Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 346 (2019) 379–391

www.elsevier.com/locate/entcs

A GRASP for the Convex Recoloring Problem in Graphs

Ana Paula dos Santos Dantas[a,1,2] Cid Carvalho de Souza[a,1,2] Zanoni Dias[a,1,2]

[a] *Institute of Computing, University of Campinas (Unicamp), Brazil*

Abstract

In this paper, we consider a coloring as a function that assigns a color to a vertex, regardless of the color of its neighbors. The Convex Recoloring Problem finds the minimum number of recolored vertices needed to turn a coloring convex, that is, every set formed by all the vertices with the same color induces a connected subgraph. The problem is most commonly studied considering trees due to its origins in the study of phylogenetic trees, but in this paper, we focus on general graphs and propose a GRASP heuristic to solve the problem. We present computational experiments for our heuristic and compare it to an Integer Linear Programming model from the literature. In these experiments, the GRASP algorithm recolored a similar number of vertices than the model from the literature, and used considerably less time. We also introduce a set of benchmark instances for the problem.

Keywords: Convex Recoloring, GRASP, Combinatorial Optimization.

1 Introduction

The convex recoloring problem has its origins in the study of phylogenetic trees. A phylogenetic tree reconstructs the evolutionary history of a set of species that have common characteristics. The leaves of these trees are known organisms, and the remaining vertices are hypothetical ancestors. An important feature of such trees is that similar organisms are expected to appear close to each other [9].

The characteristics common to the species represented in a phylogenetic tree can manifest in different states, with each species manifesting only one of the possible states. By representing each state with a distinct color, we can build a coloring

[1] This work was supported by grants from *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (304727/2014-8, 400487/2016-0, 425340/2016-3, and 140466/2018-5), *Fundação de Amparo à Pesquisa do Estado de São Paulo* (2013/08293-7, 2015/11937-9, 2017/12646-3, 2017/16246-0, and 2018/04760-3), *Coordenação de Aperfeiçoamento do Pessoal do Ensino Superior* and the CAPES/COFECUB program (831/15).

[2] Email: ana.dantas@students.ic.unicamp.br, cid@ic.unicamp.br, zanoni@ic.unicamp.br

https://doi.org/10.1016/j.entcs.2019.08.034

of the phylogenetic tree. As similar organisms are expected to stay close in such trees, equally colored vertices are also supposed to be close to each other in the tree. In graph terms, the "closeness" requirement corresponds to constraining that the vertices with the same color induce a single subtree.

Unfortunately, many phylogenetic trees do not have this property, because of errors either in the construction of the tree or in the classification of the organisms [8]. Considering the case where the tree's structure is reliable, Moran and Snir [12] defined the term *recoloring distance*, which is the minimal number of vertices from a phylogenetic tree that need to change colors so that the tree is ideal, i.e., all similar organisms are close to each other. The Convex Recoloring Problem aims to find the recoloring distance.

The problem is mainly studied on trees, due to its origin. Moran and Snir [12] proved that the version on trees is $\mathbb{NP}$-hard. Chopra *et al.* [4] presented an integer programming model to the convex recoloring on trees that has good computational results, compared to the previous approaches. The best approximation factor known for the problem on trees is $2 + \epsilon$, presented by Bar-Yehuda *et al.* [1].

Many versions of the problem were considered by modifying: (i) the class of graphs on which the recoloring is applied, such as general graphs and paths; or (ii) how the coloring is applied. One example of this situation is discussed by Kammer and Tholey [10], who investigated the problem for general graphs that map computer networks, adding restrictions on which vertices can be recolored and which can be only uncolored.

In this paper, we focus on the convex recoloring problem on general graphs, without any restrictions on the recoloring. This version of the problem is also proved to be $\mathbb{NP}$-hard [13]. To our knowledge, only exact algorithms have been proposed to solve this problem so far, but no computational results were reported.

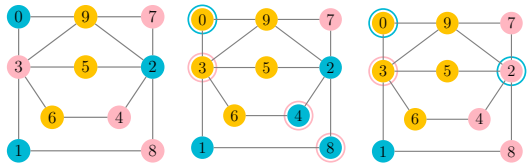
Our contribution is an algorithm based on the meta-heuristics GRASP for this variation. Besides presenting the method in detail, we also define a class of instances for the problem that can be used as a benchmark.

The remaining of this paper is organized as follows. In Section 2, we give basic definitions for the convex recoloring problem that are used throughout the text. In Section 3, we discuss the methods currently available for the problem. In Section 4, we detail the algorithm proposed. In Section 5, we report on our experiments and discuss the results. Lastly, in Section 6, we draw our conclusions and point out future research directions.

