3 TABU SEARCH FOR GRAPH COLORING, T-COLORINGS AND SET T-COLORINGS

Raphaël Dorne and Jin-Kao Hao

LGI2P, EMA-EERIE
Parc Scientifique Georges Besse, F-30000 Nîmes, France.
dorne@eerie.fr, hao@eerie.fr

Abstract: In this paper, a generic tabu search is presented for three coloring problems: graph coloring, T-colorings and set T-colorings. This algorithm integrates important features such as greedy initialization, solution re-generation, dynamic tabu tenure, incremental evaluation of solutions and constraint handling techniques. Empirical comparisons show that this algorithm approaches the best coloring algorithms and outperforms some hybrid algorithms on a wide range of benchmarks. Experiments on large random instances of T-colorings and set T-colorings show encouraging results.

3.1 INTRODUCTION

The graph coloring problem is one of the most studied *NP-hard* problems and can be defined informally as follows. Given an undirected graph, one wishes to color the nodes of the graph with a minimal number of colors in such a way that two colors assigned to two adjacent nodes must be different, i.e., have a minimal distance greater than zero. Graph coloring has many practical applications such as timetabling and resource assignment. Given the NP-completeness of the coloring problem, it is natural to design heuristic methods. Indeed many heuristic methods have been developed: constructive methods in the 1960s and 1970s [1, 23], local search meta-heuristics in the 1980s and 1990s [15, 3, 18, 10] and genetic or population-based local search methods in the 1990s [9, 5, 24, 6]. Moreover, there are a large number of well-known benchmarks for evaluating and comparing different algorithms.

Several extensions of this classical graph coloring problem exist allowing more applications to be embraced. T-colorings and set T-colorings are two important extensions allowing one to model the frequency assignment problem. In T-colorings, the forbidden separation distance for colors assigned to adjacent nodes is no longer limited to the singleton $\{0\}$, but may be any set of positive integers. In set T-colorings, a node may receive several colors verifying some forbidden separation distances. There is extensive literature on the application side, i.e., the frequency assignment problem, including studies on specific heuristic algorithms [13, 12, 22], meta-heuristic algorithms [20, 8, 2, 21, 14] and lower bounds [11, 16]. On the contrary, studies on heuristic methods for the general problems of T-colorings and set T-colorings are much limited. Costa experimented with some known methods such as *Dsatur*, tabu search, and simulated annealing for T-colorings [4]. Jiang studied various methods including greedy, dynamic ordering, and tabu search as well as some combinations of these methods for set T-colorings [17]. Until now, there are very few well-established benchmarks available for evaluating algorithms for T-colorings and set T-colorings.

In this paper, we present a generic tabu search algorithm for these three coloring problems. To evaluate its performance, we use well-known benchmarks for graph coloring and introduce a set of random instances for T-colorings and set T-colorings. Moreover, we make these instances (and our instance generator) available to other researchers in the hope that these instances may serve as benchmarks for further studies.

The paper is organized as follows. Section 3.2 defines the three families of coloring problems. Section 3.3 introduces our random instance generator for the T-colorings and set T-colorings problems. Section 3.4 presents our generic tabu search algorithm for these coloring problems. Section 3.5 shows experimental results on a wide range of random instances. Section 3.6 gives some conclusions.

3.2 COLORING PROBLEMS

3.2.1 Graph coloring

Given an undirected graph $G{=}(V,E)$ with $V{=}\{v_1,...,v_N\}$ the set of nodes and $E{=}\{e_{ij}|\ \exists$ an edge between v_i and $v_j\}$ the set of edges. The graph coloring problem is determine a partition of V with a *minimum* number of color classes $C_1, C_2, ..., C_k$ such that for each edge $e_{ij} \in E$, v_i and v_j are not in the same color class [25]. Given $c(v_i)$ which is the color (a positive integer) assigned to the node v_i, a proper coloring must meet the following *color constraint*:

$$\forall e_{ij} \in E, \quad |c(v_i) - c(v_j)| \neq 0 \tag{3.1}$$

The *chromatic number* $\chi(G)$ corresponds to the smallest value of k such that G is *k-colorable.*

3.2.2 T-colorings

Given an undirected graph G=(V, E) as above, a collection of sets T=$\{T_{ij} \in I\!N|$ for each $e_{ij} \in E\}$ is now defined to determine for each edge e_{ij} the color separations which are not allowed between the nodes v_i and v_j. Each T_{ij} is a set of unsigned integers such as $\{0, 2, 4, 7\}$, and the *color constraint* to be met is:

$$\forall e_{ij} \in E, \quad |c(v_i) - c(v_j)| \notin T_{ij} \tag{3.2}$$

The separation of colors assigned to two adjacent nodes v_i and v_j must be different from those of T_{ij}. A T-coloring of a graph is a partition of V in different color classes $C_1, C_2, ..., C_k$ such that the property (3.2) is verified for each edge of G. The *chromatic number* $\chi_T(G)$ corresponds to the minimum number of different color values used to color G. The *span* of a T-coloring is the difference between the smallest and the highest color values needed to obtain the T-coloring of G. The *T-colorings problem* is determine the *minimum span* $sp_T(G)$ for all the possible colorings of G [13].

