

Coloration de graphes d'intervalle

Léo le Douarec, Antoine Reilhes



Figure 1 – Long Thanh International Airport (Vietnam). Source : <https://www.azuremagazine.com/article/8-global-airports-ushering-in-a-new-era-of-aviation-and-design/>

Résumé

Les aéroports occupent souvent de vastes terrains (32 km² pour l'aéroport Charles de Gaulle). Les projets d'extension ou de nouveaux aéroports sont souvent sujets à des controverses, liées notamment à l'artificialisation des sols (souvent cultivables), aux impacts sur les agriculteurs, ainsi qu'aux forts coûts de construction de l'infrastructure et bien sûr à l'impact écologique. Nous proposons ici une modélisation par graphe de la capacité d'accueil d'un aéroport, afin de minimiser son impact tout en assurant l'opérabilité de l'aéroport. Le problème d'optimisation abordé a donc des conséquences majeures d'un point de vue économique, écologique et territorial.

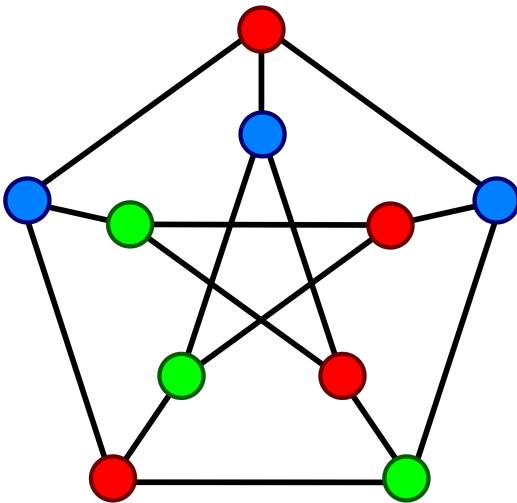


Figure 2 – Coloriage du graphe de Petersen avec 3 couleurs. Source : <https://commons.wikimedia.org/w/index.php?curid=1386753>

Table des matières

Présentation du problème et introduction	3
I Modélisation du problème	3
I.1 Modélisation sans retards	3
I.2 Modélisation avec retards	4
I.2.1 1 ^{re} approche : retard moyen	4
I.2.2 2 ^e approche : retard exceptionnel	4
II Graphes d'intervalles : résultats théoriques	4
II.1 Coloriage de graphes : Algorithme glouton	5
II.2 Application de l'algorithme glouton aux graphes d'intervalles	5
II.3 Ajout d'intervalles <i>sans augmenter le nombre chromatique</i> : proposition d'un algorithme naïf	6
III Application au problème concret (aéroports)	8
III.1 Implémentation Python et NetworkX	8
III.1.1 Récupération des données, coloriage et affichage du graphe	8
III.1.2 Prise en compte des retards	11
III.2 Analyse des résultats	11
III.2.1 Problème initial (sans retards)	11
III.2.2 Problème avec retards	11
III.2.3 Etude de l'ajout de vols	11
Conclusion	12
Bibliographie	12

Présentation du problème et introduction

Un aéroport prévoit un certain nombre de vols programmés chaque jour. L'objectif du problème est de déterminer le nombre d'emplacements d'avions à affecter pour la construction de l'aéroport afin de vérifier qu'il peut accueillir chaque vol. On prendra en compte le retard moyen des vols au départ de l'aéroport (fourni). De plus, l'aéroport devra toujours avoir 2 emplacements libres en cas d'atterrissements d'urgence.

Ce problème peut se modéliser à l'aide de graphes d'intervalles. C'est une modélisation couramment utilisée pour résoudre des problèmes d'ordonnancement (comme le nombre de voies à mettre en place dans une gare, par exemple). Leur popularité est en partie due à l'existence d'un algorithme *exact* de complexité $\mathcal{O}(|V| \log |V|)$ pour la coloration des graphes d'intervalles (résultat présenté dans ce rapport).

Le dimensionnement de la capacité d'accueil d'un aéroport dépend de nombreux facteurs. Les plus grands aéroports sont naturellement situés aux carrefours économiques et touristiques de la planète, mais d'autres dimensions propres à l'aviation sont à prendre en compte, avec l'aviation de tourisme (biplanes, ULM), l'aviation privée (hélicoptères, jets), et l'aviation militaire, le fret... Nous abordons ici une version simplifiée de ce problème en ne considérant que l'aviation civile **commerciale** (avions de ligne). De plus, même si les aéroports des grandes métropoles concentrent une grande partie du trafic aérien, les aéroports régionaux gardent un enjeu important pour l'accessibilité et l'attractivité de certaines régions, et conservent un nombre significatif de vols. Lors de la conception d'un aéroport, les constructeurs doivent souvent estimer le trafic prévisionnel.

