A tabu search heuristic for the Equitable Coloring Problem

I. Méndez Díaz[3], G. Nasini[1,2] and D. Severín[1,2]

[1] Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario, Argentina
{nasini,daniel}@fceia.unr.edu.ar
[2] CONICET, Argentina
[3] Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires, Argentina
imendez@dc.uba.ar

Abstract. The *Equitable Coloring Problem* is a variant of the Graph Coloring Problem where the sizes of two arbitrary color classes differ in at most one unit. This additional condition, called equity constraints, arises naturally in several applications. Due to the hardness of the problem, current exact algorithms can not solve large-sized instances. Such instances must be addressed only via heuristic methods.
In this paper we present a tabu search heuristic for the Equitable Coloring Problem. This algorithm is an adaptation of the dynamic TabuCol version of Galinier and Hao. In order to satisfy equity constraints, new local search criteria are given.
Computational experiments are carried out in order to find the best combination of parameters involved in the dynamic tenure of the heuristic.
Finally, we show the good performance of our heuristic over known benchmark instances.

Keywords: equitable coloring · tabu search · combinatorial optimization

1 Introduction

The *Graph Coloring Problem* (GCP) is a very well-studied $\mathcal{NP}$-Hard problem since it models many applications such as scheduling, timetabling, electronic bandwidth allocation and sequencing problems.

Given a simple graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a *k-coloring of G* is a partition of V into k sets $V_1, V_2, \dots, V_k$, called *color classes*, such that the endpoints of any edge lie in different color classes. The GCP consists of finding the minimum number k such that G admits a k-coloring, called the *chromatic number* of G and denoted by $\chi(G)$.

Some applications impose additional restrictions. For instance, in scheduling problems, it may be required to ensure the uniformity of the distribution of workload employees. Suppose that a set of tasks must be assigned to a set of workers so that pairs of tasks may conflict each other, meaning that they should not be assigned to the same worker. The problem is modeled by building a graph containing a vertex for every task and an edge for every conflicting pair of tasks. Workers are represented by colors. Then, in order for a coloring of this graph to represent a valid assignment of tasks to workers, the same number of tasks must be assigned to each worker. Since this is impossible when the number of tasks is not divisible by the number of workers, one can ask for the number of tasks assigned to two arbitrary workers can not differ by more than one. It is called *equity constraint* and the resulting problem is called *Equitable Coloring Problem* (ECP).

ECP was introduced in [1], motivated by an application concerning *garbage collection* [2]. Other applications of the ECP concern *load balancing problems* in multiprocessor machines [3] and results in *probability theory* [4]. An introduction to ECP and some basic results are provided in [5].

Formally, an *equitable k-coloring* (or just k-eqcol) of a graph G is a k-coloring satisfying the *equity constraint*, i.e. the size of two color classes can not differ by more than one unit. The *equitable chromatic number* of G, $\chi_{eq}(G)$, is the minimum k for which G admits a k-eqcol. The ECP consists of finding $\chi_{eq}(G)$ which is an $\mathcal{NP}$-Hard problem [5].

There exist some differences between GCP and ECP that make the latter harder to solve. It is known that the chromatic number of a graph is greater than or equal to the chromatic number of any of its induced subgraphs. Unfortunately, in the case of ECP, this property does not hold. For instance, if G is the graph shown in Figure 1, by deleting v_5 from G, $\chi_{eq}(G)$ increases from 2 to 3.

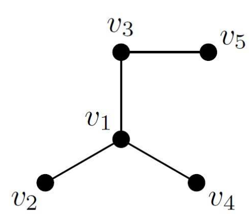

Fig. 1.

As far as we know, there are few approximate and exact algorithms available in the literature related to ECP.

It was proved that, for any graph G, $\Delta(G) + 1$ is an upper bound of $\chi_{eq}(G)$ [6], where $\Delta(G)$ is the maximum degree of vertices in G. Based on this fact, a polynomial time algorithm for obtaining a k-eqcol of a graph G with $k \geq \Delta(G)+1$ is described in [7].

Two constructive heuristics called Naive and SubGraph are given in [5] to generate greedily an equitable coloring of a graph. There also exist heuristic algorithms for constructing colorings that are "nearly" equitable [8, 9], making emphasis on achieving a small difference between the sizes of the biggest class and the smallest one, although the equity constraint still might be violated.

