Backtracking Based Iterated Tabu Search for Equitable Coloring

Xiangjing Lai [a], Jin-Kao Hao [a,b,*], Fred Glover [c]

[a] *LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers, Cedex 01, France*

[b] *Institut Universitaire de France, Paris, France*

[c] *OptTek Systems, Inc., 2241 17th Street Boulder, Colorado 80302, USA*

Abstract

An equitable k-coloring of an undirected graph $G = (V, E)$ is a partition of its vertices into k disjoint independent sets, such that the cardinalities of any two independent sets differ by at most one. As a variant of the graph coloring problem (GCP), the equitable coloring problem (ECP) concerns finding a minimum k for which an equitable k-coloring exists. In this work, we propose a backtracking based iterated tabu search (BITS) algorithm for solving the ECP approximately. BITS uses a backtracking scheme to define different k-ECP instances, an iterated tabu search approach to solve each particular k-ECP instance for a fixed k, and a binary search approach to find a suitable initial value of k. We assess the algorithm's performance on a set of commonly used benchmarks. Computational results show that BITS is very competitive in terms of solution quality and computing efficiency compared to the state-of-the-art algorithm in the literature. Specifically, BITS obtains new upper bounds for 21 benchmark instances, while matching the previous best upper bound for the remaining instances. Finally, to better understand the proposed algorithm, we study how its key ingredients impact its performance.

Keywords: Equitable coloring; tabu search; heuristics; graph coloring; combinatorial optimization.

1 Introduction

Given an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$ and edge set $E \subset V \times V$, a k-coloring of G is a partition of its vertices into

* Corresponding author.

Email addresses: laixiangjing@gmail.com (Xiangjing Lai), hao@info.univ-angers.fr (Jin-Kao Hao), glover@opttek.com (Fred Glover).

k disjoint independent sets $\{V_1, V_2, \ldots, V_k\}$ (also called color classes), i.e., any two vertices of any V_i $(i = 1, , \ldots, k)$ are not linked by an edge. For a given k, G is k-colorable if a k-coloring exists for G. The problem of finding a k-coloring for a given number of k colors is called the graph k-coloring problem. The classical graph coloring problem (GCP) is to determine the smallest k (the chromatic number of G) such that G is k-colorable.

An equitable k-coloring of G (or k-eqcol) is a k-coloring $\{V_1, V_2, \ldots, V_k\}$ such that the numbers of vertices in any two color classes differ by at most one (i.e., $||V_i| - |V_j|| \leq 1$, $\forall\ i \neq j$. This is called the equity constraint [25]). In other words, an equitable k-coloring is a conflict-free k-coloring satisfying the equity constraint. The equitable coloring problem (ECP) is to determine the smallest number k of colors such that an equitable k-coloring of G exists [26]. This minimum k is called the *equitable chromatic number* of G and denoted by $\chi_e(G)$.

Clearly the ECP is tightly related to the classical GCP. Like the GCP, the ECP is NP-hard [12] and thus computationally challenging. In addition to its theoretical significance, the ECP has a variety of applications, like garbage collection [28], partitioning and load balancing [4], scheduling [8,18,26], etc. For a review of possible applications of the ECP, the reader is referred to recent papers like [2,24].

Due to its relevance, much effort has been devoted to the studies of the ECP from a theoretical point of view. For example, Meyer suggested a conjecture that $\chi_e(G) < \Delta(G)$ for any connected graph except the complete graphs and the odd circuits, where $\Delta(G)$ is the maximum vertex degree of G [26]. Lih and Wu showed that $\chi_e(G) = \chi(G)$ holds for any connected bipartite graph $G(X, Y)$, where $\chi(G)$ is the chromatic number of G [22]. Lam et al. obtained an explicit formula to calculate the equitable chromatic number of a complete n-partite graph [21]. Bodlaender and Fomin proved that the ECP can be solved in polynomial time for graphs with bounded treewidth [5]. Kostochka and Nakprasit proved that a graph with maximum degree $\Delta(G)$ is equitably k-colorable for every $k \geq \Delta + 1$ if the average degree of vertices are at most $\Delta/5$ [20]. Furmańczyk discussed the computational complexity of the ECP for some special graphs [12]. Wu and Wang investigated the planar graphs with large girth [29]. Chen and Lih investigated equitable colorings for trees [7] and Chang studied equitable colorings for forests [6]. Nakprasit and Nakprasit obtained some results on the ECP for planar graphs with some special properties [27]. Yan and Wang investigated the ECP for Kronecker products of the complete multipartite graphs and complete graphs [30].

From the computational point of view, several exact and heuristic algorithms have been proposed in the literature for solving the ECP for arbitrary graphs. Specifically, Furmanczyk and Kubale proposed two constructive heuristics

(Naive and Subgraph) [11]. Bahiense et al. presented two effective branch-and-cut algorithms [1,2]. Méndez-Díaz et al. investigated a polyhedral approach [23], a DSatur-based exact algorithm [25], and a tabu search heuristic [24]. Kierstead et al. proposed a fast algorithm ($O(n^2)$) to find an equitable k-coloring where $k = \Delta(G) + 1$ [19].

Given the NP-hard nature of the ECP, it is unlikely that an exact algorithm will be found that is able to determine the equitable chromatic number of arbitrary graphs in polynomial time. Consequently, as for any NP-hard problem, heuristics constitute a very appealing and indispensable alternative which can be used to approximate the equitable chromatic number of a graph. On the other hand, the literature review shows that there are only limited studies on heuristic algorithms for the ECP.

In this paper, we provide a method to solve the ECP approximately by means of a backtracking based iterated tabu search (BITS) algorithm. BITS solves a series of k-ECP, i.e., equitable k-coloring instances with different fixed k values. For a given k-ECP (with a particular k), the purpose of the iterated tabu search (ITS) is to seek a conflict-free equitable k-coloring. The backtracking scheme is used to adjust k to an appropriate value. A technique based on binary search is used to determine a good initial k value.

We perform a computational study testing our proposed BITS algorithm on a large number of benchmark instances widely used in the literature. The computational results show that our new algorithm is very competitive in terms of both solution quality and computation efficiency compared to the state-of-the-art heuristics. Specifically, we are able to find new upper bounds of the equitable chromatic number for 21 benchmark instances, matching the previous best upper bound for the remaining instances.

The rest of the paper is organized as follows. In Section 2, we describe the components of the BITS algorithm, including the search space and evaluation function, the initial solution generation procedure, the iterated tabu search procedure, and the binary search procedure for determining an initial k. In Section 3, we present the computational benchmark assessments and compare our results with those of the state-of-the-art heuristic algorithms in the literature. Section 4 analyzes and discusses some important components of the proposed algorithm. Finally, we provide concluding comments in Section 5.

2 Backtracking Based Iterated Tabu Search for the ECP

In this section, we present the general solution approach and the supporting procedures that compose it.