Pour traiter ce problème de manière pratique avec NetworkX, nous allons étudier l'aéroport de **Strasbourg-Entzheim (SXB)** sur une journée classique, afin de discuter de la capacité d'accueil nécessaire pour assurer les vols de cet aéroport (toujours en minimisant les impacts de la structure). Nous pourrons également comparer les résultats théoriques à une observation de l'aéroport réel (via images satellite).

I - Modélisation du problème

Dans cette partie, nous expliquons comment construire un modèle mathématique de l'occupation d'un aéroport à l'aide de graphes d'intervalles. On définit tout d'abord les graphes d'intervalles :

Definition 1 (Graphe d'intervalle) Soit $I = I_1, \dots, I_n$ un ensemble d'intervalles.

Le graphe d'intervalles $G = (V, E)$ correspondant est défini par :

- $V = I$ (*sommets = intervalles*)
- $\forall (\alpha, \beta) \in [|1, n|], (I_\alpha, I_\beta) \in E \iff I_\alpha \cap I_\beta \neq \emptyset$ (*deux intervalles sont reliés dans le graphe si et seulement si leur intersection est non nulle*)

Passons maintenant aux deux modélisations choisies : la première ne prend pas en compte les retards, la deuxième oui.

I.1 Modélisation sans retards

L'idée générale de la modélisation est d'utiliser les heures d'arrivées et de départ théoriques des avions. Les intervalles du graphe construit correspondent alors aux intervalles [heure_arrivée, heure_départ] (pour chaque avion). Par exemple, pour l'aéroport de Strasbourg, on peut représenter les intervalles comme indiqué en figure 3.

Pour réaliser la modélisation sur une journée, deux décisions ont été prises :

- Les avions ayant atterri la veille et passant la nuit à l'aéroport sont marqués présents dès le début de la journée (00h)
- Les avions ayant une date d'atterrissement le jour d'étude et repartant le lendemain sont pris en compte comme s'ils décollaient à 23h59.

Ces "hypothèses" permettent de découper l'activité de l'aéroport sur une journée de 24h.

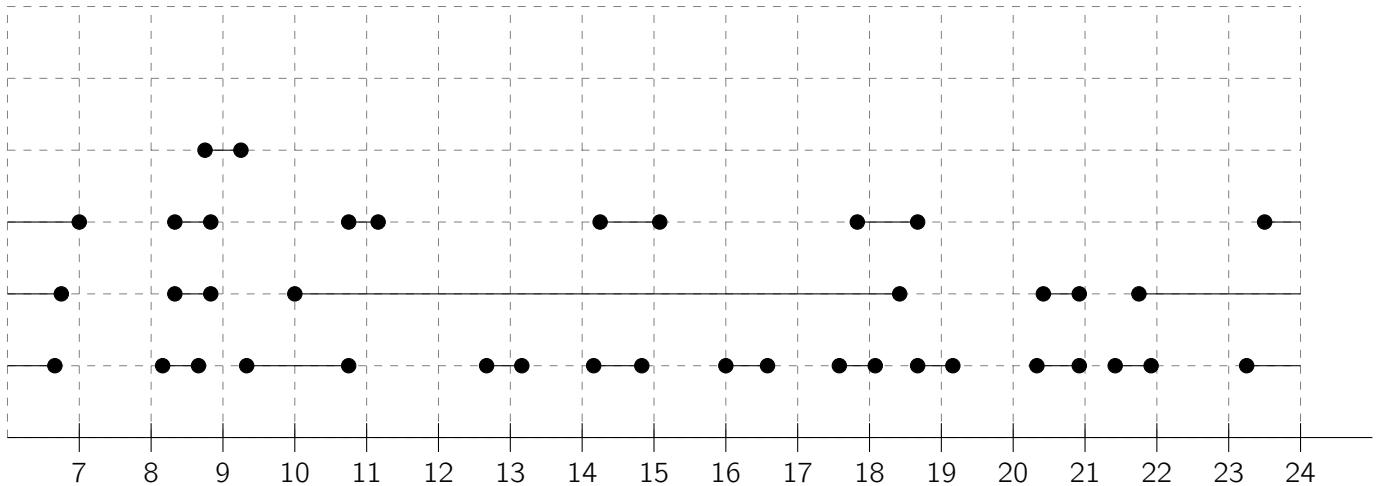


Figure 3 – Modélisation de l'activité de l'aéroport SXB par graphes d'intervalles (les heures sont données en abscisse).