The authors of [10] propose a tabu search heuristic to initialize an exact algorithm that solves ECP via Integer Linear Programming (ILP) techniques. Other exact algorithms for solving ECP are given in [11] and [12]. The first one also uses IPL techinques and the second one is based on a DSATUR enumeration scheme.

In this work, we propose a new heuristic based on the dynamic TabuCol version of Galinier and Hao [13], one of the best tabu search algorithms for GCP [14]. Then, computational experiments are carried out in order to find the best combination of parameters involved in the dynamic tenure of our heuristic and to show the good performance of it over known benchmark instances.

The paper is organized as follows. In Section 2, we present TabuCol and the dynamic variant of Galinier and Hao. In Section 3, we give our variant for ECP which we call TabuEqCol. Finally, in Section 4 we report computational experiences and conclusions.

2 TabuCol and its variants

Tabu search is a metaheuristic method proposed by Glover [15] that guides a local search algorithm equipped with additional mechanisms that prevent from visiting a solution twice and getting stuck in a local optimum.

Let S be the solution space of the problem and $f : S \to \mathbb{R}$ be the objective function. The goal is to obtain a solution $s \in S$ such that $f(s)$ is minimum.

For each solution $s \in S$, consider a *neighborhood* $N(s) \subset S$ with two desirable (but not exclusionary) properties: 1) two solutions s and s' are neighbors when it is easy (from the computational point of view) to obtain s' from s, and to obtain $f(s')$ from $f(s)$ (for instance, in constant time), and 2) for any $s, s' \in S$, there exists a path $s = s_1, s_2, \ldots, s_m = s'$ such that $s_{i+1} \in N(s_i)$ for $i = 1, \ldots, m-1$.

In general, neighbor solutions are very similar in some sense, and the difference between them can be seen as *features* that both solutions do not share. Consider a set of *features* P and a set $R \subset S \times P$ such that $(s, p) \in R$ if solution s presents a feature p.

Starting from an initial solution $s_0 \in S$, tabu search consists of generating a sequence of solutions $s_1, s_2, \ldots$ such that $s_{i+1} = \arg\min_{s \in N'(s_i)} f(s)$, where $N'(s_i)$ is a subset of $N(s_i)$ described below. In each iteration of this algorithm, a *movement* from s_i to s_{i+1} is performed and some feature of s_i is stored in a *tabu list* $L \subset P$. This list indicates whether a movement is allowed or forbidden: a solution s can be reached in the future only if s does not present any feature from L (this rule avoids from visiting a solution previously visited), except when s is better than the best solution found so far. This exception is called *aspiration* and the aspiration criterion is usually to check if the objective value of s is less than the value of currently-known best solution. Now, the set of allowed movements from s_i, $N'(s_i)$, is defined as

$$N'(s) = \{s' \in N(s) : f(s') < f(s^*) \ \vee \ (s', p) \notin R \ \ \forall\, p \in L\},$$

where s^* is the best solution found so far.

However, after several iterations, old features are no longer needed and it is better to remove them from the tabu list. This mechanism is usually implemented by assigning a "time of live" to each feature of the tabu list. Consider $live : L \to \mathbb{Z}$ and let $live(p)$ be the number of remaining iterations that p belongs to L. When a new feature p is inserted into L, $live(p)$ is assigned a value referred to as *tabu tenure* t. Then, in each iteration, the value of $live(p)$ is decreased by one unit until it reachs zero and p is removed from L. Above, we sketch a generic tabu search algorithm.

```
Data: initial solution s_0
Result: best solution found s*
begin
    L ← ∅
    s, s* ← s_0
    while stopping criterion is not met do
        for p ∈ L do
            live(p) ← live(p) − 1
            if live(p) = 0 then L ← L\{p}
        end
        N'(s) ← {s' ∈ N(s) : f(s') < f(s*)  ∨  (s', p) ∉ R  ∀ p ∈ L}
        choose a feature p ∈ P such that (s, p) ∈ R
        L ← L ∪ {p}
        live(p) ← t
        s ← arg min_{s' ∈ N'(s)} f(s')
        if f(s) < f(s*) then s* ← s
    end
end
```

Algoritmo 1: TabuSearch

In order to implement a tabu search algorithm, some decisions must be taken: neighborhood of a solution, features of a solution, stopping criterion, how to choose the feature p to be stored in the tabu list and how to compute the tabu tenure t. In particular, the value of tabu tenure directly impacts *diversification* of the algorithm. A tabu search with low tenures behaves as a standard local search, where it frequently get trapped in local minima. On the other hand, a tabu search with high tenures tends to wander across solution space without converging towards the optimal solution.

