

Algorithms for Super-coloring in Directed Graphs

Wesly Carmesini Ataide

Programa de Pós-Graduação em Ciência da Computação
Universidade Federal de Santa Catarina (UFSC)
Cx.P. 476 – 88040-370 Florianópolis – SC – Brasil
wesly.ataide@posgrad.ufsc.br

Rafael de Santiago

Programa de Pós-Graduação em Ciência da Computação,
Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
(UFSC)
Cx.P. 476 – 88040-370 Florianópolis – SC – Brasil
r.santiago@ufsc.br

Álvaro Junio Pereira Franco

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
Cx.P. 476 – 88040-370 Florianópolis – SC – Brasil
alvaro.junio@ufsc.br

ABSTRACT

Given a directed graph with colored vertices, for each vertex v , we want to know the largest number of colors that we can find in the vertices of a single path starting from v . Unlike other classical problems in colored graphs, there is no constraint that adjacent vertices must have different colors. Among the applications in literature, there is interest in Anthropology to investigate interaction between individuals from families and/or clans. This article seeks to design and analyze computational methods and some properties for this problem, particularly in directed acyclic graphs (DAG). As contributions, we constructed two algorithms and an integer programming model for this problem, and experimented them on real instances of genealogical data.

KEYWORDS. Graphs. Graph Coloring. Integer Linear Programming.

Paper topics: TAG, PM, OC

1. Introduction

Graph coloring has found applications in many different areas. One important example for us is in anthropology, where part of studying a community is analyzing the kin relationships between people, whether by consanguinity or affinity (Dal Poz et al. [2009]). Marriages in communities not only create new kin relationships, they can also be determined by pre-existing relationships: different societies may prefer or avoid marriages between the same clan or family, prefer or avoid marriage patterns of other family members, or have a specific set of conventions to follow. Part of an ethnography is to document and describe genealogy networks, according to Hamberger et al. [2004].

A particular property of each individual and the relationship (parental or affinity) between them can be represented with vertex-colored graphs, with each vertex representing an individual, each edge or arc representing a relation (which may be marriage or a parent-child relationship), and each color representing a particular property of an individual, for example, a *clan*. A question regarding the number of clans in a lineage may be then interpreted as: given a directed graph $G = (V, A)$ and a vertex coloring, what is the largest number of colors that a path starting from v can reach, for each $v \in G$?

Although there are numerous works on proper coloring problems, this specific modeling of the problem is, to the best of the author's knowledge, less explored in the literature. There are works that deal with paths in specific graphs in specific colored graphs, such as Monnot [2005], which studies paths in bipartite graphs, and works that focus on a family of very similar problems, in particular on naming labels instead of colors Hassin et al. [2007]. We mainly expand on the work of Franco and Vendramin [2021], that presents the problem treated here and proposes an algorithm for directed acyclic graphs.

We have designed and implemented two algorithms, one for the general case, in any directed graph, and one for a limited amount of colors in directed acyclic graphs. In addition, we created an integer programming model considering directed acyclic graphs. These three methods were tested, and we used them to find solutions for graphs based on real genealogical data.

In the next sections, we describe the problem, discuss the algorithms and model that we designed, and present our datasets and experiments.

2. Problem and Definitions

Let $G = (V, A)$ be a directed graph with an associated color function $V \rightarrow \{1, \dots, k\}$. A path is a sequence of vertices $\langle p_1, \dots, p_l \rangle$ where each vertex is unique, and for each consecutive pair p_i, p_{i+1} , there is an arc $(p_i, p_{i+1}) \in A$. We refer to the number of vertices in a path as its size. If G has no cycles, we say it is a DAG (directed acyclic graph).

The super-coloring value of a vertex $v \in V$ (denoted by $super(v)$) is the maximum number of colors a path starting in v can reach. We also denote the color associated with a vertex v as $color(v)$. The main problem that we discuss in this article is: for a directed graph G , what is $super(v)$ for each vertex v ? This problem is NP-hard, both for general directed graphs (as discussed in Franco and Vendramin [2021]) and for DAGs, from a reduction based on the tropical path problem, as discussed in Cohen et al. [2021].

3. Algorithm for General Directed Graphs

One approach we can use to find the super-coloring values in general graphs is by analyzing all paths from each vertex. We implement this idea with the help of sets containing structures that represent paths. For each vertex u , we maintain a set S_u , which contains the paths starting from the vertex u . Each of the paths is represented by its initial vertex, its final vertex, the set of colors of the vertices in the path, and the set of vertices

in the path. The algorithm is based on using these structures to extend the paths, until all of them are explored. If P is a path from v to w , and there is an arc (u, v) , it is trivial to create a path P' , from u to w , and that includes, possibly, one more color, the one of the vertex u . This is possible, as long as u is not already part of P . Our idea is to apply this extending process iteratively to analyze all paths in the graph, from ones made by single vertices to the longest ones in the graph. Our proposal is described in Algorithm 1. Its correctness is based on two results, that we prove in Lemmas 3.1 and 3.2.

