Scatter Search for Graph Coloring*

Jean-Philippe Hamiez[1] and Jin-Kao Hao[2]

[1] LGI2P, École des Mines d'Alès, EERIE
Parc Scientifique Georges Besse
F-30035 Nîmes Cedex 01, France
hamiez@site-eerie.ema.fr
[2] LERIA, Université d'Angers
2, Bd. Lavoisier
F-49045 Angers Cedex 01, France
Jin-Kao.Hao@univ-angers.fr

Abstract. In this paper, we present a first *scatter search* approach for the *Graph Coloring Problem* (GCP). The evolutionary strategy scatter search operates on a set of configurations by combining two or more elements. New configurations are improved before replacing others according to their quality (fitness), and sometimes, to their *diversity*. Scatter search has been applied recently to some combinatorial optimization problems with promising results. Nevertheless, it seems that no attempt of scatter search has been published for the GCP. This paper presents such an investigation and reports experimental results on some well-studied DIMACS graphs.

1 Introduction

Scatter search [13, 14] is an evolutionary approach related to the tabu search metaheuristic [12]. It is based on strategies proposed in the 1960s for combining decision rules and constraints. This approach has only been applied recently to a few optimization problems. Applications of this method include, e.g., vehicle routing [27] and unconstrained optimization [23]; see also [13] for more references.

Like other population-based methods, scatter search uses combination of configurations to generate new configurations which can replace others in the population. But the way combinations and replacements are made differs from the traditional strategies used in genetic algorithms. Combinations operate on multiple parents, and replacements rely on the improvement of a fitness function as well as the improvement of the population diversity. Furthermore, scatter search generally works with a small set of configurations and uses deterministic heuristics as much as possible in place of randomization to make a decision.

Graph k-coloring can be stated as follows: given an undirected graph G with a set V of vertices and a set E of edges connecting vertices, k-coloring

* This work was partially supported by the Sino-French Joint Laboratory in Computer Science, Control and Applied Mathematics (LIAMA) and the Sino-French Advanced Research Programme (PRA).

G means finding a partition of V into k classes $V_1, \ldots, V_k$, called *color classes*, such that no couple of vertices $(u, v) \in E$ belongs to the same color class. Formally, $\{V_1, \ldots, V_k\}$ is a valid k-coloring of the graph $G = (V, E)$ if $\forall i \in [1..k]$ and $\forall u \in V_i, \nexists v \in V_i / (u, v) \in E$. The *graph coloring problem* (GCP for short) is the **optimization problem** associated with k-coloring. It aims at searching for the minimal k such that a proper k-coloring exists. This minimum is the *chromatic number* $\chi(G)$ of graph G.

k-coloring and the GCP are well-known *NP-hard* problems [20] and only small problem instances can be solved exactly within a reasonable amount of time in the general case [5]. It is also hard even to approximate the chromatic number of a graph. In [25], it is proved that for some $\epsilon > 0$, approximating the chromatic number within a factor of n^ϵ is NP-hard. Indeed, one of the best known approximation algorithm [15] provides an extremely poor performance guarantee[1] of $O(n(\log\log n)^2/(\log n)^3)$ for a graph with n vertices.

Graph coloring has many real applications, e.g., timetable construction [24] or frequency assignment [9]. There are many resolution methods for this problem: greedy constructive approaches, e.g., DSATUR [1] and the Recursive Largest First algorithm [24], hybrid strategies like HCA [4, 8] and those proposed in [6, 26], local search meta-heuristics, e.g., simulated annealing [18] or tabu search [3, 16], neural network attempts [17], ... See [2] for a more extensive list of other references about the GCP.

Despite the fact that the literature on graph coloring is always growing, there exists, to our knowledge, no approach relying on scatter search for the GCP. The goal of our study is then to provide an experimental investigation of scatter search applied to the GCP. The paper is organized as follows: Sect. 2 recalls the general template of scatter search; Sect. 3 presents our first scatter search algorithm for graph coloring; next section gives the results we obtained on some of the well-known DIMACS benchmark graphs [19], before concluding.