TabuCol, the first tabu search algorithm designed for solving GCP, was proposed by Hertz and de Werra [16]. For a given graph $G = (V, E)$ and number $k \in \{1, \ldots, n\}$, where $n = |V|$, the goal of this algorithm is to find a k-coloring of G. In order to obtain a coloring that uses as few colors as possible, it is usual to embed TabuCol in a routine that, once a k-coloring is found, the algorithm can be restarted with $k \leftarrow k-1$ and so on, until some criterion is met. Details of TabuCol are given below:

- *Search space and objective function.* A solution s is a partition $(V_1, V_2, \ldots, V_k)$ of the set of vertices. Let $E(V_i)$ be the set of edges of G with both endpoints in V_i. The objective function is defined as

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

  Clearly, s is a k-coloring if and only if $f(s) = 0$.
- *Stopping criterion.* The algorithm stops when $f(s) = 0$ or when a maximum number of iterations is reached. Sometimes, a time limit is imposed.
- *Initial solution.* It is generated randomly. A suitable procedure given in [17] is the following. Start with empty sets $V_1, V_2, \ldots, V_k$ and, at each step, choose a non-considered vertex v randomly and put it into V_i with the smallest possible i such that $E(V_i)$ is not incremented. If it is not possible, choose a random number $j \in \{1, \ldots, k\}$ and put v into V_j.
- *Set of features.* It is $P = V \times \{1, \ldots, k\}$. A solution s presents a feature (v, i) if and only if $v \in V_i$, i.e. if v is assigned color i.
- *Neighborhood of a solution.* Let $C(s)$ be the *set of conflicting vertices* of a solution s, i.e.

$$C(s) = \{v \in V : v \text{ is incident in some edge of } E(V_1) \cup E(V_2) \cup \ldots \cup E(V_k)\}.$$

  From a solution $s = (V_1, V_2, \ldots, V_k)$, a neighbor $s' = (V'_1, V'_2, \ldots, V'_k)$ is generated as follows. Choose a conflicting vertex $v \in C(s)$. Let i be the color of v in s. Next, choose a color $j \in \{1, \ldots, k\} \backslash \{i\}$ and set

$$V'_j = V_j \cup \{v\}, \quad V'_i = V_i \backslash \{v\}, \quad V'_l = V_l \ \ \forall\, l \in \{1, \ldots, k\} \backslash \{i, j\}.$$

  In other words, s' is a copy of s except that v is moved from class color V_i to V_j. We denote such operation with $s' = s(i \xrightarrow{v} j)$. Note that objective value can be computed in linear time from $f(s)$:

$$f(s') = f(s) + |\{vw \in E : w \in V_j\}| - |\{vw \in E : w \in V_i\}|.$$

  Note also that searching all the neighbors of s requires exploring $(k-1)|C(s)|$ solutions. Original TabuCol only explores a random subset of $N(s)$ while newer versions explore $N(s)$ completely.
- *Selection of feature to add in the tabu list.* Once a movement from s to $s(i \xrightarrow{v} j)$ is performed, $p = (v, i)$ is stored on tabu list and $live(p)$ is set to a fixed tabu tenure $t = 7$.

Later, Galinier and Hao [13] improved TabuCol by using a dynamic tabu tenure that depends on the quality of the current solution, encouraging diversification of the search when solution is far from optimal. They proposed to assign a tenure of $t = \alpha|C(s)| + Random(\beta)$ where $Random(\beta)$ returns an integer randomly chosen from $\{0, \ldots, \beta - 1\}$ with uniform distribution. Based on experimentation, they suggest to use $\alpha = 0.6$ and $\beta = 10$. Other variants of TabuCol are discussed in [14, 17].

3 TabuEqCol: A tabu search for ECP

In this section, we present a new tabu search algorthm for ECP based on TabuCol with dynamic tabu tenure, which we call TabuEqCol.

Given a graph $G = (V, E)$ and a number $k \in \{1, \dots, n\}$, where $n = |V|$, the goal of TabuEqcol is to find a k-eqcol of G.

