only once. There are $O(n)$ operations to create the initial candidate list *CL*.

A dynamic linked list is used to keep track of the colored vertices. The list creates a new pointer every time a vertex is colored. This mechanism creates a direct access to each vertex in the current coloring. In our empirical tests, the improvement phase required on average less than 20% of the amount of time spent in a complete GRASP iteration.

4. Computational Results

Two families of graphs were chosen for computational testing. The first class includes instances with known optimal solutions found in Michael Trick's "Graph Coloring Instances" page (http://mat.gsia.cmu.edu/COLOR/instances.html).

LEI: Leighton Graphs (Leighton, 1979)
MYC: Graphs based on the Mycielski transformation (Michael Trick).
REG: Graphs based on register allocation for variables in real code (Gary Lewandowski).
SGB: Graphs from Donald Knuth's Stanford GraphBase.

The LEI instances are generated by a procedure proposed by Leighton (1979), which constructs graphs of known chromatic number. Given the number of nodes n, the desired chromatic number k, the average node degree d, and a random vector of non-

negative integers (b_k, b_{k-1}, ..., b_2) with $b_k \geq 1$, the method introduces edges such that there are b_i cliques of size i and the chromatic number and average vertex degree are k and d, respectively. Since the true chromatic number is known, the graphs are useful when assessing the performance of a heuristic. The instances consist of twenty seven 150-node graphs and twelve 450-vertex graphs with chromatic number ranging from 5 to 25, and with average vertex degrees ranging from 11 to 77. We restrict our testing to the twelve instances with 450 vertices.

The MYC set consists of 5 instances with known optima. We use all five instances for testing. The REG set consists of 14 instances with known optima. We also use all 14 instances for testing. The SGB instances are divided into four sets: book graphs, game graphs, mile graphs, and queen graphs. In our testing we do not use the queen graph instances because optimal solutions are known for all of them. We use 10 out of the 11 remaining graphs, because we encountered problems reading the book graph labeled "homer" (based on Homer's Iliad).

The second set of test graphs is drawn from the class of random graphs $G_{n,p}$, with n vertices and edge probability p. This means that each edge in the graph appears with probability p, independently of any other edge. These graphs are generated as follows. For each (unordered) pair $i, j \in \{1, ..., n\}$ a uniform random number r is drawn from the interval [0, 1]. The edge (i, j) is included in E if and only if $r \leq p$. While the actual chromatic number of these graphs is not known, Bollobas and Thompson (1985) derive probabilistic estimates for the expected size of an optimal coloring. Given the interest in sparse graphs, $p = 0.1$ and $n = 50$, 100, 250 and 500 were chosen. For each n, twenty five random graphs were generated from $G_{n,0.1}$, for a total of 100 instances.

We performed a preliminary experiment to determine the best values for the parameters in **GraspColor**. For this experiment we used the 41 instances from the LEI, MYC, REG and SGB sets. For each search parameter, we tested 3 values:

$GIter = 5, 15, 25$
$CIter = 5, 15, 25$
$CSize = 3, 6, 9$
$NoImpIter = n, n/2, 2n$

We ran the procedure 81 times on each instance, to test all combinations of the parameter values, resulting in 3,327 runs. The parameter combination that yielded the best results was:

$GIter = 15$
$CIter = 15$
$CSize = 3$
$NoImpIter = n/2$

These values are used throughout the rest of our experimentation. The parameters for other procedures were set as recommend by their creators. We start by comparing our GRASP implementation with other procedures reported in the literature. We use the procedures RLF (Leighton, 1979), XRLF (Johnson, et al. 1991), a simulated annealing (SA) with the Fixed-k objective function (Johnson, et al. 1991), TABUCOL (Hertz and de Werra, 1988) and GA (Fluerent and Ferland, 1996). We chose the Fixed-k objective function for SA, because it was reported as the best among three schemes when dealing with sparse graphs (Johnson, et al. 1991). The results of this experiment are reported in Table 1.