Algorithm 1 Main Scheme of the Proposed BITS Algorithm for the ECP

Input: Graph $G = (V, E)$, the cutoff time t_{max}, the depth of backtracking m, the depth of tabu search α, the number of perturbations β.
Output: the best number of colors (k^*) and an equitable k^*-coloring (s^*).
/* Find an initial k (k^r) and an equitable k^r-coloring using the binary search */
$(k^r, s^r) \leftarrow$ Binary_Search(V, E, α) /* Determine an initial k^r and s^r, Section 2.3 */
$k^* \leftarrow k^r$, $k \leftarrow k^r$, $s^* \leftarrow s^r$ /* k^* and s^* keep the best results found */
repeat
  if $k = k^* - m$ or $k = 2$ **then**
    $k \leftarrow k^* - 1$ /* backtrack to $k^* - 1$ once $k = k^* - m$ or 2 */
  else
    $k \leftarrow k - 1$
  end if
  /* solve the corresponding k-ECP using the ITS algorithm */
  $s \leftarrow$ Iterated_Tabu_Search(k, G, α, β) /* Section 2.4 */
  if $f(s) = 0$ **then**
    $k^* \leftarrow k$, $s^* \leftarrow s$
  end if
until $Time() \geq t_{max}$
return (k^*, s^*)

As a basis for determining the equitable chromatic number of a graph by finding the smallest number k of colors such that an equitable k-coloring exists, we employ the fact that, like the GCP [14,16,17], the ECP can be approximated by solving a series of k-ECP problems with *decreasing* k values, where a k-ECP problem aims at searching for a legal equitable k-coloring for a given fixed k value. This approach is called k-fixed penalty approach in the context of the GCP [14,16] and used in TabuEqCol [24] for the ECP. Our BITS algorithm, however, adopts this general solution approach with a notable difference. Unlike previous studies, where the search stops once a k-ECP cannot be solved and returns $k + 1$ as the final solution, our BITS algorithm continues with $k - 1$, $k - 2$... even if the k-ECP is not solved. As a result, BITS iteratively tries to find an equitable k-coloring with $k \in \{k^* - 1, k^* - 2, \ldots, k^* - m\}$ where k^* is the current smallest number of colors admitting an equitable k^*-coloring and $m > 1$ is a parameter. We provide arguments in favor of this approach in Section 2.5.

Specifically, starting from an initial k determined with the method given in Section 2.3, our BITS algorithm seeks to solve the k-ECP to find an equitable k-coloring. This leads to one of two possible situations. First, if an equitable k-coloring is found, then after recording k as k^* (k^* records the current smallest number of colors admitting an equitable k^*-coloring), we set k to $k - 1$ and solve the new k-ECP. Second, if no legal equitable k-coloring is found for the

current k after a given amount of computational effort, we still set k to $k-1$ and try to solve the new k-ECP. If no k-ECP is solved for m consecutive $k = k^* - 1, k^* - 2, ..., k^* - m$ values (where the parameter m is called the backtracking depth and fixed to 4 in this paper), we perform a backtracking step by resetting k to $k^* - 1$ and start a new round of search with $k \in \{k^* - 1, k^* - 2, \ldots, k^* - m\}$. Backtracking also occurs when $k = 2$ (which is the smallest lower bound for $\chi_e(G)$).

To solve each k-ECP, our BITS algorithm employs an iterated tabu search (ITS) procedure (Section 2.4) which operates in a solution space where the equity constraint is always satisfied and only the coloring constraint can be violated (Section 2.2). ITS undertakes to find a conflict-free k-coloring where the equity constraint is always maintained during the search.

The pseudo-code of the BITS algorithm is given in Algorithm 1, consisting of two stages. In the first stage, a binary search method is employed to rapidly determine an appropriate initial number of colors (k^r) which admits an equitable k^r-coloring (lines 3-4, see Section 2.3). In the second stage, the algorithm enters a loop which primarily consists of three parts, the backtrack mechanism for setting k (lines 6-10), the iterated tabu search procedure for solving the associated k-ECP (line 11, see Section 2.4), and the procedure for updating k^* (lines 12-14). BITS terminates once a given timeout limit is reached (line 15) and returns the smallest number k^* of colors admitting an equitable k^*-coloring as the approximate equitable chromatic number.

2.2 Search Space and Evaluation Function

For a given k-ECP, the BITS algorithm searches a space Ω_k composed of all possible k-partitions (denoted by $P = \{V_1, V_2, \ldots, V_k\}$) satisfying the equity constraint, which are called *k-eqpartitions* in this work. Formally, Ω_k can be written as:

$$\Omega_k = \{P :|| V_i | - | V_j ||\leq 1, i \neq j, 1 \leq i, j \leq k\} \quad (1)$$

For the ECP, the complete search space Ω explored by our BITS algorithm can be written as:

$$\Omega = \bigcup_{k=2}^{n} \Omega_k \quad (2)$$

Notice that any candidate k-eqpartition of the search space satisfies the equity constraint, but does not necessarily satisfy the coloring constraint, i.e., some adjacent vertices may receive the same color.

To evaluate the quality of a candidate k-eqpartition in Ω, we adopt an evalu-

ation function which counts the total number of conflicting edges induced by the k-eqpartition. Specifically, letting $s = \{V_1, V_2, \ldots, V_k\}$ be a k-eqpartition in Ω, the evaluation function $f(s)$ used by our BITS algorithm is written as [14,16]:

$$f(s) = \sum_{\{v_i,v_j\}\in E} \delta(i,j) \tag{3}$$

$$\delta(i,j) = \begin{cases} 1, & \text{if } \exists y \in \{1,2,\ldots,k\} \text{ such that } v_i \in V_y \text{ and } v_j \in V_y; \quad (4) \\ 0, & \text{otherwise;} \quad (5) \end{cases}$$

Therefore, a k-eqpartition $s \in \Omega$ with $f(s) = 0$ corresponds to an equitable k-coloring satisfying both the equity and coloring constraints and is thus a legal solution to the k-ECP. To locate such an equitable k-coloring, our ITS procedure explores the search space of the given k-ECP by transitioning between various k-eqpartitions while minimizing the evaluation function with the purpose of attaining a solution s with $f(s) = 0$.

2.3 Binary Search Method for Finding an Initial Value of k

Algorithm 2 Binary Search (V, E, α)

```
1: Input: Graph G = (V, E), the depth of tabu search α.
2: Output: the obtained k^r and an equitable k^r-coloring s^r
3: U^K ← |V|, L^K ← 0
4: while U^K > L^K + 1 do
5:    k ← ⌊(U^K + L^K)/2⌋
6:    s ← Solution_Initialization(V,E,k)        /* Section 2.4.1 */
7:    s ← Tabu_Search(s,α)                      /* Section 2.4.2 */
8:    if f(s) = 0 then
9:       s^r ← s, k^r ← k, U^K ← k
10:   else
11:      L^K ← k
12:   end if
13: end while
14: return (k^r, s^r)
```

Clearly, it would be excessively time-consuming for BITS to solve all k-ECP problems ($2 \leq k \leq |V|$). Hence, to speed up the search, we employ a binary search method to determine an appropriate initial value of k.

Our binary search method (Algorithm 2) uses two variables (respectively denoted by U^K and L^K) to record the upper and lower bounds of k. Specifically, we begin by setting $U^K = |V|$, $L^K = 0$ and $k = \lfloor (U^K + L^K)/2 \rfloor$ and then solve the associated k-ECP by means of a short tabu search starting with a random initial solution in Ω_k. If an equitable k-coloring is obtained by the tabu search procedure, we set U^K to the current k, and set L^K to the current k otherwise.

The above process is repeated until the condition $U^K \leq L^K + 1$ holds. Finally, U^K is returned as the result of the binary search method.

2.4 Iterated Tabu Search Procedure for k-ECP

Algorithm 3 Iterated Tabu Search for k-ECP

```
1: Input: Graph G = (V, E), the number of colors k, the parameter β, the depth of
   tabu search α
2: Output: The best solution s found during this search process
3: s ← Solution_Initialization(V,E,k)                    /* Section 2.4.1 */
4: s ← Tabu_Search(s,α)                                  /* Section 2.4.2 */
5: d ← 0    /* d counts the consecutive perturbations where s is not updated */
6: repeat
7:    s' ← Perturbation_Operator(s)                      /* Section 2.4.3 */
8:    s'' ← Tabu_Search(s',α)                            /* Section 2.4.2 */
9:    if f(s'') < f(s) then
10:      s ← s''
11:      d ← 0
12:   else
13:      d ← d + 1
14:   end if
15: until d = β or f(s) = 0
16: return s
```

Our ITS algorithm starts with an initial solution (see Section 2.4.1) and then applies a basic tabu search (TS^0) procedure (in Section 2.4.2) to improve the incumbent solution (lines 3-4). Then, ITS repeatedly applies a perturbation operator to modify the incumbent solution (in Section 2.4.3) followed by an improvement phase by TS^0. The resulting solution is accepted as the new incumbent solution as long as it is better than the incumbent solution in terms of the evaluation function f (Eq. 3) (lines 5-15). Our ITS procedure terminates if a conflict-free (and equitable) k-coloring is found or if the best solution found so far cannot be improved after a number β of consecutive perturbations.