If each $T_{ij} \in T$ is a set of consecutive integers of the form T_{ij} = {0,1,2,..., t_{ij} -1}, the *restricted T-colorings problem* can be defined where constraint (3.2) becomes:

$$\forall e_{ij} \in E, \quad |c(v_i) - c(v_j)| \geq t_{ij} \tag{3.3}$$

It is easy to see that the graph coloring problem is a special case of the T-colorings problem where all $T_{ij} = \{0\}$.

$sp(G)$=$\chi(G) - 1$.

3.2.3 Set T-colorings

Given an undirected graph G=(V, E) and a collection of sets T as above, a set of *demand* values D=$\{d_1, ..., d_N|\ d_i \in I\!N\}$ is now added for each node v_i corresponding to the number of colors required by the node that is:

$$\forall v_i \in V \quad (d_i = l \implies c(v_i) = \{c_{i,1}, ..., c_{i,l}\}) \tag{3.4}$$

And for each set of colors assigned to the same node, a set of forbidden separations called *co-node constraints*, is also defined:

$$\forall c_{i,m}, c_{i,n} \in c(v_i), m \neq n \quad |c_{i,m} - c_{i,n}| \notin T_{ii} \tag{3.5}$$

Finally, the color constraint between two adjacent nodes can be stated as:

$$\forall e_{ij} \in E, \forall c_{i,m} \in c(v_i), \forall c_{j,n} \in c(v_j) \quad |c_{i,m} - c_{j,n}| \notin T_{ij} \tag{3.6}$$

The *problem of set T-colorings* of G consists of finding a coloring such that the properties (3.4), (3.5), (3.6) are satisfied for the graph. The *chromatic number* $\chi_T^D(G)$ corresponds to the minimum number of different color values used to color G. The *span* is the difference between the smallest and the highest color values needed to obtain such a coloring of G. The *problem of set T-colorings* is the optimization problem of finding the *minimum span* $sp_T^D(G)$ of the graph G [17]. As stated above, if each T_{ij} is a set of consecutive integers, the *restricted set T-colorings* problem can be defined by replacing constraints (3.5) and (3.6) with:

$$\forall c_{i,m}, c_{i,n} \in c(v_i), m \neq n \quad |c_{i,m} - c_{i,n}| \geq t_{ii} \tag{3.7}$$

$$\forall e_{ij} \in E, \forall c_{i,m} \in c(v_i), \forall c_{j,n} \in c(v_j) \quad |c_{i,m} - c_{j,n}| \geq t_{ij} \tag{3.8}$$

It is easy to see that the T-colorings problem is a special case of the set T-colorings problem where each node v_i has the same demand d_i=1. Note that a graph of set T-colorings G=(V, E) can be transformed into a graph of T-colorings G'=(V', E') by creating a node for each demand ($|V'|$=$\sum_{v_i \in V} d_i$). A co-node constraint becomes a set of edges forming a clique of d_i nodes with a separation equal to T_{ii} for each edge. And for each color separation e_{ij}, each new node of v_i is connected to each new node v_j with a color value separation equal to T_{ij} (see Fig. 3.1).

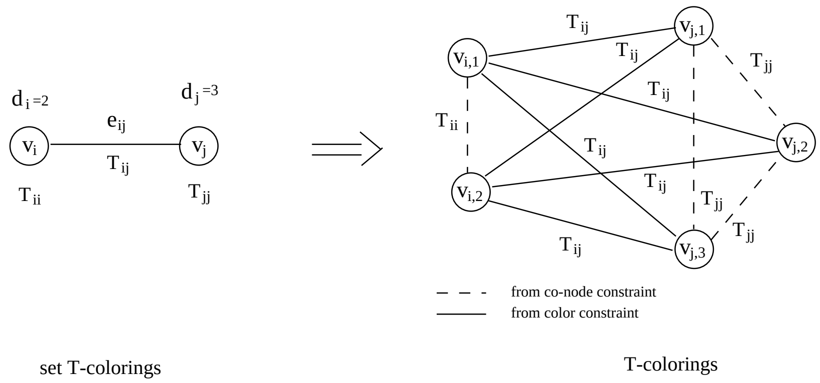

Figure 3.1 Transformation of a graph of set T-colorings into a graph of T-colorings

3.3 BENCHMARKS

3.3.1 Graph coloring

For graph coloring, there exists a large number of well-known benchmarks [15, 18][1]. Instances used in this study will be introduced in Section 3.5 when experimental results are presented.

[1]Most of these benchmarks can be downloaded via ftp from: dimacs.rutgers.edu/pub/challenge/graph/benchmarks/

3.3.2 T-colorings and Set T-colorings

Definitions

For these two classes of problems, no benchmark is available. Therefore, we developed a random instance generator for *restricted T-colorings* and *restricted set T-colorings*, following the same principles defined for generating random graphs for graph coloring [18].

For restricted T-colorings, each instance is defined by three parameters: N, the number of nodes; $d \in [0, 1]$, the edge density; and Sep, the largest color separation. To generate such an instance, we first build a graph of N nodes with $d(N(N\text{-}1))/2$ edges uniformally distributed on these nodes. Then, a uniform random value from $[1, Sep]$ is assigned to each edge.