I.2 Modélisation avec retards

On peut vouloir raffiner ce modèle afin d'avoir des résultats plus fins. Nous proposons donc deux approches pour prendre en compte le retard des avions dans le modèle.

I.2.1 1^{re} approche : retard moyen

Une première manière simple de procéder est de considérer une donnée déjà connue sur l'aéroport : le retard moyen au décollage (27 minutes dans le cas de Strasbourg¹). Pour ce faire, il suffit de changer la borne droite de l'intervalle. On ajoute ainsi `retard_moyen` à l'heure de décollage théorique de chaque avion, et le graphe a la même allure que la Figure 3, avec des intervalles plus longs.

I.2.2 2^e approche : retard exceptionnel

Il arrive également que des événements exceptionnels (incidents, conditions météo, retard passager...) amènent à des retards ponctuels plus ou moins importants sur un vol donné. Pour modéliser cela, nous procédons de la même manière que pour le retard moyen, sauf que l'on ajoute une durée aléatoire (entre 0 et 200 minutes) à l'heure d'atterrissement et de décollage théoriques d'un avion donné (on peut aussi ne changer que l'heure de décollage). Cela permet ensuite d'étudier l'impact de ce retard exceptionnel sur la capacité d'adaptation dont a besoin l'aéroport pour fonctionner normalement (places supplémentaires).

Nous allons maintenant étudier les propriétés de cette modélisation (Section II), et ainsi justifier l'implémentation NetworkX réalisée (Section III.1).

II - Graphes d'intervalles : résultats théoriques

Dans cette section, nous allons présenter quelques résultats théoriques propres aux graphes d'intervalles et à la coloration des graphes d'intervalles. Ces résultats sont inspirés de nos lectures (en particulier l'article publié en 2008 par Kosowski, Adrian and Krzysztof Manuszewski [3], qui présente l'algorithme glouton et les méthodes de coloration séquentielles) pour les sections II.1 et II.2. La section II.3 est le fruit de nos propres idées (nous ne nous sommes pas inspiré d'un article particulier pour la développer). Tous ces résultats sont centraux pour justifier le choix des algorithmes utilisés par la suite.

1. Source des données : <https://airportinfo.live/fr/statistiques-aeroport/sxb-strasbourg-international-de-strasbourg>

II.1 Coloriage de graphes : Algorithme glouton

L'algorithme utilisé est un algorithme glouton (`greedy_color`). Il se base sur un principe relativement simple : étant donné un ordre de parcours $O = v_1, \dots, v_n$ des noeuds d'un graphe G , l'algorithme attribue au sommet v la plus petite couleur² possible en analysant les couleurs déjà attribuées aux voisins de v . La figure 4 présente le pseudo-code de cet algorithme.

```
algorithm greedy_color(G,O);
begin
for v := v1 to vn do
give vertex v the smallest possible color2;
end;
```

Figure 4 – Algorithme glouton pour le coloriage

De plus, pour résoudre notre problème nous utilisons un algorithme dit de **coloration séquentielle**. Autrement dit, nous définissons un ordre de parcours des sommets de G spécifique à notre problème pour que l'algorithme glouton renvoie le nombre chromatique le plus petit possible :

Definition 2 (Coloration séquentielle) Soit G un graphe.

Un **algorithme de coloration séquentielle** est un algorithme de coloration réalisant deux étapes :

- déterminer un ordre de parcours O des sommets de G
- appliquer `greedy_color(G,O)`.

II.2 Application de l'algorithme glouton aux graphes d'intervalles

Pour les graphes d'intervalle, on utilise une méthode spécifique pour déterminer l'ordre de parcours, qu'on appelle "Méthode Left Sort (LS)" :

Definition 3 (Méthode Left Sort (LS)) Soit G un graphe d'intervalle.

On appelle *Méthode LS* (pour "Left Sort") la méthode qui parcourt les sommets de G par borne inférieure d'intervalle croissante.