2 General Design of Scatter Search

We briefly recall here the components of scatter search; fundamental concepts and motivations are described in [13]. See also [14] for an exhaustive illustration of these components on a non-linear optimization problem.

1. **Diversification Generation Method**. This step is used first to initialize the population and, eventually, to rebuild a subset of the population during the search. Configurations are built in order to respect a maximal diversity[2]. See [21], e.g., for an illustration of an appropriate generator for a 0-1 knapsack problem;

[1] The performance guarantee is the maximum ratio, taken over all inputs, of the color size over the chromatic number.

[2] One of the lessons provided in [22] suggests to "consider the use of *frequency memory* to develop *effective diversification*".

2. **Improvement Method**. New configurations, obtained in step 1 or by combination, are improved in quality;
3. **Reference Set Update Method**. Improved configurations are checked for replacing others in a *reference set* ($RefSet$ for short) according to their quality (line 2 in Algorithm 1) or their diversity (line 3)[3]. $RefSet$ consists of b best evaluated configurations ($BestSet$) and d most diverse configurations ($DivSet$). See [23] for other update methods;
4. **Subset Generation Method**. This step produces subsets of configurations (with two elements or more) from the reference set to be combined;
5. **Configuration Combination Method**. Subsets of configurations built in step 4 are combined, generally using a problem-dependent combination operator. Laguna and Armentano [22] also suggest that "the use of *multiple* combination methods can be effective".

Algorithm 1: Scatter search outline

begin
  $P \leftarrow \emptyset$ /* Start with an empty population P */
  while P is not full **do**
    Build a new configuration c with the *Diversification generation method*
    Apply the *Improvement method* to c to obtain c^*
    if $c^* \notin P$ **then** $P \leftarrow P \cup \{c^*\}$
  Build $BestSet$ by selecting / removing in P the b best evaluated configurations
  Build $DivSet$ by selecting in P the d most diverse configurations
  $RefSet \leftarrow BestSet \cup DivSet$
[1]   **while** *RefSet* contains new elements **do**
    for each subset i built by applying the *Subset generation method* on *RefSet* **do**
      Use the *Configuration combination method* with i to obtain c_i
      Apply the *Improvement method* on c_i to obtain c_i^*
[2]   **if** $c_i^* \notin RefSet$ **and** evaluation of c_i^* is better than the worst evaluated element c_{worst} in $BestSet$ **then**
        $BestSet \leftarrow BestSet \cup \{c_i^*\}; BestSet \leftarrow BestSet - \{c_{worst}\}$
[3]   **else if** $c_i^* \notin DivSet$ **and** c_i^* add diversity to $DivSet$ **then**
        $DivSet \leftarrow DivSet \cup \{c_i^*\}$
        Remove the element with minimal diversity in $DivSet$
end

Algorithm 1 gives an overall view of one single iteration of a generic scatter search procedure, and thus, the way the components are linked[4]. The main loop 1 controls the termination of the procedure: the process stops when the population evolves no more in quality. Note that some scatter search procedures also use

[3] Note that some scatter search procedures only use the quality criterion to update $RefSet$. In this case, remove line 3.
[4] See [10] for helpful information on practical implementation.

the evolution of the $RefSet$ diversity as a stop criterion. More iterations can be done by restarting loop 1 with a new set of configurations composed of the b best evaluated configurations found by the previous iteration and d new diverse configurations built using the diversification generation method.

3 Scatter Search for Graph Coloring

In this section we describe the five components of our scatter search procedure dedicated to the graph coloring problem. These components are organized in the same way as in Algorithm 1. Let us first describe some concepts useful for the understanding of the overall procedure and its composing elements.

Configuration: a configuration c is any partition $\{V_1, \ldots, V_k\}$ of V into k subsets. $V_i (i \in [1..k])$ is an *independent set* if $\forall u, v \in V_i, (u, v) \notin E$. c is a *proper k-coloring* if each $V_i \in c$ is an independent set. We will refer to *partial coloring* for configurations in which some vertices are not assigned a color.

Evaluation Function: two configurations can be compared in terms of quality using an evaluation (or a fitness) function f: $f(c) = |\{(u, v) \in V_i (V_i \in c, i \in [1..k])/(u, v) \in E\}|$. In other words, f counts the edges having both endpoints in the same color class. Solving a k-coloring instance means finding a particular configuration c^* such as $f(c^*) = 0$.