For *restricted set T-colorings*, two more parameters are necessary: D, the maximum number of colors required by a node and $CoSep$, the maximum separation required between two colors of the same node. As above, to generate a restricted set T-colorings instance, we make for each node two uniform random choices from $[1, D]$ and $[1, CoSep]$.

Note that from an instance for restricted set T-colorings, we can get an instance for restricted T-colorings (with an identical edge topology) by setting for each node v_i its separation d_i to 1. In the same way, we can obtain a random *graph coloring* instance with an identical edge topology by setting for each edge e_{ij} the separation T_{ij} to $\{0\}$.

Probabilistic estimations for the minimum span

Unlike graph coloring, there is no estimation available for the minimum span for T-colorings. There are some theoretical lower and upper bounds, but these bounds are of little use in our case because they are often far from the real minimum span. Establishing a good estimation for T-colorings is very helpful for us, but seems to be difficult for the general T-colorings. We thus limit our attention to restricted T-colorings with a color separation uniformly chosen in $[1..Sep]$. So we can define the following estimation:

$$\widetilde{sp}_T(G_{n,p}) = (\widetilde{\chi}(G_{n,p}) - 1) * Sep_{ave} \tag{3.9}$$

where Sep_{ave} is the average of color separations over the graph. This estimation considers that from an edge topology which allows a k-coloring, if each edge has a color separation equal to Sep_{ave} on average, then the minimum span of T-colorings is equal to the difference between the highest color value $1 + (k - 1) * Sep_{ave}$ and the lowest color value, i.e., 1. For the instances used in this study, Sep_{ave} is equal to 3 because the color separation is uniformly chosen between 1 and 5 (see next subsection).

As for T-colorings, we try to establish an estimation for the minimum span for random set T-colorings graphs. In order to do this, we need to transform the graph of set T-colorings into a graph of T-colorings. As explained in Section 3.3, a co-node constraint becomes a clique of d_i nodes with a separation equal to t_{ii} on each edge. And for each color separation e_{ij}, each new node of v_i is connected to each new node of v_j with a color value separation equal to t_{ij}. Since the color separations for co-nodes and for adjacent nodes are very close in our case, the influence of the demand of each node, thus the estimation, depends mainly on the edge density. The lower the edge density is, the higher the influence of the demand is on the *minimum span* $sp_T^D(G_{n,p})$ of G.

Given D_{ave} the average demand per node, and Sep_{ave} the average color separation over the graph, we can define two estimations: one for low edge density (3.10) and one for high edge density (3.11).

$$\widetilde{sp}_T^D(G_{n,p}) = ((\widetilde{\chi}(G_{n,p}) - 1) * Sep_{ave}) * D_{ave} \tag{3.10}$$

The first estimation considers that the graph is divided into D_{ave} similar and strongly connected subgraphs.

$$\widetilde{sp}_T^D(G_{n,p}) = (\widetilde{\chi}(G_{D_{ave}*n,p}) - 1) * Sep_{ave} \tag{3.11}$$

In the second case, because we have a high edge density, only the number of nodes in the transformed graph of T-colorings is modified and the edge density before and after the transformation remains nearly the same.

For the instances used in this study, the demand for each node and the co-node separation are on average equal to 3 (demands and co-node separations are uniformly chosen between 1 and 5) (see next subsection).

Random instances

For the purpose of this study, we generate 15 random instances of restricted set T-colorings with the following possible values: $N \in \{30, 100, 300, 500, 1000\}$, $d \in \{0.1, 0.5, 0.9\}$, *Sep*=*D*=*coSep*=5. Each instance is defined by a name of the form $N.D_{sum}.d.STcol$, where $STcol$ means set T-colorings and D_{sum} is the sum of all the demands ($D_{sum} = \sum_{i \in [1,N]} d_i$).

From these instances, we built 15 random instances of T-colorings denoted by $N.d.Tcol$ (resp. graph coloring instances denoted by $N.d.col$) by assigning $\forall i \in [1, N], d_i = 1$ (resp. by assigning $\forall i \in [1, N], d_i = 1$ and $\forall T_{ik} \in T, T_{ik} = \{0\}$ for coloring). So we have the same graph topology (edge structure) of different densities for the three families of problems. These 45 instances[2] enable us to study the differences among them, in terms of separation distance, demands and edges density.

3.4 GENERIC TABU SEARCH FOR COLORING PROBLEMS

Given the fact that graph coloring and T-colorings are special cases of set T-colorings, the following presentation is oriented to set T-colorings. It should be clear that some components of the algorithm are not necessary for graph

[2]These instances and the generator are available from the authors via email and via internet at http://www.eerie.fr/∼dorne

coloring or T-colorings.

Configuration and search space

Given a graph involving N nodes, with d_i (i $\in$ {1..N}) demand per node and NC available colors numbered from 1 to NC, a configuration $s = < c_{1,1}...c_{1,d_1}...c_{i,1}...c_{i,d_i}...c_{N,1}, c_{N,d_N} >$ is a complete coloring satisfying the following condition:

$$\forall c_{i,m}, c_{i,n} \in c(v_i), m \neq n \quad |c_{i,m} - c_{i,n}| \notin T_{ii}$$

The search space S is thus composed of all possible configurations meeting the *co-node constraints*. Integrating co-node constraints into configurations is an important factor in improving the search efficiency for the set T-colorings problem. This point is developed in [7] for the frequency assignment problem.

