RAIRO-Oper. Res. 54 (2020) 845–871
https://doi.org/10.1051/ro/2019037

RAIRO Operations Research
www.rairo-ro.org

A HEURISTIC FOR THE MINIMUM COST CHROMATIC PARTITION PROBLEM

Celso C. Ribeiro[1,*] and Philippe L. F. dos Santos[2]

Abstract. The graph coloring problem consists in coloring the vertices of a graph $G = (V, E)$ with a minimum number of colors, such as that any two adjacent vertices receive different colors. The minimum cost chromatic partition problem (MCCPP) is an extension of the graph coloring problem in which there are costs associated with the colors and one seeks a vertex coloring minimizing the sum of the costs of the colors used in each vertex. The problem finds applications in VLSI design and in some scheduling problems modeled on interval graphs. We propose a trajectory search heuristic using local search, path-relinking, and perturbations for solving MCCPP and discuss computational results.

Mathematics Subject Classification. 90C27, 68R10.

Received January 16, 2019. Accepted March 27, 2019.

1. Introduction

The vertex coloring problem consists in coloring the vertices of a graph $G = (V, E)$ with a minimum number of colors, such as that any two adjacent vertices receive different colors. The chromatic number $\chi(G)$ of G is the minimum number of colors used to color the vertices in V, see [24] for a survey.

The minimum cost chromatic partition problem (MCCPP) is an extension of the vertex coloring (VCP) problem in which there are costs associated with the colors and one seeks a vertex coloring minimizing the sum of the costs of the colors used in each vertex. The problem was introduced by Mead and Conway [25] and has applications in the design of VLSI circuits [33,35]. Kroon *et al.* [19] showed that MCCPP for interval graphs is equivalent to the fixed interval scheduling problem with machine-dependent processing costs.

The weighted vertex coloring problem (WVCP) amounts to the minimization of the sum of the costs of the colors used, where the cost of each color is the maximum weight of the vertices assigned to that color. Malaguti *et al.* [23] proposed three alternative integer linear programming formulations for WVCP and a 2-phase heuristic algorithm that embeds fast refinement procedures aimed at improving the quality of the solutions found.

No exact algorithms are reported for MCCPP in the literature. In this work, we propose a trajectory search heuristic with path-relinking for MCCPP. In Section 2, the minimum cost chromatic partition problem is formulated as an integer programming problem and some complexity results are presented. Section 3 presents

Keywords. Minimum cost chromatic partition problem, graph coloring problem, metaheuristics, trajectory search heuristic, path-relinking.

[1] Institute of Computing, Universidade Federal Fluminense, Niterói 24210-346, Brazil.
[2] Instituto Federal de Educação, Ciência e Tecnologia Fluminense, Campos dos Goytacazes 28030-130, Brazil.
*Corresponding author: celso@ic.uff.br

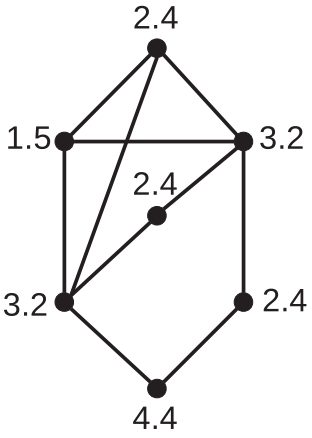


FIGURE 1. Graph with seven vertices colored with four colors, the solution cost is 19.5.

the trajectory search heuristic with path-relinking. Section 4 reports on the computational experiments and numerical results. Concluding remarks are drawn in the last section.

2. Formulation and complexity

Let $G = (V, E)$ be a graph with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E \subseteq V \times V$. We suppose that a set $\mathcal{C} = \{C_1, \ldots, C_n\}$ of colors is available and there is a cost w_c associated with each color C_c, $c = 1, \ldots, n$. Let x_{ic} be a binary variable such that $x_{ic} = 1$ if color C_c is assigned to vertex v_i, $x_{ic} = 0$ otherwise. Sen *et al.* [33] formulated the minimum cost chromatic partition problem as the following 0–1 integer programming problem:

$$\min \quad \sum_{i=1}^{n} \sum_{c=1}^{n} w_c \cdot x_{ic} \tag{2.1}$$

subject to:

$$\sum_{c=1}^{n} x_{ic} = 1, \ \forall i = 1, \ldots, n \tag{2.2}$$

$$x_{ic} + x_{jc} \leqslant 1, \ \forall i, j = 1, \ldots, n : i \neq j, (v_i, v_j) \in E, \ \forall c = 1, \ldots, n \tag{2.3}$$

$$x_{ic} \in \{0, 1\}, \ \forall i, c = 1, \ldots, n. \tag{2.4}$$

Constraints (2.2) impose that each vertex is colored with exactly one color. Constraints (2.3) guarantee that if some color is used, then only one of any two adjacent vertices can receive this color. The objective function (2.1) minimizes the sum of the costs of the color assignments.

Figure 1 depicts an example of a graph with seven vertices that is colored with colors whose weights are 3.2, 2.4, 1.5, and 4.4. The total cost of the solution is 19.5.

Figure 2 illustrates the MCCPP on a tree with eight vertices, for which three colors with costs 1.2, 2.5 and 3.5 are available. Figure 2a illustrates a solution using the minimum number of colors and the colors with smaller costs, with total cost of 14.8, while the solution in Figure 2b uses three colors and has total cost of 13.2.

The minimum cost chromatic partition problem was shown to be NP-hard for general graphs by Sen *et al.* [33] from a restriction to the chromatic sum problem, which was proved to be NP-hard in [20]. It can be polynomially solved for trees, co-graphs, graphs with constant tree-width and the complements of bipartite graphs and triangle free graphs [15, 16, 19]. The problem is also polynomially solvable on interval

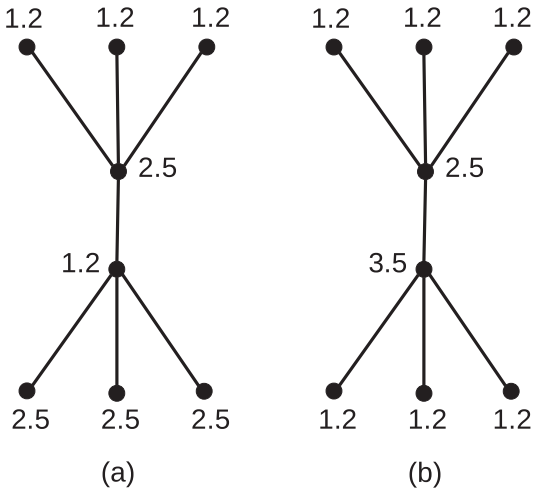


FIGURE 2. Tree (*Panel a*) colored with two colors with cost 14.8 and (*Panel b*) colored with three colors with optimal cost 13.2.

graphs if there are only two different values for the color costs, but remains NP-hard if there are at least four distinct values for the color costs.

Jansen [17] showed that there can be no polynomial approximation algorithm with ratio $O(|V|^{0.5-\varepsilon})$ for MCCPP, with $\varepsilon > 0$, for bipartite and interval graphs, unless P = NP. It is also showed that there can be no polynomial approximation algorithm with ratio $O(|V|^{1-\varepsilon})$, with $\varepsilon > 0$, for split, chordal, permutation, and comparability graphs, unless P = NP. Approximation algorithms with ratio $O(|V|^{0.5})$ are proposed for bipartite, interval, and uni-modular graphs.

Section 3 presents a trajectory search with path-relinking heuristic for MCCPP.

3. Trajectory search heuristic with path-relinking

The trajectory search heuristic TSH+PR developed in this section makes use of a randomized greedy heuristic combined with a simple neighborhood for local search to build diversified colorings, hybridized with perturbation and path-relinking strategies. A short-term, one-pass tabu list is used to avoid that the current solution is revisited in the next iteration. All components of the heuristic are described in the next sections.

3.1. Solution representation and evaluation function

Any solution to MCCPP in $G = (V, E)$ may be represented by a partition S: $\langle V_1, \ldots, V_n \rangle$ of the vertices in V, with $V_i \cap V_j = \emptyset$ for every $i, j = 1, \ldots, n : i \neq j$ and $V_1 \cup \ldots \cup V_n = V$, such that all vertices in color class $V_j \subseteq V$ are colored with color C_j, for $j = 1, \ldots, n$. A feasible solution to MCCPP using exactly k different colors is called a proper k-coloring.

The trajectory search heuristic TSH+PR considers an evaluation function associated with any (proper or improper) coloring S: $\langle V_1, \ldots, V_n \rangle$ that is given by

$$f(S) \;=\; \sum_{c=1}^{n} (w_c \cdot |V_c| + M \cdot |E(V_c)|), \tag{3.1}$$

where $E(V_c) \subseteq E$ is the subset of conflicting edges with both extremities in color class V_c and M is a sufficiently large penalty. The first term of the evaluation function (3.1) to be minimized accounts for the total cost of the

vertices colored with color C_c, while the second is a penalty cost associated with the conflicting edges connecting vertices in color class V_c. Since $\sum_{c=1}^{n} |E(V_c)| = 0$ in any proper coloring, the penalty M should be large enough to discard the presence of conflicting edges in any solution minimizing $f(S)$.

3.2. Initial solution

A variant of the greedy Recursive Largest First (RLF) algorithm of Leighton [22] is used to build an initial solution for MCCPP. The pseudo-code in Algorithm 1 describes the procedure in detail. The number of color classes and the set of uncolored vertices are initialized in lines 1 and 2, respectively. The outer loop in lines 3–15 is performed for as many times as the resulting number of color classes k in the solution. At any step, the next color class V_k to be built is initialized in lines 4–5. The yet uncolored vertex v' with the largest number of uncolored adjacent vertices is selected from V' in line 6 and placed in color class V_k in line 7. All uncolored vertices adjacent to v' are moved to a set of temporarily uncolored vertices U in line 8. The inner loop in lines 9–13 repeats the above steps to complete color class V_k. The uncolored vertices temporarily kept in U that could not be placed in color class V_k are restored to set V' in line 14 and a new iteration of the outer loop resumes. Algorithm RLF would always return a proper coloring. In order to introduce diversity in the construction of the initial solution, the algorithm is interrupted when there are less than $\lceil RF \times |V| \rceil$ uncolored vertices, where the randomization factor $RF \in (0, 1)$. At this point, the uncolored vertices are randomly assigned to the color classes already created. Typically, the initial solution obtained in line 16 will be an improper k-coloring.