2 Definitions

A graph G is defined as an ordered pair (V, E), with V as the set of vertices and $E \subseteq V \times V$ as the set of edges. We say that two vertices u and v are adjacent, or neighbors, if the edge (uv) is in E. The subgraph *induced* by $S \subseteq V$ is the graph whose vertex set is S, and whose edges are all $(uv) \in E$ with $u, v \in S$.

Let us represent a color by an integer number and the absence of color by the symbol $\emptyset$. A *coloring* of a graph $G = (V, E)$ is a function $C : V \to \mathcal{C}$ that assigns a



(a) initial coloring (b) convex recoloring (c) optimal recoloring

Fig. 1. Example of instance. Figure 1(a) shows a colored graph. Figures 1(b) and 1(c) show possible recolorings.

color to the vertices of G, where $\mathcal{C}$ is the set of colors. A *partial coloring* $C : V \to \mathcal{C} \cup \{\emptyset\}$ of G is a coloring function that allows for a vertex to be uncolored, that is, no color is assigned to it. The *color class* c is the set of all vertices that have the color $c \in \mathcal{C}$. A color is said to be *convex* if its color class induces a connected subgraph, otherwise the color is called *bad*. A coloring is convex if every color is convex. Given the graph G and a coloring of it, (G, C) is dubbed a *colored graph*. A coloring C' of a colored graph (G, C) is a *recoloring* of G. Given a colored graph (G, C) and a recoloring C' of G, $R_C(C')$ is the subset of colored vertices of (G, C) that changed colors in the recoloring, i.e., $R_C(C') = \{v \in V \mid C(v) \neq \emptyset \text{ and } C(v) \neq C'(v)\}$. Note that, although uncolored vertices in (G, C) are not in $R_C(C')$, the set may contain vertices that are colored by C but not by C'. Now, let $w : V \to \mathbb{Q}^+$ be a function that assigns costs to recoloring vertices in V. The convex recoloring problem is defined as follows.

Definition 2.1 Convex Recoloring Problem *(*crp*)*
INPUT: *a graph G, a color set $\mathcal{C}$, a partial coloring C, and a cost function w;*
OUTPUT: *a convex recoloring C' with minimum cost, that is,* $\min(\sum_{v \in R_C(C')} w(v))$.

Figure 1 shows an instance of the convex recoloring problem. Assume that all recoloring costs are one. Figure 1(a) shows a colored graph with three colors. The recoloring in Figure 1(b) recolors four vertices ($\{0, 3, 4, 8\}$). The recoloring in Figure 1(c) is optimal and recolors only three vertices ($\{0, 2, 3\}$).

3 Integer Linear Programming (ilp) Model

Campêlo *et al.* [3] model the crp as an integer program having an exponential number of constraints. Moura [14] also gives an ilp formulation for the problem, but with an exponential number of variables and suggests a column generation algorithm to solve the model restricted to the case of trees. Computational experiments with both formulations focused on phylogenetic trees. Since the pricing subproblem by Moura [14] does not deal with general graphs, our experiments only involve the formulation by Campêlo *et al.* [3].

Campêlo's model is given by Equations 1 to 4. In this formulation, the constant $w_{v,c}$ is equal to $w(v)$, if $C(v) = c$, and 0, otherwise. The unweighted version of the crp requires that $w(v) = 1$ for all $v \in V$. The binary variable $x_{v,c}$ is set to 1 if and only if c is the color assigned to vertex v. To understand the model, recall that, for a pair (u, v) of non adjacent vertices, a (u, v) *vertex-cut* is a set of vertices Z that,

when removed from G, leaves u and v in separate components. If $\Gamma(u, v)$ is the set of all minimal (u, v) vertex-cuts in G, the model reads:

$$\max \quad \sum_{v \in V} \sum_{c \in \mathcal{C}} w_{v,c} x_{v,c} \tag{1}$$

$$s.t. \quad \sum_{c \in \mathcal{C}} x_{v,c} \leq 1 \qquad \forall v \in V \tag{2}$$

$$x_{u,c} + x_{v,c} - \sum_{z \in Z} x_{z,c} \leq 1 \; \forall uv \notin E, \; Z \in \Gamma(u, v), \; c \in \mathcal{C} \tag{3}$$

$$x_{v,c} \in \{0, 1\} \qquad \forall v \in V, \; \forall c \in \mathcal{C} \tag{4}$$

Equation 1 defines the objective function as maximizing the number of vertices that keep their initial color, which is equivalent to minimizing the number of vertices that switch colors. Equation 2 forces each vertex to receive at most one color. Equation 3 guarantees that any two non adjacent vertices of the same color are connected. Equation 4 restricts the values of the decision variable $x_{v,c}$ to be binary. Campêlo *et al.* [3] presents a proof of correctness to this formulation, as well as a polyhedral study. We implemented the model and use it to assess the results with our proposed heuristic.