Cost function

For each configuration $s \in S$, f(s) is simply the total number of unsatisfied color constraints.

$$f(s) = \sum_{e_{ij} \in E} \sum_{\substack{c_{i,m} \in c(v_i) \\ c_{j,n} \in c(v_j)}} q(c_{i,m}, c_{j,n})$$

$$\text{where } q(c_{i,m}, c_{j,n}) = \begin{cases} 1 & \text{if } |c_{i,m} - c_{j,n}| \in T_{ij} \\ 0 & \text{otherwise} \end{cases}$$

Neighborhood and candidate list

Given $s \in S$, let $s(i,m)$ be equal to the value of the m^{th} color of the node v_i in s. Then the neighborhood N is defined as follows. *s'* $\in$ *N(s)* if and only if the following condition is verified:

$$\exists \,!\; \mathrm{i} \in \{1..\mathrm{N}\},\ \exists \,!\; \mathrm{m} \in \{1..d_i\} \text{ such that } s(i,m) \neq s'(i,m)$$

A neighbor of s can thus be obtained by changing the value of a color of a node in s and in such a way that the new value always satisfies the co-node constraint. A move is thus characterized by a triplet $< i, m, v >$, i, m and v being a node, a demand of the node, and a color value, respectively. Note that the number of neighbors of a configuration may be very high for large graphs.

In this work, the following strategy is used to define the candidate list V^* = {*s'* $\in$ *N(s)* such that *s'* and s are different only at the color of a *conflicting* demand of a node in s}. A demand of a node is said to be conflicting if its color value violates some color constraints. Let CD be the set of conflicting demands of the nodes in s, then $|V^*|$=$|CD| * (NC - 1)$. Clearly, $|V^*|$ varies during the search according to the number of conflicting demands. This candidate list strategy reduces the number of neighbors to be considered at each iteration;

more importantly, it helps the search to concentrate on influential moves.

Incremental evaluation and neighborhood examination

To evaluate the configurations, we use an approach inspired by a technique proposed in [10]. The main idea consists of maintaining incrementally δ, the *move value* or *cost variation* for each possible move from the current configuration s, where W = $\sum_{i=1}^{N} d_i$ in a $NC * W$ matrix. Each time a move is carried out, the elements affected by the move are updated accordingly. Initializing δ takes time $O(NC * W^2)$. The matrix can be updated in time $O(NC * W)$ in the worst case. Now a best neighbor in V^* can be evaluated in time $O(|V^*|)$. Thus each iteration takes time $O(NC * W + |V^*|)$.

Tabu list and tabu tenure

When the color of a demand m of a node v_i in a configuration s is changed to a new value, the triplet $< v_i, m, old\_value >$ is classified tabu for l (tabu tenure) iterations. That is, the old value will not be allowed to be re-assigned to v_i during this period. The tabu tenure l is dynamically adjusted by a function defined over the number of conflicting demands of the nodes. More precisely, l = $\alpha * |CD| + random(g)$ where the function $random(g)$ returns an integer value uniformly chosen in $[1..g]$. The values of α and g are empirically determined and l is limited by $|V^*|$. Since CD varies during the search, so does l for any fixed α.

Aspiration criteria

A very simple aspiration criterion is used to override the tabu restriction. The tabu status of a move will be cancelled if the move leads to a better configuration than the best configuration $s*$ encountered so far.

Generic Tabu Search (GTS)

Our Generic Tabu Search algorithm is composed of three parts: greedy construction of initial coloring, configuration re-generation and searching for proper coloring (see Fig. 3.2).

- Initial configuration: GTS uses a *Dsatur-based* greedy algorithm [1] to generate initial configurations. This greedy step is fast and provides, with the tabu algorithm, a far better initial configuration than a random approach.

- Configuration re-generation: the re-generation aims at producing quickly a k-1 coloring with a minimum of color conflicts. It proceeds as follows: given a coloring with k colors, the nodes colored with the k^{th} color are given a new color from $[1..k-1]$ in such a way that the new color minimizes the color conflicts over the graph (break ties randomly).

Input: G, a graph
Output: NC, the minimum number of color used (minimum span for T-colorings and set T-colorings is equal to NC-1)

%**Variables**
% f,f^*: objective function and its best value encountered so far
% s,s^*: current configuration and the best configuration encountered so far
% V^*, l, MAX: candidate list, the size of tabu list and the limit of the iterations
% NC: the minimum number of color used
% colors(s): returns the highest color value

```
begin
    M = 0 /* to initialize tabu matrix */
    NB = 0 /* to initialize iteration counter */
    s = generate() /* initial configuration generated with a greedy algorithm */
    NC = colors(s)-1
    s = re-generate(s,NC) /* re-generation from s with NC colors */
    while (NB < MAX) do
        s* = s
        f* = f(s*)
        while ( f* > 0 and NB < MAX) do
            if ( f(best_neighbor(s)) < f* ) then
                s = best_neighbor(s) /* Aspiration criterion, s(i,m) = new_v */
            else
                s = best_non_tabu_neighbor(s) /* s(i,m) = new_v */
            M[i,m,old_v] = NB + l; /* < v_i, m, old_v > becomes tabu */
            l = α * |CD| + random(g);
            if (f(s) < f*) then
                s* = s
                f* = f(s*)
            NB = NB + 1
        if (f* = 0) then
            NC = NC - 1
            s = re-generate(s,NC)
            NB = 0
    return NC+1
end
```

