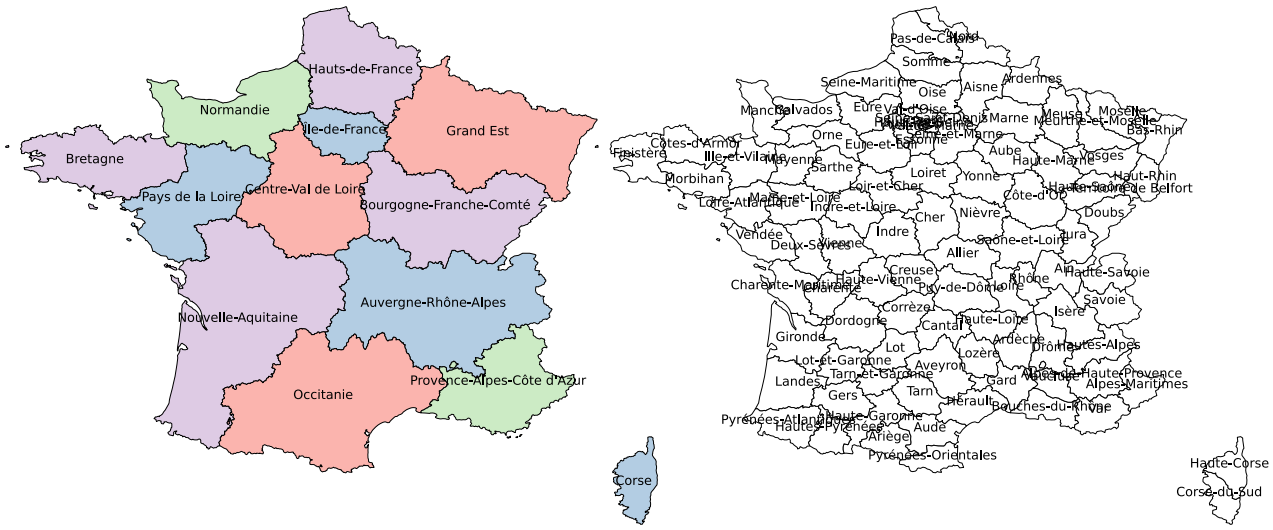
TP-Cours d'Informatique 10

Algorithmes gloutons : application à la coloration de carte

ITC – PCSI[2] 2025-2026 – Lycée Fabert (Metz) – S.Roby – R.Sinteff – D.Mengel – S.Limal

Objectif



La France est découpée en 5 niveaux : région (figure de gauche), département (figure de droite), arrondissement, canton et commune. Si la coloration manuelle du découpage en région est envisageable, dès le découpage en départements, l'apport d'une coloration automatisée devient intéressant.

On souhaite colorer une carte pour mettre en évidence son découpage :

- Les zones ayant une frontière commune ne doivent pas avoir la même couleur (**contrainte**),
- Il faut minimiser de nombre de couleurs (**coût**).

Ressources

Le dossier fourni contient :

- des « fichiers de formes » $*$.shp, format de fichier pour les systèmes d'informations géographiques contenant notamment des informations de géométrie. Ils sont complétés par des fichiers de données complémentaires extension $*$.dbf, Ces fichiers proviennent du site https://gadm.org/ ,
- une bibliothèque `shapefile.py` (https://github.com/GeospatialPython/pyshp) permettant de lire les « fichier de formes »,
- des fichiers textes $*$.csv répertoriant les voisins des zones de chaque niveau,
- une bibliothèque `ModuleTPGlouton.py` à importer avec l'alias `TP`,
- un fichier à compléter.

Construction du programme

On rappelle comment récupérer le contenu d'un fichier texte :

```
file = open("fichier.csv",'r',encoding="utf-8")
ListeLignes = file.readlines()
file.close()
```

1. Lire et mémoriser dans la variable `ListeLignes` la liste les données contenues dans le fichier voisins_1.csv.

- Par application de la fonction `mise_en_forme_liste_voisins` de la bibliothèque aux lignes importées du fichier *.csv, générer la liste des voisins de chaque zone.

```
L_voisins=TP.mise_en_forme_liste_voisins(ListeLignes)
```

2. À partir des données associées à `L_voisins`, construire le dictionnaire `dict_voisins` qui associe à chaque clé nom de zone la valeur liste des noms de ses voisins. On pourra utiliser une construction par compréhension ou une boucle pour ajouter successivement les couples (clé,valeur).

On souhaite désormais construire un dictionnaire dans lequel chaque clé correspond à un nom de zone et la valeur associée à une couleur. La fonction de coloration utilisera les paramètres d'entrée suivants :

- un dictionnaire dans lequel chaque clé correspond à un nom de zone et la valeur associée à la liste de ses voisins,
- une liste de couleurs.