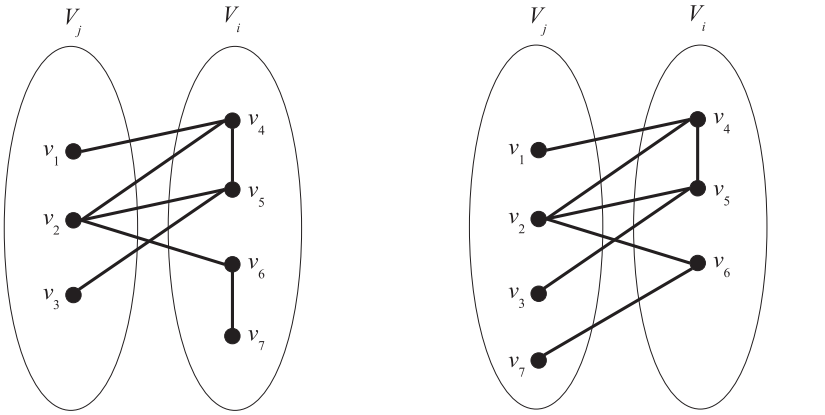
Algorithm 1. *InitialSolution.*

Input: graph $G = (V, E)$
Output: coloring $S : \langle V_1, \ldots, V_n \rangle$
1: $k \leftarrow 0$;
2: $V' \leftarrow V$;
3: **while** $|V'| \geq \lceil RF \times |V| \rceil$ **do**
4: $k \leftarrow k + 1$;
5: $V_k \leftarrow \emptyset$;
6: Select $v' \in V'$ with the largest number of adjacent vertices in V';
7: Move v' from V' to V_k;
8: $U \leftarrow$ all vertices in V' adjacent to v';
9: **while** $V' \neq \emptyset$ **do**
10: Select $v \in V'$ with the largest number of adjacent vertices in U;
11: Move v from V' to V_k;
12: $U \leftarrow U \cup$ all vertices in V' adjacent to v;
13: **end while**
14: $V' \leftarrow U$;
15: **end while**
16: Randomly assign the uncolored vertices in V' to color classes $V_1, \ldots, V_k$;
17: $V_i \leftarrow \emptyset$, $i = k + 1, \ldots, n$;

3.3. Feasibility search

Let S be a k-coloring obtained as the initial solution. We denote by $S(v) = i$ the color class V_i of vertex $v \in V$ and by $N_j(v) \subseteq V_j$ the subset of vertices adjacent to v in color class V_j, for any $j = 1, \ldots, k$: $j \neq S(v)$.

A proper coloring can be obtained from S by the application of a sequence of moves in the *Critical One-Move Neighborhood*, widely used in coloring problems [3,5,6,11]. A neighbor solution is obtained by moving a vertex v involved in a color conflict from its color class V_i to another class V_j, with $1 \leqslant j \leqslant k+1$, $i \neq j$, and $|N_j(v)| = 0$. The vertex selected to be moved is always the one that promotes the largest reduction in the evaluation function (3.1). Furthermore, the selected pair $\langle v, j \rangle$ cannot be classified as a tabu move (as the result of the last sequence



(a) Two conflicting edges in class V_i (b) One conflicting edge in class V_i

FIGURE 3. Vertex v_7 is moved from color class V_i to V_j and one color conflict in class V_i is eliminated.

of perturbation moves, see Sect. 3.7), unless it satisfies an aspiration criterion leading to a neighbor solution that improves upon the best solution found.

The cost of a solution S' obtained by moving vertex v from color class V_i to V_j in solution S can be computed by a straightforward update as

$$f(S') \;=\; f(S) - (w_i + M \cdot |N_i(v)|) + w_j. \tag{3.2}$$

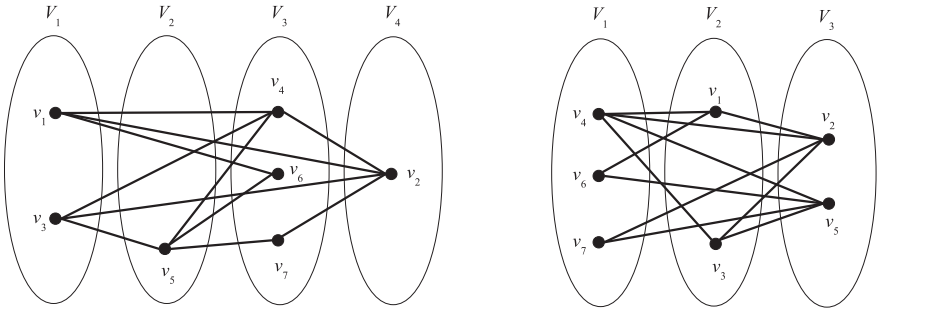
We note that moving a vertex v to color class V_{k+1} corresponds to the creation of a new color class with v as a single vertex. Since one conflicting vertex is moved at each step, the procedure always stops with a proper coloring. Figure 3 illustrates a typical move in this neighborhood.

Algorithm 2 investigates one by one all conflicting vertices in the current improper coloring S and selects the best move in the critical one-move neighborhood. The pseudo-code describes the procedure in detail. S^* denotes the best proper coloring already found by the algorithm. S_{best} denotes the best neighbor of the current solution and is initialized in line 1. The outer loop in lines 2–11 investigates the moves originated by each conflicting vertex in the critical one-move neighborhood. For each conflicting vertex v investigated, the current solution S is temporarily copied to S' in line 3 and the color class $S(v)$ of vertex v in the current solution S is saved to

Algorithm 2. *FeasibilitySearch.*

Input: k-coloring S: $\langle V_1, \ldots, V_n \rangle$, best coloring S^*
Output: coloring S_{best}

```
 1: S_best ← S;
 2: for each conflicting vertex v ∈ V do
 3:     S' ← S;
 4:     i ← S(v);
 5:     for j = 1, ..., k + 1 : |N_j(v)| = 0, j ≠ i do
 6:         S'(v) ← j;
 7:         if (⟨v, j⟩ is not tabu and f(S') < f(S_best)) or f(S') < f(S*) then
 8:             S_best ← S';
 9:         end if
10:     end for
11: end for
```



(a) Proper coloring S with cost 22.5 (b) Proper coloring S' with cost 16.0

FIGURE 4. Solution improvement.

i in line 4. In the inner loop 5–10, the moves of vertex v to each color class $V_j, j = 1, \ldots, k+1$: $j \neq i$, whose vertices are not involved in conflicts with vertex v are evaluated. Vertex v is temporarily moved to color class V_j in neighbor S' in line 6. If (a) the move of vertex v to class color V_j is not forbidden and solution S' is better than the best neighbor already found in the neighborhood of v or (b) solution S' is strictly better than the best solution found S^*, then the best neighbor S_{best} is updated in line 8. The outer loop proceeds until all conflicting vertices are examined once. The procedure returns the best neighbor S_{best}, with at least one color conflict less than in the initial coloring S.

3.4. Solution improvement

After a proper coloring has been obtained, a solution improvement phase attempts to reassign vertices to smaller cost color classes, while preserving feasibility. First, the least cost color is assigned to the largest cardinality color class [10]. At each next iteration, the procedure attempts to reassign vertices of the next largest cardinality color class to a smaller cost color class of the solution under construction that does not create a conflict (or, otherwise, to a new color class with the yet unused least cost color).

The example in Figure 4a displays a proper 4-coloring S with four color classes obtained at the end of the feasibility search phase, where $w_1 = 1.2$, $w_2 = 2.4$, $w_3 = 3.8$, and $w_4 = 6.3$ are, respectively, the costs of vertices in color classes V_1, V_2, V_3, and V_4, with $f(S) = 22.5$. Following the solution improvement phase, the least cost color ($w_1 = 1.2$) is assigned to the largest cardinality color class (formed by v_4, v_6, and v_7) in the new class V_1 of solution S' of Figure 4b. Next, the procedure attempts to move each vertex of the second largest cardinality color class of S (formed by v_1 and v_3) to the previously created class V_1 of S'. However, since both create conflicting edges (v_1 is adjacent to v_4 and v_6, and v_3 is adjacent to v_4), they receive the second least cost ($w_2 = 2.4$) to form the second color class V_2 of solution S'. Since the remaining color classes in S have both one vertex each, the order we consider them is not relevant (v_5 or v_2). We first attempt to reassign v_5 to any of the two classes already created in S'. Since in both cases there will be conflicting edges, a third color class V_3 with $w_3 = 3.8$ is created in S'. Finally, we notice that the last remaining vertex v_2 can also be reassigned to this third color class, since it is not adjacent to v_5. The new solution S' in Figure 4b has cost $f(S') = 16$ and is better than the initial solution S.

The pseudo-code in Algorithm 3 describes in detail the solution improvement procedure that starts from a proper k-coloring S. We denote by V_i and V'_i the color classes of colorings S and S' associated with color C_i, $i = 1, \ldots, n$, respectively. The color classes $V_1, \ldots, V_k$ of solution S are indexed in non-increasing order of their cardinalities in line 1. The loop in lines 2–4 creates initially empty color classes $V'_1, \ldots, V'_n$ of the new solution S' indexed in non-decreasing order of their color costs. Indices j and i are used to visit all color classes of the initial and new solutions, respectively S and S'. The largest cardinality color class of solution S is copied to the least cost color class of solution S' in lines 5 and 6. The loop in lines 7–24 reassigns the vertices of each

Algorithm 3. *SolutionImprovement.*

Input: k-coloring S: $\langle V_1, \ldots, V_n \rangle$
Output: coloring S

```
 1: Sort the color classes of solution S by their cardinalities: |V_i| ≥ |V_{i+1}|, 1 ≤ i < k;
 2: for i = 1, ..., n do
 3:     V'_i ← ∅;
 4: end for
 5: j ← 1; i ← 1;
 6: V'_j ← V_i;
 7: for i = 2, ..., k do
 8:     for each v ∈ V_i do
 9:         ℓ ← 1; find ← false;
10:         while ℓ ≤ j and find = false do
11:             if v is adjacent to any vertex in V'_ℓ then
12:                 ℓ ← ℓ + 1;
13:             else
14:                 find ← true;
15:             end if
16:         end while
17:         if find = false then
18:             j ← j + 1;
19:             V'_j ← v;
20:         else
21:             V'_ℓ ← V'_ℓ ∪ {v};
22:         end if
23:     end for
24: end for
25: if f(S') < f(S) then
26:     S ← S';
27: end if
```

remaining color class V_i, $i = 2, \ldots, k$, of solution S. The loop in lines 8–23 picks each vertex $v \in V_i$. The inner loop 10–16 determines the least cost color class V'_ℓ in S' without any conflict with vertex v. If none is found, then a new color class V'_{j+1} formed exclusively by vertex v is created in lines 18–19, otherwise vertex v is moved to color class V_ℓ in line 21.