Figure 3.2 Algorithm: Generic Tabu Search (GTS)

- Searching for proper coloring: beginning from such an improper coloring, the tabu algorithm tries to reduce the number of color conflicts to zero. If this happens, the algorithm finds a proper coloring and proceeds to re-generate a new improper coloring with one less color.

It stops when an optimal known coloring is obtained or when MAX iterations have been carried out without finding a conflict-free (proper) coloring for the current number of colors k. The lowest number of colors used to find a conflict-free coloring is returned.

3.5 EXPERIMENTAL RESULTS

3.5.1 Settings for experiments

In this section, we report experimental results on various instances for the three coloring problems. All the tests have been performed on an Ultra Sparc station with a 143 MHz processor and 128 MB of memory. The GTS algorithm was implemented in C++ and compiled by CC compiler with option -O5. The computing time reported corresponds to the average running time of the entire algorithm including the above three steps.

For graph coloring, GTS runs with a dynamic tabu tenure determined by $2 * |CN| + random(10)$ for all the instances except for the three flat1000... graphs where the chosen value is $4 * |CN| + random(10)$ (CN corresponds to the set of conflicting nodes). For T-colorings and set T-colorings, tabu tenure is set to $l{=}4 * |CD| + random(10)$ for all instances (CD is the set of conflicting demands). The maximum number of iterations for an attempt of finding a proper coloring is fixed at 10,000,000 (for some large or hard graph coloring instances this value is increased to 20,000,000).

3.5.2 Graph coloring

For graph coloring, we used benchmarks from 2nd Dimacs Challenge, Hertz and De Werra [15], and Johnson et al. [18]. Results are compared with the best ones published in the literature:

1. *Fleurent and Ferland*, a tabu algorithm [10] (denoted by 1a in the tables) and a genetic tabu algorithm (denoted by 1b) [9]. These algorithms use an efficient pre-processing technique of [15], which reduces the initial graphs by removing a large number of independent sets. Coloring algorithms are then used to color the residual graphs. This technique is applied to graphs larger than 300 nodes and systematically used by many existing coloring algorithms.

2. *Costa*, an evolutionary hybrid algorithm *EDM* (denoted by 2) with the above pre-processing technique [5].

3. *Morgenstern*, distributed local search algorithms (denoted by 3) based on a particular neighborhood and initialized by a parallelized version of *Johnson et al's XRLF* algorithm [24].

4. *Johnson et al.*, the Successive Augmentation Method *XRLF* (4a) and a set of methods based on simulated annealing: *Penalty Function Annealing* (4b), *Kempe Chain Annealing* (4c), and *Fixed-K Annealing* (4d) [18].

problems	Best Known			Generic Tabu Search			
	k	Method	Time(sec.)	$runs(fail.)$	k	Iterations	Time(sec.)
R125.1.col	5	3,1a	1	10(0)	5	1	1
R125.5.col	35	1a	1,380	10(0)	36	147,000	65
R125.1c.col	46	3,1a	1	10(0)	46	1	1
R250.1.col	8	3,1a	1	10(0)	8	1	1
R250.5.col*	65	3	181	5(2)	66	7,800	6
R250.1c.col	64	3,1a	60	5(1)	64	462	1
R1000.1.col	20	3	18	3(0)	20	1	1
R1000.5.col*	241	3	2,078	3(0)	242	6,027,000	18,758
R1000.1c.col*	98	3	1,240	3(0)	98	1,623,000	4,500
flat300_20_0.col	20	3	1	5(0)	20	33,000	17
flat300_26_0.col	26	3	22	5(0)	26	1,723,000	850
flat300_28_0.col*	31	3	4,214	2(0)	31	17,000,000	9,200
flat1000_50_0.col*	50	3	1	10(0)	50	1,664,000	3,020
flat1000_60_0.col*	60	3	1	5(0)	60	5,548,000	10,579
flat1000_76_0.col*	84	1b	14,520	10(5)	89	5,410,000	8,015

Table 3.1 2nd Dimacs Challenge instances

Tables 3.1-3.3 give comparative results for Dimacs, Hertz and De Werra, and Johnson et al. graphs. Table 3.1 shows our results on Dimacs instances with the best-known results given in the above papers. The best known results are summarized in columns 2-4: the smallest number of colors ever obtained, methods which produced such a coloring and the best computing time required. For example, the third line indicates that two methods find a coloring with 46 colors for the problem R125.1c.col, and the best method requires 1 second. Note that information about computing time is only for indicative purpose because these methods have been run on different machines. The last four columns report results obtained by our Tabu algorithm. We give the number of total runs with the number of failures (unsuccessful runs) in parentheses (5^{th} column), the smallest number of colors obtained (6^{th} column), the number of iterations and computing time averaged over successful runs (7^{th} and 8^{th} columns).