Table 1. Average number of colors used (optimal known).										
Name	*Instances*	*n*	*m*	OPT	RLF	XRLF	SA	TABUCOL	GA	GRASP
LEI	12	450.0	11005.5	15.0	17.8	16.9	18.0	19.0	15.8	16.5
MYC	5	73.4	688.4	6.0	6.0	6.0	6.0	6.0	6	6.0
REG	14	362.1	7608.1	37.4	37.4	37.4	40.1	57.1	54.5	37.4
SGB	10	113.9	2835.2	22.6	22.6	22.9	26.0	24.1	23.1	22.7
Tot./Avg.	41	249.9	5534.3	20.2	20.9	20.8	22.5	26.6	24.9	20.6

The columns in Table 1 consist of: (1) identifier of instances, (2) number of instances, (3) number of vertices, (4) average number of edges, (5) average number of colors used in the optimal coloring, (6) to (11) average number of colors used per each procedure.

It is interesting to note that the simple heuristics RLF and XRLF perform remarkably well in this set of problems (with average number of colors equal to 20.9 and 20.8, respectively). We should also mention that the performance of SA and TABUCOL is highly dependent on the chosen values of their search parameters. Most notably is the selection of the initial k (i.e., the number of colors for which the procedure is attempting to find a legal coloring). Note also that the GA approach was only marginally better than TABUCOL. The number of optima found by each procedure and the average computer time are reported in Table 2. The times (in seconds) were obtained by running the procedures (coded in C) on a personal computer with a 350 MHz Pentium processor.

Table 2. Number of optima / computer time (in seconds).						
Name	RLF	XRLF	SA	TABUCOL	GA	GRASP
LEI	3 / 0.10	5 / 2.23	2 / 142.36	0 / 13.64	6 / 138.10	6 / 27.59
MYC	5 / 0.00	5 / 0.08	5 / 0.12	5 / 0.79	5 / 6.93	5 / 0.47
REG	14 / 0.07	14 / 1.22	3 / 20.93	0 / 43.83	0 / 101.27	14 / 16.65
SGB	10 / 0.01	8 / 0.20	6 / 0.23	8 / 2.46	7 / 0.23	9 / 2.05
Total / Avg.	32 / 0.05	32 / 0.93	16/ 40.91	13 / 15.18	18 / 71.70	34 / 11.69

Table 2 reveals the inferior performance of SA, TABUCOL and GA when considering the current set of problems. These procedures find the least number of optimal solutions and require the most amount of computer time. GRASP is competitive in terms of number of optimal solutions found (34 out of 41), however, it requires several orders of magnitude more time than RLF or XRLF.

In our second set of experiments, we employ the 100 $G_{n,0.1}$ instances generated as described above. The results of this experiment are reported in Table 3, where the values indicate the average number of colors used by each procedure.

Table 3. Average number of colors used ($G_{n,0.1}$ graphs).							
n	*m*	RLF	XRLF	SA	TABUCOL	GA	GRASP
50	218.5	5.20	5.16	5.84	5.00	4.84	4.92
100	889.8	7.88	7.56	8.60	7.76	7.00	7.08
250	5654.5	14.08	13.48	14.36	14.00	12.04	13.04
500	22708.8	23.20	22.16	21.48	23.48	20.40	21.88
Average	7367.9	12.59	12.09	12.57	12.56	11.07	11.73

The results in Table 3 disclose the advantage of using GRASP when dealing with sparse graphs, when compared to RLF, XRLF, SA and TABUCOL. The GA approach marginally outperforms GRASP in this set. However, the quality of the solutions found by GA comes at a high price in terms of computational time. The times (in seconds) and the number of best solutions found by each procedure are reported in Table 4. The best solutions are found by considering the solutions generated by all procedures.