General Resolution Strategy for Graph Coloring: k-coloring aims at finding a complete assignment of k colors to the vertices that satisfies all the constraints. Such an assignment is said consistent (*proper k-coloring*). The generalized GCP can then be stated as solving successive k-colorings with decreasing values of k until no proper k-coloring can be obtained.

3.1 Diversification Generation Method

We generate conflicting configurations with k colors by means of *random* independent sets built using Algorithm 2.

Algorithm 2: A diversification generation method for the GCP

begin
  for $i = 1$ to k, by 1 **do**
    $A \leftarrow \emptyset; V_i \leftarrow \emptyset$
    while $V \neq \emptyset$ **do**
[1]   Choose (randomly) a vertex $v \in V$
      $V \leftarrow V - \{v\}$; $V_i \leftarrow V_i \cup \{v\}$
      for each $u \in V/(u, v) \in E$ **do**
        $A \leftarrow A \cup \{u\}$; $V \leftarrow V - \{u\}$
    $V \leftarrow A$
  Put each $v \in V$ in a color class such as it minimizes the conflicts over the graph
end

Randomization is used here (line 1) only to insure diversity. Other meaningful choice rules can easily replace it. For instance, the vertices can be selected in decreasing order of their saturation degree, like in DSATUR [1].

3.2 Improvement Method

We use a tabu search algorithm to improve new configurations. This algorithm iteratively makes *best 1-moves*, changing the current color of a conflicting vertex[5] to another one, until achieving a proper coloring. "Best moves" are those, which minimize the difference between the fitness of the configuration before the move is made and the fitness of the configuration after the move is performed. In case of multiple best 1-moves, choose one randomly. A tabu move leading to a configuration better than the best configuration found so far, within the same execution of the improvement method or within the overall scatter search procedure, is always accepted (*aspiration criterion*).

The tabu tenure l is dynamically computed by the formula proposed in [3]:

$$l = \alpha \times |E_c| + random(g) \qquad (1)$$

where E_c is the set of conflicting edges in configuration c. $random(g)$ is a function which returns an integer value uniformly chosen in $[1..g]$. α weights the number of conflicting edges.

A move m can be characterized by a triplet (u, V_{old}, V_{new}), $u \in V$, V_{old} and V_{new} being, respectively, the previous and the new colors of the conflicting vertex u. So, when a move m is performed, assigning u to the color class V_{old} is forbidden for the next l iterations by introducing the (u, V_{old}) couple in the tabu list.

The algorithm stops when a solution is obtained or when a maximum number of moves have been carried out without finding a solution. Algorithm 3 gives an outline of the procedure, which is extracted from an effective *generic tabu search* [3] designed for various coloring problems (k-coloring, GCP, T-coloring and set T-coloring).

Algorithm 3: A tabu search for graph coloring

```
begin
    Let c be the configuration to improve
    TL ← ∅ /* Initialize the tabu list TL to empty */
    c* ← c
    while f(c*) > 0 and not Stop condition do
        Update c by performing a best 1-move m(u, V_old, V_new)
        TL ← TL ∪ (u, V_old)
        if f(c) < f(c*) then
            c* ← c
end
```

[5] A vertex $u \in V_i$ is said conflicting if $\exists v \in V_i / (u, v) \in E$.

3.3 Reference Set Update Method

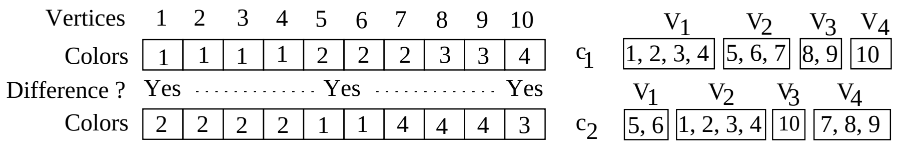
We use the same reference set update method as the one described in Algorithm 1 (Sect. 2):

1. An improved configuration c_{new} replaces another one in $BestSet$ if its evaluation is better than the worst evaluated configuration c_{worst} in $BestSet$.
2. If c_{new} is not kept due to its quality (step 1), it is checked for diversity. c_{new} replaces the configuration with lowest diversity c_{worst} in $DivSet$ if c_{new} has a higher diversity than c_{worst}.