From Table 3.1, we see that the results of GTS are very competitive on these instances. Indeed, except for four instances, GTS manages to produce the best-known result. This is remarkable if we compare these results with those of Fleurent and Ferland (1a): GTS gives better colorings on a wide range of instances (all these instances are marked with a star "*") and in particular we find the optimal configuration for the problems flat1000_50_0.col and flat1000_60_0.col while the method 1a produces a solution with 90 colors.

Table 3.2 shows a comparison on Hertz and De Werra instances between GTS and the methods 1a, 1b and 2. These instances belong to four classes of 100, 300, 500 and 1000 nodes, respectively composed of 20 (g1-g20), 10 (gg1-gg10), 5 (ggg1-ggg5[3]), and 2 instances (gggg1-gggg2[4]). k^* corresponds to

[3]ggg5 is identical to DSJC500.5.col
[4]gggg2 is identical to DSJC1000.5.col

problems	Best Known				Generic Tabu Search				
	k^*	k_{moy}	Method	Time	$runs$ $(fail.)$	k^*	Iterations	Time	k_{moy}
g1-20	15	14.95	1b	9.5 sec.	10(0)	15	8,000	2,5 sec	14.95
gg1-10	34	33.3	2	11,000 sec.	5(0)	33	1,302,000	635.9 sec	32.90
ggg1-5	49	49	1b,3	0 sec.	3(0)	50	4,037,000	3,500 sec	49.75
gggg1-2	84	84	1b	41 hours	2(0)	90	12,355,000	19,514 sec	90.0
residual	23	23	1b	5 hours	5(0)	23	834,000	382	23.0 sec

Table 3.2 Hertz and De Werra instances

the smallest number of colors used for all the instances of one class and k_{moy} is the average. On small instances (gg1-gg10), GTS gives better results and brings down the value of k^* to 33. For large instances, we get worse results. However, remember that these methods (1a, 1b and 2) use the "independent sets removing" pre-processing, this makes it impossible to compare these results directly. To get a fair comparison, we run GTS on the residual graph of gggg2 (or DSJC1000.5.col). This residual graph is obtained after having removed 61 independent sets and is given in [9]. GTS finds a 23 coloring leading to an 84 coloring to the initial graph. GTS took 382 seconds and 834,000 iterations to reach a 23-coloring, while 1a needs more than 19 hours and about 50,000,000 iterations, and 1b more than five hours. Note also that Costa's algorithm obtains only 85-colorings.

problems	Best Known			Generic Tabu Search			
	k	Method	Time	$runs(fail.)$	k	Iterations	Time(sec.)
DSJC125.1.col	5	4a	1	10(0)	5	5,000	<1
DSJC125.5.col	17	3	14	10(0)	17	348,000	136
DSJC125.9.col	44	4d	1,080	10(0)	44	9,000	5
DSJC250.1.col	8	4d	9,360	5(0)	8	168,000	32
DSJC250.5.col*	28	3	591	5(0)	28	3,604,000	1,716
DSJC250.9.col	72	4c	72,000	5(0)	72	720,000	591
DSJC500.1.col	12	3	5,452	3(0)	13	16,000	5
DSJC500.5.col*	48	3	49,000	3(0)	50	3,078,000	2,327
DSJC500.9.col	126	3	158,400	3(0)	127	4,211,000	6,150
DSJC1000.1.col	21	3	210	3(0)	21	290,000	154
DSJC1000.5.col*	84	3,1b	118,000	3(1)	90	11,211,000	16,799
DSJC1000.9.col	226	3	65,500	2(0)	226	13,283,000	39,554

Table 3.3 Johnson et al. instances

Table 3.3 shows the results on random instances from Johnson et al. [18] with different edge densities of 10, 50, and 90 %. GTS remains competitive on instances for graphs of low or high edge density. On large instances with 50 % of edges, our method has some difficulties and certainly needs an independent set pre-processing to improve its results.

3.5.3 T-colorings and set T-colorings

Before giving our results on the 15 random T-colorings and set T-colorings instances (Section 3.3), Table 3.4 shows the results on the instances considered as graph coloring. The chromatic number $\widetilde{\chi}$ is estimated with the Johri and

Matula probabilistic method [19]. Note that the results obtained by GTS are similar to those of the Table 3.3.

problems	$\tilde{\chi}$	Generic Tabu Search $runs(fail.)$	k_{best}	k_{ave}	Iterations	Time(sec.)
30.1.col	4	3(0)	3	3.0	2	<1
30.5.col	8	3(0)	7	7.0	14	<1
30.9.col	15	3(0)	15	15.0	10	<1
100.1.col	6	3(0)	5	5.0	81	<1
100.5.col	16	3(1)	14	14.33	2,514,000	1,875
100.9.col	36	3(0)	37	37.0	29,000	32
300.1.col	11	3(0)	9	9.0	15,597	6
300.5.col	35	3(2)	32	32.67	4,746,000	2,262
300.9.col	84	3(0)	84	84.0	790,000	376
500.1.col	14	3(0)	13	13.0	12,000	9
500.5.col	50	3(2)	49	49.67	10,951,000	8,279
500.9.col	124	2(1)	127	127.5	2,908,000	4,247
1000.1.col	22	1(0)	21	21.0	2,256,000	1,198
1000.5.col	85	1(0)	91	91.0	4,013,000	6,013
1000.9.col	222	1(0)	226	226.0	16,484,000	49,085