Inputs : A colored directed graph $G = (V, A)$.

Output: A vector *super* containing the super-coloring value for each vertex v .

```

1 FindSuperGeneral ( $G$ )
   /* Each path  $P = \langle u, \dots, v \rangle$  is represented by its initial vertex,
   final vertex, set of colors and the set of vertices.      */
2 struct {
3   | (initial_vertex, final_vertex, colors, vertices) ;      /* a 4-tuple */
4 } Path;
5 new_iteration  $\leftarrow \{\}$ ;
6 foreach vertex  $u \in V$  do
7   | new_path  $\leftarrow \text{Path}(u, u, \{\text{color}(u)\}, \{u\})$  ;
8   |  $S_u \leftarrow \text{new\_path}$ ;
9   | new_iteration  $\leftarrow \text{new\_iteration} \cup \{\text{new\_path}\}$  ;
10 end
11 found_path  $\leftarrow \text{True}$ ;
12 while found_path = True do
13   | found_path  $\leftarrow \text{False}$ ;
14   | last_iteration, new_iteration  $\leftarrow \text{new\_iteration}, \{\}$ ;
15   | foreach arc  $(i, j) \in A$  do
16     | foreach path  $P \in \text{last\_iteration}$  do
17       |  $u, v, C, Q \leftarrow$ 
18         |  $P.\text{initial\_vertex}, P.\text{final\_vertex}, P.\text{colors}, P.\text{vertices}$ ;
19       | if  $j = u$  AND  $i \notin Q$  then
20         |  $\text{new\_path} \leftarrow \text{Path}(i, v, C \cup \{\text{color}(i)\}, Q \cup \{i\})$ ;
21         |  $S_i \leftarrow S_i \cup \{\text{new\_path}\}$  ;
22         | found_path  $\leftarrow \text{True}$ ;
23         | new_iteration  $\leftarrow \text{new\_iteration} \cup \{\text{new\_path}\}$  ;
24       | end
25     | end
26   | end
27   | foreach vertex  $v$  do
28     |  $\text{super}(v) \leftarrow \max_{P_v \in S_v} (\{|P_v.\text{colors}|\})$ .
29   | end
30 return super
  
```

Algorithm 1: Algorithm for finding the super-coloring value in directed graph.

Lemma 3.1. In l -th iteration of Algorithm 1, in line 12, all simple paths of size $l + 1$ are

added to the S sets.

Proof. We proceed by induction.

(Base step): Let $P = \langle u, v \rangle$ be a path of size 2 that exists in the graph. Naturally, there is an arc (u, v) with $u \neq v$. Before the first iteration of the loop, the *new_iteration* contain only the paths of individual vertices, including $P' = \langle v \rangle$, a path of size 1. This set of paths will be saved in *last_iteration*. Since there is an arc (u, v) , this path P' will be considered, and since v is the start of P' , and $u \notin \{v\}$, P will be added to S_u .

(Inductive step): Suppose that the theorem is valid for iteration $l - 1$, that is, all paths of size l were added in the last iteration. Let $P = \langle p_0 = u, p_1 = v, p_2, \dots, p_l = w \rangle$ be a path of size $l + 1$ from vertex u to vertex w . By our hypothesis, $P' = \langle v, p_2, \dots, p_l = w \rangle$, of size l , has just been added, and therefore we will consider it for extension. Furthermore, (u, v) is an arc, and u is distinct from all vertices in P' , or P would not be a path. This way, the path P is added to S_u . $\square$

Lemma 3.2. *The loop in line 12 of Algorithm 1 runs for d iterations, with d being the length of the longest path in number of vertices of the input graph.*

Proof. Suppose that the loop runs for $r < d$ iterations, and that $P = \langle p_1, \dots, p_r, \dots, p_d \rangle$ is a simple path with the maximum length in number of vertices in the input graph. By Lemma 3.1, all paths of size $r + 1$ would have been added last. Since our algorithm has stopped, there are no paths of this length. But P has, at least, $r + 1$ vertices, and, therefore, it is possible to construct a path with exactly $r + 1$ vertices using part of vertices of P , which is a contradiction.