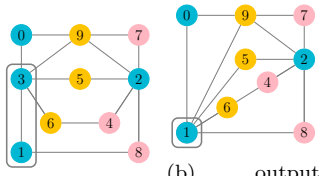
4 A Greedy Randomized Adaptive Solution Procedure

In this section, we propose an algorithm to solve the CRP in general graphs based on the Greedy Randomized Adaptive Search Procedure (GRASP) [7]. Each iteration of the procedure consists of two phases that are executed in sequence. The first phase builds a feasible solution while the second one applies a local search to this solution. The iterations repeat until a stopping condition, usually the total of iterations, is reached and the best solution found is returned.

The construction algorithm in the initial phase is also iterative, and has three key components. The first is the greedy component: at each iteration, a partial solution is available and the candidate elements to be added to it are put in a list, that is sorted according to a greedy criterion which measures how good an element is compared to the other possible ones. The second key component is the randomization: instead of picking the candidate at the top of the previously sorted list, a *Restricted Candidate List* (RCL) is formed with only a percentage of the best candidates from which one is chosen randomly. The third key component is the adaptivity: each iteration is affected by the previous ones, since each choice modifies the state of the solution.

We chose this meta-heuristic because of its simplicity and the small number of tuning parameters, namely, the percentage of best candidates considered for selection and the stopping condition. Another advantage of GRASP is its potential to combine it with other techniques, such as Variable Neighborhood Search and parallelization [15]. Next, we describe the main ingredients of our GRASP to solve the CRP.

4.1 Greedy Randomized Adaptive Solution. A key operation in our procedure



(a) input graph (b) output graph

Fig. 2. Shrink procedure. Figure 2(a) the graph before the operation and Figure 2(b) shows the new graph.

to build a solution is the *shrinking* of a vertex. The *shrinking* of vertex u into a neighbor vertex v removes u from the graph and adds its adjacency to that of v. In this case, v is said to represent u. Figure 2 illustrates the operation, where vertex 3 is shrunk into vertex 1. Before the shrinking (Figure 2(a)), vertices $\{0, 1, 5, 6, 9\}$ were adjacent to vertex 3 and vertices $\{3, 8\}$ were adjacent to vertex 1. The resulting graph is seen in Figure 2(b), where vertex 3 is removed and vertex 1 becomes adjacent, not only to its old neighbors, but also to those of vertex 3.

The construction phase is detailed in Algorithm 1. The greedy selection recolors the vertices aiming to obtain large monochromatic components. The algorithm operates on a colored graph (H, D), that is initialized with the original graph G and its coloring C. Next, every monochromatic component of H is shrunk into a single vertex. Then, the following steps are repeated until no bad colors remain in H: vertices are sorted according to a greedy criterion, a vertex v is chosen among a fraction of the best candidates following the greedy criterion, the color of v is switched and, finally, its neighbors with the same color are shrunk into it. The criteria we use to select the next candidate, to sort the candidate list, and to recolor the chosen vertices are detailed below. After a solution is constructed, all vertices from G are either in H or represented by a vertex of H. To extract a recoloring of G, we assign the color of a vertex v from H to its copy in G and to the vertices v represents. We envisaged two alternative criteria to select, sort, and recolor candidate vertices. One is called *ratio* while the other is named *union*.

The *ratio* criterion considers as a candidate every vertex $v \in H$ such that $D(v) \neq \emptyset$ and is based on two values: *weight* and *ratio*. The *weight* of a vertex is given by the number of vertices it represents. The *ratio* of a vertex is the frequency of its neighborhood's most common color divided by the neighborhood's size. The sorting of the candidates orders the vertices in ascending order of their weights, with ties broken by the descending order of their ratio. Accordingly, the first vertex of the sorted list of candidates is the one with the smallest weight and, in case of a tie, the one with largest ratio. After the RCL is created, a vertex v is selected and its new color is chosen based on the frequency of its neighbors' color. If the most frequent color appears more than once, then v receives this color. If no two neighbors of v have the same color, but there is a convex color in its adjacency, v receives this color; otherwise, v is uncolored. Figure 3 exemplifies the *ratio* criterion.