Table 3.4 Results for graph coloring

To evaluate the GTS algorithm on the T-colorings instances, the estimation of the minimum span, defined in Section 3.3.2, allows us to have a rough idea about the performance of the GTS algorithm. In order to further evaluate the performance of GTS, we adapted *Dsatur* algorithm for T-colorings and set T-colorings during the constructive process, when a color c is assigned to a node v_i the values included in $[c-(t_{ij}-1), c+(t_{ij}-1)]$ are now forbidden for any adjacent node v_j. This is the only difference from *Dsatur* for coloring. Table 3.5 gives comparative results on the 15 T-colorings instances.

From Table 3.5, several remarks may be made. First, GTS gives far better results than *Dsatur*. Indeed, GTS requires much fewer colors (up to -231 colors) for these graphs. Second, the estimation given in the equation (3.9) seems reasonably good. Finally, the computing time necessary to get good T-colorings is high for large and dense graphs.

Table 3.6 gives comparative results for the 15 set T-colorings instances. From the data, we may make similar remarks as for T-coloring. In particular, we see that GTS outperforms *Dsatur* on all the instances, especially on large

		Dsatur		Generic Tabu Search				
problems	$\tilde{sp}_T$	sp_{best}	sp_{ave}	$runs(fail.)$	sp_{best}	sp_{ave}	Iterations	Time(sec.)
30.1.Tcol	9	9	9.0	3(0)	9	9.0	1	<1
30.5.Tcol	21	23	25.2	3(0)	17	17.0	55,000	21
30.9.Tcol	42	40	44.9	3(0)	31	31.0	293,000	114
100.1.Tcol	15	23	23.0	3(0)	14	14.0	154,000	82
100.5.Tcol	45	63	66.1	3(0)	43	43.67	936,000	365
100.9.Tcol	105	124	125.0	3(1)	82	82.33	4,327,000	2,317
300.1.Tcol	30	40	43.8	3(0)	29	29.0	4,720,000	899
300.5.Tcol	102	159	161.3	3(2)	110	111.0	11,967,000	6,285
300.9.Tcol	249	304	318.9	3(2)	216	217.0	32,734,000	26,869
500.1.Tcol	39	61	62.5	3(0)	43	43.0	2,363,000	711
500.5.Tcol	147	240	247.5	3(0)	175	175.67	6,900,000	5,216
500.9.Tcol	369	483	493.1	2(0)	351	351.0	14,523,000	20,811
1000.1.Tcol	63	104	105.1	1(0)	77	77.0	12,537,000	6,657
1000.5.Tcol	252	436	441.4	1(0)	328	328.0	31,554,000	47,281
1000.9.Tcol	663	896	904.2	1(0)	665	665.0	18,949,000	56,426

Table 3.5 Comparative results for T-colorings

problems	$\tilde{sp}_T^D$		Dsatur		Generic Tabu Search				
	low	high	sp_{best}	sp_{ave}	$runs$ $(fail.)$	sp_{best}	sp_{ave}	Iterations	Time (sec.)
30.86.1	27	15	28	33.6	2(0)	26	26.0	193,000	101
30.95.5	63	42	74	76.1	2(1)	61	61.50	2,468,000	2,721
30.90.9	126	99	145	153.7	3(1)	102	102.33	2,004,000	2,050
100.275.1	45	30	62	62.0	2(1)	45	45.50	1,563,000	1,294
100.304.5	135	102	170	176.2	2(1)	114	114.50	7,332,000	17,974
100.299.9	315	249	320	340.3	3(1)	224	224.33	2,690,000	6,020
300.937.1	90	57	112	114.6	2(1)	80	80.50	11,526,000	20,890
300.905.5	306	234	431	443.9	2(1)	267	268.50	17,169,000	129,291
300.940.9	747	609	902	921.2	3(2)	572	573.33	19,350,000	96,800
500.1507.1	117	84	157	166.0	2(1)	112	113.0	19,300,000	20,308
500.1484.5	441	351	677	677.0	1(0)	402	402.0	14,750,000	133,673
500.1536.9	1,107	915	1,409	1,432.3	1(0)	931	931.0	3,593,000	42,938
1000.3049.1	189	138	281	285.7	2(1)	178	178.50	27,879,861	126,854
1000.3024.5	756	615	1,220	1,226.3	1(0)	846	846.0	19,179,000	382,898
1000.2975.9	1,989	1,665	2,522	2,541.0	1(0)	1,723	1,723.0	9,000,000	670,000

Table 3.6 Comparative results for set T-colorings. All problem instances are of type STcol.

instances with a difference of about 500 colors. The computing times to color these instances are high because the search space is huge (up to about 3,000 variables and 4,000,000 constraints). Indeed, each node requires several colors and the number of colors needed for the graph is very high (up to about 2,000).