By Lemma 3.1, in the d -th iteration, all paths of size $d + 1$ were added to the S sets. Since d is the size of the longest path in the input graph, there are no paths with size $d + 1$, and so, the algorithm must stop on the d -th iteration. $\square$

Theorem 3.3. *The Algorithm 1 finds $\text{super}(v)$ for each vertex v .*

Proof. Let $P = \langle v = p_1, p_2, \dots, p_q \rangle$ be a path with $\text{super}(v)$ colors. If the path P was found at some point in the algorithm, there is a corresponding structure in S_v , which includes a set of colors with $\text{super}(v)$ colors. This value is the maximum, as a path that had a larger set of colors would imply a larger value for $\text{super}(v)$. If P was found, therefore, the value of $\text{super}(v)$ would be obtained correctly. So, if the algorithm returned a wrong value for $\text{super}(v)$, the path P was not found. According to our Lemma 3.1, this implies the algorithm performed a maximum of $q - 2$ iterations. But the path P exists and has size $q > q - 2$, which is a contradiction with Lemma 3.2. Thus, P was found, and is part of S_v , which leads the algorithm to return the value $\text{super}(v)$ correctly. $\square$

Although Algorithm 1 is fairly simple in concept, the fact that it has to search and store all paths in the graph leads to a very large space and time complexity. In the worst case (with a complete undirected graph), it may have to explore and store $\mathcal{O}(n!)$ paths.

4. Algorithm for k Colors

For a directed graph $G = (V, A)$, $G' = (V, A')$ is the *transitive closure* of G if there is an arc $(u, v) \in A'$ if and only if there is a directed path from u to v in G . For a fixed number of colors, it is possible to solve the problem for DAGs using the transitive graph closure and an additional procedure. This way, determining $\text{super}(v)$ is reduced to running a modified depth-limited search using v as the root. This idea is described in the Algorithms 2 and 3.

Inputs : A DAG D , with vertices colored with k colors.

```

1 FindSuperKColors ( $D$ )
2    $D' \leftarrow$  transitive closure of  $D$ ;
3    $k \leftarrow$  number of colors in  $D$ ;
4   foreach  $v \in D$  do
5      $super(v) \leftarrow FindSuperKColorsAux(D', v, k, \{\});$ 
6   end
7   return  $super$ 

```

Algorithm 2: Main algorithm for finding super-coloring values for DAGs with k colors

Inputs : A connected DAG D , a vertex v , a limit value $limit$, and a set of visited colors S .

```

1 FindSuperKColorsAux ( $D, v, limit, S$ )
2    $S' \leftarrow S \cup \{color(v)\};$ 
3   if  $limit = 1$  then
4     return  $|S'|$ ;
5   end
6    $super \leftarrow |S'|$ ;
7   foreach vertex  $u$  such that  $(v, u)$  is an arc in  $D$  do
8      $super_u \leftarrow FindSuperKColorsAux(D, u, limit - 1, S');$ 
9     if  $super_u > super$  then
10       $super \leftarrow super_u$ ;
11    end
12  end
13  return  $super$ 

```

Algorithm 3: Auxiliary algorithm for k colors in DAGs.

The complexity of this strategy can be analyzed in two parts. First, it depends on the complexity of the chosen algorithm for the transitive closure. This may be $\mathcal{O}(n^3)$ if the Floyd-Warshall algorithm is used (Cormen et al. [2009]), or, if a matrix multiplication algorithm is used, it approaches $\tilde{\mathcal{O}}(n^w)$, where w comes from complexity of the multiplication algorithm. The most recent bound, obtained in Williams et al. [2024], is $w \leq 2.371552$.

Secondly, the complexity depends on the depth-first search, which searches all paths of size k . It should be noted that this applies to the transitive closure of the original graph, and so, there may be many more paths, leading to a higher complexity, although it is exponential in either case.

We proceed to prove the correctness of Algorithm 2. Two definitions are necessary. Firstly, for a DAG $D = (V, A)$, a vertex $w \in V$ and a value l , let $\mathcal{P}_w^l$ be the set of all paths in D starting in w with at most l vertices. Second, if there are two paths $P_1 = \langle u_1, \dots, u_q \rangle$ and $P_2 = \langle v_1, \dots, v_s \rangle$, with all vertices being distinct, and there is an arc (u_q, v_1) , the concatenation of P_1 with P_2 (denoted $P_1 \cdot P_2$) is the path $\langle u_1, \dots, u_q, v_1, \dots, v_s \rangle$.

Theorem 4.1. *If v is a vertex in a DAG D , l a positive integer value, and $u_1, \dots, u_q$ are all the vertices such that there is an arc (v, u_i) for all i where $1 \leq i \leq q$, then*

$$\mathcal{P}_v^{l+1} = \{\langle v \rangle\} \cup \bigcup_{1 \leq i \leq q} \{\langle v \rangle \cdot P \text{ for all } P \in \mathcal{P}_{u_i}^l\}.$$

Proof. For any path $P = \langle u_i, \dots, w \rangle \in P_{u_i}^l$, it may be extended through the arc (v, u_i) . This is possible, because if v is already part of P , that would imply a cycle $\langle v, u_i, \dots, v \rangle$, which is a contradiction. All of these extensions start at v , and have size of $l + 1$, at the most, so they are in P_v^{l+1} . With this, we have that $\{\langle v \rangle\} \cup \bigcup_{1 \leq i \leq q} \{\langle v \rangle \cdot P \text{ for all } P \in \mathcal{P}_{u_i}^l\} \subseteq \mathcal{P}_v^{l+1}$.