2.4.1 Initialization Procedure for the k-ECP

The purpose of the initialization step (Algorithm 4) is to generate an initial solution (i.e., a k-eqpartition) with as few conflicts as possible for the given k-ECP problem. This is achieved as follows. First, we randomly pick k distinct vertices from the set U of unassigned vertices (U is set to V initially) and assign them respectively to k color classes. Then, the unassigned vertices are assigned one by one to the current color class V_i in a greedy way, where V_i denotes the ith color class, and the initial value of i is set to 1. More specifically,

Algorithm 4 Solution Initialization for k-ECP

Input: Graph $G = (V, E)$, the number k of colors used
Output: a k-eqpartition $\{V_1, V_2, \ldots, V_k\}$ satisfying the equity constraint
for each $i \in [1, k]$ **do**
  $V_i \leftarrow \emptyset$
end for
$U \leftarrow V$ /* U is the set of unassigned vertices */
for each $i \in [1, k]$ **do**
  Randomly pick a vertex $v \in U$
  $V_i \leftarrow V_i \cup \{v\}$, $U \leftarrow U \setminus \{v\}$
end for
$i \leftarrow 1$
while $U \neq \emptyset$ **do**
  $v \leftarrow argmin\{|\Gamma^i(v')| : v' \in U\}$
  /* $\Gamma^i(v)$ is the set of neighbors of v in V_i, ties are broken randomly */
  $V_i \leftarrow V_i \cup \{v\}$, $U \leftarrow U \setminus \{v\}$
  $i \leftarrow 1+ i \bmod k$.
end while

for the current color class V_i, we choose an unassigned vertex v that has the smallest number of neighbors in V_i and assign it to V_i, breaking the ties at random. After that, we move to the next color class in a rotating fashion by setting $i \leftarrow 1+ i \bmod k$, and repeat the former operations.

2.4.2 Basic Tabu Search Procedure

Algorithm 5 Tabu Search(s_0, α)

Input: Input solution s_0, the neighborhood N, the depth of tabu search α
Output: The best solution s_b found during the tabu search process
$s \leftarrow s_0$ /* s is the current solution */
$s_b \leftarrow s$ /* s_b is the best solution found so far */
$d \leftarrow 0$ /* d counts the consecutive iterations where s_b is not updated */
repeat
  Choose a best admissible neighboring solution $s' \in N(s)$
  /* s' is admissible if it is not forbidden by the tabu list or better than s_b */
  $s \leftarrow s'$
  Update tabu list
  if $f(s) < f(s_b)$ **then**
    $s_b \leftarrow s$,
    $d \leftarrow 0$
  else
    $d \leftarrow d + 1$
  end if
until $d = \alpha$ or $f(s) = 0$
return s_b

The basic tabu search procedure TS^0 used in our ITS algorithm is based

on the classic TabuCol algorithm for the GCP [17]. The goal of TS^0 is to find a legal k-coloring (i.e., a conflict-free k-eqpartition) starting from a given initial conflicting k-coloring in the search space Ω_k (see Section 2.2). Our TS^0 procedure shares the same evaluation function and neighborhood structure as TabuEqCol [24]. However, in contrast to TabuEqCol (and TabuCol), our TS^0 procedure employs a hybrid tabu list management strategy to control the tabu tenures dynamically. For the sake of completeness, we describe our TS^0 procedure as follows.

Given a neighborhood function N (see below), the evaluation function f (Section 2.2), and a given initial solution s_0 (Section 2.4.1), our TS^0 iteratively replaces the incumbent solution s by a neighboring solution $s^{'}$ (initially set to s_0) until a stopping condition is met, i.e., an equitable k-coloring is found or the best solution found so far is not updated during α consecutive iterations (α is called the depth of tabu search). At each iteration of TS^0, the incumbent solution s is replaced by a best admissible neighboring solution $s^{'}$ generated by a move operator, and the associated move is recorded on the tabu list to prevent the reverse move from being performed for the next tt iterations, where tt is called the tabu tenure and is dynamically controlled by the tabu list management strategy (see below). Here, a move is considered to be admissible if it is not forbidden by the tabu list or if it leads to a solution better than the best solution found so far. The pseudo-code of TS^0 is described in Algorithm 5, and its ingredients are described as follows.

Neighborhood Structures. Our tabu search procedure exploits the union of two basic neighborhoods, i.e., the critical one-move neighborhood N_1 and critical swap neighborhood N_2. The neighborhood N_1 can be described by a constrained $OneMove$ operator. Given a conflicting k-coloring (or a k-eqpartition) $s = \{V_1, V_2, \ldots, V_k\}$, the constrained $OneMove$ operator transfers a conflicting vertex v from its current color class V_i to another color class V_j ensuring that the equity constraint is always respected, i.e., $|V_i| \geq \lfloor \frac{n}{k} \rfloor$ and $|V_j| \leq \lceil \frac{n}{k} \rceil$. Let $< v, V_i, V_j >$ designate such a move and $s \oplus < v, V_i, V_j >$ be the resulting neighboring solution when applying the move to s. Let $C(s)$ be the set of conflicting vertices in the current solution s, a vertex being conflicting if it belongs to the same color class as at least one of its adjacent vertices. Then the neighborhood N_1 of s is composed of all possible solutions that can be obtained by applying the constrained $OneMove$ operator to s. i.e.,

$$N_1(s) = \{s \oplus < v, V_i, V_j > : v \in V_i \cap C(s), i \neq j, |V_i| > \lfloor \tfrac{n}{k} \rfloor, |V_j| < \lceil \tfrac{n}{k} \rceil\}$$

where n is the number of vertices in the graph. In other words, $N_1(s)$ is the set of k-eqpartitions which can be reached by transferring a conflict vertice of s to another color class. Note that this neighborhood is empty and not applicable if $n = l \times k$ holds, where l is a positive integer. Clearly N_1 is bounded by $O(|C(s)| \times k)$ in size.

The neighborhood N_2 is defined by the constrained $Swap$ operator. Given two vertices v and u which are located in two different color classes and either v or u is a conflict vertex, $Swap(v,u)$ exchanges their color classes to produce a neighboring solution. Thus, the neighborhood N_2 of a solution s is composed of all possible neighboring solutions that can be obtained by applying the constrained $Swap$ operator to s, i.e.,

$$N_2(s) = \{s \ \oplus Swap(v,u) : v \in V_i, u \in V_j, i \neq j, \{v,u\} \cap C(s) \neq \emptyset\}$$

$N_2(s)$ is thus the set of k-eqpartitions which can be reached from s by swapping two vertices such that at least one of them is a conflict vertex. Clearly the size of N_2 is bounded by $O(|C(s)| \times n)$.

Our TS^0 procedure explores the combined neighborhood $N_3(s)$ which is the *union* of N_1 and N_2: $N_3(s) = N_1(s) \cup N_2(s)$.