Exemple 1 Pour le graphe dont les intervalles sont [arrivée-avion, départ-avion], la méthode LS parcourt les avions par heure d'arrivée croissante.

Cette méthode nous permet de garantir l'optimalité de l'algorithme glouton sur les graphes d'intervalle. C'est un résultat central pour notre approche de modélisation :

Proposition 1 (Optimalité) Soit G un graphe d'intervalle. L'algorithme Glouton sur les sommets, ordonnés par date d'arrivée croissante, renvoie une coloration optimale de G .

Exemple 2 Autrement dit, la coloration séquentielle avec méthode LS renvoie une coloration optimale.

Ce résultat nous permet d'appliquer l'algorithme glouton avec méthode LS sur n'importe quel graphe d'intervalle, en particulier un graphe modélisant l'activité d'un aéroport, et d'obtenir ainsi une coloration optimale (avec un minimum de couleurs), ce qui présente un très grand avantage ! Voici une preuve de cette proposition :

Proof Notons $glouton(G)$ le nombre chromatique donné par l'algorithme glouton sur G . On veut montrer que $glouton(G) = \chi(G)$. On va raisonner par double inégalité :

- $glouton(G) \geq \chi(G)$ car l'algorithme glouton renvoie un coloriage.

2. On représente les couleurs par les entiers $0, 1, 2, \dots$, d'où la notion de "plus petite couleur".

- Montrons que $glouton(G) \leq \chi(G)$. Par l'absurde, supposons $glouton(G) > \chi(G)$. Soit I le 1^{er} intervalle qui utilise la couleur $\chi(G) + 1$ (on numérote les couleurs en partant de 1).
- Puisque le parcours des intervalles se fait par date de début croissante**, cela signifie qu'il y a au moins $\chi(G)$ intervalles qui commencent avant I , et finissent après son début ($\chi(G)$ conflits). Il y a donc au moins $\chi(G) + 1$ tâches en parallèle, et on a donc besoin d'au moins $\chi(G) + 1$ couleurs pour G . C'est-à-dire que $\chi(G) + 1 = \chi(G)$ par définition du nombre chromatique : **absurde**.
- Ainsi, $glouton(G) = \chi(G)$: l'algorithme glouton renvoie bien un coloriage optimal.

□

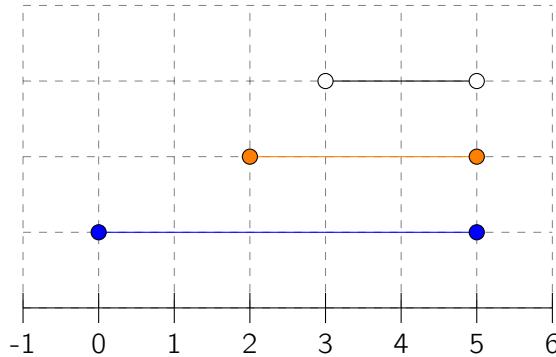


Figure 5 – Illustration de la preuve d'optimalité, dans un cas simple de conflit. Explication : si l'algorithme glouton est obligé d'utiliser une 3^e couleur, c'est qu'il rencontre une situation où l'intervalle du haut (le prochain à colorier, en blanc) est "en conflit" (relié) à deux autres intervalles déjà coloriés avec deux couleurs différentes (ils ont été coloriés avant car leur heure d'arrivée est inférieure à celle de l'intervalle blanc). L'algorithme glouton ajoute donc une nouvelle couleur. Mais on voit bien qu'il est impossible de colorier ce graphe d'intervalle avec moins de 3 couleurs. D'où la preuve par l'absurde, qui montre que l'algorithme glouton avec méthode LS ne peut pas utiliser plus de couleurs que le nombre chromatique.

On termine cette section avec la question de la complexité de l'algorithme glouton dans le cas du problème de l'aéroport.

Proposition 2 (Complexité de l'algorithme glouton dans le cas du problème de l'aéroport) Soit $G = (E, V)$ un graphe représentant le problème de gestion de l'aéroport.
L'algorithme proposé pour le problème de gestion de l'aéroport est de complexité $\mathcal{O}(|V| \log |V|)$.

Proof L'algorithme de coloriage glouton est de complexité $\mathcal{O}(|V|)$, où V est l'ensemble des sommets de G (on fait un unique parcours de l'ensemble des noeuds. On considère que le parcours des voisins pour le choix de la couleur est en $\mathcal{O}(1)$). Quant à elle, la méthode LS requiert un tri de la liste des sommets, et est donc de complexité $\mathcal{O}(|V| \log |V|)$. Cette dernière complexité majore donc la complexité de l'algorithme.