Considering the initial coloring of Figure 3(a), every vertex has weight 1, since no shrinking was made. The ratio is calculated based on the same topology. For instance, vertex 3 has five adjacent vertices, three of which have the same color, so, its ratio is 3/5. Figure 3(b) shows the vertices sorted using the *ratio* criterion.

Algorithm 1. GRASP Construction Phase

Construction(G, C)
  let D be a copy of C ▷ coloring backup
  let H be a copy of G ▷ graph backup
  shrink all adjacent vertices of H with equal colors
  while there is a bad color in D **do**
    $L \leftarrow$ sorted list of candidates vertices from H
    $RCL \leftarrow$ percentage of best candidates in L
      select a v at random from RCL
    change the color of v in D
    shrink adjacent vertices of v that have
      the same color into v
  end while
  $C' \leftarrow$ expand D to a coloring of G
  return C'

Alg. 1. GRASP construction phase.

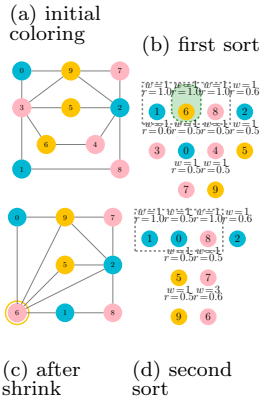


Fig. 3. Construction phase using the *ratio* criterion.

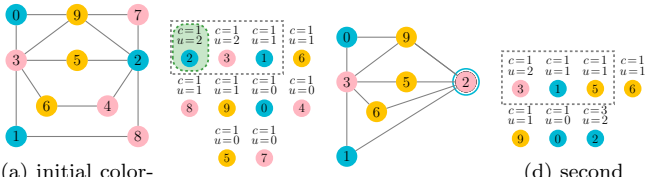


Fig. 4. Construction phase using the *union* criterion.

Above each vertex, we have their respective values of weight (w) and ratio (r). Suppose we consider the 30% best candidates (those contoured with dashed lines in Figure 3(b)) to pick one at random. Also, suppose that vertex 6 (dashed and shaded in Figure 3(b)) is selected to be recolored. Vertex 6 is recolored with the most frequent color on its neighborhood, i.e., the color of vertices 3 and 4. Next, vertices 3 and 4 are shrunk into 6, as shown in Figure 3(c). Figure 3(d) shows how this recoloring affects the sorting of the next iteration of the construction procedure. Note that the weight of vertex 6 is 3, since now it represents vertices 3, 4 and itself.

The *union* criterion considers as a candidate every vertex whose color is not convex or has at least one neighbor whose color is not convex. Again, the sorting process relies on two values. The first is the *cost of recoloring* a vertex given by the number of vertices it represents and that were not previously recolored. The second one is the *union factor* of a vertex v, which is the maximum number of vertices added to a color component if v is assigned to this same color. This quantity can be computed as follows. Given a color c, the factor of c relative to v is the sum of the weights of the neighbors of v with color c excluding the largest of them. The union factor of v is defined as the maximum factor of a color relative to v. If c is the color associated to this value, the new color of v becomes c. The *union* criterion prioritizes the vertices with smaller recoloring costs, with ties broken following the descending order of the union factors.

Figure 4 explains the use of the *union* criterion. Figure 4(a) depicts the initial coloring of the graph. In this case, every vertex represents only itself and no vertex was recolored yet, so, each vertex has cost one. The union factors are calculated

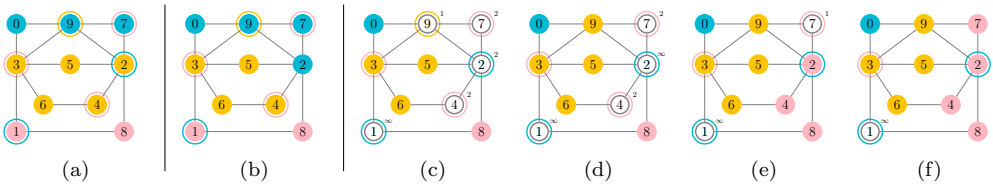


Fig. 5. Simple (Figure 5(b)) and extended (Figures 5(c), 5(d), 5(e) and 5(f)) neighborhood of the recoloring from Figure 5(a).

as describe above. To exemplify one such calculation, consider vertex 3. It has five neighbors, three of the same color and two of another. Since the weight of each vertex is 1, the factor of the yellow color is $1+1+1-1=2$ and of the blue color is $1+1-1=1$. The union factor of vertex 3 is the largest of the two, and, if chosen to be recolored, it will receive the yellow color. Figure 4(b) shows the sorting of the vertices. Once again we consider the top 30% as best. Suppose now, that vertex 2 is randomly selected. It will be recolored as red and its red-colored neighbors will be shrunken into it. Figure 4(c) shows the resulting graph. On the next sorting phase, Figure 4(d), vertex 2 will have weight 4, since, besides itself, it represents the vertices $\{4, 7, 8\}$, but the cost of recoloring will be only 3, because vertex 2 was already recolored.