In other to prove the equality, notice that the path of size 1 $\langle v \rangle$ is trivially part of the right hand side. Next, suppose any other path $P = \langle v, u_i, \dots, w \rangle$, starting with v , with at most $l + 1$ vertices. By definition, there is the path $\langle u_i, \dots, w \rangle$ in $P_{u_i}^l$, and so, since there is an arc (v, u_i) , P will be part of the union. This implies $\mathcal{P}_v^{l+1} \subseteq \{\langle v \rangle\} \cup \bigcup_{1 \leq i \leq q} \{\langle v \rangle \cdot P \text{ for all } P \in \mathcal{P}_{u_i}^l\}$, which finishes our proof. $\square$

Corollary 4.1.1. *Given a colored DAG D , and a path P , let $colors(P)$ be the set of colors of the vertices of P . If v is a vertex in D , l a positive integer value, and $u_1, \dots, u_q$ are all the vertices such that there is an arc (v, u_i) for all i where $1 \leq i \leq q$, then*

$$\max_{P \in \mathcal{P}_v^{l+1}} |colors(P)| = \max\{|\{\{color(v)\}\} \cup \bigcup_{1 \leq i \leq q} \{\{color(v)\} \cup colors(P) \text{ for all } P \in \mathcal{P}_{u_i}^l\}|\}.$$

Proof. This follows from Theorem 4.1, as every path is replaced by the cardinality of its set of colors. The two sets of paths are the same in Theorem 4.1, so replacing each path with its set of colors results in two sets that are also congruent. This also works for the cardinality and the maximum function, so the equality is maintained, resulting in the expression we want. $\square$

Theorem 4.2. *Let $D = (V, A)$ be a vertex-colored DAG, v be a vertex in D , an integer limit, and S_C a set of colors. If Algorithm 3 is called with D , v , limit and S_C as its parameters, it returns exactly $\max_{P \in \mathcal{P}_v^{limit}} |colors(P) \cup S_C|$.*

Proof. We proceed by induction on the *limit* value.

(Base step): For $limit = 1$, P_v^1 consists of a single path: $\langle v \rangle$. Since $limit = 1$, the algorithm returns, in line 4, $|color(v) \cup S_C|$, which is the correct value.

(Inductive step): Let the theorem be valid for a value r . We must consider the call to Algorithm 3 with $limit = r + 1$. Since $limit > 1$, we execute the loop in line 7, and, for each u where there is an arc (v, u) , we make a recursive call. Notice the parameters of this call: D , u , r , and $S_C \cup \{color(v)\}$. Each call returns $\max_{P \in \mathcal{P}_u^r} |colors(P) \cup S_C \cup \{color(v)\}|$, by our inductive hypothesis.

After the loop, we have stored in the variable *super* the maximum cardinality between $|S_C \cup \{color(v)\}|$ and each of the return values of the recursive calls. This corresponds to

$$\max\{|\{\{color(v)\} \cup S_C\}| \cup \bigcup_{(v,u) \in A} \{\max_{P \in \mathcal{P}_u^r} |\{color(v)\} \cup colors(P) \cup S_C|\}\}. \quad (1)$$

We expand the inner maximum function and rearrange, obtaining the equivalent expression

$$\max\{|\{\{color(v)\} \cup S_C\}| \cup \bigcup_{(v,u) \in A} \{|\{\{color(v)\} \cup colors(P) \cup S_C\}| \text{ for all } P \in \mathcal{P}_u^r\}\}. \quad (2)$$

Notice this expression is equivalent to the right hand side from Corollary 4.1.1, with the exception that every set is part of an union with S_C . Since this is the case for every path, we can still apply it by also changing the left hand side. We obtain that Equation 2 is equal to $\max_{P \in \mathcal{P}_v^{r+1}} |colors(P) \cup S_C|$. $\square$

Lemma 4.3. *Let D be a DAG, D' the transitive closure of D , and k the number of colors among the vertices of D . If, for a $v \in D$, $\text{super}(v) = l \leq k$, then there is a path P' in D' , starting from v , with exactly l vertices of l different colors.*

Proof. If $\text{super}(v) = l$, there is a path in D starting from v with l colors ($l \leq k$). Let that be $P = \langle v = q_1, \dots, q_2, \dots, q_l, \dots, w \rangle$, where each q is the first vertex of each color in the path. It is evident that there is a path from q_1 to q_2 , from q_2 to q_3 and so on. This implies that there are arcs (q_i, q_{i+1}) in D' for each i such that $1 \leq i \leq l-1$, and so, there is a path $P' = \langle q_1, q_2, \dots, q_l \rangle$ in D' , with exactly l vertices, and each has a different color. $\square$