To compute the diversity $D(c_1, c_2)$ between two configurations c_1 and c_2, we use the distance measure introduced in [8]. The distance between c_1 and c_2 is the minimum number of 1-moves necessary to transform c_1 into c_2. Let us call it the *move distance* (D_M).

Note that the Hamming distance $D_H(c_1, c_2)$ is not well adapted to compare, in terms of diversity, two configurations c_1 and c_2 of the GCP. This is due to the definition itself of this distance. For two strings (configurations) c_1 and c_2 of the same length, $D_H(c_1, c_2)$ is the number of different positions in the two strings. To illustrate the difference between these two distances, let us consider the example of Fig. 1.

Left drawing gives a representation of the traditional assignment strategy in which colors are assigned to vertices. In Fig. 1(a), and according to the definition of the Hamming distance, $D_H(c_1, c_2) = 10$. So, c_1 is quite far from c_2. Right drawing shows a representation of the partition approach for the same configurations. In this case, the labeling of the colors is indifferent. In Fig. 1(b), moving the vertex 7 from the class V_2 to V_3 in c_1 leads to c_2 (up to a permutation of the colors). Then, $D_M(c_1, c_2) = 1$.



(a) Assignment approach (b) Partition approach

Fig. 1. Comparing Hamming and move distances

Note that the diversity is a measure over all the configurations in $RefSet$. So, it is updated each time a new configuration is added to $RefSet$.

3.4 Subset Generation Method

The smallest subsets (denoted by *2-subsets*) consist of all the couples of configurations in $RefSet$. Intermediate subsets (*a-subsets*, $a \in \{3, 4\}$) are built by augmenting each *(a-1)-subset* with the best evaluated configuration not included in the subset. Finally, *c-subsets* ($c \in [5..|RefSet|]$) contain the best c elements. In the scatter search template, each subset is generated only once, while, in the context of genetic algorithms, combinations based on the same subset are allowed. See [10], from which this description is extracted, for motivations about this method.

3.5 Configuration Combination Method

Combination may be viewed as a generalization upon multiple parents of classical crossovers which usually operate with only two parents. We used a generalization of the powerful *greedy partition crossover* (GPX), proposed in [8] within an evolutionary algorithm. GPX has been especially developed for the graph coloring problem with results reaching, and sometimes improving, those of the best known algorithms for the GCP.

Given a subset p generated by the subset generation method, the generalized combination operator builds the k color classes of the offspring one by one. First, choose arbitrarily (and temporarily remove) a configuration $c \in p$. Temporarily remove from c a minimal set A_c of conflicting vertices such as c becomes a partial proper k-coloring. Next, fill in a free color class $V_i (i \in [1..k])$ of the offspring with all conflict-free vertices of the color class $V_*^c \in c$ having maximum cardinality (break ties randomly) and remove these vertices from all configurations in p. Then, make A_c available for c and repeat these steps until the k color classes of the offspring contain at least one vertex. Finally, to complete the new configuration if necessary, assign to each free vertex in the offspring a color such that it minimizes the conflicts over the graph.

The chosen configuration c is temporarily removed from p to balance the origins of the color classes of the new configuration; all removed configurations become enable when p is empty. Fig. 2 summarizes the combination mechanism. See also [8] for an illustration of GPX on two parents.

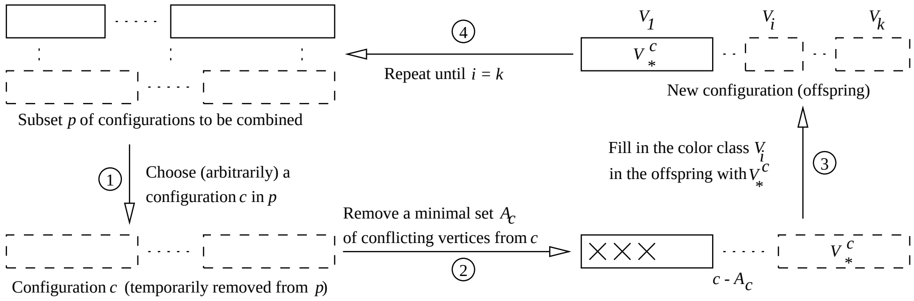


Fig. 2. Generalization of GPX

We used randomization to give each configuration the same chance to be selected when building the color class V_i of the new configuration. Note that, if we choose configurations following the order of two different permutations π_1 and π_2 issued from the same subset p, this may lead to two different offsprings. Another possible way of choosing configurations could be to consider their costs.