4.2 Local Search. The local search routine is executed after each solution is constructed to explore a neighborhood of the solution in search for a local maximum. We propose two neighborhoods for the CRP: the *simple* and the *extended* neighborhood. These neighborhoods are detailed below. The basic idea is to revert the color change of a vertex, i.e., to find a vertex that was recolored and have it returned to its original color. The difference between *simple* and *extended* is how they find such vertices.

Given a convex recoloring of a colored graph, a *simple* neighborhood is constructed by searching for a vertex that was recolored and that can be removed from its new color class without disconnecting it. If such vertex v is found, then it must be verified if v is adjacent to a vertex colored with v's original color so that it can be recolored with its original color. The local search procedure with the simple neighborhood is repeated until no more vertices can have their color changed back. Figures 5(a) and 5(b) show an example of this neighborhood.

An outlined vertex in Figure 5(a) means that it was recolored. The color of the outline points out the original color. Vertex 2 was recolored and, if removed from its current color class, will not disconnect it. Also, 2 is adjacent to a vertex with its original color, hence, it can be reverted. Note that we could have chosen 3 as well. Figure 5(b) exhibits the new recoloring. As no more vertices can be reverted, the procedure ends with a recoloring one unit cheaper than the starting one.

The *extended* neighborhood is constructed by systematically removing the color of all vertices that were recolored and can be removed from their new class without disconnecting them. After removing the color of every possible vertex, we iterate over the uncolored vertices and find a path to its original color. This path must contain only uncolored vertices. Then, we choose the vertex with the smallest path to be reverted, recoloring also the path. This process of finding paths and connecting

is repeated until there are no vertices that can be connected to its original color. Figures 5(c)–5(f) show an example of application of this neighborhood.

Considering the colored graph from Figure 5(a) as input, Figure 5(c) is the result of the uncoloring phase of the *extended* local search, where white vertices are uncolored. Above each uncolored vertex v in this figure is given the smallest distance from v to a vertex having its original color. Note that, for vertex 1, this distance is ∞, as there is no path made only of uncolored vertices joining 1 to a blue vertex. Since vertex 9 has the smallest such distance, its color is changed back, and the new paths are calculated as shown in Figure 5(d). The next smallest distance is 2, and either the color of vertex 4 or 7 is changed back. Suppose we revert the color of vertex 4. Figure 5(e) depicts the result of this operation. Lastly, vertex 7 is reverted to its original color and the final recoloring in Figure 5(f) is obtained, with a cost reduced by 3 units.

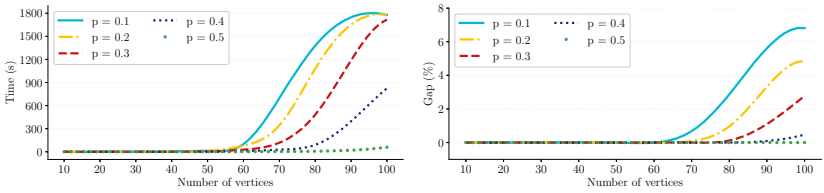
5 Computational Results

This section reports the results of our computational experiments with the ILP model and the GRASP algorithm.

5.1 Integer Linear Programming Model. First, we describe the implementation details of the formulation proposed by Campêlo *et al.* [3] (ILP) and present results of our computational tests. The model was coded in the programming language C++, compiled using g++ (version 5.4), with flag C++11, and solved by IBM's CPLEX (version 12.8). The experiments were carried out on an Intel$^{\text{R}}$ Core$^{\text{TM}}$ i7 3.40GHz machine with 32GB of memory running Ubuntu 18.04 LTS. We set a time limit of 30 minutes for the solver, with presolve and multithread settings turned off.