Solution space consists of partitions of V into k sets $V_1, V_2, \dots, V_k$ such that they satisfy the equity constraint, i.e. for any pair of classes V_i and V_j, $\big||V_i| - |V_j|\big| \leq 1$. Objective function f is the same as in TabuCol, so any solution s such that $f(s) = 0$ is indeed an equitable coloring. Also, set of features P is the same as in TabuCol.

Stopping criterion depends on the experiment carried out. Usually, a time limit is imposed.

Let $s \in S$. Denote $W^+(s) = \{i : |V_i| = \lfloor n/k \rfloor + 1\}$ and $W^-(s) = \{i : |V_i| = \lfloor n/k \rfloor\}$, where V_i are the color classes of s. Since s satisfies the equity constraint, we have that $W^+(s)$ and $W^-(s)$ determine a partition of $\{1, \dots, k\}$ and, in particular, $|W^+(s)| = r$ where $r = n - k\lfloor n/k \rfloor$. From now on, we just write W^+ and W^-. These sets will be useful in the development of the algorithm.

We propose two greedy procedures for generating initial solution s_0.

Procedure 1. Start with empty sets $V_1, V_2, \dots, V_k$ and an integer $\tilde{r} \leftarrow 0$ (this value will have the cardinal of W^+). At each step, define set $I = \{i : |V_i| \leq M - 1\}$, where M is the maximum allowable size of a class:

$$M = \begin{cases} \lfloor n/k \rfloor + 1, & \text{if } \tilde{r} < r \\ \lfloor n/k \rfloor, & \text{if } \tilde{r} = r \end{cases}$$

(once we already have r class of size $\lfloor n/k \rfloor + 1$, the size of the remaining classes must not exceed $\lfloor n/k \rfloor$). Then, choose a non-considered vertex v randomly and put it into a class V_i such that $i \in I$ is the smallest possible and $E(V_i)$ is not incremented. If it is not possible, i is chosen ramdonly from I. To keep $\tilde{r}$ up to date, each time a vertex is added to a set V_i such that $|V_i| = \lfloor n/k \rfloor$, $\tilde{r}$ is incremented by one unit.

The previous procedure works fine for generating initial solutions from scratch. However, at this point it is common to know a $(k+1)$-eqcol (i.e. in the cases where we previously ran tabu search with $k+1$ and reached an equitable coloring) and we can exploit this coloring in order to improve the quality of the initial solution as follows.

Procedure 2. Let $\mathfrak{p} : \{1, \dots, k+1\} \to \{1, \dots, k+1\}$ be a bijective function (i.e. a random permutation) and let $V_1^*, V_2^*, \dots, V_k^*, V_{k+1}^*$ be the color classes of the known $(k+1)$-eqcol. Set $V_i = V_{\mathfrak{p}(i)}^*$ for all $i \in \{1, \dots, k\}$, and $\tilde{r} = |W^+|$. Then, run Procedure 1 to assign a color to the remaining vertices which are those belonging to $V_{\mathfrak{p}(k+1)}^*$.

Regarding neighborhood of a solution $s \in S$ notice that, if n does not divide k, $W^+ \neq \varnothing$ and it is possible to move a vertex from a class of W^+ to W^-, keeping equity. That is, for all $v \in \cup_{i \in W^+} V_i$ and all $j \in W^-$, we have $s(i \xrightarrow{v} j) \in S$. However, the number of allowed movements is rather limited when r is very low (for instance, $r = 1$) or very high ($r = k - 1$), so we need to add supplementary movements. Swapping the colors of two vertices simultaneously seems to work fine and as well can be used when n divides k.

From a solution $s = (V_1, V_2, \ldots, V_k)$, a neighbor $s' = (V'_1, V'_2, \ldots, V'_k)$ is generated with two schemes:

- *1-move* (only applicable when n does not divide k). Choose a conflicting vertex $v \in C(s) \cap (\cup_{i \in W^+} V_i)$. Let i be the color of v in s. Next, choose a color $j \in W^-$. We have $s' = s(i \xrightarrow{v} j)$. Searching all the neighbors of s with this scheme requires exploring $(k - r)|C(s) \cap (\cup_{i \in W^+} V_i)|$ solutions.
- *2-exchange*. Choose a conflicting vertex $v \in C(s)$. Let i be the color of v in s. Next, choose another vertex u such that either $i < j$ or $u \notin C(s)$, where j is the color of u in s (the condition imposed to u prevents from evaluating 2-exchange on u and v twice). Then, set

  $$V'_j = (V_j \backslash \{u\}) \cup \{v\}, \;\; V'_i = (V_i \backslash \{v\}) \cup \{u\}, \;\; V'_l = V_l \;\; \forall\, l \in \{1, \ldots, k\} \backslash \{i, j\}.$$