4 Preliminary Results

We give in this section some preliminary results obtained with our scatter search algorithm (called SSGC hereafter). Results of SSGC are reported for some of the DIMACS benchmark graphs [19][6] together with those of the best-known algorithms and those of a *generic tabu search* algorithm:

- *Morgenstern* [26], two local search algorithms based on particular neighborhoods (denoted by 3a and 3b in Table 1), and a distributed population-based algorithm (3c);
- *Funabiki and Higashino* [7], a descent algorithm with various heuristics mixed with a greedy construction stage and the search for a maximum clique (denoted by MIPS_CLR);
- *Dorne and Hao* [3], a generic tabu search (GTS for short) which solves successive k-colorings, k decreasing as long as a proper k-coloring is found.

Columns 5-7 in Table 1 summarize the best-known results given in the above papers for some DIMACS graphs: the smallest coloring ever obtained, the method which produced such a coloring, and the best computing time required. Columns 1-4 recall the characteristics of each graph: its name, number of vertices and edges, and its chromatic number (or its best known lower bound when unknown).

Results of GTS are reported in Table 2 (columns 3-5), including mean number of moves. The last four columns give the results of our scatter search algorithm[7]. Time entries and number of moves and combinations are averaged over five to ten runs. A maximum of 100000 moves were allowed for the improvement method. We only make, for each execution, one iteration of scatter search, i.e., main loop 1 in Algorithm 1 was performed on a single reference set. The α and g parameters (Sect. 3.2) used in computing the dynamic tabu tenure l (1) were empirically determined, respectively 2 and 10 at most. The size of the starting population was no more than 20 configurations and the b and d parameters (Sect. 2) were both empirically fixed to 10 at most. For the flat300_20_0 and dsjc125.5 graphs, the improvement method was allowed to perform 1% of random walks, i.e.: moving a random vertex into a random color class. Information about computing time (in seconds) in Table 1 and 2 is only for indicative purpose because results were obtained on different machines. "–" marks signal that no result is available.

[6] Available via FTP from: dimacs.rutgers.edu/pub/challenge/graph/benchmarks/.

[7] SSGC is implemented in C (CC compiler with -O5 flag) and runs on a Sun Ultra 1 (256 RAM, 143 MHz).

Table 1. Best known results for some instances of the 2nd DIMACS challenge

Graph name	$\|V\|$	$\|E\|$	χ	k	Method	Time (sec.)
school1	385	19095	14	14	MIPS_CLR	< 1
school1_nsh	352	14612	14	14	MIPS_CLR	< 1
dsjr500.1c	500	121275	84	85	3c	1240
r125.5	125	3838	36	36	3c	< 1
r250.1c	250	30227	64	64	3c	60
r250.5	250	14849	65	65	3c	181
r1000.1c	1000	485090	≥ 90	98	3c	1240
r1000.5	1000	238267	≥ 234	237	MIPS_CLR	3266
le450_15a	450	8168	15	15	3b	< 60
le450_15b	450	8169	15	15	3b	< 60
le450_15c	450	16680	15	15	3b	126
le450_15d	450	16750	15	15	3b	80
flat300_20_0	300	21375	20	20	3a	1
dsjc125.5	125	3891	≥ 10	17	3c	14