Fast Neighborhood Evaluation Technique. To effectively calculate the move value (Δ_f) which identifies the change in the evaluation function f (Equation 3), our TS^0 procedure adopts a fast incremental evaluation technique first developed for the graph coloring problem [9,10,13]. Specifically, a $n \times k$ matrix M is maintained in which the entry $M[v][q]$ ($1 \leq v \leq n, 1 \leq q \leq k$) is the number of adjacent vertices to v which are colored in color q in the current solution $s = \{V_1, V_2, \ldots, V_k\}$, i.e., $M[v][q] = \sum_{\{v,u\} \in E} \delta_1(q, P(u))$, where $P(u)$ represents the color class of vertex u in s and the function $\delta_1(x,y)$ is defined as follows:

$$\delta_1(x,y) = \begin{cases} 1, & \text{for } x = y; \quad (6) \\ 0, & \text{otherwise}; \quad (7) \end{cases}$$

With this memory structure, each move value can be rapidly calculated. First, when a $OneMove$ move (i.e., $< v, V_i, V_j >$) is performed, its move value, $\Delta_f(< v, V_i, V_j >)$, can be calculated as $\Delta_f(< v, V_i, V_j >) = M[v][V_j] - M[v][V_i]$, and then the matrix M is accordingly updated as follows. For each neighbor u of vertex v, $M[u][V_i] \leftarrow M[u][V_i] - 1$, and $M[u][V_j] \leftarrow M[u][V_j] + 1$. Clearly, updating M can be done in $O(n)$. Second, if a $Swap$ move (i.e., $Swap(u,v)$) is performed, its move value $\Delta_f(Swap(u,v))$ can be calculated as $\Delta_f(Swap(u,v)) = (M[v][P(u)] - M[v][P(v)]) + (M[u][P(v)] - M[u][P(u)]) - 2\delta_2(v,u)$, where the function $\delta_2(v,u)$ is defined as

$$\delta_2(v,u) = \begin{cases} 1, & \{v,u\} \in E; \quad (8) \\ 0, & \text{otherwise}; \quad (9) \end{cases}$$

It should be noted that the $Swap(u,v)$ move can be performed as a combined move which consists of two consecutively performed $OneMove$ moves, i.e., $s \ \oplus Swap(v,u) = (s \oplus < v, P(v), P(u) >) \oplus < u, P(u), P(v) >$, thus matrix M can be updated in $O(n)$.

Tabu List Management Strategy. The tabu list is used to impart vigor to the search and prevent (or strongly discourage) the search from revisiting previously encountered solutions. If a $OneMove$ move ($< v, P_i, P_j >$) is performed, i.e., vertex v is displaced from its current color class V_i to color class V_j, then vertex v is forbidden to move back to V_i for the next tt (tabu tenure) iterations. Similarly, if a $Swap(v, u)$ move is performed, vertices v and u are forbidden to join their respective color classes for the next tt iterations. During the search process, we use a hybrid tabu list management strategy to dynamically tune the tabu tenure tt, which integrates the following rules.

By the first rule, if the current iteration is x, the tabu tenure of a move is determined as: $tt(x) = C_0 + rand(C_1)$, where C_0 and C_1 are two constant numbers which are empirically set to 5, and $rand(C_1)$ is a random number between 0 to C_1. This rule is used to generate small tabu tenures ($tt \leq 10$), thus favoring an intensified search of tabu search.

The second rule, first proposed in [9,13] and adapted in [24] consists of setting $tt(x) = \alpha_0 \times |C(s)| + rand(\beta)$, where $C(s)$ is the set of conflicting vertices in the current solution s, α_0 and β are two parameters which are respectively set to 0.9 and 5.

The third rule adapts the technique proposed in [15] where the tabu tenure is adjusted by a periodic step function. Specifically, the tabu tenure function is defined, for each period, by a sequence of values $(a_1, a_2, \cdots, a_{p+1})$ and a sequence of interval margins $(x_1, x_2, \cdots, x_{p+1})$ such that for each iteration x in $[x_i, x_{i+1} - 1]$, $tt(x) = a_i + rand(2)$. Here, p is fixed to 15, $(a)_{i=1,\cdots,15} = \frac{T_{max}}{8}(1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)$, where T_{max} is a parameter and represents the maximum tabu tenure. The interval margins are defined by $x_1 = 1$, $x_{i+1} = x_i + 3a_i$,$(i \leq 15)$.

To enhance the robustness of the tabu search algorithm, the three preceding rules are applied in alternation and each rule is employed for γ ($\gamma = 3 \times 10^4$) consecutive iterations.

2.4.3 Perturbation Operators of Iterated Tabu Search

During the TS^0 procedure, if the best solution found so far cannot be improved during α (i.e., the depth of tabu search) consecutive iterations, the search is estimated to be trapped in a deep local optimum. To jump out of the local optimum trap, we follow the strategy of the breakout local search method [3] and apply both random perturbations and directed perturbations to change the incumbent solution. These two perturbations can be described as follows.

Given a solution s to be perturbed, the directed perturbation iteratively performs η_1 (empirically set to 5×10^3) $OneMove$ or $Swap$ moves (see Section

2.4.2 for these operators), and at each iteration it chooses a best unforbidden move to perform, breaking ties randomly. First, if a $OneMove$ move (denoted by $< v, V_i, V_j >$) is chosen, i.e., vertex v is moved from the current color class V_i to the new color class V_j, then vertex v is forbidden to move back to V_i for the next tt moves. Second, if a $Swap$ move (denoted by $Swap(v, u)$) is performed, then vertices v and u are forbidden to move back to their respective color class for the next tt moves. Here, tt is the tabu tenure and is dynamically determined as $tt = 2000 + rand(1000)$, where $rand(1000)$ represents a random number between 0 and 1000. In addition, a forbidden move is always accepted if it leads to a better solution than the best solution found so far.

The goal of the directed perturbation is twofold. In addition to leading the search toward a new search region, the directed perturbation also serves as a local optimization procedure for some special problem instances.

The random perturbation is composed of η_2 (empirically set to $0.3 \times n$) consecutively performed $Swap$ moves that are randomly chosen from the set of available moves, and for each swap move the colors of two vertices v and u belonging to two different color classes are exchanged.

In our ITS algorithm, the random and directed perturbations are applied with a probability of $p \in [0, 1]$ and $1 - p$, respectively.

2.5 Motivation of the Backtracking Mechanism

As explained in Section 2.1, we approximate the equitable chromatic number by searching for k within an interval $[k^* - m, k^* - 1]$ where k^* is the current smallest number of colors admitting an equitable k^*-coloring. This strategy is different from the commonly used technique for the graph coloring problem where the search stops and returns $k^* = k + 1$ as the approximate chromatic number as soon as no k-coloring can be found. We justify our adopted strategy as follows.

For the GCP, the $(k - 1)$-coloring problem is necessarily more difficult than the k-coloring problem. Hence in this case it is suitable to stop the search for a $(k - 1)$-coloring if no k-coloring can be found.

However, unlike the GCP, due to the equity constraint in the ECP, it is possible that the $(k - 1)$-ECP is easier to solve than the k-ECP for some graphs and some k values (This is experimentally confirmed as we show in Section 4.3). Therefore, for the ECP, it is not appropriate to solve a series of k-ECP with strictly decreasing k values, since such a search process can be blocked by a very difficult k-ECP, but can be unblocked if we continue with $(k - j)$-ECP $(j > 1)$. For this reason, our BITS algorithm uses a backtracking mechanism

to allow the search to iteratively consider several k-ECP problems with $k \in \{k^* - 1, k^* - 2, \ldots, k^* - m\}$. As we show in Sections 3 and 4.3, this strategy proves to be useful for a number of problem instances.

3 Computational Results and Comparisons

In this section, we present computational results and comparisons to assess the performance of the proposed BITS algorithm.

3.1 Benchmark Instances

We tested a set of 73 benchmark instances which are commonly used in the literature. These instances, which were initially proposed for the conventional graph coloring problems, are available at `http://www.info.univ-angers.fr/pub/porumbel/graphs/` and `http://www.cs.hbg.psu.edu/txn131/graphcoloring.html`.