  Note that objective value can be computed in linear time from $f(s)$:

  $$\begin{aligned} f(s') = f(s) + |\{uw \in E : w \in V_i \backslash \{v\}\}| - |\{uw \in E : w \in V_j\}| \\ + |\{vw \in E : w \in V_j \backslash \{u\}\}| - |\{vw \in E : w \in V_i\}|. \end{aligned}$$

  Searching all the neighbors of s with this scheme requires exploring a quadratic number of solutions.

Now, let s' be the next solution in the sucession; s' is obtained by applying either 1-move or 2-exchange to s, where vertex $v \in V_i$ in s and $v \notin V'_i$ in s'. In both schemes, $p = (v, i)$ is stored on tabu list and $live(p)$ is set to a dynamic tabu tenure $t = \alpha|C(s)| + Random(\beta)$ where α and β are parameters to be determined empirically. This is one of the purposes of the next section.

4 Computational experiments and conclusions

This section is devoted to perform and analyze computational experiments. They were carried out on an Intel i5 CPU 750@2.67Ghz with Ubuntu Linux O.S. and Intel C++ Compiler. We considered graphs from [18], which are benchmark instances difficult to color.

First, we test different combinations of values for parameters α and β from the dynamic tabu tenure in order to determine the combination that makes TabuEqCol perform better. Then, we report the behaviour of TabuEqCol over known instances by using the best combination previously found. We also compare its performance against tabu search algorithm given in [10].

Tuning parameters

We run TabuEqCol over 16 instances with a predetermined value of k and an initial solution s_0 generated with Procedure 1. The same initial solution is used in all executions of TabuEqCol for the same instance.

Results are reported in Table 1. First column is the name of the graph G. Second and third columns are the number of vertices and edges of G. Fourth and fifth columns are known lower and upper bound of $\chi_{eq}(G)$ (obtained by other means). The remaining columns are the time elapsed in seconds by the execution of TabuEqCol when a k-eqcol is found within the term of 1 hour, for each combination. In the case TabuEqCol is not able to find a k-eqcol, $f(s^*)$ is displayed between braces where s^* is the best solution found. Three last rows indicate the sum of objective function $f(s^*)$ over non-solved instances, percentage of instances TabuEqCol solved successfully and the average time elapsed for these instances to be solved.

For the sake of simplicity, we refer to each combination with a capital letter.

Note that combination D has the least average time, however it has solved less instances than other combinations and the sum of objective values is also worse. We discard A, B, C, D, E and H with this criterion. By comparing the three remaining combinations, we have that G is faster than the other two. Even if we restrict the comparison to those 11 instances the 3 combinations solve simultaneously, we have 807 seconds for F, 562 seconds for G and 730 seconds for I, so G is still better.

We consider combination G ($\alpha = 0.9$ and $\beta = 5$) for TabuEqCol.

Testing tabu search heuristic

For each instance, the following process is performed. First, execute Naive algorithm (described in [5]) in order to find an initial equitable coloring c of the current instance. Suppose that $k+1$ is the number of colors of c. Then, obtain an initial solution s_0 of k color classes generated from c with Procedure 2, and run TabuEqcol with parameters $\alpha = 0.9$ and $\beta = 5$. If a k-eqcol is found, start over the process with $k-1$ color classes by running Procedure 2 and TabuEqcol again. This process is repeated until 1 hour is elapsed or a $\underline{\chi_{eq}}$-eqcol is reached, and the best coloring found so far is returned.

In Table 2 we report results over 76 benchmark instances with at least 50 vertices (75 from [18] and one Kneser graph used in [10]). First 5 columns have the name of the graph G, number of vertices and edges, and best known lower and upper bound of $\chi_{eq}(G)$. Sixth column displays the number of colors of the initial equitable coloring c. Seventh and eighth columns display the value k of the best k-eqcol found after 30 seconds of execution of our algorithm and the time elapsed in seconds until such k-eqcol is reached. If the coloring is optimal, k is displayed in boldface. Next two columns show the same information after 1 hour of execution, but if the best coloring is found within the first 30 seconds, these columns are left empty.