Theorem 4.4. *Given a DAG D , Algorithm 2 returns the super-coloring value for all vertices.*

Proof. Let D' be the transitive closure of D . For every vertex v , Algorithm 2 calls Algorithm 3 with D' , v , k and $\{\}$ as its parameters. By Theorem 4.2, the return value is $\max_{P \in \mathcal{P}_v^k} |\text{colors}(P)|$, which is the maximum amount of colors in all paths starting in v , with at most k vertices. By Lemma 4.3, this is enough to determine $\text{super}(v)$. $\square$

Although these algorithms may give correct solutions, performance is an important issue. As an alternate approach, we have designed an integer programming model for DAGs. This model may take advantage of methods used in integer programming, so that, even if it is not faster than the algorithmic methods, it may give bounds on the optimal solution.

5. Integer programming model for DAGs

In this section, we describe an integer programming model designed to solve the super-coloring problem where the input is a DAG. As part of designing and explaining the integer programming model, we employ two important definitions: an arc (u, v) is a **shortcut** if there is a path from u to v that contains at least one other vertex; and the **level** of a vertex v given a vertex r , which we denote $\text{level}_r(v)$, is the size in number of vertices of the longest path from r to v .

Although levels are not part of the model, they are useful for proving the theorems that relate to it. The shortcuts, on the other hand, are important to consider when building the constraints. It is important for our model that there are no shortcuts in our DAG, as we will explain when presenting the next section. Theorem 5.1 guarantees that removing them will not interfere with the answer to our problem.

Theorem 5.1. *Let $D = (V, A)$ be a DAG, and $D' = (V', A')$ be the graph such that $V' = V$, and A' is the result from taking A and removing all arcs of D that are shortcuts. For each $v \in V$ and the corresponding $v' \in V'$, $\text{super}(v)$ in D is equal to $\text{super}(v')$ in D' .*

Proof. First, notice that every path that exists in D' also exists in D , and therefore, for any pair of corresponding vertices $u \in V$ and $u' \in V'$, $\text{super}(u) \geq \text{super}(u')$.

Now, let P be a path in D starting at u that contains $\text{super}(u)$ colors. If this path does not contain any shortcut arcs, it can be directly transformed into a path in D' , starting at u' , and having $\text{super}(u)$ colors, and so $\text{super}(u) = \text{super}(u')$. Otherwise, for each shortcut (x, y) , we swap it with the longest path between x and y , and repeat until there are no more shortcuts. This is possible as long as the graph has no cycles, as we will see next. Obviously, making this switch does not decrease the number of colors in the path, as the new path still contains all original vertices.

Suppose the arc (x, y) in P was replaced by the subpath $\langle x, s_1, s_2, \dots, s_m, y \rangle$. None of the vertices s_i is part of the original path, as this would imply a cycle $\langle s_i, \dots, x, \dots, s_i \rangle$ or $\langle s_i, \dots, y, \dots, s_i \rangle$. At the end of the process, we have a path that exists in D' , starting at u' , and that has at least $\text{super}(u)$ colors, and so, $\text{super}(u') \geq \text{super}(u)$. Since $\text{super}(u) \geq \text{super}(u')$ and $\text{super}(u') \geq \text{super}(u)$, it follows that $\text{super}(u) = \text{super}(u')$. $\square$

To identify the shortcuts, a simple algorithm using breadth-first search is used. For each arc (x, y) , a graph $G_a = (V, A - \{(x, y)\})$ is created, and the search is used to determine whether there is a path between x and y . If it exists, the arc (x, y) is a shortcut, and can be removed from the graph. Since this requires a search for each edge, the complexity of this process is $\mathcal{O}(m(n + m))$, where n and m mean the number of vertices and number of edges in the graph, respectively.

5.1. Integer programming model

To explain the model, it is interesting to first consider a simplified version of the problem, which asks for the solution only for a vertex r , which we call the root. The model we have developed tries to build the optimal path (that is, the path that passes through the greatest number of colors.) This is represented through the binary variables p_i , which represent, for a given vertex i , whether i is on the path (when $p_i = 1$) or not (when $p_i = 0$.) To ensure that the path P generated by these variables is valid, it is necessary to ensure that some constraints apply.