3.2 Parameter Settings and Experimental Protocol

Table 1
Settings of important parameters

Parameters	Section	Description	Values
m	2.1	depth of backtracking	4
β	2.4	number of perturbations	30
α	2.4.2	depth of tabu search	$\{10^2, 10^5\}$
T_{max}	2.4.2	maximum tabu tenure of TS	80
γ	2.4.2	number of iterations for each tabu rule	3×10^4
η_1	2.4.3	strength of directed perturbation	5×10^3
η_2	2.4.3	strength of random perturbation	$0.3 \times n$
p	2.4.3	probability of choosing the random perturbation	0.7

First, Table 1 shows the parameters used in our algorithm as well as their settings which are determined by preliminary experiment. Note that the tabu search depth parameter α takes 10^2 for the tabu search procedure employed in the binary search method introduced in Section 2.3, and 10^5 in other cases.

Our BITS algorithm was coded in C++ and compiled using the g++ compiler with the '-O2' option [1], and all experiments were carried out with an Intel Xeon E5440 processor (2.83 GHz CPU and 2 Gb RAM), running the Linux operating system. The user times of the DIMACS machine benchmark

[1] The source code of our BITS algorithm will be available at `http://www.info.univ-angers.fr/pub/hao/ecp.html`.

procedure [2] on our machine are 0.23, 1.42 and 5.42 seconds for graphs r300.5, r400.5, and r500.5, respectively.

Two experiments were carried out to assess and compare the performance of the proposed BITS algorithm. In the first experiment, we followed [24] and ran our BITS algorithm one time to solve each instance with a cutoff time of 3600 seconds. This allowed us to compare our results with those of [24]. Secondly, to further assess the search capacity of the proposed algorithm under a longer timeout limit, in the second experiment we used a cutoff time of 10^4 seconds for the instances with $n \leq 500$ and 2×10^4 seconds for larger instances with $n > 500$, respectively. Note that such cutoff limits (and even larger values) were typically used in the literature for testing graph coloring algorithms. Moreover, given the random nature of our BITS algorithm, each instance was independently solved 20 times in the second experiment.

3.3 Computational Results and Comparison

Table 2 reports the computational results achieved by the BITS algorithm on the set of 73 benchmark instances [3]. The first two columns give the name and size of each instance. Columns 3 and 4 indicate the best lower bound (LB) and upper bound (UB) of χ_e reported in [24,25], which are yielded by some very recent algorithms or CPLEX. Column 5 reports the best results (k_{pre}) obtained in [24] by means of a recent and highly effective tabu search algorithm which is the current best performing heuristic for the ECP, with a one hour cutoff time. The results of our first experiment with a one hour cutoff time are given in column 6 (k_1), and the results of the second experiment with relaxed cutoff times are given in columns 7 to 10, including the best result for each instance (i.e., the smallest number of colors achieved by the BITS algorithm) over 20 independent runs (k_{best}), the average results (k_{avg}), the success rate (SR) to achieve k_{best} over 20 runs, and the average computing time (in seconds) to obtain our best results ($t(s)$). A bold entry highlights an improved result relative to the published best upper bound. An entry in italics indicates that our upper bound matches the current best lower bounds and consequently the optimality of the solution is proved. Entries with "-" mean that the corresponding results are not available in the literature.

Table 2 discloses that the outcomes from our BITS algorithm are noteworthy compared to the current best known results in the literature. First, BITS improves the best known upper bound for 21 instances, while matching the best known upper bound for the other instances that have been investigated

[2] dmclique: `ftp://dimacs.rutgers.edu/pub/dsj/clique`.

[3] The best results reported in this work is available at: `http://www.info.univ-angers.fr/pub/hao/ECP/ecpresults.zip`

Table 2
Computational results of the proposed BITS algorithm on the set of 73 benchmark instances. Improved results are indicated in bold compared to the previous best upper bound. The optimum results obtained in this work are indicated in italic.

Instance	N	LB [24,25]	UB [24,25]	k_{pre} [24]	BITS				
					k_1	k_{best}	k_{avg}	SR	$t(s)$
DSJC125.1	125	5	5	5	*5*	*5*	5.0	20/20	0.96
DSJC125.5	125	9	18	18	**17**	**17**	17.5	10/20	5169.38
DSJC125.9	125	43	45	45	**44**	**44**	44	20/20	0.16
DSJC250.1	250	5	8	8	8	8	8.0	20/20	5.50
DSJC250.5	250	12	32	32	32	**30**	31.9	1/20	3265.63
DSJC250.9	250	63	83	83	**72**	**72**	72.0	20/20	1179.92
DSJC500.1	500	5	13	13	13	13	13.0	20/20	6.96
DSJC500.5	500	13	62	63	**57**	**56**	56.95	1/20	484.60
DSJC500.9	500	101	148	182	**130**	**129**	129.9	2/20	3556.53
DSJR500.1	500	12	12	12	12	12	12.0	20/20	0.58
DSJR500.5	500	120	131	133	**126**	**126**	126.3	14/20	3947.61
DSJC1000.1	1000	5	22	22	22	**21**	21.95	1/20	3605.49
DSJC1000.5	1000	15	112	112	112	**103**	105.1	3/20	18078.94
DSJC1000.9	1000	126	268	329	**254**	**252**	253.3	1/20	4064.65
R125.1	125	-	-	-	5	5	5.00	20/20	0.01
R125.5	125	-	-	-	36	36	36.00	20/20	0.39
R250.1	250	-	-	-	8	8	8.00	20/20	0.01
R250.5	250	-	-	-	67	66	66.65	7/20	6275.08
R1000.1	1000	-	-	-	20	20	20.00	20/20	3.09
R1000.5	1000	-	-	-	269	250	250.40	12/20	10723.29
le450_5a	450	5	5	-	*5*	*5*	5.00	20/20	45.86
le450_5b	450	5	5	7	*5*	*5*	5.00	20/20	74.43
le450_5c	450	-	-	-	5	5	5	20/20	1877.73
le450_5d	450	5	8	8	**5**	**5**	5.00	20/20	2231.59
le450_15a	450	15	15	-	*15*	*15*	15.00	20/20	4.44
le450_15b	450	15	15	15	*15*	*15*	15.00	20/20	4.16
le450_15c	450	-	-	-	15	15	15.1	18/20	410.35
le450_15d	450	15	16	16	**15**	**15**	15.70	6/20	629.83
le450_25a	450	25	25	-	*25*	*25*	25.00	20/20	0.72
le450_25b	450	25	25	25	*25*	*25*	25.00	20/20	0.78
le450_25c	450	-	-	-	26	26	26.00	20/20	16.50
le450_25d	450	25	27	27	**26**	**26**	26.00	20/20	14.08
wap01a	2368	41	46	46	**43**	**42**	42.60	8/20	4183.29
wap02a	2464	40	44	44	**42**	**41**	41.80	4/20	6829.03
wap03a	4730	40	50	50	**46**	**45**	45.05	19/20	11267.27
wap04a	5231	-	-	-	46	44	44.15	17/20	11345.30
wap05a	905	-	-	-	50	50	50.00	20/20	8.46
wap06a	947	-	-	-	42	41	41.70	6/20	6892.09
wap07a	1809	-	-	-	43	43	43.05	19/20	718.25
wap08a	1870	-	-	-	43	43	43.05	19/20	951.85
flat300_28_0	300	11	36	36	**35**	**34**	34.70	6/20	4407.62
flat1000_50_0	1000	-	-	-	112	101	102.80	1/20	9206.28
flat1000_60_0	1000	-	-	-	112	102	102.90	5/20	10201.53
flat1000_76_0	1000	14	112	112	112	**102**	103.40	3/20	13063.39
latin_square_10	900	90	130	130	**129**	**115**	120.00	1/20	17859.13
C2000.5	2000	-	-	-	202	201	201.65	7/20	4808.96
C2000.9	2000	-	-	-	504	502	502.45	11/20	7772.04
mulsol.i.1	197	49	49	50	*49*	*49*	49.00	20/20	14.82
mulsol.i.2	188	34	39	48	**36**	**36**	36.35	13/20	3633.61
fpsol2.i.1	496	65	65	78	*65*	*65*	65.00	20/20	830.30
fpsol2.i.2	451	47	47	60	*47*	*47*	47.00	20/20	976.07
fpsol2.i.3	425	55	55	79	*55*	*55*	55.00	20/20	729.47
inithx.i.1	864	54	54	66	*54*	*54*	54.00	20/20	1468.27
inithx.i.2	645	30	93	93	**36**	**36**	36.35	13/20	12412.83
inithx.i.3	621	-	-	-	38	37	37.45	11/20	9214.61
zeroin.i.1	211	49	49	51	*49*	*49*	49.00	20/20	1367.14
zeroin.i.2	211	36	36	51	*36*	*36*	36.00	20/20	96.99
zeroin.i.3	206	36	36	49	*36*	*36*	36.00	20/20	109.11
myciel6	95	7	7	7	*7*	*7*	7.00	20/20	0.00
myciel7	191	8	8	8	*8*	*8*	8.00	20/20	0.02
4-FullIns_3	114	7	7	-	*7*	*7*	7.00	20/20	0.00
4-FullIns_4	690	6	8	8	8	8	8.00	20/20	0.23
4-FullIns_5	4146	6	9	9	9	9	9.00	20/20	54.49
1-Insertions_6	607	3	7	7	7	7	7.00	20/20	0.34
2-Insertions_5	597	3	6	6	6	6	6.00	20/20	0.11
3-Insertions_5	1406	3	6	6	6	6	6.00	20/20	0.57
school1	385	15	15	15	*15*	*15*	15.00	20/20	1.30
school1_nsh	352	14	14	14	*14*	*14*	14.00	20/20	2.63
qg.order40	1600	40	40	40	*40*	*40*	40.00	20/20	4.73
qg.order60	3600	60	60	60	*60*	*60*	60.00	20/20	21.57
ash331GPIA	662	4	4	4	*4*	*4*	4.00	20/20	2.02
ash608GPIA	1216	3	4	4	4	4	4.00	20/20	0.50
ash958GPIA	1916	3	4	4	4	4	4.00	20/20	23.31