Table 2. Results of GTS and SSGC

Graph name	χ	GTS			SSGC			
		k	Moves	Time (sec.)	k	Moves	Combinations	Time (sec.)
school1	14	–	–	–	14	14	4	25
school1_nsh	14	–	–	–	14	18	9	31
dsjr500.1c	84	85	21000	70	85	103349	2	97
r125.5	36	36	147000	65	36	52	43	9
r250.1c	64	64	462	1	64	15497	146	349
r250.5	65	66	7800	6	65	180617	3604	3864
r1000.1c	≥ 90	98	1623000	4500	98	50395691	484	–
r1000.5	≥ 234	242	6027000	18758	240	20800000	189	–
le450_15a	15	15	103000	25	15	1064916	69	216
le450_15b	15	15	33000	8	15	677910	43	128
le450_15c	15	16	–	–	15	2135839	68	1091
le450_15d	15	16	–	–	15	8507576	280	3980
flat300_20_0	20	20	33000	17	20	732618	130	1849
dsjc125.5	≥ 10	17	348000	136	17	6805080	1352	805

From Table 1 and 2 we can make several remarks. First, SSGC manages to reach the results of the best-known algorithms (column k), except on the r1000.5 graph for which a 237-coloring has been published recently. The sophisticated algorithm used to reach this coloring includes, among other components, the search for a maximum clique. Nevertheless, SSGC obtains here a better coloring than GTS. Note that the previous best result for this graph was reported in [26] with 241 colors. Our scatter search approach also improves in quality on the results obtained with tabu search (GTS) on four graphs. This means that tabu search, the *improvement method* we used, surely benefits from the other general components of scatter search (Sect. 2).

Nevertheless, SSGC seems to be quite slow (regarding the number of moves) to reach the same results as those of GTS. This can partially be explained by the time spent in building the initial reference set, which include the improvement of **all** the configurations in the population. Furthermore, the generation method used in SSGC (random independent sets, see Sect. 3.1) differs from the one performed by GTS which is based on DSATUR [1]. It would certainly be interesting to follow the approach used in GTS to build the initial reference set. Another reason to the slowness of SSGC may be allotted with the diversity update step which occurs each time a new configuration c_{new} is inserted in $RefSet$. One possible way to speed up this step could be to regulate it according to a minimal diversity D_0 (fixed parameter or dynamically computed). In other words, c_{new} will be added to $DivSet$ only if its minimal diversity is greater than D_0.

Table 3. Results of SSGC using a *descent* improvement method

Graph name	χ	k	Moves	Combinations	Time (sec.)
school1	14	14	900	1	6
school1_nsh	14	14	408	2	7
r125.5	36	36	7	66	7
r250.1c	64	64	83600	37499	–
r250.5	65	65	25	580	452

We also experimented scatter search with a less elaborated improvement method, the simplest *descent* heuristic, which makes best 1-moves until achieving no improvement. This approach was only able to solve a few "easy" instances (see Table 3) implying that the global efficiency of scatter search is greatly dependent on the particular efficiency of its improvement algorithm.

5 Conclusions and Perspectives

We have developed a first adaptation of the scatter search method to the graph coloring problem. The preliminary results we obtained on some well-studied DIMACS graphs using this technique show that scatter search may be effective for some of them since SSGC reached most of the results of the best-known approaches. SSGC obtains also better results than those of a generic tabu search on some graphs.

At the same time, we observed that scatter search is quite slow due, essentially, to the numerous components it uses. Efforts must then be made to develop effective procedures to make the scatter search quicker. Another difficulty is to identify meaningful rules when choices are needed.

We are now working on replacing some random choices made in SSGC by deterministic rules. A first modification have been carried out in the diversification generation method (Sect. 3.1) by selecting vertices to be included in the same independent set according to Brélaz heuristics [1]. Vertices are inserted in color classes in decreasing order of their saturation degrees. Ties are broken using the connection degree and, only if needed, randomness. Few results are available since tests are currently running. However, the modified procedure was able to solve the school1, school1_nsh, and r125.5 graphs even while the improvement method was left out. In other words, solutions were found only by combining sets of configurations. This suggests that the generalization of the effective *greedy partition crossover* may be a suitable combination operator.

This research has highlighted the need for further investigation to make scatter search more effective for graph coloring. By using the *generic tabu search* in the initialization step for building the reference set, one may speed up the overall procedure. Another possible way could be to use maximal independent sets in place of selecting vertices at random to identify independent sets. Indeed, this later way shares the same objective as the *greedy partition crossover*. Lessons provided in [22] could also be useful for further attempts. Finally, some choice rules proposed in [11] may be used as fast heuristics either to generate the initial population or to color free vertices after each combination.