The first constraint is that, for each vertex v that is part of P and that is not the root r , the path P must also contain one of its parents, i.e., one of the vertices u such that (u, v) is an arc in the graph. These constraints, which we refer to as parent constraints, are represented as

$$p_v \leq \sum_{(u,v) \in A} p_u \text{ for all } v \in V - \{r\}. \quad (3)$$

Note that this can lead to the inclusion of multiple vertices without forming a single path. In order to force it to be a proper path, we add constraints so that only one of the children of each vertex is selected at most. This leads to what we call the children constraints:

$$\sum_{(u,v) \in A} p_v \leq 1 \text{ for all } u \in V. \quad (4)$$

To induce a path starting with the root, the constraint that $p_r = 1$ is added. Note that, for these constraints to encompass all paths, there must be no shortcuts in the graph. This is explored in Theorem 5.3

Next, we present three results. The first one is about level and will help us to prove other results. The second shows that any values associated to p variables satisfying the parent and children constraints are paths in the DAG. Finally, the last one shows that for any path in the DAG, the associated values to the p variables satisfy the children and parent constraints.

Lemma 5.2. *For a vertex r and vertices u and v , if there is a path from r to u , a path from r to v , and an arc (u, v) , then $\text{level}_r(u) < \text{level}_r(v)$.*

Proof. Suppose $\text{level}_r(u) \geq \text{level}_r(v)$. There is a path $\langle r, \dots, u \rangle$ with length $\text{level}_r(u)$. Since there is an arc (u, v) , there is a path $\langle r, \dots, u, v \rangle$ with length $\text{level}_r(u) + 1$. So, $\text{level}_r(v) \geq \text{level}_r(u) + 1$, making $\text{level}_r(v) > \text{level}_r(u)$, which is a contradiction. $\square$

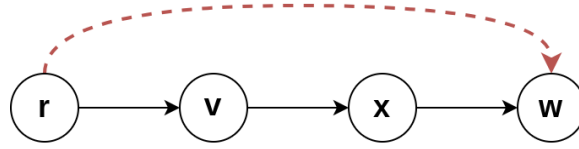


Figure 1: A graph with four vertices, including a shortcut (r, w) .

Theorem 5.3. *If G has no shortcuts, a choice of variables p that respects constraints (3) and (4) induces a path, that is, there is a path P in which $p_v = 1$ for every vertex v in P , and $p_w = 0$ for every vertex w that is not in P .*

Proof. Let $Q = \{v \in V | p_v = 1\}$ for a choice of variables p that respects the constraints, but does not induce a path. Let be a sequence of vertices $P' = \langle q_0, q_1, \dots, q_s \rangle$, where the vertices correspond to the vertices in Q ordered by level. We assert that P' is a path, or, rather, that $\langle q_0, \dots, q_j \rangle$ is a path for all j such that $0 \leq j \leq s$. We proceed by induction:

(Base step) $\langle q_0 \rangle$ is a path of size 1. Furthermore, $\langle q_0, q_1 \rangle$ is a path, since q_1 must have a parent by the parent constraints, and by Lemma 5.2, q_0 is the only option.

(Inductive step) Suppose $S = \langle q_0, \dots, q_j \rangle$, with $j \geq 2$ is a path. If there is no arc (q_j, q_{j+1}) , by the parent constraints, there must be a $q_s \neq q_j$ such that there is an arc (q_s, q_{j+1}) . This also implies $level(q_s) < level(q_{j+1})$, and so, q_s is part of S . There is, therefore, a subpath $S' = \langle q_s, q_{s+1}, \dots, q_j \rangle$. But if this is the case, q_s has two children in Q , which contradicts the children constraints. Therefore, there must be an arc (q_j, q_{j+1}) , and the path $\langle q_0, \dots, q_{j+1} \rangle$ exists. $\square$

Theorem 5.4. *Let G be a DAG with no shortcuts, and $P = \langle q_0, \dots, q_s \rangle$ any path in G . If q_0 is defined as the root and the p variables are chosen so that*

$$p_v = \begin{cases} 1, & \text{if } v \text{ is a vertex in } P; \\ 0, & \text{otherwise,} \end{cases}$$

then all constraints are satisfied.

Proof. Firstly, from the construction of the variables, it is clear that $p_{q_i} = 1$ for $1 \leq i \leq s$, which covers the parent constraints for vertices in P . For other vertices, this constraint is trivially satisfied.

Secondly, suppose that constraint (4) is not satisfied for a certain u , and let w and x be vertices such that there are arcs (u, w) and (u, x) , and $p_w = p_x = 1$, which is true only if w and x are part of P . Considering w , we have three cases:

- w appears in the path *before* u . This would imply a cycle $\langle w, \dots, u, w \rangle$, and so, it cannot happen in G , which is a DAG;
- w appears in the path *after* u , but does not immediately succeed it. This implies in a subpath $\langle u, \dots, w \rangle$ with length larger than 2. But this means (u, w) is a shortcut, which is a contradiction, since G has no shortcuts;
- w is the immediate successor of u in the path. This is the only possible option.