Time spent by Naive is not considered in the computation. However, Naive rarely spent more than 1 sec. (and never more than 4 sec.).

Last two columns show the same information for the tabu search described in [10]. If such information is not available, these columns are left empty. We recall that the values provided in [10] were computed on a different platform (1.8 Ghz AMD-Athlon with Linux and GNU C++ compiler).

Note that our approach reachs optimality in 29 instances and a gap of one unit between $\underline{\chi_{eq}}$ and the best solution in 7 instances. In other words, it reachs a gap of at most one unit in roughly a half of the evaluated instances. Note also that TabuEqcol improves the initial solution given by Naive in most cases (precisely, 63 instances).

On those instances the value of the best solution given by tabu search of [10] is known, our algorithm gives the same value or a better one. Despite the difference between platforms, it seems that our approach also runs faster.

An interesting fact is that each execution of TabuEqCol needs no more than 500000 iterations to reach the best value since the largest number of iterations performed was 493204 and took place when TabuEqcol found a 18-eqcol of `DSJC125.5`.

In the same sense, TabuEqCol needs no more than 30000 iterations in each execution and the overall process needs no more than 30 seconds to reach the best value on 56 instances; justly those ones such that columns 9 and 10 are empty. On these instances, the largest number of iterations performed was 28791 and took place when TabuEqcol found a 10-eqcol of `queen9_9`.

Conclusion

The Equitable Coloring Problem is a variation of the Graph Coloring Problem that naturally arises from several applications where the cardinalities of color classes must be balanced. Just like Graph Coloring, the need to solve applications associated to this new NP-Hard problem justifies the development of exact and approximate algorithms. On large instances, known exact algorithms are unable to address them and heuristics such as Naive delivers poor solutions. Our tabu search heuristic based on TabuCol has shown to improve these solutions and presented a fairly good performance, even if a limit of 30 seconds is imposed. In addition, an iteration limit of 30000 (for a time limit of 30 seconds) and 500000 (for a time limit of 1 hour) can be imposed in order to save time.

Acknowledgements. This work is partially supported by grants UBACYT 20020100100666, PICT 2010-304, PICT 2011-817, PID-UNR ING416 and PIP-CONICET 241.

References

1. Meyer W.: Equitable Coloring. Amer. Math. Monthly 80, 920-922 (1973)
2. Tucker A.: Perfect graphs and an application to optimizing municipal services. SIAM Review 15, 585-590 (1973)
3. Das S.K., Finocchi I., Petreschi R.: Conflict-free star-access in parallel memory systems. J. Parallel Distrib. Comput. 66, 1431-1441 (2006)
4. Pemmaraju S.V.: Equitable colorings extend Chernoff-Hoeffding bounds. Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, LNCS vol. 2129, pp. 285-296 (2001)
5. Furmanczyk H., Kubale M.: Equitable coloring of graphs. Graph Colorings, Providence, Rhode Island: American Mathematical Society, 35-53 (2004)
6. Hajnal A., Szemerédi E.: Proof of a conjecture of P. Erdös. Combin. theory and its app. 2, 601-623 (1970)
7. Kierstead H., Kostochka A., Mydlarz M., Szemerédi E.: A fast algorithm for equitable coloring. Combinatorica 30, 217-224 (2010)
8. Brélaz D., Nicolier Y., de Werra D.: Compactness and balancing in scheduling. Math. Methods Oper. Res. 21, 65-73 (1977)
9. Sulong G.B.: Some balanced colouring algorithms for examination timetabling. Jurnal Teknologi 19, 57-63 (1992)
10. Bahiense L., Frota Y., Noronha T.F., Ribeiro C.: A branch-and-cut algorithm for the equitable coloring problem using a formulation by representatives. Discrete Appl. Math. 164(1), 34-46 (2014)
11. Méndez Díaz I., Nasini G., Severín D.: A polyhedral approach for the Equitable Coloring Problem. Discrete Appl. Math. 164(2), 413-426 (2014)
12. Méndez Díaz I., Nasini G., Severín D.: An exact DSatur-based algorithm for the Equitable Coloring Problem. Electron. Notes Discrete Math 44, 281-286 (2013)
13. Galinier P., Hao J-K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. 3(4), 379-397(1999)
14. Galinier P., Hertz A.: A survey of local search methods for graph coloring. Comput. Oper. Res. 33(9), 2547-2562 (2006).
15. Glover F., McMillan C., Novick B.: Interactive decision software and computer graphics for architectural and space planning. Ann. Oper. Res. 5(3), 557-573 (1985)
16. Hertz A., de Werra D.: Using tabu search techniques for graph coloring. Computing 39(4), 345-351 (1987)
17. Blöchliger I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput. Oper. Res. 35(3), 960-975 (2008)
18. Graph Coloring Benchmark Instances. `http://www.cs.hbg.psu.edu/txn131/graphcoloring.html`