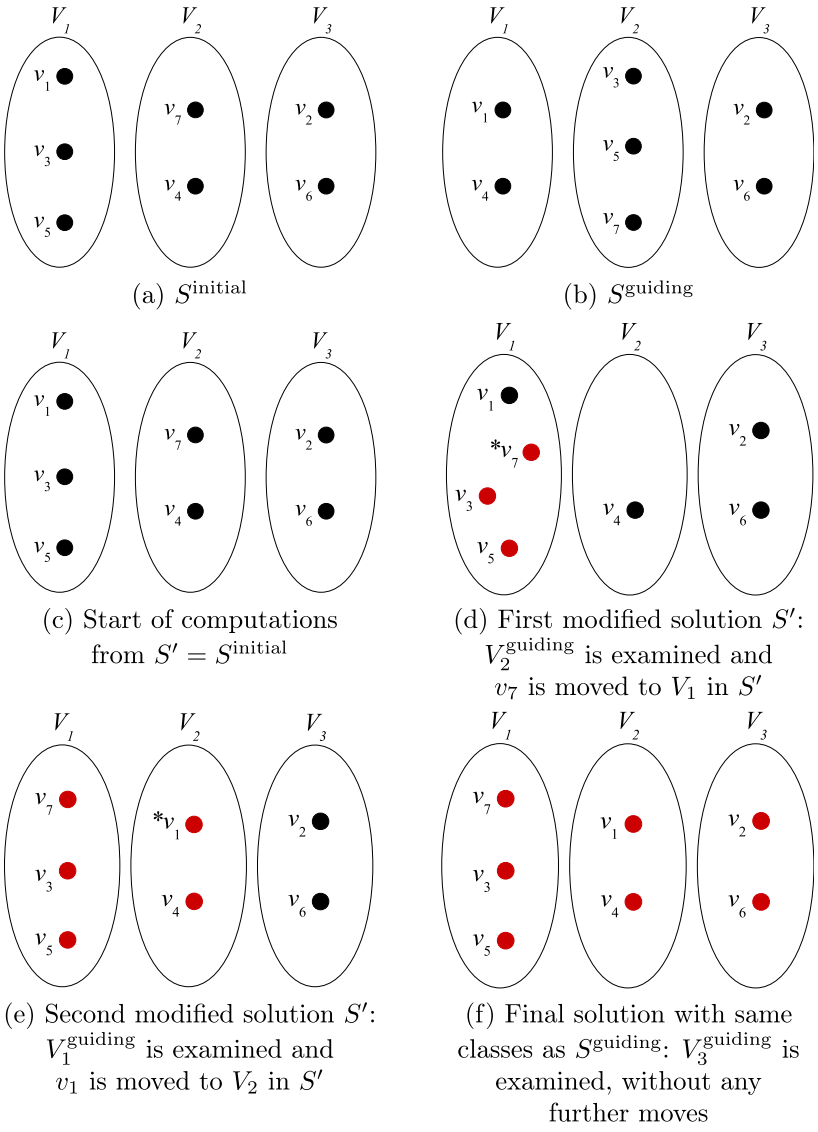
3.5. Path-relinking

Given two feasible solutions for some optimization problem, the move distance between them is defined as the minimum number of elementary transformations necessary to transform one of them into the other. For the case of MCCPP, Galinier and Hao [7] showed that an elementary transformation corresponds to the operation of moving one vertex from a color class to another class. The algorithm outlined by Hamiez and Hao [9] for computing the move distance between two proper colorings is described in Section 3.5.1. This algorithm will be used in Section 3.5.2 as part of the procedure used to perform path-relinking between two proper colorings.

3.5.1. Move Distance

Given two proper colorings S^{initial} and S^{guiding}, the move distance MD between them corresponds to the number of vertices that should be moved from a color class of S^{initial} to another, until a solution with the vertices arranged together as they should be in S^{guiding} is obtained [7,9].

The example in Figure 5 illustrates the computation of the move distance from solution S^{initial} in Figure 5a to S^{guiding} in Figure 5b. The initial solution is set as $S' = S^{\text{initial}}$ in Figure 5c, with $V_1 = \{v_1, v_3, v_5\}$, $V_2 = \{v_4, v_7\}$, and $V_3 = \{v_2, v_6\}$. The color classes of S^{guiding} are examined in the non-increasing order of their cardinalities:



(a) S^{initial} (b) S^{guiding}

(c) Start of computations from $S' = S^{\text{initial}}$

(d) First modified solution S': V_2^{guiding} is examined and v_7 is moved to V_1 in S'

(e) Second modified solution S': V_1^{guiding} is examined and v_1 is moved to V_2 in S'

(f) Final solution with same classes as S^{guiding}: V_3^{guiding} is examined, without any further moves

FIGURE 5. Solutions S^{initial} and S^{guiding} with move distance $MD = 2$ (vertices v_1 and v_7 are moved).

$V_2^{\text{guiding}} = \{v_3, v_5, v_7\}$ is the first to be examined and its vertices will be moved to the first class V_1 of S'. Since vertices v_3 and v_5 are already in V_1, but vertex v_7 is not, then v_7 should be moved to V_1, leading to the first modified solution S' in Figure 5d (the move of vertex v_7 is the first elementary transformation). Since the two remaining classes in S^{guiding} have the same cardinality, we can proceed from any of them. Let $V_1^{\text{guiding}} = \{v_1, v_4\}$ be the next class to be examined. Vertices v_1 and v_4 are in distinct classes of solution S' in Figure 5d. Vertex v_4 cannot be moved to the same class of v_1, because vertices v_3, v_5, and v_7 are already fixed in this class. However, since there are no other vertices in the second class V_2 of S' except v_4, vertex v_1 can be moved to class V_2, resulting in a new, second modified solution S' depicted in Figure 5e (the move of vertex v_1 is the second elementary transformation). The last class of S^{guiding} to be examined is $V_3^{\text{guiding}} = \{v_2, v_6\}$. Since all its

vertices are in class V_3 of solution S', they do not have to be moved. Figure 5f illustrates the final solution S', where the vertices are arranged together as they are in S^{guiding}.

Algorithm 4 describes in detail the procedure that computes the move distance between S^{initial} and S^{guiding}. We denote by V_i^{initial} and V_i^{guiding} the color classes of solutions S^{initial} and S^{guiding}, for $i = 1, \ldots, n$. The initial solution $S' = S^{\text{initial}}$ is set in line 1 and will be progressively modified by elementary transformations, until all vertices are arranged together as they are in S^{guiding}. Set A is initialized in line 2 and contains the vertices that are definitely arranged as they should be in S^{guiding}. Set Δ is initialized in line 3 and contains the vertices that will be effectively moved from a class to another. The color classes in S^{guiding} are indexed in non-increasing order of their cardinalities in line 4. Each of them will be treated in one iteration of the outer loop in lines 5–27. At each loop iteration, the largest still untreated color class V_i^{guiding} will guide the moves of vertices still incorrectly placed in S'. A partition of the vertices in V_i^{guiding} is created in line 6, with each set P_j containing the vertices of $V_i^{\text{guiding}} \cap V_j$. The sets P_j are indexed in non-increasing order of their cardinalities in line 7. A move flag is set in line 8. Each set $P_j, j = 1, \ldots, n$, is analyzed in the inner loop in lines 9–18. In line 10, we check if the vertices in $V_i^{\text{guiding}} \setminus P_j$ can be moved to color class V_j in S'. This will be possible if (a) the vertices in P_j and V_j coincide or (b) the vertices in $V_j \setminus P_j$ are not yet definitely arranged as they should be in S^{guiding}. If this is the case, then the move flag is updated in line 11 and all vertices in $V_i^{\text{guiding}} \setminus P_j$ are moved to class V_j in solution S' in lines 12–14 and inserted into set Δ of moved vertices in line 15. All vertices in V_i^{guiding} are

Algorithm 4. *MoveDistance.*

Input: coloring S^{initial}, coloring S^{guiding}
Output: coloring S', set Δ of vertices moved, move distance MD

1: S': $\langle V_1, \ldots, V_n \rangle \leftarrow S^{\text{initial}}$;
2: $A \leftarrow \emptyset$;
3: $\Delta \leftarrow \emptyset$;
4: Sort color classes of solution S^{guiding} by their cardinalities: $|V_i^{\text{guiding}}| \geq |V_{i+1}^{\text{guiding}}|$, $1 \leq i < n-1$;
5: **for** $i = 1, \ldots, n : V_i^{\text{guiding}} \neq \emptyset$ **do**
6: Create a partition $P_j = V_j \cap V_i^{\text{guiding}}, j = 1, \ldots, n$, of the vertices in V_i^{guiding};
7: Sort the sets P_j by their cardinalities: $|P_j| \geq |P_{j+1}|, 1 \leq j < n-1$;
8: $move \leftarrow$ `false`;
9: **for** $j = 1, \ldots, n : P_j \neq \emptyset$ **while** $move =$ `false` **do**
10: **if** $P_j = V_j$ **or** $(V_j \setminus P_j) \cap A = \emptyset$ **then**
11: $move =$ `true`;
12: **for** each $v \in V_i^{\text{guiding}} \setminus P_j$ **do**
13: $S'(v) \leftarrow j$;
14: **end for**
15: $\Delta \leftarrow \Delta \cup V_i^{\text{guiding}} \setminus P_j$;
16: $A \leftarrow A \cup V_i^{\text{guiding}}$;
17: **end if**
18: **end for**
19: **if** $move =$ `false` **then**
20: $\ell \leftarrow \text{argmin}\{w_j, j = 1, \ldots, n : V_j = \emptyset\}$;
21: **for** each $v \in V_i^{\text{guiding}}$ **do**
22: $S'(v) \leftarrow \ell$;
23: **end for**
24: $\Delta \leftarrow \Delta \cup V_i^{\text{guiding}}$;
25: $A \leftarrow A \cup V_i^{\text{guiding}}$;
26: **end if**
27: **end for**
28: $MD \leftarrow |\Delta|$;

inserted in set A in line 16, since all of them are now arranged in S' as they should be in S^{guiding}. If no move was found, then the vertices in V_i^{guiding} are placed in an empty class in lines 19–26.

To conclude this section, we observe that the move distance $MD = |\Delta|$ computed by Algorithm 4 gives, in fact, an approximation to the number of elementary transformations that are necessary to transform S^{initial} into S^{guiding}. This is so because it computes the exact number of vertices that have to be moved between the color classes of S^{initial} until the vertices of the graph are arranged as they are in S^{guiding}, but not necessarily in the same color classes.

3.5.2. Path-relinking

Path-relinking is an intensification strategy originally proposed by Glover [8] that explores trajectories connecting elite solutions produced by metaheuristics. Path-relinking is usually carried out between two solutions: one is the initial solution, while the other is the guiding solution. A path that connects these solutions is constructed in the search for better solutions that incorporate attributes of the guiding solution into the initial solution. Local search may also be applied to the best solution in the path, since there is no guarantee that this solution is locally optimal. Accounts of successful applications of path-relinking can be seen *e.g.* in [4, 12, 21, 26, 27, 29–32].