This argument works identically for x , which means it also must be the immediate successor of u . But it is not possible for more than one vertex to occupy this position, making it a contradiction. $\square$

To encode the colors, a matrix M identifies the colors for each vertex:

$$M_{i,k} = \begin{cases} 1, & \text{if } color(i) = k; \\ 0, & \text{otherwise.} \end{cases}$$

The presence of each of the k colors in the path is indicated using variables, which we represent with T_k for each color k . Here, we define T_k as a binary variable, with $T_k = 1$ indicating that some vertex of the path has the color k , or $T_k = 0$ indicating that no vertex of the path has the color k .

To ensure that colors that are **not** in the path are **not** counted, we make $T_k \leq \sum_{v \in V} M_{v,k} \cdot p_v$ for all k ; to ensure that colors in the path P are accounted for, we could make $T_k \geq M_{v,k} \cdot p_v$ for all v, k . This last group of constraints is not necessary, however, since our objective function tries to maximize the value of each T_k .

Finally, we define the objective function, to be maximized, which corresponds to the number of distinct colors that are on the defined path: $\max \sum_{i=1, \dots, k} T_i$.

Note that this model is extendable to all vertices in a graph creating several independent models, each with a different vertex being the root.

6. Experiments

We experimented on graphs that were extracted from real genealogical data. The available data included a list of individuals, their sex, and their parents. Four of the available instances included clan data for each individual, while five others did not. We can see in Table 1 the number of vertices and arcs for the corresponding graph of each instance, as well as the source for each network.

Table 1: Summary of the instances in our dataset

Network	$ V $	$ A $	Clan Data	# of Colors	Source
N1	1214	1792	Absent	253	Lima [2015]
N2	392	733	Absent	13	Teixeira-Pinto [1997]
N3	1252	2280	Absent	109	Córdoba and Villar [2013]
N4	875	1589	Absent	70	Florida [2013]
N5	789	1386	Present	10	Silva [2012]
N6	618	1003	Absent	113	Bueno [2015]
N7	1444	2377	Present	34	Melatti [1970]
N8	459	713	Present	2	Maybury-Lewis [1974]
N9	755	1224	Present	8	[Unpublished]

For instances that did not include information regarding clans, we colored the graphs artificially. We do this by coloring each of the source vertices (that is, vertices without any parent vertex) with a color, and propagating it through the graph. This can be done by setting the color of each child as the color of their father (which we call patrilinear) or their mother (which we call matrilinear). As such, we create two graphs corresponding to these two options. Occasionally, some vertices are left uncolored by this process. We then handpick the color from one of its parents. The output from this process is a kinship network, which is the input for our algorithms.

We also use these graphs to generate the corresponding integer programming models for each genealogical network. These files are then used as input for the solver. We used Gurobi Optimization, LLC [2024], version 10.0.3, with an AMD Ryzen 5 CPU, and 6 physical cores.

Table 2 gives a summary of the results, comparing our different approaches according to size, given in number of vertices. For the networks without clans, the name includes the type of heritage, with M representing matrilinear, and P representing patrilinear. In order to minimize fluctuations of time, the values are extracted from 30 runs of the general algorithm and the k -color algorithm, and from 20 runs (that is, 20 independent calls to the solver to optimize the model) in the case of the solver. The mean time is shown in seconds, along with the standard deviation, also in seconds. For the solver, the time is split in time reading the model, time solving it.

We note that all the instances are easy to solve in the sense of how experimentally fast the proposed methods were performed. However, it is important to say that our main interest is in solving real world instances. All the presented instances are real, gathered by our anthropologist partners, for whom the solutions found here will be presented. The algorithms must be tested with other challenging real kinship networks for a more accurate analysis.