in previous studies. Second, for 26 instances, our BITS algorithm attains the optimum solutions, i.e., our upper bounds equal the best lower bounds. In addition, our BITS algorithm obtains a success rate of at least 10/20 except for 17 instances, which shows the robustness of the algorithm. Finally, the average computing time is related to the hardness of the instances, and varies between 0 and 2×10^4 seconds.

Compared to the highly effective TabuEqCol algorithm of [24] which reports the previously best known upper bounds for the 50 tested instances, our algorithm under the same one hour cutoff time delivers a better result (upper bound) (k_1) for 26 instances, while achieving the same bounds for the remaining 24 instances. TabuEqCol was run on a computer with an Intel i5 CPU 750@2.67Ghz, which is comparable to the computer used to run our BITS algorithm (with an Intel Xeon E5440 CPU 2.83GHz).

4 Analysis and Discussions

In this section, we study several key ingredients of the BITS algorithm to get some insight into its behavior.

4.1 Influence of the Tabu List Management Strategy

The effectiveness of the critical tabu search procedure depends on its tabu list management strategy. To show the impact of our multiple tabu list strategy, we carried out an additional experiment based on a set of 40 representative instances by means of our BITS algorithm and its three variants, i.e., BITS1, BITS2, and BITS3 which respectively employ the first, second, and third tabu list management rule alone (Section 2.4.2 for details). In this experiment, each algorithm was independently performed 20 times to solve each instance with the same cutoff time limits given in Section 3.2.

The experimental results are summarized in Table 3. The first column indicates the names of instances, and the second column shows the best results (k^*) yielded in this experiment. The results of BITS and its three variants are respectively listed in columns 3 to 14, including the best results in the number of colors used (k_{best}), the success rate to reach k_{best} over 20 runs (SR), and the difference between k_{best} and k^* ($\Delta = k_{best} - k^*$). The rows *Equal* and *Worse* respectively indicate the number of instances for which the associated algorithm attains an equal and worse result compared to k^*. The row *Sum* denotes the sum of Δ over the tested instances.

Table 3
Influence of tabu list management strategies on the performance of the BITS algorithm. The performances of the BITS algorithms with four different tabu list management strategies are assessed on a set of 40 representative instances.

		BITS1			BITS2			BITS3			BITS		
Instance	k^*	k_{best}	SR	Δ	k_{best}	SR	Δ	k_{best}	SR	Δ	k_{best}	SR	Δ
DSJC125.5	17	17	20/20	0	17	3/20	0	17	1/20	0	17	10/20	0
DSJC250.5	29	29	2/20	0	32	20/20	3	32	20/20	3	30	1/20	1
DSJC250.9	72	72	20/20	0	72	20/20	0	72	3/20	0	72	20/20	0
DSJC500.1	13	13	20/20	0	13	20/20	0	13	13/20	0	13	20/20	0
DSJC500.5	56	56	7/20	0	57	20/20	1	57	1/20	1	56	1/20	0
DSJC500.9	128	128	2/20	0	129	14/20	1	131	10/20	3	129	2/20	1
DSJR500.5	126	126	11/20	0	126	9/20	0	126	2/20	0	126	14/20	0
DSJC1000.1	21	21	20/20	0	21	5/20	0	22	4/20	1	21	1/20	0
DSJC1000.5	101	101	1/20	0	112	20/20	11	112	20/20	11	103	3/20	2
DSJC1000.9	252	252	5/20	0	253	7/20	1	254	1/20	2	252	1/20	0
R250.5	66	67	6/20	1	67	2/20	1	66	19/20	0	66	7/20	0
R1000.5	250	250	4/20	0	250	16/20	0	250	2/20	0	250	12/20	0
le450_15c	15	15	1/20	0	15	16/20	0	15	19/20	0	15	18/20	0
le450_15d	15	16	18/20	1	15	8/20	0	15	7/20	0	15	6/20	0
le450_25c	26	26	20/20	0	26	20/20	0	27	20/20	2	26	20/20	0
le450_25d	26	26	20/20	0	26	20/20	0	27	20/20	2	26	20/20	0
wap01a	42	43	20/20	1	42	2/20	0	42	11/20	0	42	8/20	0
wap02a	41	42	2/20	1	41	4/20	0	41	8/20	0	41	4/20	0
wap03a	45	49	8/20	4	45	15/20	0	45	20/20	0	45	19/20	0
wap04a	44	48	4/20	4	44	20/20	0	44	20/20	0	44	17/20	0
wap06a	41	41	4/20	0	41	9/20	0	41	5/20	0	41	6/20	0
wap07a	42	43	19/20	1	42	5/20	0	43	19/20	1	43	19/20	1
wap08a	42	43	4/20	1	42	7/20	0	42	1/20	0	43	19/20	1
flat300_28_0	34	34	16/20	0	35	20/20	1	35	4/20	1	34	6/20	0
flat1000_50_0	101	101	2/20	0	104	8/20	3	105	5/20	4	101	1/20	0
flat1000_60_0	101	101	1/20	0	104	1/20	3	106	3/20	5	102	5/20	1
flat1000_76_0	102	102	10/20	0	104	2/20	2	108	1/20	6	102	3/20	0
latin_square_10	113	113	1/20	0	129	12/20	16	130	20/20	17	115	1/20	2
C2000.5	201	201	19/20	0	202	20/20	1	203	1/20	2	201	7/20	0
C2000.9	502	503	1/20	1	503	20/20	1	504	6/20	2	502	11/20	0
mulsol.i.2	36	36	10/20	0	36	10/20	0	36	13/20	0	36	13/20	0
fpsol2.i.1	65	65	20/20	0	65	20/20	0	65	20/20	0	65	20/20	0
fpsol2.i.2	47	47	20/20	0	47	20/20	0	47	20/20	0	47	20/20	0
fpsol2.i.2	55	55	20/20	0	55	20/20	0	55	20/20	0	55	20/20	0
inithx.i.1	54	54	20/20	0	54	20/20	0	54	20/20	0	54	20/20	0
inithx.i.2	36	36	15/20	0	36	11/20	0	36	14/20	0	36	13/20	0
inithx.i.3	37	37	13/20	0	37	15/20	0	37	9/20	0	37	11/20	0
zeroin.i.1	49	49	20/20	0	49	20/20	0	49	20/20	0	49	20/20	0
zeroin.i.2	36	36	20/20	0	36	20/20	0	36	20/20	0	36	20/20	0
zeroin.i.3	36	36	20/20	0	36	20/20	0	36	20/20	0	36	20/20	0
#Equal		31			27			24			33		
#Worse		9			13			16			7		
Sum				15			45			63			9