Path-relinking between two solutions S^{initial} and S^{guiding} of MCCPP will always be preceded by the computation of the move distance between them by Algorithm 4, which also returns a solution S' with the vertices arranged as they are in S^{guiding} and the set Δ of vertices to be moved between color classes until S' is obtained.

Algorithm 5 describes in detail the pseudo-code of the path-relinking procedure. The current S^{path} and best $\overline{S}$ solutions visited by path-relinking are initialized with S^{initial} in lines 1 and 2, respectively. The outer loop in lines 3–18 is performed while the final solution S' is not reached, *i.e.*, while the move set is not empty. The best vertex still available to be moved is determined in lines 4–12. The inner loop in lines 5–12 investigates all possible vertex moves. In line 7, each available vertex v is tentatively moved to the same color class it occupies in the final solution S'. The best move is updated in lines 8–11. Once the best vertex v^* to be moved in this iteration has been determined, the current solution S^{path} is updated in line 13 and the move set is updated in line 14. If the new current solution improves the best solution $\overline{S}$ found by path-relinking, then $\overline{S}$ is updated in lines 15 and 16.

Algorithm 5. *PathRelinking.*

Input: coloring S^{initial}, coloring S^{guiding}, coloring S', set Δ of moves
Output: coloring $\overline{S}$

```
 1: S^path ← S^initial;
 2: S̄ ← S^initial;
 3: while Δ ≠ ∅ do
 4:     f_min ← ∞;
 5:     for each v ∈ Δ do
 6:         S^temp ← S^path;
 7:         S^temp(v) ← S'(v);
 8:         if f(S^temp) < f_min then
 9:             f_min ← f(S^temp);
10:             v* ← v;
11:         end if
12:     end for
13:     S^path(v*) ← S'(v*);
14:     Δ ← Δ \ {v*};
15:     if f(S^path) < f(S̄) then
16:         S̄ ← S^path;
17:     end if
18: end while
```

3.6. Elite set update

Heuristic TSH+PR makes use of an elite set $\mathcal{E}$ formed by at most *maxElite* high-quality solutions. Whenever a new high-quality solution is found, the elite set is updated. If the elite set is not full, then the new solution is inserted into the elite set $\mathcal{E}$. Otherwise, the new solution replaces the worst solution S_{worst} in $\mathcal{E}$ if it is different from every solution in $\mathcal{E}$ and better than S_{worst}.

Algorithm 6 describes in detail the procedure that updates the elite set after a new high quality solution is found. The worst solution in the elite set is determined in line 1. The flag *equal* is initialized with `false` in line 2. The loop in lines 3–7 determines if solution S is equal to at least one elite solution $S' \in \mathcal{E}$, in which case *equal* is set to `true` in line 5. If solution S is not equal to any solution $S' \in \mathcal{E}$ and is better than the worst, then the elite set is updated in line 9.

Algorithm 6. *EliteSetUpdate.*

Input: coloring S, elite set $\mathcal{E}$
Output: elite set $\mathcal{E}$
1: Let S_{worst} be the worst solution in $\mathcal{E}$;
2: *equal* $\leftarrow$ `false`;
3: **for** each $S' \in \mathcal{E}$ **while** *equal* $=$ `false` **do**
4: **if** $S' = S$ **then**
5: *equal* $\leftarrow$ `true`;
6: **end if**
7: **end for**
8: **if** *equal* $=$ `false` **and** $f(S) < f(S_{\text{worst}})$ **then**
9: $\mathcal{E} \leftarrow (\mathcal{E} \setminus S_{\text{worst}}) \cup S$;
10: **end if**

3.7. Perturbations

The diversification strategy adopted by TSH+PR amounts to creating perturbations after a feasible solution has been obtained and improved by path-relinking. Vertices of a proper k-coloring are randomly selected and reassigned to another color class of the current solution or to a new color class. Moves leading to the reversal of the perturbations should be forbidden and are placed in a one-pass tabu list for the next application of the feasibility search procedure. The number *maxPerturbations* of perturbations is handled as a parameter of the algorithm.

Algorithm 7 describes the pseudo-code of the perturbation procedure. The loop in lines 1–13 applies perturbations to exactly *maxPerturbations* vertices of the initial proper k-coloring. In line 2 a yet unperturbed vertex is selected. The index of the current color C_i of vertex v is saved in line 3. The inner loop in lines 4–10 is repeated until a new color C_j distinct from C_i is determined and assigned to vertex v in line 11. The move associated to reassigning vertex v to color C_i is placed in the one-pass tabu list in line 12.

3.8. Full trajectory search heuristic with path-relinking (TSH+PR)

Algorithm 8 describes heuristic TSH+PR. In line 1, the algorithm sorts the colors in non-decreasing order of their costs. An initial solution S is built in line 2 by Algorithm 1 – *InitialSolution*(G, w, S) – described in Section 3.2. The best solution S^* and its cost $f(S^*)$ are initialized in line 3. The counter of elite solutions and the elite set are initialized in line 4. If the initial solution S is a proper coloring, then the counter of elite solutions and the elite set are updated in line 6. The outer loop in lines 8–34 is performed while a stop criterion is not satisfied. If the current solution S is a proper coloring, then the loop in lines 9–12 is skipped. Otherwise, Algorithm 2 – *FeasibilitySearch*$(S, S^*, S_{\text{best}})$ – presented in Section 3.3 is applied in line 10 to reduce the infeasibility of the current solution S, which is replaced by its neighbor S_{best} in line 11 until it becomes feasible. The tabu list with

Algorithm 7. *Perturbations.*

Input: k-coloring S, number of perturbations *maxPerturbations*
Output: coloring S

```
 1: for i = 1, ..., maxPerturbations do
 2:     v ← random{1, ..., |V| : v not yet selected};
 3:     i ← S(v);
 4:     repeat
 5:         if k < |V| then
 6:             j ← random{1, ..., k + 1};
 7:         else
 8:             j ← random{1, ..., k};
 9:         end if
10:     until j ≠ i;
11:     S(v) ← j;
12:     Insert move ⟨v, i⟩ in the tabu list;
13: end for
```

the forbidden moves is emptied out in line 13. Algorithm 3 – *SolutionImprovement*(S) – presented in Section 3.4 is applied to the current proper coloring S in line 14 as an attempt to improve it.

Line 15 checks if the elite set $\mathcal{E}$ is full. If it is not, then the counter of elite solutions is updated and the locally optimal solution S is simply inserted into the elite set in line 16. Otherwise, solution S is used in line 18 to update the elite set $\mathcal{E}$ using Algorithm 6 – *EliteSetUpdate*$(S, \mathcal{E})$ – described in Section 3.6. Path-relinking is applied in lines 20–29 only if there are at least two or more solutions in the elite set. In this case, an elite solution S_e is randomly selected from the elite set in line 21. S^{initial} is set as the best among S and S_e in line 22, and S^{guiding} takes the other. Algorithm 4 – *MoveDistance*$(S^{\text{initial}}, S^{\text{guiding}}, S', \Delta, MD)$ – presented in Section 3.5.1 is applied in line 23 to compute the move distance MD between S^{initial} and S^{guiding}, as well as the solution S' with the vertex set arranged as in S^{guiding} and the set Δ of vertices that will be effectively moved from a class to another. Resende and Ribeiro [29] showed that path-relinking should be applied between two solutions only if the number of elementary transformations between them is greater than or equal to four. Therefore, line 24 discards the application of path-relinking if the move distance between solutions S^{initial} and S^{guiding} is smaller than four. Otherwise, Algorithm 5 – *PathRelinking*$(S^{\text{initial}}, S^{\text{guiding}}, S', \Delta, \overline{S})$ – described in Section 3.5.2 is applied in line 25, the current solution S is updated in line 26 with the solution $\overline{S}$ obtained by path-relinking and, once again, submitted to Algorithm 3 – *SolutionImprovement*(S) in line 27. The best solution S^* and its cost $f(S^*)$ are updated in lines 30–32 and Algorithm 7 – *Perturbations*$(S, maxPerturbations)$ – described in Section 3.7 is applied in line 33 to generate a new perturbed solution, before a new iteration resumes. The best solution S^* and its cost $f(S^*)$ are returned in lines 35.

4. Computational experiments

Heuristic TSH+PR was implemented in C and compiled with gcc version 5.4.0. An Intel Core i7-4790K processor with a 4 GHz clock and 16 GB of RAM memory running Linux Ubuntu 16.04 LTS 64 bits was used in the computational experiments. We considered 62 benchmark graphs from `http://mat.gsia.cmu.edu/COLOR03` and `http://mat.gsia.cmu.edu/COLOR/instances.html` [18]. We generated integer random color cost values in the interval $[1, 1000]$ associated with $|V|$ colors for each test instance.

4.1. Tuning

We selected ten instances out of the 62 benchmark graphs with color cost values for tuning the main parameters of heuristic TSH+PR, as shown in Table 1. For each instance in this table, we give its number of vertices, its number of edges, and its optimal value obtained by CPLEX 12.8.0 running model (2.1)–(2.4) in Section 2

Algorithm 8. TSH+PR.

Input: graph $G = (V, E)$, color costs w
Output: coloring S^* (proper)

Sort the colors in non-decreasing order of their costs;
InitialSolution(G, w, S);
$S^* \leftarrow S$; $f(S^*) \leftarrow f(S)$;
countElite $\leftarrow 0$; $\mathcal{E} \leftarrow \emptyset$;
if S is a proper coloring **then**
    countElite $\leftarrow 1$; $\mathcal{E} \leftarrow \{S\}$;
end if
while stop criterion is not satisfied **do**
    while S is not a proper coloring **do**
        FeasibilitySearch$(S, S^*, S_{\text{best}})$;
        $S \leftarrow S_{\text{best}}$;
    end while
    Make the tabu list empty;
    SolutionImprovement(S);
    if *countElite* $<$ *maxElite* **then**
        countElite $\leftarrow$ *countElite* $+ 1$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{S\}$;
    else
        EliteSetUpdate$(S, \mathcal{E})$;
    end if
    if *countElite* ≥ 2 **then**
        Select solution $S_e \in \mathcal{E}$ at random;
        $S^{\text{initial}} \leftarrow \text{best}\{S, S_e\}$; $S^{\text{guiding}} \leftarrow \text{worst}\{S, S_e\}$;
        MoveDistance$(S^{\text{initial}}, S^{\text{guiding}}, S', \Delta, MD)$;
        if $MD \geq 4$ **then**
            PathRelinking$(S^{\text{initial}}, S^{\text{guiding}}, S', \Delta, \overline{S})$;
            $S \leftarrow \overline{S}$;
            SolutionImprovement(S);
        end if
    end if
    if $f(S) < f(S^*)$ **then**
        $S^* \leftarrow S$; $f(S^*) \leftarrow f(S)$;
    end if
    Perturbations$(S, maxPerturbations)$;
end while
Return $S^*, f(S^*)$;

with a time limit of 3600 s on a computer with the Intel Xeon E5-2690 v4 processor with a 2.6 GHz clock and 48 GB of RAM. Since CPLEX was not able to prove the optimality of the best solution found for the six last instances in Table 1, for these instances the value in the last column is that of the best known solution (and not necessarily the optimal value).