Finally, we mention that in addition to these tests, a variant of GTS was also applied to solve frequency assignment instances in mobile-radio networks coming from French National Center for Telecommunications [14]. GTS was compared with methods based on constraint programming, simulated annealing, Gamst's constructive method and evolutionary algorithms. Experimental results showed that GTS finds the best results for all the tested instances.

3.6 CONCLUSIONS

In this paper, a robust and effective tabu search algorithm has been presented for three coloring problems: graph coloring, T-colorings and set T-colorings. The algorithm integrates some important features such as greedy construction of initial configurations, re-generating configurations, dynamic tabu tenure, and co-node constraint handling. Compared with many best known algorithms, this algorithm remains simpler and easier to tune.

The performance of the algorithm was evaluated on a wide range of random graphs. For graph coloring, the algorithm produces highly competitive results compared with some well-known and more complicated algorithms. The "independent sets" pre-processing technique should improve the performance of the algorithm. For T-colorings and set T-colorings, experimental results show that GTS outperforms largely a *Dsatur* algorithm adapted to these problems for all the instances tested. Given that there are few benchmarks for T-colorings and set T-colorings, the random generator and instances used in this study may help to improve this situation.

A last remark to conclude the paper: the independent sets extracting is a widely-used technique for the graph coloring problem and has proven to be important for all well-known algorithms to color large graphs. Unfortunately, no equivalent technique is available for T-colorings or set T-colorings. It will certainly be interesting and important to develop such techniques in the future.

Acknowledgements

We would like to thank the referees for their useful comments which helped to improve the paper.

References

[1] D. Brélaz. New methods to color vertices of a graph. *Communications of ACM*, 22:251–256, 1979.

[2] D.J. Castelino, S. Hurley, and N.M. Stephens. A tabu search algorithm for frequency assignment. *Annals of Operations Research*, 63:301–320, 1996.

[3] M. Chams, A. Hertz, and D. De Werra. Some experiments with simulated annealing for coloring graphs. *European Journal of Operational Research*, 32:260–266, 1987.

[4] D. Costa. On the use of some known methods for T-colorings of graphs. *Annals of Operations Research*, 41:343–358, 1993.

[5] D. Costa, A. Hertz, and O. Dubuis. Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1):105–128, 1995.

[6] R. Dorne and J.K. Hao. *A new genetic local search algorithm for graph coloring.* submitted to PPSN'98, Amsterdam, Sept. 1998.

[7] R. Dorne and J.K. Hao. Constraint handling in evolutionary search: A case study of the frequency assignment. In *Intl. Conf. on Parallel Problem Solving From Nature*, volume 1141 of *Lectures Notes in Computer Science*, pp 801–810, Berlin, Germany, 1996.

[8] M. Duque-Anton, D. Kunz, and B. Rüber. Channel assignment for cellular radio using simulated annealing. *IEEE Transactions on Vehicular Technology*, 42:14–21, 1993.

[9] C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1995.

[10] C. Fleurent and J.A. Ferland. *Object-Oriented Implementation of Heuristic Search Methods for Graph Coloring, Maximum Clique, and Satisfiability*, volume 26 of *Discrete Mathematics and Theoretical Computer Science*, pp 619–652. American Mathematical Society, 1996.

[11] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, 35:8–14, 1986.

[12] A. Gamst. A ressource allocation technique for FDMA systems. *Alta Frequenza*, 57(2):89–96, 1988.

[13] W.K. Hale. Frequency assignment: Theory and applications. *IEEE Transactions on Vehicular Technology*, 68(12):1497–1514, 1980.

[14] J.K. Hao, R. Dorne, and P. Galinier. Tabu search for frequency assignment in mobile radio networks. *Journal of Heuristics*, 4(1):47–62, 1998.

[15] A. Hertz and D. De Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[16] S. Hurley and D.H. Smith. Bounds for the frequency assignment problem. *Discrete Mathematics*, 167-168:571–582, 1997.

[17] M. Jiang. Méthodes heuristiques pour le problème du T-coloriage avec intervalles. *to appear in RAIRO Operational Research*.

[18] 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. *Operations Research*, 39(3):378–406, 1991.

[19] A. Johri and D.W. Matula. Probabilistic bounds and heuristic algorithms for coloring large random graphs. Technical Report 82-CSE-06, Southern Methodist University, Department of Computing Science, Dallas, 1982.

[20] D. Kunz. Channel assignment for cellular radio using neural networks. *IEEE Transactions on Vehicular Technology*, 40:188–193, 1991.

[21] W.K. Lai and G.G. Coghill. Channel assignment through evolutionary optimization. *IEEE Transactions on Vehicular Technology*, 45(1):91–95, 1996.

[22] R.A. Leese. Tiling methods for channel assignment in radio communication networks. *Zeitschrift fur Angewandte Mathematik und Mechanik*, 76:303–306, 1996.

[23] F.T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau Standard*, 84:79–100, 1979.

[24] C. Morgenstern. *Distributed Coloration Neighborhood Search*, volume 26 of *Discrete Mathematics and Theoretical Computer Science*, pp 335–358. American Mathematical Society, 1996.

[25] C.H. Papadimitriou and K. Steiglitz. Combinatorial optimization - algorithms and complexity. *Prentice Hall*, 1982.