Instance	$\vert V\vert$	$\vert E\vert$	$\underline{\chi_{eq}}$	k	$\alpha = 0.3$			$\alpha = 0.6$			$\alpha = 0.9$		
					$\beta = 5$	$\beta = 10$	$\beta = 15$	$\beta = 5$	$\beta = 10$	$\beta = 15$	$\beta = 5$	$\beta = 10$	$\beta = 15$
					A	B	C	D	E	F	G	H	I
DSJR500.1	500	3555	12	12	{3}	1	1	1	1	1	1	1	1
DSJR500.5	500	58862	120	131	{14}	{3}	{1}	{8}	{3}	3242	{5}	{3}	{1}
DSJR500.1c	500	121275	126	195	{4}	{1}	427	{3}	78	747	66	8	11
DSJC500.1	500	12458	5	13	{2}	55	41	38	63	47	39	83	57
DSJC500.5	500	62624	13	62	61	530	{1}	{1}	{1}	{2}	{1}	{2}	{1}
DSJC500.9	500	112437	101	148	{1}	106	104	94	91	80	100	90	121
DSJC1000.1	1000	49629	5	22	767	411	509	551	423	858	710	691	1059
DSJC1000.5	1000	249826	15	112	543	968	623	518	999	{2}	1853	{2}	{1}
DSJC1000.9	1000	449449	126	268	1850	1751	1822	1926	1725	1250	1808	1723	983
inithx.i.1	864	18707	54	54	{8}	{8}	{8}	{8}	{7}	{7}	{8}	{8}	{7}
latin_square_10	900	307350	90	131	1182	1080	1013	796	782	946	895	1298	778
flat300_28_0	300	21695	11	37	238	{1}	{1}	143	{1}	{1}	{1}	{2}	{2}
flat1000_76_0	1000	246708	14	112	228	548	1255	154	600	1681	245	780	3298
abb313GPIA	1557	53356	8	9	{27}	{44}	{15}	{2}	{10}	2801	1796	{1}	1304
qg.order40	1600	62400	40	40	26	31	17	25	26	20	24	25	26
wap01a	2368	110871	41	47	{21}	477	501	{6}	451	446	499	744	397
Sum of objective values					80	57	26	28	22	12	15	18	12
Success					50%	69%	69%	63%	69%	75%	75%	63%	69%
Average Time					612	542	574	425	476	1010	670	544	730