We considered the tuning of three parameters, corresponding to 36 different configuration settings:

- Three possible values for the randomization factor in Algorithm 1: $RF = 5\%, 10\%, 20\%$;
- Three possible values for the size of the elite set in Algorithm 6: *maxElite* $= 10, 20, 30$; and
- Four possible values for the number of perturbations in Algorithm 7: $maxPerturbations = \sqrt{V}, \log_2 V, V/20, 20$.

First, the algorithm was run for each instance once for each of the 36 parameter configuration settings. The average running time over the 36 runs for each instance was set as the reference time for that instance. Next, each instance was run ten times for each configuration for the reference time (each run starting with a different seed). The best value for each instance was collected after the 360 runs of the instance. For the six last instances in Table 1, this value is also used to generate the entry in the table. Using the optimal or the best known value

TABLE 1. Instances used for tuning the parameters of TSH+PR.

Instance	$\|V\|$	$\|E\|$	Optimal/best value
mug100_25	100	166	3177
games120	120	638	4351
anna	138	493	1137
ash331GPIA	662	4185	1513
DSJC125.1	125	736	1095
DSJC250.1	250	3218	2048
DSJC500.1	500	12 458	6081
2-FullIns_5	852	12 201	1334
wap05	905	43 081	12 760
DSJC1000.1	1000	49 629	10 803

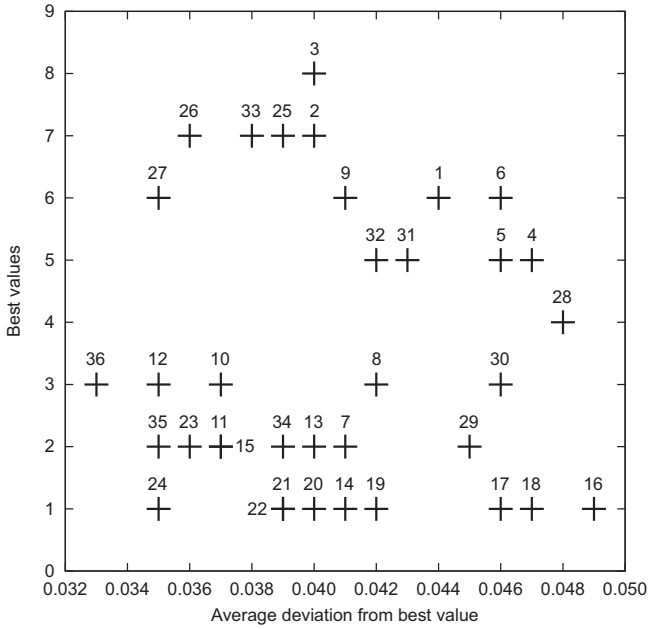


FIGURE 6. Comparative analysis of the 36 parameter configurations of TSH+PR with different settings for the randomization factor, the number of perturbations, and the size of the elite set. The best parameter values led to an average deviation from the best value of 0.04 and to eight matches of the best value (point labeled with 3).

for each instance in Table 1, we computed for each parameter configuration the number of best values (at most ten) found and the average deviation from the best value. Extending the calculations over the ten instances, we computed for each parameter configuration the number of best values (at most 100) found and the average deviation from the best value. For each triple of parameter values, Figure 6 displays a point whose abscissa represents the average deviation over the 100 runs (ten runs for each of the ten instances) of the solution value obtained by the heuristic with respect to the best value in Table 1 and the ordinate corresponds to the number of runs in which this configuration matched the best value. The configurations that present the best results are those with either the smallest abscissas or the largest ordinates, when these two evaluation criteria and

TABLE 2. Results obtained with heuristic TSH+PR – Part I.

				TSH+PR			
Instance	$\|V\|$	$\|E\|$	CPLEX	Best	Average	#Matches	Average time (s)
school1-nsh	352	14 612	13 193	**7647**	7769.20	–	1.36
school1	385	19 095	14 687	**7158**	7274.00	–	1.37
3-FullIns_4	405	3524	**2951**	**2951**	2957.00	4/10	0.24
fpsol2.i.3	425	8688	**3738**	**3738**	3739.10	4/10	1.00
le450_5c	450	9803	**2610**	**2610**	2610.00	10/10	0.40
le450_5d	450	9757	3421	**2707**	2719.50	–	0.47
le450_15c	450	16 680	11 335	**9834**	9952.10	–	4.26
le450_15d	450	16 750	13 011	**11 067**	11 207.80	–	4.08
le450_25a	450	8260	**9730**	9994	10 074.80	–	2.79
le450_25b	450	8263	**7564**	7812	7893.40	–	3.98
le450_25c	450	17 343	11 776	**10 547**	10 813.20	–	3.59
le450_25d	450	17 425	12 791	**11 916**	11 990.40	–	3.24
fpsol2.i.2	451	8691	**4694**	4699	4708.40	–	0.98
4-Insertions_4	475	1795	**999**	1035	1035.00	–	0.20
fpsol2.i.1	496	11 654	**8364**	**8364**	8365.90	1/10	1.05
DSJC500.5	500	62 624	22 845	**19 032**	19 357.50	–	8.76
C500.9	500	112 332	77 015	**64 470**	65 419.50	–	22.05
DSJC500.9	500	112 437	78 869	**67 195**	67 966.90	–	21.10
DSJR500.1	500	3555	**6253**	6510	6580.00	–	3.00
DSJR500.1c	500	121 275	35 554	**27 650**	27 865.90	–	9.89
DSJR500.5	500	58 862	64 892	**55 515**	56 089.30	–	19.80
2-Insertions_5	597	3936	**2999**	3103	3195.10	–	0.64
1-Insertions_6	607	6337	**1367**	1370	1403.30	–	0.75
inithx.i.3	621	13 969	**3633**	**3633**	3641.60	4/10	2.13
inithx.i.2	645	13 979	**4073**	**4073**	4093.90	1/10	2.44
4-FullIns_4	690	6650	**2443**	2455	2460.40	–	0.89

fixed running times (for all configurations) are considered. We selected as the best configuration that with the randomization factor, the number of perturbations, and the size of the elite population set at 5%, $\sqrt{V}$, and 30, respectively, for which the average relative deviation from the best value in Table 1 was 0.04 and the number of matches of the best value was eight (out of a possible maximum of 100) (point labeled 3 in Fig. 6). These parameter settings will be used in the remainder of the computational experiments.

4.2. Numerical results: short runs

Heuristic TSH+PR with the randomization factor set at 5%, the number of perturbations set at $\sqrt{V}$, and the size of the elite population set at 30 was applied to the remaining 52 benchmark instances. Ten independent runs were performed for each instance, with the stopping criterion being 300 iterations without improvement in the best solution found.

Detailed results are presented in Tables 2 and 3. For each instance in these tables, we first give its number of vertices, its number of edges, and the best solution value obtained by CPLEX 12.8.0 running model (2.1)–(2.4) with a time limit of 3600 s on a computer with the Intel Xeon E5-2690 v4 processor with a 2.6 GHz clock and 48 GB of RAM (underlined solution values are proved optimal values, values in boldface are the best among the best solution values found by CPLEX and TSH+PR, and blank cells are those for which no feasible solution was produced in the maximum time limit). The next columns indicate the best and average solution values obtained by TSH+PR over the ten runs for each instance, the number of times the optimal or best solution value obtained by CPLEX was matched, and the average running time of the heuristic in seconds.

TABLE 3. Results obtained with heuristic TSH+PR – Part II.

Instance	$\|V\|$	$\|E\|$	CPLEX	TSH+PR Best	TSH+PR Average	TSH+PR #Matches	TSH+PR Average time (s)
will199GPIA	701	7065	5428	**4918**	5000.70	–	3.93
inithx.i.1	864	18 707	**3934**	**3934**	3936.20	1/10	3.50
qg.order30	900	26 100	**11 940**	11 944	11 948.90	–	5.02
latin_sqr_10	900	307 350	–	**49 672**	50 149.50	–	37.96
wap06	947	43 571	20 434	**18 552**	18 682.10	–	31.78
DSJC1000.5	1000	249 826	–	**48 680**	49 089.80	–	98.15
flat1000_50_0	1000	245 000	–	**43 356**	43 992.00	–	93.12
flat1000_60_0	1000	245 830	–	**42 324**	42 825.00	–	108.24
flat1000_76_0	1000	246 708	–	**42 959**	43 269.10	–	73.07
DSJC1000.9	1000	449 449	–	**109 434**	110 602.60	–	321.28
C1000.9	1000	450 079	–	**112 205**	114 381.90	–	195.18
5-FullIns_4	1085	11 395	**2212**	**2212**	2217.50	2/10	2.28
ash608GPIA	1216	7844	4215	**3940**	3952.20	–	13.78
3-Insertions_5	1406	9695	**1406**	**1406**	1406.00	10/10	2.93
abb313GPIA	1557	65 390	–	**4865**	4912.70	–	59.84
qg.order40	1600	62 400	–	**15 282**	15 283.80	–	31.53
wap07	1809	103 368	–	**13 602**	13 667.70	–	108.42
wap08	1870	104 176	–	**14 581**	14 705.80	–	152.75
ash958GPIA	1916	12 506	3171	**2911**	2934.20	–	50.20
3-FullIns_5	2030	33 751	5845	**4082**	4082.20	–	13.88
wap01	2368	110 871	–	**18 733**	18 937.30	–	197.49
wap02	2464	111 742	–	**17 567**	17 676.30	–	250.30
qg.order60	3600	212 400	–	**35 950**	35 953.00	–	353.94
4-FullIns_5	4146	77 305	–	**6271**	6279.00	–	133.05
wap03	4730	286 722	–	**20 889**	20 970.90	–	1458.34
wap04	5231	294 902	–	**24 025**	24 160.20	–	2162.80

Results in these tables show that heuristic TSH+PR matched or improved the best solution found by CPLEX for 43 out of the 52 instances and found the optimal solution for six of them, in significantly less time than CPLEX.

Run time distributions (or time-to-target plots – or ttt-plots, for short) display on the ordinate axis the probability that an algorithm will find a solution at least as good as a given target value within a given running time, shown on the abscissa axis. Run time distributions have also been advocated in Hoos and Stützle [13] as a way to characterize the running times of stochastic local search algorithms for combinatorial optimization. In this experiment, TSH+PR is made to stop whenever a solution with cost smaller than or equal to a given target value was found. The target is set as the cost of the best known solution for the instance. The heuristic was run 200 times each, with different initial seeds for the pseudo-random number generator. Next, the empirical probability distributions of the time taken by the heuristic to find a target solution value is plotted. To plot the empirical distribution for each heuristic, we followed the methodology proposed by [1, 2]. We associate a probability $p_i = (i - \frac{1}{2})/200$ with the i-th smallest running time t_i and plot the points (t_i, p_i), for $i = 1, \ldots, 200$. The more to the left is a plot, the better is the algorithm corresponding to it.