Table 3 discloses that the multiple tabu list strategy leads to the best performance among all 4 compared strategies. First, in terms of the number of instances for which the associated algorithm yields a worse result compared to k^*, BITS misses the current best known results (k^*) for only 7 instances, whereas BITS1, BITS1, and BITS3 miss the current best known results for 9, 13, 16 instances, respectively. It is interesting to note that the best results obtained in Section 3.3 are further improved in this experiment for 7 instances. Second, concerning the sum of Δ over all instances, the result of BITS is 9, whereas the results of BITS1, BITS2, and BITS3 are 15, 45, 63 respectively. On the other hand, when comparing the three single tabu list management strategies, one finds that the first strategy (i.e., BITS1) performs the best, and the third strategy (i.e., BITS3) performs the worst.

Table 4
Influence of perturbation operators on the performance of the BITS algorithm. Note that our BMTS algorithm is obtained by replacing its perturbation operators of the BITS algorithm with the initialization procedure for k-ECP. Better results between the BITS and BMTS algorithms are indicated in bold.

		BITS				BMTS			
Instance	N	k_{best}	k_{avg}	SR	$t(s)$	k_{best}	k_{avg}	SR	$t(s)$
DSJC125.5	125	17	17.50	10/20	5169.38	17	**17.45**	11/20	5511.65
DSJC250.5	250	30	31.90	1/20	3265.63	30	31.90	1/20	4318.98
DSJC250.9	250	72	72.00	20/20	1179.92	72	72.00	20/20	1884.94
DSJC500.1	500	13	13.00	20/20	6.96	13	13.00	20/20	7.41
DSJC500.5	500	56	56.95	1/20	484.60	56	56.95	1/20	1249.59
DSJC500.9	500	129	**129.90**	2/20	3556.53	129	130.00	2/20	5427.04
DSJR500.5	500	126	**126.30**	14/20	3947.61	126	126.50	10/20	2048.89
DSJC1000.1	1000	21	21.95	1/20	3605.49	21	**21.85**	3/20	2088.95
DSJC1000.5	1000	103	**105.10**	3/20	18078.94	**102**	106.00	1/20	18887.27
DSJC1000.9	1000	252	253.30	1/20	4064.65	252	**253.20**	2/20	4412.62
R250.5	250	66	**66.65**	7/20	6275.08	66	66.90	2/20	7315.66
R1000.5	1000	**250**	**250.40**	12/20	10723.29	268	271.05	1/20	17131.15
le450_15c	450	15	15.10	18/20	410.35	15	**15.05**	19/20	399.87
le450_15d	450	15	15.70	6/20	629.83	15	**15.50**	10/20	1143.19
le450_25c	450	26	26.00	20/20	16.50	26	26.00	20/20	14.77
le450_25d	450	26	26.00	20/20	14.08	26	26.00	20/20	16.85
wap01a	2368	42	**42.60**	8/20	4183.29	42	42.95	1/20	15801.24
wap02a	2464	**41**	**41.80**	4/20	6829.03	42	42.00	20/20	1621.23
wap03a	4730	45	**45.05**	19/20	11267.27	45	45.95	1/20	12994.88
wap04a	5231	44	**44.15**	17/20	11345.30	44	44.70	6/20	14063.33
wap06a	947	**41**	**41.70**	6/20	6892.09	42	42.00	20/20	78.90
wap07a	1809	43	43.05	19/20	718.25	43	**43.00**	20/20	1899.08
wap08a	1870	43	**43.05**	19/20	951.85	43	43.10	18/20	2538.18
flat300_28_0	300	34	34.70	6/20	4407.62	34	**34.65**	7/20	3644.83
flat1000_50_0	1000	101	102.80	1/20	9206.28	101	102.80	1/20	11486.31
flat1000_60_0	1000	102	**102.90**	5/20	10201.53	102	103.00	5/20	11549.44
flat1000_76_0	1000	**102**	**103.40**	3/20	13063.39	103	103.75	8/20	12468.99
latin_square_10	900	**115**	**120.00**	1/20	17859.13	116	121.55	1/20	18863.40
C2000.5	2000	201	**201.65**	7/20	4808.96	201	201.75	5/20	5577.63
C2000.9	2000	502	**502.45**	11/20	7772.04	502	502.70	6/20	10229.98
mulsol.i.2	188	**36**	**36.35**	13/20	3633.61	37	37.50	10/20	3340.18
fpsol2.i.1	496	**65**	**65.00**	20/20	830.30	84	85.90	3/20	5326.77
fpsol2.i.2	451	**47**	**47.00**	20/20	976.07	66	72.35	1/20	8753.76
fpsol2.i.3	425	**55**	**55.00**	20/20	729.47	79	83.55	1/20	9991.45
inithx.i.1	864	**54**	**54.00**	20/20	1468.27	68	70.95	1/20	12028.89
inithx.i.2	645	**36**	**36.35**	13/20	12412.83	56	58.15	6/20	13287.27
inithx.i.3	621	**37**	**37.45**	11/20	9214.61	58	60.10	4/20	9183.48
zeroin.i.1	211	**49**	**49.05**	20/20	1367.14	58	58.85	3/20	6531.84
zeroin.i.2	211	**36**	**36.00**	20/20	96.99	42	45.25	1/20	958.29
zeroin.i.3	206	**36**	**36.00**	20/20	109.11	45	45.05	19/20	1742.31
#Better		15	26			1	7		
#Equal		24	7			24	7		
#Worse		1	7			15	26		

4.2 Effectiveness of Perturbation Operators

Perturbation is another important ingredient of the proposed BITS algorithm. In order to show the effect of the perturbation operators on the performance of the BITS algorithm, we compared BITS with a variant called BMTS where instead of perturbing the incumbent solution, we generate a new solution by the initialization procedure introduced in Section 2.4.1, while keeping the other ingredients of BITS are unchanged. We ran both BITS and BMTS 20 times on each of the 40 representative instances used in Section 4.1 and reported the computational results in Table 4.

Table 4 shows that BITS substantially outperforms BMTS. First, in terms of the best result k_{best}, BITS obtains a better result on 15 out of the 40 instances compared to BMTS, while matching the results of BMTS for the other 24 instances. Second, BITS has a better average result (k_{avg}) than BMTS on 26

instances, while obtaining a worse result for only 7 instances. Concerning the success rate in the case where both algorithms obtain the same best results (k_{best}), BITS outperforms or matches BMTS in all but 6 instances. These outcomes imply that the perturbation operators of the BITS algorithm play an important role for its performance.