As the formulation has an exponential number of restrictions, we implemented an approach that relaxes the restrictions on Equation 3 and add them later as Lazy Constraints. To identify a violated constraint, we build a directed graph D from the input graph G, such that, for each vertex $v \in V(G)$, we add two vertices v^+ and v^- to $V(D)$ connected by an arc (v^-, v^+), and for each edge $e = uv \in E(G)$, we add the arcs (u^+, v^-) and (v^+, u^-). For each color c, we assign weights to the arcs of D and execute a min-cut/max-flow algorithm using the implementation from the Library for Efficient Modeling and Optimization in Networks (LEMON) [6]. For every arc of the form (w^-, w^+), the weight is the value of $x_{w,c}$, and for every other arc the weight is 1. If the maximum flow from u^+ to v^- is less than 1, the minimum cut arcs form the vertex-cut set.

Since no reports on experimental results with general graphs were available in the literature, we seek to generate hard instances for the CRP. In preliminary studies, we noted that the number of bad colors greatly influences the difficulty of an instance. With that in mind, we tried to find extreme cases of bad color classes and noticed that a proper coloring, i.e., one in which no two equal-colored vertices are adjacent, fulfills this requirement. So, we created 100 Erdos-Renyi connected random graphs for each $n \in \{10, 20, \ldots, 100\}$ and



(a) average time (b) average gap

Fig. 6. Average values obtained after executing the ILP model.

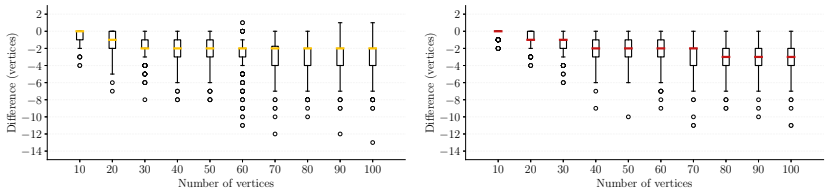
$p \in \{0.1, 0.2, \ldots, 0.5\}$, with n being the number of vertices of the graph and p the probability of adding an edge, hence, the expected density of the graph. For each of these graphs, we produced a proper coloring based on the greedy algorithm introduced in Bondy and Murty [2]. Starting with such coloring, we apply a balancing procedure that moves vertices from the largest to the smallest color class, whilst maintaining the proper coloring property. Our benchmark is available at `www.loco.ic.unicamp.br/files/instances/convex-recoloring`.

Figures 6(a) and 6(b) show the average time in seconds used to solve the 100 instances of each pair (n, p) and the average optimality gap, respectively. In each of these figures, the x axis exhibits the number of vertices n and each line represents a different density p. Figure 6(a) shows that, as we increase the density of the graph, the instances become easier for the model, as they are solved before the time limit. We also observed a decrease in the totals of recolored vertices and lazy constraints. When an instance finishes executing before the time limit, it means that an optimal solution was found for it and the gap is zero. Conversely, instances that hit the time limit have positive gap. Figure 6(b) displays the average gaps for all instance sizes.

To check that the previous instances were indeed hard due to the use of proper colorings, we created similar instances but with different colorings. These instances had the same number of colors, the same distribution of vertices per color class and every class was bad. We created instances for $n \in \{10, 20, \ldots, 100\}$ and $p \in \{0.1, 0.2\}$, values for which we could assure the aforementioned properties. The model solved all instances to optimality before the time limit, required less recolorings and the addition of less lazy constraints.

5.2 GRASP. Section 4.1 discusses two sets of criteria to build a solution. This section analyzes the results of the tests with both of these sets. They were implemented using the programming language C++, and the compiler flags C++11 and -O3, compiler g++ (version 5.4). The experiments reported below were run on the same machine as those with the ILP model and use the same instances.

There are two GRASP parameters that need to be tuned prior to execute the experiments. The first is the stop condition. For this version, we used $2n^2$ iterations, with n being the number of vertices of the input. The second parameter is the percentage to be considered in the random selection of the candidates, and for this we used the R package Iterated Race Automatic Algorithm Configuration (irace) [11]. To setup the irace, we randomly selected 20% of the instances, and divided them into train (80%) and test (20%) sets. The density of the graphs were equally dis-



(a) *ratio* (b) *union*

Fig. 7. Difference on the number of recolored vertices after adding randomization for each criteria.

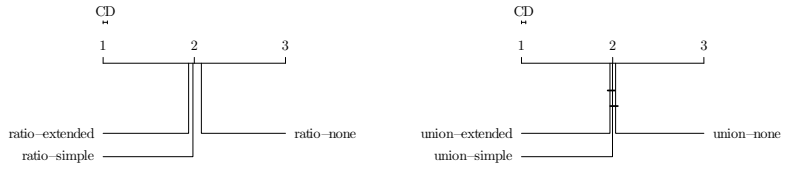
tributed in this sample. The values returned by irace and used in the experiments were 13.954% for the *ratio* criterion and 10.229% for the *union* criterion.