Figures 7 and 8 display time-to-target plots for instance qg.order30 with targets 11 952 and 11 958 that are, respectively, 0.10% and 0.15% greater than the optimal value listed for this instance in Table 3. We note that the empirical and theoretical distributions coincide for about the first shorter 80% runs, but do not for the longer runs, when the empirical distribution grows slower than the theoretical. Stützle and Hoos [34] observed that

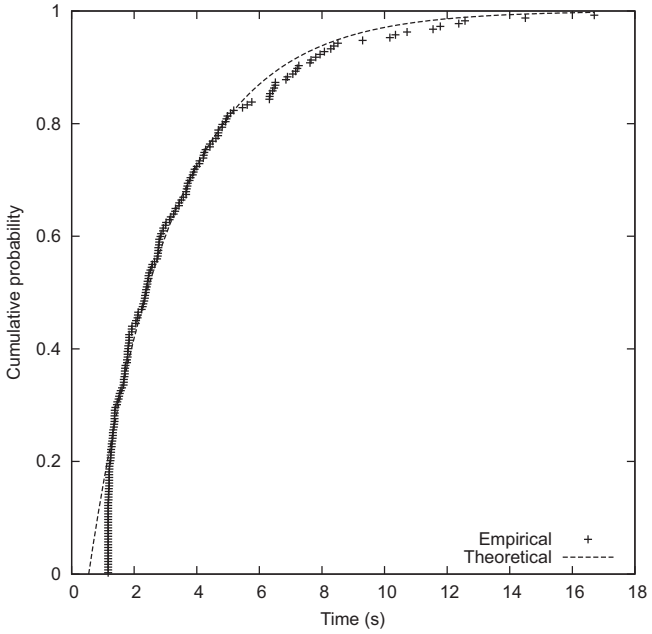


FIGURE 7. Time-to-target plot for instance qg.order30 with target 0.10% greater than the optimal value 11 940.

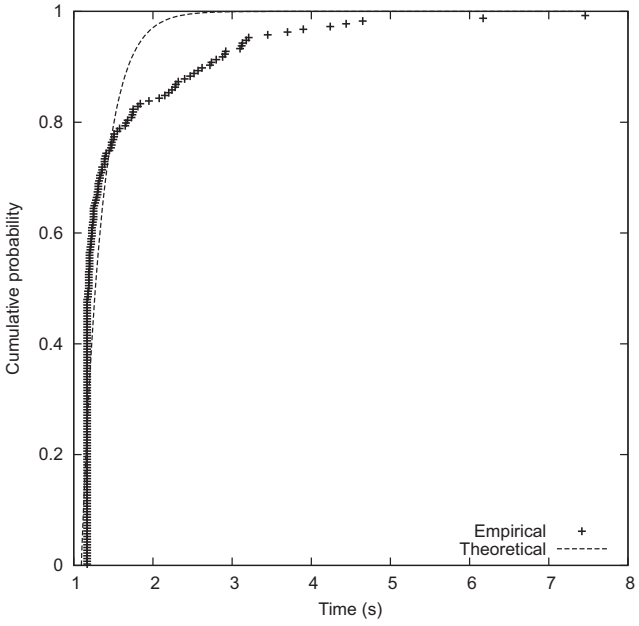


FIGURE 8. Time-to-target plots for instance qg.order30 with target 0.15% greater than the optimal value 11 940.

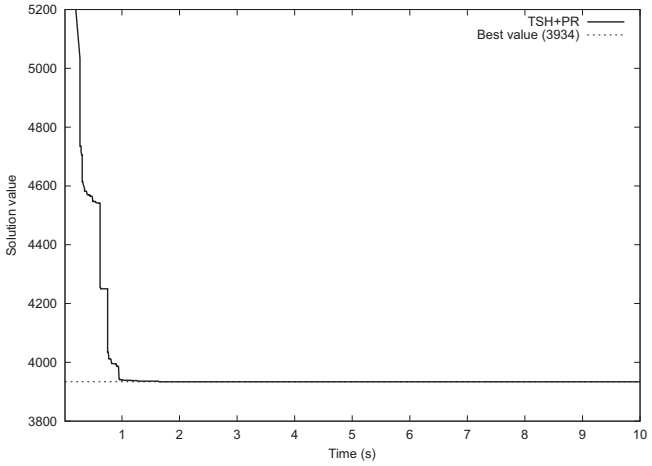


FIGURE 9. Best value *vs.* running time for instance inithx.i.1: TSH+PR converged to the optimal value 3934 in 1.64 s.

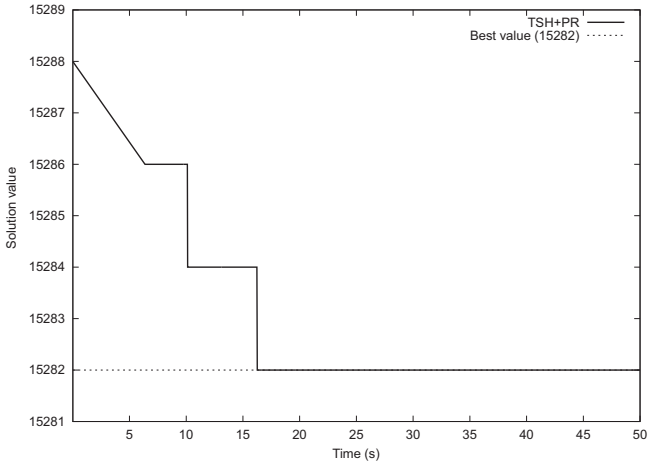


FIGURE 10. Best solution value *vs.* running time for instance qg.order40: TSH+PR converged to the best solution value 15 282 in 16.26 s.

this is an indication of search stagnation, possibly due to the stabilization of the elite set used by path-relinking. This result points out to the need of a restart strategy, as it will be discussed in Section 4.4.

Figures 9 and 10 display two plots showing how the value of the best solution obtained by heuristic TSH+PR evolves with the running time. Figure 9 shows that heuristic TSH+PR converged to the optimal value of instance inithx.i.1 (3934) in less than ten seconds (1.64 s). Figure 10 shows that TSH+PR converged to the best solution value of instance qg.order40 (15 282) in 16.26 s.

4.3. Experiments with path-relinking

In this section, we evaluate the efficiency and the effectiveness of the path-relinking component of heuristic TSH+PR (basically, lines 15–29 of Algorithm 8). In addition to the original version of heuristic TSH+PR using path-relinking, two variants without path-relinking are considered:

- Variant TSH-time contains only the construction, improvement, and perturbation phases. In the numerical experiments, the stopping criterion for each instance is the average time taken by TSH+PR over the ten runs for the same instance.
- Variant TSH-iterations contains, again, only the construction, improvement, and perturbation phases. In the numerical experiments, the stopping criterion is the same of TSH+PR: the heuristic is interrupted after 300 successive iterations without improvement in the best solution found.

Table 4 reports computational results for the three variants (TSH+PR, TSH-time, and TSH-iterations) on ten instances of Table 3 among those with the largest number of vertices. These results show that the full heuristic TSH+PR with path-relinking is the only variant that matched the best solution values for all instances in the table. In addition, TSH+PR systematically obtained the best average solution values over the ten runs. These results illustrate the relevance of the path-relinking component of the heuristic.

4.4. Restart strategies for minimum cost chromatic partition

Resende and Ribeiro [28, 29] showed that restart strategies are able to reduce the running time to reach a target solution value for many problems. We apply the same type of restart(κ) strategy, in which the elite set is emptied out and the heuristic restarted from scratch after κ consecutive iterations have been performed without improvement in the best solution found. We evaluate the performance of the restart strategy for the minimum cost chromatic partition problem for $\kappa = 50, 100, 200$.

Figures 11 and 12 display typical time-to-target plots for instances wap08 (with the target value set to 14855) and 4-FullIns_5 (with the target value set to 6334) for TSH+PR without restarts and TSH with the restart(κ) strategy for $\kappa = 50, 100, 200$. These plots make the case for the use of the restarts strategy, showing that at least one of the improved strategies always finds the target with the same probability systematically faster (in particular, for the long runs).

As already observed in [14, 28], the effect of restart strategies can be mainly noticed in the longest runs. Considering the 200 runs for each of the two instances whose time-to-target plots are depicted in Figures 11 and 12, the long runs are associated with the columns corresponding to the fourth quartile in Tables 5 and 6, respectively. Entries in this quartile correspond to those in the heavy tails of the runtime distributions. The restart strategies in general do not affect too much the other quartiles of the distributions, which is a desirable characteristic. Compared to the no restart strategy, the restart(200) strategy was the one that lead to the most significant reductions in the average running time. It was able not only to reduce the average running time in the fourth quartile, but also in all other quartiles. Strategy restart(200), very close to restart(100), performed the best among the strategies tested, with the smallest average running times over the 200 runs.

Bearing this result in mind, Table 7 recalls, for the same instances in Table 4, the best and average results over ten runs of TSH+PR and presents the best and average results over ten runs of TSH+PR with restarts using the same stopping criterion of TSH+PR without restarts, *i.e.*, stop after 300 iterations without improvement in the best known solution value. The best known solution values are marked in boldface. These results show that strategy restart(100) contributed to further improve the best known solution value for only one instance (4-FullIns_5), while restart(200) recovered the same best solution values found for two instances (abb313GPIA and qg.order40). For all other instances, the strategies with restarts found less quality solutions when compared with the original TSH+PR without restarts heuristic. However, in terms of the running times, the strategies with restarts are consistently faster. This is a clear indication that the restart strategies do contribute to speedup the convergence to a high-quality local optimum, although this convergence might be premature.

Since the stopping criterion considered in the previous experiment seems to lead to premature convergence of the strategies with restarts, we allow them to run for longer. Table 8 recalls the best and average results over ten runs of TSH+PR and presents the best and average results over ten runs of TSH+PR with restarts running for as long as the average time taken by the original heuristic TSH+PR without restarts for every instance in Table 7.

TABLE 4. Comparative results obtained with and without path-relinking.