Table 1. Execution of TabuEqCol with different combination of values

Instance	$\|V\|$	$\|E\|$	$\underline{\chi_{eq}}$	$\overline{\chi_{eq}}$	Naive	≤ 30 sec. k	Time	≤ 1 hour k	Time	[10] k	Time
miles750	128	2113	31	31	33	**31**	0.0			35	13
miles1000	128	3216	42	42	47	43	0.1			49	13
miles1500	128	5198	73	73	74	**73**	0.0			77	13
zeroin.i.1	211	4100	49	49	51	51	0.0			74	22
zeroin.i.2	211	3541	36	36	51	51	0.0			95	22
zeroin.i.3	206	3540	36	36	49	49	0.0			97	21
queen8_8	64	728	9	9	18	**9**	1.2			10	7
jean	80	254	10	10	10	**10**	0.0			**10**	3
anna	138	493	11	11	11	**11**	0.0			13	14
david	87	406	30	30	40	**30**	0.0			**30**	9
games120	120	638	9	9	9	**9**	0.0			11	6
kneser9_4	126	315	3	3	4	**3**	0.0			6	2
2-FullIns_3	52	201	5	5	9	**5**	0.0			8	1
3-FullIns_3	80	346	6	6	7	**6**	0.0			9	2
4-FullIns_3	114	541	7	7	12	**7**	0.1			11	5
5-FullIns_3	154	792	8	8	9	**8**	0.0			13	8
2-FullIns_5	852	12201	4	7	15	7	2.5				
3-FullIns_5	2030	33751	5	8	13	8	25				
4-FullIns_4	690	6650	6	8	14	8	0.4				
4-FullIns_5	4146	77305	6	9	21	14	20	9	254		
1-Insertions_6	607	6337	3	7	14	7	0.2				
2-Insertions_5	597	3936	3	6	6	6	0.0				
3-Insertions_5	1406	9695	3	6	8	6	1.2				
homer	561	1628	13	13	13	**13**	0.0				
huck	74	301	11	11	11	**11**	0.0				
latin_square_10	900	307350	90	130	460	169	30	130	1301		
DSJC125.1	125	736	5	5	8	**5**	0.8				
DSJC125.5	125	3891	9	18	27	19	0.1	18	788		
DSJC125.9	125	6961	42	45	66	45	0.4				
DSJC250.1	250	3218	4	8	13	9	0.1	8	32		
DSJC250.5	250	15668	11	32	65	33	7.2	32	69		
DSJC250.9	250	27897	63	83	136	83	1.2				
DSJR500.1	500	3555	12	12	12	**12**	0.0				
DSJR500.5	500	58862	120	131	135	133	0.1				
DSJR500.1c	500	121275	126	195	349	257	0.3				
DSJC500.1	500	12458	5	13	23	14	3.5	13	33		
DSJC500.5	500	62624	13	62	128	63	11				
DSJC500.9	500	112437	101	148	284	182	0.7				
DSJC1000.1	1000	49629	5	22	38	26	26	22	500		
DSJC1000.5	1000	249826	15	112	265	128	27	112	2261		
DSJC1000.9	1000	449449	126	268	575	329	20				
flat300_20_0	300	21375	11	34	81	38	9.2	34	463		
flat300_28_0	300	21695	11	36	65	39	3.3	36	3222		
flat1000_76_0	1000	246708	14	112	223	127	24	112	1572		
fpsol2.i.1	496	11654	65	65	85	78	0.1				
fpsol2.i.2	451	8691	47	47	62	60	0.0				
fpsol2.i.3	425	8688	55	55	80	79	0.0				
inithx.i.1	864	18707	54	54	70	66	0.1				
inithx.i.2	645	13979	30	93	158	93	7.2				
le450_15b	450	8169	15	15	17	16	0.3	**15**	107		
le450_15d	450	16750	15	16	30	22	9.6	16	599		
le450_25b	450	8263	25	25	25	**25**	0.0				
le450_25d	450	17425	25	27	31	27	29				
le450_5b	450	5734	5	5	12	7	7.2				
le450_5d	450	9757	5	8	18	8	15				
mug100_25	100	166	4	4	4	**4**	0.0				
mug88_25	88	146	4	4	4	**4**	0.0				
mulsol.i.1	197	3925	49	49	63	50	0.0				
mulsol.i.2	188	3885	31	48	58	48	0.1				
myciel6	95	755	7	7	11	**7**	0.0				
myciel7	191	2360	8	8	12	**8**	0.1				
qg.order40	1600	62400	40	40	64	42	22	**40**	47		
qg.order60	3600	212400	60	60	64	64	0.0	**60**	267		
queen8_12	96	1368	12	12	20	**12**	0.1				
queen9_9	81	1056	10	10	15	**10**	9.2				
queen10_10	100	1470	10	11	18	12	0.1	11	143		
school1	385	19095	15	15	49	**15**	12				
school1_nsh	352	14612	14	14	40	**14**	14				
wap01a	2368	110871	41	46	48	46	15				
wap02a	2464	111742	40	44	49	47	18	44	83		
wap03a	4730	286722	40	50	58	57	18	50	464		
abb313GPIA	1557	53356	8	9	17	13	28	10	283		
ash331GPIA	662	4181	3	4	8	4	2				
ash608GPIA	1216	7844	3	4	10	4	12				
ash958GPIA	1916	12506	3	4	10	5	11	4	41		
will199GPIA	701	6772	7	7	9	**7**	2.2				

Table 2. Execution of TabuEqCol over benchmark instances