Our first experiment was to compare each criteria with their purely greedy version, that chooses the first element of the sorted list, instead of a random one from the RCL. Our goal with this experiment is to show that it is better to use a randomized solution. We executed a GRASP version without any local search and the purely greedy algorithm. Figures 7(a) and 7(b) show the difference in number of recolored vertices after adding the randomized part of GRASP for the *ratio* and *union* criteria, respectively. The x axis of the graphs from Figure 7 shows the number of vertices of the input graph, and the boxes indicate the difference in the number of recolored vertices after adding the randomization. All densities are represented together in these graphs.

For the *ratio* criterion, Figure 7(a) shows that there were a few instances where the greedy was better than the randomized version, that is, the greedy solution recolored less vertices. On the graph, these are the instances with value greater than zero. The graph indicates that it is better to use randomization. To confirm that there is a significant difference, we performed the Wilcoxon's signed test. Our null hypothesis is that there is no statistically significant difference between the two approaches. Our z statistic value for the test was -59.83, considering a confidence level of 0.95, we must have $z < -1.96$ to reject the null hypothesis [5]. Therefore, we can reject the null hypothesis. For the *union* criterion, Figure 7(b) shows that, for every instance, the randomized version was at least as good as the purely greedy version. We also performed the Wilcoxon's signed test, and found a z statistic of -59.06, also rejecting the null hypothesis.

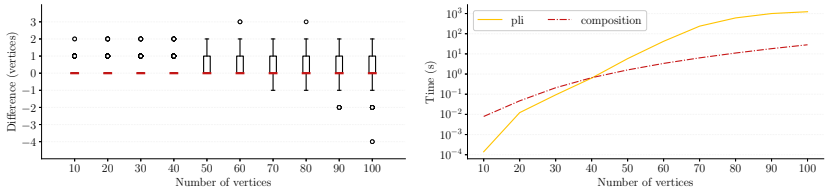
Next, we executed experiments to measure the impact of using the local search, at each GRASP iteration, to improve the solution for both criteria. We executed each criterion with each local search neighborhood defined in Section 4.2. Since we have now three approaches (no local search, simple local search and extended local search), we used the Iman-Davenport test to compare these approaches. The Iman-Davenport statistic, F_F, follows the F-distribution with $(k-1)$ and $(k-1)(N-1)$ degrees of freedom, with N as the number of instances and k as the number of approaches. The null hypothesis is the same from the previous tests.

With 3 approaches and 5000 instances, and 0.95 confidence level we need $F_F > 3.00$. We compared the solutions of the *ratio* criterion using this test and found $F_F = 25.44$, and therefore we reject the null hypothesis. Using the same test for solutions to the *union* criterion, we found $F_F = 4.81$ and, again, reject the null



(a) *ratio* (b) *union*

Fig. 8. Critical Difference (CD) for the different local search neighborhoods for each criteria.



(a) difference on recolored vertices (b) time

Fig. 9. Comparison of the best GRASP approaches with the ILP model.

hypothesis. So, we conclude that local search has a significant impact on the quality of the solutions.

Next, we executed the Nemenyi post-hoc test with each criterion. The graphical representation of these tests are shown in Figures 8(a) and 8(b). The Nemenyi test measures the Critical Difference (CD) between approaches using the average rank. In these graphs, if there is no significant difference between two approaches, the vertical lines representing these approaches are connected by a horizontal line segment. These figures show that, for the *ratio* criterion, there are no equivalent approaches. For the *union* criterion, the one that uses the extended neighborhood and the one that does not use any local search have a significant difference.

Also from the Nemenyi test, we can extract the information that the approaches with best average ranks are the ones that use the extended neighborhood for local search. We now investigate if a composition of the best approaches from each criteria yields a better result. This composition executes all $2n^2$ GRASP iterations of each approach in sequence. To answer this, we first counted how many times one was better than the other and we found that *union* was the best in approximately 40.0% of the instances and *ratio* in 4.5% , the remaining instances were ties. We then applied Iman-Davenport statistical test comparing *ratio* criteria, *union* criteria and the composition. We can reject the null hypothesis if $F_F > 3.00$. Since we found $F_F = 576.83$, we have that executing both criteria is better than using just one or the other. Executing the post-hoc Nemenyi test, we found that no two of these approaches are equivalent, and the composition has the best average rank.