Instance	TSH+PR				TSH-time (average time)				TSH-iterations (300 iterations)			
	Best	Average	#Matches	Average time (s)	Best	Average	#Matches	Average time (s)	Best	Average	#Matches	Average time (s)
abb313GPIA	**4865**	4912.70	1/10	59.84	4873	4891.50	–	59.85	4894	4936.90	–	23.84
qg.order40	**15 282**	15 283.80	2/10	31.53	15 286	15 291.50	–	31.54	15 286	15 291.50	–	12.41
wap07	**13 602**	13 667.70	1/10	108.42	14 268	14 331.10	–	108.43	14 268	14 331.10	–	14.29
wap08	**14 581**	14 705.80	1/10	152.75	15 281	15 308.80	–	152.77	15 331	15 401.80	–	21.96
wap01	**18 733**	18 937.30	1/10	197.49	19 503	19 569.60	–	197.51	19 503	19 569.60	–	27.48
wap02	**17 567**	17 676.30	1/10	250.30	18 289	18 334.90	–	250.32	18 289	18 334.90	–	29.52
qg.order60	**35 950**	35 953.00	1/10	353.94	35 964	35 967.50	–	353.99	35964	35 967.50	–	126.71
4-FullIns_5	**6271**	6279.00	4/10	133.05	**6271**	6275.00	2/10	133.07	**6271**	6276.60	1/10	86.97
wap03	**20 889**	20 970.90	1/10	1458.34	21 661	21 724.40	–	1458.43	21 661	21 724.40	–	181.11
wap04	**24 025**	24 160.20	1/10	2162.80	24 781	24 800.40	–	2162.88	24 860	24 887.00	–	310.52

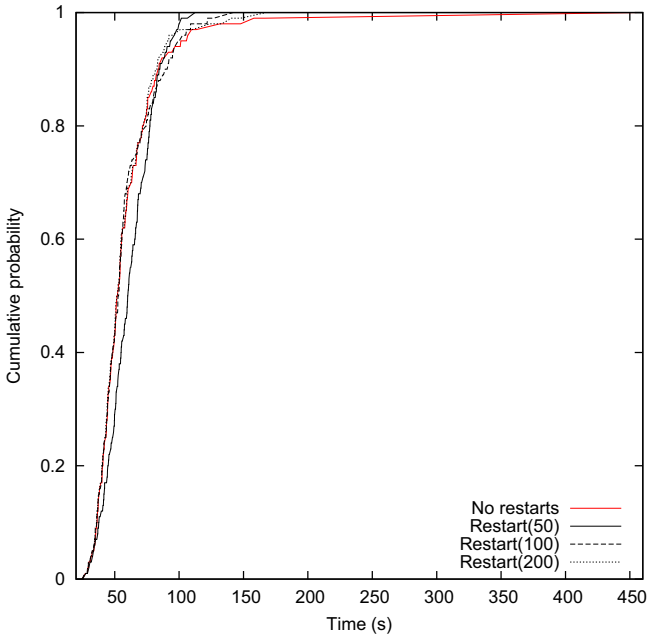


FIGURE 11. Time-to-target plots for instance wap08 with target value set to 14 855.

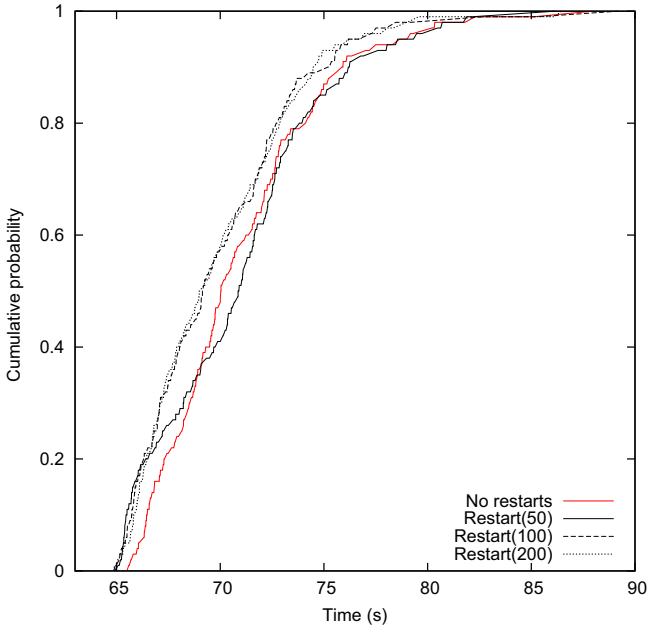


FIGURE 12. Time-to-target plots for instance 4-FullIns_5 with target value set to 6334.

TABLE 5. For instance wap08, 200 independent runs were executed for each strategy.

Strategy	1st	2nd	3rd	4th	Average
No restarts	36.56	47.42	58.07	95.20	59.31
Restart(50)	39.23	54.08	66.82	**85.42**	61.39
Restart(100)	36.35	47.51	**57.02**	87.87	57.19
Restart(200)	**36.26**	**47.07**	57.75	85.64	**56.68**

Notes. Each run was made to stop when a solution as good as the target solution value 14 855 was found. For each quartile, the table gives the average running times in seconds over all runs in that quartile. The average running times over the 200 runs are also given for each strategy.

TABLE 6. For instance 4-FullIns_5, 200 independent runs were executed for each strategy.

Strategy	1st	2nd	3rd	4th	Average
No restarts	66.72	69.14	71.53	76.28	70.92
Restart(50)	**65.85**	69.29	71.97	76.40	70.88
Restart(100)	**65.85**	67.94	70.71	75.05	69.89
Restart(200)	65.97	**67.82**	**70.66**	**74.99**	**69.86**

Notes. Each run was made to stop when a solution as good as the target solution value 6334 was found. For each quartile, the table gives the average running times in seconds over all runs in that quartile. The average running times over the 200 runs are also given for each strategy.

Again, the best known solution values are marked in boldface. These results show that, given the same amount of time, the strategies with restarts become very effective. In particular strategy restart(100) contributed to further improve the best known solution value for one instance (wap08), while restart(50) obtained new best solution values for four instances (wap02, 4-FullIns_5, wap03, and wap04).

4.5. Long runs

Table 9 presents the results for long runs for all instances in Tables 2 and 3 for which the value of the best solution found by the heuristic did not match the optimal value obtained by CPLEX. Also included in this table is one additional very large instance (qg.order100) with 10 000 vertices and 990 000 edges. Two variants of TSH+PR are compared. The strategy without restarts is made to stop after 1000 iterations without improvement in the best solution found. Strategy restart(50) is made to stop after the same running time taken by the strategy without restarts. The best solution value for each instance is marked in boldface. Given the same running time, strategy restart(50) found the best solution for 34 out of the 47 instances and performed much better than the strategy without restarts, which obtained only 21 best solutions.

5. CONCLUDING REMARKS

The minimum cost chromatic partition problem is an extension of the vertex coloring problem. The problem finds applications in VLSI design and in some scheduling problems modeled on interval graphs. We proposed a trajectory search heuristic for the minimum cost chromatic partition problem. Path-relinking and restart strategies were used to improve the efficiency of the trajectory search heuristic. Extensive computational results on a set of 62 benchmark test problems built from graph coloring instances were reported. The instances and best solution values reported in this article may be used in future benchmark studies.

TABLE 7. Restart strategies with $\kappa = 50, 100, 200$: ten runs for each instance.

	No restarts			Restart(50)			Restart(100)			Restart(200)		
Instance	Best	Average	Average time (s)	Best	Average	Average time (s)	Best	Average	Average time (s)	Best	Average	Average time (s)
abb313GPIA	**4865**	4912.70	59.84	4890	4918.00	46.64	4888	4911.70	52.94	**4865**	4907.40	59.69
qg.order40	**15 282**	15 283.80	31.53	15 284	15 288.30	19.82	15 283	15 285.90	25.53	**15 282**	15 284.30	28.61
wap07	**13 602**	13 667.70	108.42	13 754	13 858.80	34.61	13 614	13 742.30	56.08	13 620	13 700.30	81.47
wap08	**14 581**	14 705.80	152.75	14 662	14 834.20	55.36	14 708	14 808.30	75.97	14 631	14 732.90	121.86
wap01	**18 733**	18 937.30	197.49	19 094	19 206.60	54.28	18 952	19 067.80	85.70	18 897	19 020.90	124.84
wap02	**17 567**	17 676.30	250.30	17 863	17 931.10	66.67	17 694	17 825.40	103.33	17 600	17 738.90	174.00
qg.order60	**35 950**	35 953.00	353.94	35 954	35 959.50	199.72	35 954	35 958.00	223.34	35 952	35 955.70	285.10
4-FullIns_5	6271	6279.00	133.05	6271	6277.90	151.01	**6254**	6275.50	145.16	6271	6278.90	137.58
wap03	**20 889**	20 970.90	1458.34	21 066	21 230.70	392.88	20 995	21 117.80	573.61	20 947	21 021.10	999.61
wap04	**24 025**	24 160.20	2162.80	24 281	24 456.80	595.39	24 140	24 301.30	894.35	24 161	24 250.40	1301.69

Notes. Stopping criterion: 300 iterations without improvement in the best value.

TABLE 8. Restart strategies with $\kappa = 50, 100, 200$: ten runs for each instance. Stopping criterion: running time is the average time of TSH+PR.

	No restarts			Restart(50)			Restart(100)			Restart(200)		
Instance	Best	Average	Average time (s)	Best	Average	Average time (s)	Best	Average	Average time (s)	Best	Average	Average time (s)
abb313GPIA	**4865**	4912.70	59.84	4889	4905.10	59.86	4883	4903.80	59.86	**4865**	4903.20	59.86
qg.order40	**15 282**	15 283.80	31.53	15 284	15 288.30	31.55	**15 282**	15 284.90	31.55	**15 282**	15 284.20	31.54
wap07	**13 602**	13 667.70	108.42	13 710	13 812.60	108.44	13 614	13 709.70	108.44	13 620	13 683.90	108.44
wap08	14 581	14 705.80	152.75	14 536	14 635.80	152.77	**14 531**	14 636.90	152.77	14 633	14 715.60	152.77
wap01	**18 733**	18 937.30	197.49	18 837	18 917.00	197.53	18 844	18 980.40	197.53	18 875	18 947.00	197.52
wap02	17 567	17 676.30	250.30	**17 564**	17 667.40	250.33	17 566	17 721.50	250.33	17 600	17 676.40	250.35
qg.order60	**35 950**	35 953.00	353.94	35954	35 959.50	354.07	35 954	35 957.50	354.08	35 952	35 954.90	354.03
4-FullIns_5	6271	6279.00	133.05	**6270**	6277.80	133.09	6271	6278.50	133.09	6271	6279.60	133.11
wap03	20 889	20 970.90	1458.34	**20 827**	20 950.00	1458.49	20 836	20 945.00	1458.47	20 849	20 942.40	1458.54
wap04	24 025	24 160.20	2162.80	**23 846**	23 931.40	2162.92	23 915	23 995.40	2163.03	24 070	24 126.40	2162.94

TABLE 9. Longruns: no restarts and restart(50), one run for each instance. Stopping criterion: 1000 iterations without improvement in the best value for no restarts, same running time as no restarts for restart(50).