Table 2: Summary of the runs of our algorithms and optimization model

Name	Size	Time (General)	Time (k -Color)	Reading Time	Solving Time
N2 (M)	392	0.016 $\pm$ 0.001	0.017 $\pm$ 0.002	0.02 $\pm$ 0.00	0.08 $\pm$ 0.01
N2 (P)	392	0.017 $\pm$ 0.001	0.015 $\pm$ 0.001	0.02 $\pm$ 0.00	0.08 $\pm$ 0.01
N8	459	0.005 $\pm$ 0.001	0.010 $\pm$ 0.003	0.01 $\pm$ 0.00	0.02 $\pm$ 0.00
N6 (M)	618	0.052 $\pm$ 0.002	0.026 $\pm$ 0.003	0.08 $\pm$ 0.01	0.15 $\pm$ 0.03
N6 (P)	618	0.059 $\pm$ 0.005	0.027 $\pm$ 0.004	0.08 $\pm$ 0.01	0.16 $\pm$ 0.03
N9	755	0.028 $\pm$ 0.002	0.032 $\pm$ 0.004	0.03 $\pm$ 0.01	0.05 $\pm$ 0.01
N5	789	0.035 $\pm$ 0.004	0.035 $\pm$ 0.004	0.04 $\pm$ 0.00	0.11 $\pm$ 0.02
N4 (M)	875	0.072 $\pm$ 0.005	0.059 $\pm$ 0.007	0.08 $\pm$ 0.00	0.36 $\pm$ 0.05
N4 (P)	875	0.071 $\pm$ 0.006	0.058 $\pm$ 0.005	0.08 $\pm$ 0.01	0.41 $\pm$ 0.05
N1 (M)	1214	0.097 $\pm$ 0.003	0.051 $\pm$ 0.006	0.21 $\pm$ 0.01	0.32 $\pm$ 0.04
N1 (P)	1214	0.096 $\pm$ 0.003	0.055 $\pm$ 0.008	0.21 $\pm$ 0.01	0.32 $\pm$ 0.04
N3 (M)	1252	0.316 $\pm$ 0.011	0.132 $\pm$ 0.011	0.15 $\pm$ 0.01	0.73 $\pm$ 0.07
N3 (P)	1252	0.330 $\pm$ 0.016	0.120 $\pm$ 0.004	0.15 $\pm$ 0.02	0.69 $\pm$ 0.08
N7	1444	0.236 $\pm$ 0.008	0.302 $\pm$ 0.020	0.13 $\pm$ 0.01	0.63 $\pm$ 0.06

7. Conclusion

In this paper, we designed and discussed algorithmic solutions for the super-coloring problem, including an algorithm for general directed graphs and an algorithm tailored to DAGs. We also developed an integer programming model for the problem, as an alternative focused on optimization. These approaches were then compared based on their performance on graphs based on real-life genealogy networks. Future work could include a revision of the integer model, focusing on reducing the number of variables, or a more theoretical investigation of the relation between the super-coloring problem and other similar problems on graph theory. Additionally, creating more difficult instances of the problem could stress our methods, and give more insight into their complexity.

References

- Bueno, A. C. V. (2015). *Fios de memórias. Um estudo sobre parentesco e história a partir da construção da genealogia Manoki (Irantxe)*. PhD thesis, Universidade de São Paulo.
- Cohen, J., Italiano, G. F., Manoussakis, Y., Thang, N. K., and Pham, H. P. (2021). Tropical paths in vertex-colored graphs. *J. Comb. Optim.*, 42(3):476–498. ISSN 1382-6905.
- Córdoba, L. and Villar, D. (2013). Some aspects of marriage alliance among the chacobo (bolivian amazonia). In *Wege im Garten der Ethnologie. Caminos en el jardín de la etnología*, p. 177–190. Hanna Heinrich; Harald Grauer.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition. ISBN 0262033844.
- Dal Poz, J., Neto, P., and Silva, M. (2009). Maqpar: a homemade tool for the study of kinship networks. *Vibrant*, 6.
- Florido, M. P. (2013). *Os Deni do Cuniuá: um estudo do parentesco*. PhD thesis, Universidade de São Paulo.
- Franco, Á. J. P. and Vendramin, M. (2021). Super-colored paths in digraphs. In *Anais do VI Encontro de Teoria da Computação*, p. 94–97, Porto Alegre, RS, Brasil. SBC.
- Gurobi Optimization, LLC (2024). Gurobi Optimizer Reference Manual. URL <https://www.gurobi.com>.
- Hamberger, K., Houseman, M., Daillant, I., White, D., and Barry, L. (2004). Matrimonial ring structures. *Mathématiques et sciences humaines*, 43:83–120.
- Hassin, R., Monnot, J., and Segev, D. (2007). Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization*, 14.
- Lima, L. M. d. (2015). *No Arapiuns, entre verdadeiros e-ranas: sobre os espaços, as lógicas, as organizações e os movimentos do político*. PhD thesis, Universidade de São Paulo.
- Maybury-Lewis, D. (1974). *Akwẽ-Shavante Society*. Oxford University Press. ISBN 9780195197297.
- Melatti, J. C. (1970). O sistema social craô. *Série Antropológica*.
- Monnot, J. (2005). The labeled perfect matching in bipartite graphs. *Information Processing Letters*, 96(3):81–88. ISSN 0020-0190.
- Silva, M. F. d. (2012). *Liga dos enawene-nawe: um estudo da aliança de casamento na amazônia meridional*. Master's thesis, Universidade de São Paulo.
- Teixeira-Pinto, M. (1997). *Ieipari: sacrifício e vida social entre os índios Arara (Caribe)*, volume 40. Editora Hucitec.
- Williams, V. V., Xu, Y., Xu, Z., and Zhou, R. (2024). New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, p. 3792–3835. SIAM.