Our final experiment was to compare the solutions found by the best GRASP approach with the solutions found by the integer linear programming model. Figures 9(a) and 9(b) compare the solution values and execution times, respectively. Figure 9(a) reports the number of recolored vertices by the GRASP relative to the ILP model. The optimal solution is known for all instances with $n \leq 60$, in such cases the GRASP recolored on average 3.21% more vertices than the optimal solution. For instances with $n > 60$, the GRASP recolored on average 2.23% more vertices than

the solution returned by the ILP model. Figure 9(b) depicts the runtime for the GRASP and the model. Note that the scale of the y axis is logarithmic. We can see that, as the number of vertices grows, the model's runtime increases rapidly, whilst that of GRASP has a much slower growth.

6 Conclusions

In this paper, we propose a GRASP to solve the convex recoloring problem on general graphs. Two sets of greedy criteria are considered to create solutions. The experiments showed that a combination of the two performs better than each one individually. We also implement the integer programming model presented by Campêlo *et al.* [3], and use it for comparison purposes. We found that GRASP yields solutions that recolor about the same number of vertices as Campêlo's model, using only a fraction of the time. So far, our implementation includes just the basic steps of GRASP, and since it has already shown a promising performance, we intend to continue improving it with new techniques such as path relinking. We also aim to adapt the heuristic to tackle different recoloring problems such as the Restricted Recoloring Problem [10].

References

[1] Bar-Yehuda, R., I. Feldman and D. Rawitz, *Improved Approximation Algorithm for Convex Recoloring of Trees*, Theory Comput. Syst. **43** (2008), pp. 3–18.

[2] Bondy, J. A. and U. S. R. Murty, "Graph Theory," Graduate Texts in Mathematics **244**, Springer, London, 2011, 1st edition.

[3] Campêlo, M. B., A. S. Freire, K. R. Lima, P. F. S. Moura and Y. Wakabayashi, *The Convex Recoloring Problem: Polyhedra, Facets and Computational Experiments*, Math. Program. **156** (2016), pp. 303–330.

[4] Chopra, S., B. Filipecki, K. Lee, M. Ryu, S. Shim and M. V. Vyve, *An Extended Formulation of the Convex Recoloring Problem on a Tree*, Math. Program. **165** (2017), pp. 529–548.

[5] Demšar, J., *Statistical comparisons of classifiers over multiple data sets*, J. Mach. Learn. Res. **7** (2006), pp. 1–30.

[6] Dezös, B., A. Jüttner and P. Kovács, *LEMON - an Open Source C++ Graph Template Library*, Electron. Notes Theor. Comput. Sci. **264** (2011), pp. 23 – 45.

[7] Feo, T. A. and M. G. C. Resende, *Greedy Randomized Adaptive Search Procedures*, J. Global Optim. **6** (1995), pp. 109–133.

[8] Frenkel, Z., Y. Kiat, I. Izhaki and S. Snir, *Convex Recoloring as an Evolutionary Marker*, Mol. Phylogenetics Evol. **107** (2017), pp. 209–220.

[9] Goldberg, L. A., P. W. Goldberg, C. A. Phillips, E. Sweedyk and T. Warnow, *Minimizing phylogenetic number to find good evolutionary trees*, in: *Combinatorial Pattern Matching* (1995), pp. 102–127.

[10] Kammer, F. and T. Tholey, *The Complexity of Minimum Convex Coloring*, Discrete Appl. Math. **160** (2012), pp. 810–833.

[11] López-Ibáñez, M., J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle and M. Birattari, *The irace package: Iterated racing for automatic algorithm configuration*, Oper. Res. Perspect. **3** (2016), pp. 43–58.

[12] Moran, S. and S. Snir, *Convex Recolorings of Strings and Trees: Definitions, Hardness Results and Algorithms*, J. Comput. Syst. Sci. **74** (2008), pp. 850–869.

[13] Moran, S., S. Snir and W.-K. Sung, *Partial Convex Recolorings of Trees and Galled Networks: Tight Upper and Lower Bounds*, ACM Trans. Algorithms **7** (2011), p. 42.

[14] Moura, P. F. S., "Graph Colorings and Digraph Subdivisions," Ph.D. thesis, Instituto de Matemática e Estatística, Universidade de São Paulo (2017).

[15] Resende, M. G. and C. C. Ribeiro, "Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications," International Series in Operations Research & Management Science, Springer US, 2010, 2nd edition pp. 283–319.