4.3 Importance of Backtracking Scheme

Table 5
Comparison of the BITS algorithms with and without the backtracking scheme on some selected instances. Better results between the two algorithms are indicated in bold.

		BITS				BITS$^-$			
Instance	N	k_{best}	k_{avg}	SR	$t(s)$	k_{best}	k_{avg}	SR	$t(s)$
flat1000_76_0	1000	**102**	**103.40**	3/20	13063.39	112	112.00	20/20	21.15
latin_square_10	900	**115**	**120.00**	1/20	17859.13	123	128.45	1/20	19758.08
DSJC1000.5	1000	**103**	**105.10**	3/20	18078.94	112	112.00	20/20	23.92
le450_5c	450	**5**	**5.00**	20/20	1877.73	7	7.00	20/20	1.21
le450_5d	450	**5**	**5.00**	20/20	2231.59	7	7.00	20/20	0.93

To assess the merit of our BITS backtracking scheme to solve the k-ECP instances with $k \in [k^* - m, k^* - 1], m > 1$, we remove this strategy and stop the search once the current k-ECP instance ($k = k^* - 1$) cannot be solved. In other words, we effectively set $m = 1$ and call this variant BITS$^-$. We ran both algorithms 20 times on each of five selected benchmark instances to highlight the fact that even if the backtracking mechanism is not needed in all situations, it is indeed very useful for finding improved solutions for some particularly difficult instances. The results of this experiment are summarized in Table 5 with the same information as in the previous tables.

As shown in Table 5, the BITS algorithm significantly outperforms the BITS$^-$ algorithm by obtaining much better results for the instances tested. These outcomes clearly indicate that the popular strategy designed for the classic graph coloring problem, i.e., solving a series of k-colorability problems with strictly decreasing k values, is not appropriate for the ECP, and our backtracking scheme helps the BITS algorithm to attain improved solutions for some difficult instances.

5 Conclusions

Our backtracking based iterated tabu search (BITS) approach to solve the equitable coloring problem (ECP) achieves a high level of performance by integrating several components: a backtracking scheme to define different k-ECP instances, a tabu search procedure with a hybrid tabu list management

strategy to solve each associated k-ECP instance, two perturbation operators to jump out of local optima, and a binary search method to determine the initial value of k.

The effectiveness of the BITS algorithm is demonstrated by a computational study on a set of 73 benchmark instances, comparing our method with the best existing heuristic algorithms in the literature. Among other features, our algorithm finds improved upper bounds for 21 out of the 73 benchmark instances.

Finally, we investigated several essential components to shed light on the behavior of the BITS algorithm. Our tests disclose that the multiple tabu list management strategy, the perturbation operators, and the backtracking scheme all contribute to the algorithm's performance.

Acknowledgments

We are grateful to the reviewers for their useful comments which helps us to improve the paper. The work is partially supported by the LigeRo project (2009-2013, Region of Pays de la Loire, France), the PGMO project (2013-2015, Jacques Hadamard Mathematical Foundation) and a post-doc grant (for X.J. Lai) from the Region of Pays de la Loire (France).

References

[1] Bahiense L., Frota Y., Maculan N., Noronha T.F., Ribeiro C.C., 2009, A branch-and-cut algorithm for equitable coloring based on a formulation by representatives. *Electronic Notes in Discrete Mathematics* 35, 347–252.

[2] Bahiense L., Frota Y., Noronha T.F., Ribeiro C.C., 2014, A branch-and-cut algorithm for the equitable coloring problem using a formulation by representatives. *Discrete Applied Mathematics* 164, 34–46.

[3] Benlic U., Hao J.K., 2013, Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence* 26(3), 1162–1173.

[4] Blazewicz J., Ecker K., Pesch E., Schmidt G., Weglarz J., 2001, Scheduling Computer and Manufacturing Processes, Springer.

[5] Bodlaender H.L., Fomin F.V., 2005, Equitable colorings of bounded treewidth graphs. *Theoretical Computer Science* 349(1), 22–30.

[6] Chang G.J., 2009, A note on equitable colorings of forests. *European Journal of Combinatorics* 30(4), 809–812.

[7] Chen B.L., Lih K.W., 1994, Equitable coloring of trees. *Journal of Combinatorial Theory, Series B* 61(1), 83–87.

[8] Ding J.-Y., Song S., Gupta J.N.D., Zhang R., Chiong R., Wu C., 2015, An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Applied Soft Computing* 30, 604–613.

[9] Dorne R., Hao J.K., 1998, Tabu search for graph coloring, T-colorings and set T-colorings. In S. Voss, S. Martello, I.H. Osman, C. Roucairol (Eds.), Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization, Chapter 6, pp77–92, Kluwer.

[10] Fleurent C., Ferland J.A., 1996. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research* 63(3), 437–461.

[11] Furmanczyk H., Kubale M., 2004, Equitable coloring of graphs. Graph colorings, Providence, Rhode Island, *American Mathematical Monthly* 35–53.

[12] Furmańczyk H., 2005, The complexity of equitable vertex coloring of graphs. *Journal of Applied Computer Science* 13(2), 95–107.

[13] Galinier P., Hao J.K., 1999, Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatioal Optimization* 3(4), 379–397.

[14] Galinier P., Hertz A., 2006, A survey of local search methods for graph coloring. *Computers & Operations Research* 33(9), 2547–2562.

[15] Galinier P., Boujbel Z., Fernandes M.C., 2011, An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research* 191(1), 1–22.

[16] Galinier P., Hamiez J.P., Hao J.K., Porumbel D., 2012, Recent advances in graph vertex coloring. In Zelinka I., Abraham A., Snasel V. (Eds.) Handbook of Optimization, Springer.

[17] Hertz A., de Werra D., 1987, Using tabu search techniques for graph coloring. *Computing* 39(4), 345–351.

[18] Irani S., Leung V., 1996, Scheduling with conflicts and applications to traffic signal control. in: *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, Atlanta. pp. 85–94.

[19] Kierstead H., Kostochka A., Mydlarz M., Szemeredi E., 2010, A fast algorithm for equitable coloring. *Combinatorica* 30, 217–224.

[20] Kostochka A.V., Nakprasit K., 2005, On equitable Δ-coloring of graphs with low average degree. *Theoretical Computer Science* 394(1), 82–91.

[21] Lam P.C.B., Shiu W.C., Tong C.S., Zhang Z.F., 2001, On the equitable chromatic number of complete n-partite graphs. *Discrete Applied Mathematics* 113, 307–310.

[22] Lih K.W., Wu P.L., 1996, On equitable coloring of bipartite graphs. *Discrete Mathematics* 151(1-3), 155–160.

[23] Méndez-Díaz I., Nasini G., Severín D., 2014, A polyhedral approach for the equitable coloring problem. *Discrete Applied Mathematics* 164, 413–426.

[24] Méndez-Díaz I., Nasini G., Severín D., 2014, A tabu search heuristic for the equitable coloring problem. *Lecture Notes in Computer Science* pp. 347–358.

[25] Méndez-Díaz I., Nasini G., Severín D., 2015, A DSATUR-based algorithm for the equitable coloring problem. *Computers & Operations Research* 57, 41–50.

[26] Meyer W., 1973, Equitable coloring. *American Mathematical Monthly* 80, 143–149.

[27] Nakprasit K., Nakprasit K., 2012, Equitable colorings of planar graphs without short cycles. *Theoretical Computer Science* 465, 21–27.

[28] Tucker A., 1973, Perfect graphs and an application to optimizing municipal services. *SIAM Review* 15, 585–590.

[29] Wu J.L., Wang P., 2008, Equitable coloring planar graphs with large girth. *Discrete Mathematics* 308(5-6), 985–990.

[30] Yan Z.D., Wang W., 2014, Equitable coloring of kronecker products of complete multipartite graphs and complete graphs. *Discrete Applied Mathematics* 162, 328–333.