□

II.3 Ajout d'intervalles sans augmenter le nombre chromatique : proposition d'un algorithme naïf

Les deux parties précédentes nous donnent un algorithme exact pour obtenir une coloration minimale pour le trafic **déjà prévu** à l'aéroport. Nous nous sommes alors demandés s'il était possible de rajouter des avions sans augmenter le nombre chromatique (=le nombre de places d'avions nécessaire). Cela permettrait par exemple à l'aéroport de proposer de nouveaux créneaux à des compagnies aériennes désireuses de créer de nouvelles liaisons aériennes au départ de l'aéroport (sans avoir à mettre en œuvre de coûteux travaux d'extension).

Cette section présente les résultats que nous avons démontrés pour répondre à cette question. Comme on le verra, cela s'est avéré plus compliqué que prévu, mais nous présentons tout de même quelques résultats préliminaires.

Par abus de langage, nous confondons "augmenter le nombre chromatique" et "augmenter la coloration" d'un graphe dans la suite.

Lemma 1 *Dans un graphe d'intervalles G , si ajouter un noeud à une composante connexe n'augmente pas sa coloration, alors ajouter ce noeud n'augmente pas le nombre chromatique de G .*

Proof Soit $G \subset G$ une composante connexe de G . Supposons que l'on ajoute le noeud n en le reliant à un sommet de G' , et que $\chi(G')$ n'augmente pas. Puisque qu'on ne change rien d'autre au graphe G , alors $\chi(G)$ n'augmente pas non plus.

Lemma 2 (Lemme pour le Théorème 2) *Soit G un graphe d'intervalles.*

*Tout intervalle ajouté à G , inclus dans l'union des intervalles d'une composante connexe est relié **uniquement** à cette composante connexe dans G .*

Proof Par l'absurde : supposons qu'un intervalle I inclus dans l'union des intervalles d'une composante connexe G' de G soit aussi lié à G'' une autre composante connexe de G . Alors il existe I'' appartenant à G'' tel que, I'' appartient à G' (car I est dans l'union des intervalles de G'). Donc G' et G'' sont reliées entre elles. Donc elles ne sont pas des composantes connexes différentes de G ce qui est absurde.

□

Theorem 1 *Soit G un graphe et G' une composante connexe non vide de G .*

Tout intervalle $I \subset G'$ en dehors de l'intersection des intervalles de G' , n'augmente pas la coloration de G' .

Proof Soit G' une composante connexe de G un graphe d'intervalles tel que $G' = (E_i)_{i \in [|1, n|]}$, où les E_i sont les intervalles représentant les noeuds de la composante. Par abus de langage on notera $G' = \bigcup_{i=1}^n E_i$.

Soit $A = \bigcap_{i=1}^n E_i$. Si A est non vide, alors $\chi(G') = n$. Soit I un intervalle à ajouter tel que $I \in G'$ et soit G'_2 la composante obtenue après ajout de I .

$$\begin{aligned} I \in G' \setminus A &\implies I \cap A = \emptyset \\ &\implies I \cap (\bigcap_{i=1}^n E_i) = \emptyset \\ &\implies \exists i \in [|1, n|], I \cap E_i = \emptyset \text{ car } I \neq \emptyset \text{ et } \bigcap_{i=1}^n E_i \neq \emptyset \text{ par hypothèse (on peut prendre la couleur de } E_i \text{ pour } I !) \\ &\implies \chi(G'_2) \leq \chi(G') \text{ car } I \text{ n'est pas relié à tous les } E_i. \end{aligned}$$

Ainsi, se placer dans $G' \setminus A$ donne un intervalle qui n'augmente pas la coloration de G' donc d'après le Lemme 2 : on peut trouver pour chaque composante connexe des intervalles qui n'augmentent pas la coloration du graphe G (Lemme 1) si on ajoute un noeud correspondant à cet intervalle. On supposera A non vide, si A est vide l'algorithme ne renverra pas d'intervalle possible pour G' .

□

Cet algorithme présente une première approche qui donne des résultats moyens (voir Section III.2). Nous avons pensé à d'autres manière d'aborder l'ajout de nouveaux avions au planning, mais ce problème, bien que facile à concevoir, nous a donné du fil à retordre : nous en parlerons lors de la présentation en classe.