Table 4. Number of best / computer time (in seconds).

n	RLF	XRLF	SA	TABUCOL	GA	GRASP
50	16 / 0.00	17 / 0.04	7 / 0.07	21 / 0.22	25 / 5.21	24 / 0.16
100	4 / 0.00	11 / 0.15	16 / 1.71	6 / 0.78	25 / 10.40	24 / 0.81
250	0 / 0.04	0 / 0.84	0 / 13.49	0 / 5.15	25 / 46.73	25 / 8.71
500	0 / 0.25	0 / 3.79	6 / 329.75	0 / 29.42	25 / 204.03	13 / 61.23
Total / Avg.	20 / 0.07	28 / 1.20	29 / 258.79	27 / 26.68	100/ 66.59	46 / 17.73

While the GA approach provides all the best solutions in the set, GRASP is able to match 46 out of 100 using 73% less computer time. While RLF and XRLF continue to be very fast, the quality of their solutions quickly deteriorates as *n* increases. SA is able to provide 29 best solutions, but it has the highest average solution time of all procedures. The largest contribution to this average comes from the solution time to the set of instances with 500 vertices.

We take advantage of the flexibility of the GRASP design to implement two variants of the method. The variants use the same general framework described in section 2, but replace the improvement phase with two different procedures: a tabu search (GRASP-TS) and a simulated annealing (GRASP-SA). We perform a final experiment to assess the merit of replacing the somewhat simple improvement phase in the original scheme. We employ the $G_{n,0.1}$ instances for this purpose and report the results in Tables 5 and 6.

Table 5. Average number of colors used ($G_{n,0.1}$ graphs).

n	GRASP	GRASP-TS	GRASP-SA
50	4.92	4.92	4.88
100	7.08	7.04	7.08
250	13.04	13.00	13.00
500	21.88	21.84	21.92
Average	11.73	11.70	11.72

Table 6. Number of best / computer time (in seconds).

n	GRASP	GRASP-TS	GRASP-SA
50	23 / 0.16	23 / 0.34	24 / 0.20
100	24 / 0.81	25 / 1.34	24 / 1.00
250	24 / 8.71	25 / 11.07	25 / 10.45
500	22 / 61.24	23 / 69.25	21 / 71.61
Average	93 / 17.73	96 / 20.50	94 / 21.06

The GRASP-TS variant seems to be marginally better than the other two in terms of solution quality, in particular as n increases. These results are not surprising, considering that the termination criteria used for the improvement phase limited the amount of exploration that either TS or SA could perform. When combining GRASP constructions with other metaheuristics, a decision must be made regarding the amount of time spent in each phase of the procedure. The tradeoff is between performing more GRASP iterations (i.e., to favor constructions) and spending more time in the improvement phase (i.e., to favor local search). This issue is equivalent to finding the right balance between search diversification (GRASP constructions) and search intensification (Improvement Phase) within the tabu search framework. In our particular implementation, we have found that the GRASP constructions are of such a high quality that the improvement procedure yields better results only about 3% of the time.

5. Conclusions

In this paper we have first review some relevant work in the graph coloring area. We focused our partial review on implementation of metaheuristics in the context of graph coloring. We then describe a GRASP implementation for coloring graphs, which exploits features of sparse graphs.

Our testing shows that the GRASP implementation is competitive when dealing with graphs that are denser that we had in mind when designing the procedure. The optimal solutions to these instances are known, and our GRASP implementation is able to match 34, out of 41, in an average of 35 seconds. The merit of the proposed procedure becomes more evident in a second experiment that uses sparse random graphs. Finally, we tested the idea of combining GRASP constructions with an improvement phase consisting of another metaheuristic. We implemented variants that combined GRASP with both tabu search and simulated annealing. Our experiments show that if the amount of time in the improvement phase is limited, metaheuritics are not able significantly improve the performance of the overall method when compared to simple descent mechanisms.

Acknowledgments

The authors would like to express their gratitude to Tom Feo and Stuart Smith for sharing their experiences with implementing GRASP in the context of graph coloring. Also, they would like to thank Charles Fleurent for sharing his genetic algorithm implementation.

References

Bollobas, B. and A. Thompson (1985) “Random Graphs of Small Order,” *Annals of Discrete Mathematics*, vol. 28, pp. 249-255.

Brelez, D. (1979) “New Methods to Color Vertices of a Graph,” *Comm. ACM*, vol. 22, pp. 251-256.