References

1. Brélaz., D.: New methods to color the vertices of a graph. *Commun. ACM* **22(4)** (1979) 251–256
2. Culberson, J.: Bibliography on graph coloring. Available on the wold wide web at `http://liinwww.ira.uka.de/bibliography/Theory/graph.coloring.html` (2000)
3. Dorne, R., Hao, J.K.: Tabu search for graph coloring, T-colorings and set T-colorings. In Voss, S., Martello, S., Osman, I.H., Roucairol, C. (editors) *Meta-heuristics: advances and trends in local search paradigms for optimization* Kluwer Academic Publishers (1998) 77–92
4. Dorne, R., Hao, J.K.: A new genetic local search algorithm for graph coloring. *Lecture Notes in Computer Science* **1498** Springer-Verlag (1998) 745–754
5. Dubois, N., de Werra, D.: EPCOT: an efficient procedure for coloring optimally with tabu search. *Computers Math. Appl.* **25(10/11)** (1993) 35–45
6. Ferland, J.A., Fleurent, C.: Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In [19] (1996) 619–652
7. Funabiki, N., Higashino, T.: A minimal-state processing search algorithm for graph colorings problems. *IEICE Trans. Fundamentals* **E83–A(7)** (2000) 1420–1430

8. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. *J. Combin. Optim.* **3(4)** (1999) 379–397
9. Gamst, A.: Some lower bounds for a class of frequency assignment problems. *IEEE Trans. Veh. Tech.* **35(1)** (1986) 8–14
10. Glover, F.: A template for scatter search and path relinking. In Hao, J.K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (editors) *Artificial evolution*. Lecture Notes in Computer Science **1363** Springer-Verlag (1998) 13–54
11. Glover, F.: Tutorial on surrogate constraint approaches for optimization in graphs. Tech. Report (February 2001)
12. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997)
13. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. *Control and Cybernetics* **39(3)** (2000) 653–684
14. Glover, F., Laguna, M., Martí, R.: Scatter search. In Ghosh, A., Tsutsui, S. (editors) *Theory and applications of evolutionary computation: recent trends* Springer-Verlag (to appear)
15. Halldórsson, M.M.: A still better performance guarantee for approximate graph coloring. *Inform. Process. Lett.* **45** (1993) 19–23
16. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. *Computing* **39** (1987) 345–351
17. Jagota, A.: An adaptive, multiple restarts neural network algorithm for graph coloring. *European J. Oper. Res.* **93** (1996) 257–270
18. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation. II. Graph coloring and number partitioning. *Oper. Res.* **39(3)** (1991) 378–406
19. Johnson, D.S., Trick., M.A. (editors): Cliques, coloring, and satisfiability: 2nd DIMACS implementation challenge, 1993. *DIMACS Series in Discr. Math. and Theoretical Comput. Sci.* **26** American Math. Soc. (1996)
20. Karp, R.M.: Reducibility among combinatorial problems. In Miller, R.E., Thatcher, J.W. (editors) *Complexity of computer computations* Plenum Press, New York (1972) 85–103
21. Laguna, M.: Scatter search. In Pardalos, P.M., Resende, M.G.C. (editors) *Handbook of applied optimization* Oxford Academic Press (to appear)
22. Laguna, M., Armento, V.A.: Lessons from applying and experimenting with scatter search. Tech. Report (March 2001)
23. Laguna, M., Martí, R.: Experimental testing of advanced scatter search designs for global optimization of multimodal functions. Tech. Report (August 2000)
24. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. *J. Res. Nat. Bur. Stand.* **84** (1979) 489–506
25. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. *Proc. 25th Annual ACM Symp. Theory of Comput.* (1993) 286–293
26. Morgenstern, C.A.: Distributed coloration neighborhood search. In [19] (1996) 335–357
27. Rego, C.: Integrating advanced principles of tabu search for the vehicle routing problem. Working paper, Faculty of Sciences, University of Lisbon (1999)