			No restarts: 1000 iterations without improvement		Restart(50): same time as no restarts	
Instance	$\|V\|$	$\|E\|$	Solution value	Time (s)	Solution value	Time (s)
school1-nsh	352	14 612	**7681**	1.95	7775	1.95
school1	385	19 095	**7164**	1.90	**7164**	1.90
le450_5c	450	9803	**2610**	1.07	**2610**	1.07
le450_5d	450	9757	**2707**	1.01	**2707**	1.01
le450_15c	450	16 680	9866	11.69	**9563**	11.69
le450_15d	450	16 750	11 069	11.15	**10 665**	11.15
le450_25a	450	8260	**9976**	8.34	10 052	8.34
le450_25b	450	8263	7935	5.45	**7885**	5.45
le450_25c	450	17 343	10 807	11.47	**10 589**	11.47
le450_25d	450	17 425	11 906	15.37	**11 684**	17.12
fpsol2.i.2	451	8691	4699	1.82	**4695**	1.82
4-Insertions_4	475	1795	**1035**	0.61	**1035**	0.61
DSJC500.5	500	62 624	19 329	10.33	**19 169**	10.33
C500.9	500	112 332	64 935	71.05	**63 865**	71.06
DSJC500.9	500	112 437	67 563	111.61	**65 498**	111.62
DSJR500.1	500	3555	**6508**	6.92	6509	6.92
DSJR500.1c	500	121 275	28 050	23.64	**27 658**	23.65
DSJR500.5	500	58 862	**54 840**	82.25	56 850	82.26
2-Insertions_5	597	3936	3124	1.85	**3115**	1.85
1-Insertions_6	607	6337	1413	1.62	**1370**	1.62
4-FullIns_4	690	6650	**2463**	1.98	**2463**	1.98
will199GPIA	701	7065	5011	15.93	**5002**	15.93
qg.order30	900	26 100	**11 944**	16.34	11 948	16.34
latin_sqr_10	900	307 350	**49 318**	121.96	49 383	121.97
wap06	947	43 571	18 535	194.52	**18 425**	194.52
DSJC1000.5	1000	249 826	49 412	236.94	**48 296**	236.97
flat1000_50_0	1000	245 000	43 682	384.67	**42 562**	384.69
flat1000_60_0	1000	245 830	42 373	173.72	**41 614**	173.73
flat1000_76_0	1000	246 708	43 325	116.32	**42 746**	116.34
DSJC1000.9	1000	449 449	111 284	476.95	**108 604**	477.02
C1000.9	1000	450 079	112 708	705.85	**110 222**	705.87
5-FullIns_4	1085	11 395	2219	5.45	**2212**	5.45
ash608GPIA	1216	7844	**3943**	21.19	3947	21.19
3-Insertions_5	1406	9695	**1406**	7.25	**1406**	7.26
abb313GPIA	1557	65 390	**4853**	224.22	4856	224.25
qg.order40	1600	62 400	**15 284**	93.80	15 291	93.82
wap07	1809	103 368	**13 687**	177.20	13 797	177.21
wap08	1870	104 176	14 621	595.14	**14 487**	595.16
ash958GPIA	1916	12 506	**2899**	146.15	2904	156.26
3-FullIns_5	2030	33 751	**4082**	28.15	**4082**	28.15
wap01	2368	110 871	**18 655**	751.14	18 680	751.15
wap02	2464	111 742	17 625	440.12	**17611**	440.17
qg.order60	3600	212 400	**35 951**	777.13	35 960	777.15
4-FullIns_5	4146	77 305	6285	174.78	**6283**	174.81
wap03	4730	286 722	20 879	2907.46	**20 865**	2907.56
wap04	5231	294 902	24 046	4478.23	**23 754**	4478.35
qg.order100	10 000	990 000	**55 601**	4725.44	**55 601**	4727.49

Acknowledgements. Work of Philippe L. Santos was sponsored by a CAPES scholarship. Work of Celso C. Ribeiro was partially supported by CNPq research grants 303958/2015-4 and 425778/2016-9 and by FAPERJ research grant E-26/202.854/2017. This work was also partially sponsored by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), under Finance Code 001.

References

[1] R.M. Aiex, M.G.C. Resende and C.C. Ribeiro, Probability distribution of solution time in GRASP: An experimental investigation. *J. Heurist.* **8** (2002) 343–373.

[2] R.M. Aiex, M.G.C. Resende and C.C. Ribeiro, TTTPLOTS: A Perl program to create time-to-target plots. *Optim. Lett.* **1** (2007) 355–366.

[3] C. Avanthay, A. Hertz and N. Zufferey, A variable neighborhood search for graph coloring. *Eur. J. Oper. Res.* **151** (2003) 379–388.

[4] M.P. Bastos and C.C. Ribeiro, Reactive tabu search with path-relinking for the Steiner problem in graphs, In: Essays and Surveys in Metaheuristics, edited by C.C. Ribeiro and P. Hansen. Springer, Boston (2002) 39–58.

[5] H. Bouziri and M. Jouini, A tabu search approach for the sum coloring problem. *Electron. Notes Discrete Math.* **36** (2010) 915–922.

[6] M. Chiarandini, I. Dumitrescu and T. Stützle, Stochastic local search algorithms for the graph colouring problem, In: Handbook of Approximation Algorithms and Metaheuristics, edited by T.F. Gonzalez. *Computer & Information Science Series*. Chapman & Hall, Boca Raton (2007) 63.1–63.17.

[7] P. Galinier and J.-K. Hao, Hybrid evolutionary algorithms for graph coloring. *J. Comb. Optim.* **3** (1999) 379–397.

[8] F. Glover, Tabu search and adaptive memory programming – Advances, applications and challenges, In: Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies, edited by R.S. Barr, R.V. Helgason and J.L. Kennington. Springer, Boston (1997) 1–75.

[9] J.-P. Hamiez and J.-K. Hao, Scatter search for graph coloring, In: Artificial Evolution, edited by P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton and M. Schoenauer. Vol. 2310 of *Lecture Notes in Computer Science*. Springer, Berlin (2002) 168–179.

[10] A. Helmar and M. Chiarandini, A local search heuristic for chromatic sum, In: Proceedings of the 9th Metaheuristics International Conference, *Udine*, edited by L.D. Gaspero, A. Schaerf and T. Stützle (2011) 161–170.

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

[12] S.C. Ho and M. Gendreau, Path relinking for the vehicle routing problem. *J. Heurist.* **12** (2006) 55–72.

[13] H. Hoos and T. Stützle, Evaluation of Las Vegas algorithms – Pitfalls and remedies, In: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, *Madison*, edited by G. Cooper and S. Moral (1998) 238–245.

[14] R. Interian and C.C. Ribeiro, A GRASP heuristic using path-relinking and restarts for the Steiner traveling salesman problem. *Int. Trans. Oper. Res.* **24** (2017) 1307–1323.

[15] K. Jansen, Complexity results for the optimum cost chromatic partition problem. Universität Trier, Mathematik/Informatik (1996) 96–41.

[16] K. Jansen, The optimum cost chromatic partition problem, In: Algorithms and Complexity, edited by G. Bongiovanni, D. Bovet and G. Di Battista. Vol. 1203 of *Lecture Notes in Computer Science*. Springer, Berlin (1997) 25–36.

[17] K. Jansen, Approximation results for the optimum cost chromatic partition problem. *J. Algorithms* **34** (2000) 54–89.

[18] Y. Jin, J.-P. Hamiez and J.-K. Hao, Algorithms for the minimum sum coloring problem: A review. *Artif. Intell. Rev.* **47** (2017) 367–394.

[19] L.G. Kroon, A. Sen, H. Deng and A. Roy, The optimal cost chromatic partition problem for trees and interval graphs, In: Graph-Theoretic Concepts in Computer Science, edited by F. d'Amore, P. Franciosa and A. Marchetti-Spaccamela. Vol. 1197 of *Lecture Notes in Computer Science*. Springer, Berlin (1997) 279–292.

[20] E. Kubicka and A.J. Schwenk, An introduction to chromatic sums. In *Proceedings of the ACM Seventeenth Annual Computer Science Conference*. ACM Press (1989) 39–45.

[21] X. Lai and J.-K. Hao, Path relinking for the fixed spectrum frequency assignment problem. *Expert Syst. Appl.* **42** (2015) 4755–4767.

[22] F.T. Leighton, A graph coloring algorithm for large scheduling problems. *J. Res. Nat. Bureau Stand.* **84** (1979) 489–506.

[23] E. Malaguti, M. Monaci and P. Toth, Models and heuristic algorithms for a weighted vertex coloring problem. *J. Heurist.* **15** (2009) 503–526.

[24] E. Malaguti and P. Toth, A survey on vertex coloring problems. *Int. Trans. Oper. Res.* **17** (2010) 1–34.

[25] C. Mead and L. Conway, Introduction to VLSI Systems. Addison-Wesley, Boston (1980).

[26] M.G.C. Resende and C.C. Ribeiro, A GRASP with path-relinking for private virtual circuit routing. *Networks* **41** (2003) 104–114.

[27] M.G.C. Resende and C.C. Ribeiro, Greedy randomized adaptive search procedures: Advances and applications, In: Handbook of metaheuristics, edited by M. Gendreau and J.-Y. Potvin, 2nd edition. Springer, New York (2010) 293–319.

[28] M.G.C. Resende and C.C. Ribeiro, Restart strategies for GRASP with path-relinking heuristics. *Optim. Lett.* **5** (2011) 467–478.

[29] M.G.C. Resende and C.C. Ribeiro, Optimization by GRASP: Greedy Randomized Adaptive Search Procedures. Springer, Boston (2016).

[30] M.G.C. Resende, C.C. Ribeiro, F. Glover, and R. Martí, Scatter search and path-relinking: Fundamentals, advances, and applications, In: Handbook of metaheuristics, edited by M. Gendreau and J.-Y. Potvin, 2nd edition. Springer, New York (2010) 87–107.

[31] C.C. Ribeiro and M.G.C. Resende, Path-relinking intensification methods for stochastic local search algorithms. *J. Heurist.* **18** (2012) 193–214.

[32] C.C. Ribeiro and D.S. Vianna, A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *Int. Trans. Oper. Res.* **16** (2009) 641–657.

[33] A. Sen, H. Deng and S. Guha, On a graph partition problem with application to VLSI layout. *Info. Proc. Lett.* **43** (1992) 87–94.

[34] T. Stützle and H.H. Hoos, Analyzing the run-time behaviour of iterated local search for the tsp. In: *III Metaheuristics International Conference*, *Angra dos Reis* (1999) 449–453.

[35] K. Supowit, Finding a maximum planar subset of a set of nets in a channel. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **6** (1987) 93–94.