Chams, M., A. Hertz and D. de Werra (1987) “Some Experiments with Simulated Annealing for Coloring Graphs,” *European Journal of Operational Research*, vol. 32, pp. 260-266.

Coleman, T. F. and J. J. More (1983) "Estimation of Sparse Jacobian Matrices and Graph Coloring Problems," *SIAM J. Numer. Anal.*, vol. 20, no. 1, pp. 187-209.

Costa, D. (1995) "An Evolutionary Tabu Search Algorithm and the NHL Scheduling Problem," *INFOR*, vol. 33, no. 3, pp. 161-178.

Costa, D., A. Hertz and O. Dubuis (1995) "Embedding of a Sequential Procedure within an Evolutionary Algorithm for Coloring Problems in Graphs," *Journal of Heuristics*, vol. 1, no. 1, pp. 105-128.

Dahl, E. D. (1987) "Neural Networks Algorithms for an NP-Complete Problem: Map and Graph Coloring," *IEEE International Conference on Neural Networks*, IEEE, New York, pp. 113-120.

Eiben, A. E., J. K. van der Hauw and J. I. van Hemert (1997) "Graph Coloring with Adaptive Evolutionary Algorithms," Dept. of Computer Science, Leiden University, The Netherlands.

Feo, T. and M. G. C. Resende (1989) "A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem," *Operations Research Letters*, vol. 8, pp. 67-71.

Feo, T. and M. G. C. Resende (1995) "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 2, pp. 1-27.

Fleurent, C. and J. A. Ferland (1996) "Genetic and Hybrid Algorithms for Graph Coloring," *Annals of Operations Research*, vol. 63, pp. 437-464.

Hertz, A. and D. de Werra (1988) "Using Tabu Search Techniques for Graph Coloring," *Computing*, vol. 39, pp. 345-351.

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

Hopfield, J. J. and D. W. Tank (1985) "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, vol. 52, pp. 141-152.

Jagota, A. (1996) "An Adaptive, Multiple Restarts Neural Network Algorithm for Graph Coloring," *European Journal of Operational Research*, vol. 93, pp. 257-270.

Johnson, D. S., C. A. Aragon, L. A. Mcgeoch and C. Schevon (1991) "Optimization by Simulated Annealing: An Experimental Evaluation — Part II (Graph Coloring and Number partitioning)," *Operations Research*, vol. 31, pp. 378-406.

Kirkpatrick, S., C. D. Gelatt Jr. and M. P. Vecchi (1983) "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680.

Leighton, F. T. (1979) "A Graph Coloring Algorithm for Large Scheduling Problems," *J. Res. Nat. Bur. Standard*, vol. 84, no. 6, pp. 489-506.

Lewandowski, G. and A. Condon (1993) "Experiments with Parallel Graph Coloring Heuristics," Technical Report 1213, Computer Science Department, University of Wisconsin, Madison.

Matula, D. W., G. Marble and J. D. Isaacson (1972) "Graph Coloring Algorithms," on *Graph Theory and Computing,* R. C. Read, ed., Academic Press, New York, pp. 104-122.

Pardalos, P. M., T. Mavridou and J. Xue (1999) "The Graph Coloring Problem: A Bibliographic Survey," in *Handbook of Combinatorial Optimization,* D.-Z. Du and P. M. Pardalos (Eds.), vol. 2, pp. 331-395.

Stecke, K. (1985) "Design Planning, Scheduling and Control Problems of Flexible Manufacturing," *Annals of Operations Research,* vol. 3, pp. 3-12.

Trick, M. (1993) "The Second DIMACS Challenge," http://mat.gsia.cmu.edu/challenge.html.

Welsh, D. J. A. and M. B. Powell (1967) "An Upper Bound for the Chromatic Number of a Graph and its Application to Timetabling Problems," *Comput. J.,* vol. 10, pp. 85-86.

Wood, D. C. (1969) "A Technique for Coloring a Graph Applicable to Large Scale Timetable Problems," *Computer Journal,* vol. 12, pp. 317-322.