L'algorithme glouton proposé consiste à :

- parcourir les différentes zones,
- pour une zone donnée,
  - construire la liste des couleurs déjà prises par ses voisins,
  - parcourir les couleurs. La première couleur trouvée non attribuée à un voisin est attribuée à la zone courante,
  - si toutes les couleurs sont attribuées à des voisins, il n'y a pas de solution de coloration. Lever une assertion pour arrêter le programme.

On rappelle comment lever une assertion :

```
assert MaNote>20, f"La note est seulement de {MaNote}"
```

3. Écrire la fonction `coloration_gloutonne(voisins:dict, couleurs:list)` qui construit et renvoie un dictionnaire associant une valeur de couleur à un nom ... ou termine le programme en levant une assertion.

Le fichier à compléter propose une liste de couleurs par défaut.

```
liste_couleurs=['#fbb4ae','#b3cde3','#ccebc5','#decbe4','#fed9a6','#
    ffffcc']
```

La page https://colorbrewer2.org/ permet de générer d'autres listes de couleurs adaptées à différents critères de cartographie. Il suffit de choisir le nombre de couleurs, choisir un schéma, régler un codage HEXadécimal et sélectionner l'onglet EXPORT avant de copier-coller dans le code le tableau destiné à JavaScript.

4. Appliquer la fonction `coloration_gloutonne` à `dict_voisins` en utilisant la liste de couleurs `liste_couleurs` et affecter le résultat renvoyé à la variable `couleurs_zones`. Visualiser le résultat obtenu dans la console.

- Afficher l'aide de la fonction `tracer_carte` pour connaître la syntaxe :

```
1 help(TP.tracer_carte)
```

5. Appliquer la fonction `tracer_carte` à `couleurs_zones` pour calculer la carte colorée et mémoriser la figure renvoyée.

- Avec la méthode `show`, afficher la figure obtenue mémorisée.

Analyse et amélioration

Un problème d'optimisation est un problème algorithmique dans lequel l'objectif est de trouver la solution optimale (**coût** minimum ou maximum) par rapport à un critère en respectant une **contrainte**.
Le principe de l'algorithme glouton est de déterminer une **solution optimale locale**. Chaque décision prise à un état donné n'est jamais remise en cause.

Les avantages sont :

- la simplicité des décisions. Dans l'exemple précédent, on choisit le 1ère couleur non utilisée par les voisins d'une zone,
- une progression vers une solution qui est mesurable. Dans l'exemple précédent, il est possible d'afficher le nombre de zones traitées sur le total de zones.

6. Mettre à jour la fonction `coloration_gloutonne` pour afficher en console l'évolution du nombre de zones affectées d'une couleur sur le nombre total de zones.

Le théorème des quatre couleurs indique qu'il est possible, en n'utilisant que quatre couleurs différentes, de colorier n'importe quelle carte découpée en zones connexes (chaque zone est en un seul morceau), de sorte que deux zones limitrophes reçoivent toujours deux couleurs distinctes. La démonstration de ce théorème a été prouvée informatiquement.

7. Vérifier si notre algorithme glouton est capable de colorier notre carte des régions avec 4 couleurs.

Il apparaît que notre algorithme ne trouve pas forcément de solution. Sans remettre en cause notre algorithme glouton, il est possible de mélanger les couleurs à chaque changement de zone.

8. Importer la fonction `shuffle` de la bibliothèque `random`.
   Mettre à jour la fonction `coloration_gloutonne` pour mélanger la liste de couleurs avant chaque parcours de cette liste.

En exécutant plusieurs fois le programme, l'algorithme trouve une solution, mais pas systématiquement. Et, en passant à un découpage de niveau 2 (à partir de voisins_2.csv et de l'appel correspondant de la fonction `tracer_carte`), ce n'est plus le cas.

L'algorithme cherche une couleur par zone dans l'ordre fourni par le fichier texte. Si la coloration était traitée manuellement, les zones les plus contraintes (qui ont le plus de voisins) seraient colorées en premier.

9. Écrire une fonction de tri qui admet en entrée un liste de couples (zones, nombre de voisins) et renvoie la liste triée par nombre décroissant de voisins. On pourra par exemple utiliser un algorithme de tri par insertion ou de tri rapide (cf. TPs précédents).

10. Réaliser des essais en exécutant plusieurs fois l'algorithme pour un couple (niveau de découpage (entre 1 et 4), le nombre de couleurs) donné.

Si le pré-traitement des données améliore la capacité de trouver une solution, on constate sur cet exemple que l'algorithme glouton utilisé ne trouve pas forcément de solution ... et ne trouve pas forcément la solution optimale.

Sources

https://www.cs.cmu.edu/ bryant/boolean/maps.html
https://ian-r-rose.github.io/mapping-california-cities.html