Voici une idée dans la continuité de celle qui a été présentée, que nous n'avons pas implémentée (nous donnons une esquisse du principe, cela reste à améliorer) :

Notons $I = \max(\text{nombre d'intervalles se coupant en même temps})$. Pour pallier le cas où A est vide, il

```

algorithm ajout_intervalle(G) ;
begin
liste_solutions = []
Recherche des composantes connexes
Pour chacune des composantes connexes G' :
Si A est non vide :
Ajouter l'intervalle  $G' \setminus A$  à liste_solutions
retourner liste_solutions
end ;

```

Figure 6 – Algorithme "naïf" renvoyant les intervalles disponibles pour de nouveaux avions

faudrait que l'on considère les points en dehors de A' , l'union des intersections d'intervalles de cardinal l (autrement dit, A' est le regroupement des intersections entre un nombre d'intervalles égal à la coloration de G'). Mais cela était plus difficile à mettre en œuvre informatiquement, et nous avons manqué de temps pour l'implémentation.

La Figure 6 présente un pseudo-code pour l'algorithme proposé, déduit des propriétés précédemment démontrées.

Après l'introduction de ces résultats théoriques utiles à la justification de nos algorithmes, nous présentons une application au problème concret de gestion de l'aéroport de Strasbourg.

III - Application au problème concret (aéroports)

Maintenant que nous avons expliqué comment modéliser le problème, et présenté les résultats théoriques permettant de répondre à la question posée ("de quelle capacité a besoin l'aéroport ?"), nous présentons dans les paragraphes suivants l'implémentation des algorithmes (coloriage, prise en compte des retards...) à l'aide de la bibliothèque Python NetworkX et les résultats obtenus.

III.1 Implémentation Python et NetworkX

Cette première sous-section a pour objectif de présenter l'implémentation Python :

- de la récupération des données (heures d'arrivée et de départ)
- la construction du graphe et son coloriage à l'aide des fonctions NetworkX que l'on détaillera
- et enfin, l'affichage du graphe avec le coloriage obtenu.

Ces différents points pourront ensuite être utilisés pour complexifier le modèle avec la prise en compte des retards, qu'on présentera dans un second temps. On abordera ensuite la détection d'intervalles candidats pour ajouter des avions (sans augmenter le nombre chromatique du graphe).

III.1.1 Récupération des données, coloriage et affichage du graphe

Les données³ ont été enregistrées au sein d'un tableau Excel contenant plusieurs informations sur les vols du 4 Janvier 2024 à l'aéroport de Strasbourg-Enthzeim. Le tableau peut être visualisé figure 7.

Une fois ces données récupérées, il suffit de créer les intervalles à l'aide des colonnes "Départ" et "Arrivée". En pratique, on convertit l'heure en nombre de minutes écoulées depuis minuit (00h) afin de pouvoir construire les intervalles. Ainsi, un avion arrivant à 08h10 et repartant à 08h40 pour Marseille sera défini par l'intervalle [490, 520] (heure de départ, heure d'arrivée - en minutes à partir de 00h00).

Ensuite, on peut utiliser les fonctions et méthodes fournies par le module NetworkX :

- `interval_graph` pour créer le graphe d'intervalles

3. Source des données : <https://www.flightradar24.com/data/airports/sxb>

Destination	Arrivée	Départ	Identifiant	Compagnie	No vol	Provenance
Amsterdam	00:00	06:40	F-HRAV	Amelia	8R6768	Amsterdam 21h45
Montpellier	00:00	06:45	NC	Volotea	V72507	Rome 23h25
Nice	00:00	07:00	NC	Volotea	V72746	Nice 23h15
Marseille	08:10	08:40	NC	Volotea	V72514	Marseille 8h10
Tarbes	08:20	08:50	NC	Volotea	V72185	Bordeaux 8h20
Toulouse	08:20	08:50	NC	Volotea	V72530	Toulouse 8h20
Nantes	08:45	09:15	NC	Volotea	V72123	Nantes 8h45
Lyon	09:20	10:45	F-HBXF	Air France	AF1685	Lyon 9h20
Amsterdam	10:00	18:25	NC	Amelia	8R6774	Amsterdam 10h
Porto	10:45	11:10	NC	Ryanair	FR7698	Porto 10:45
Bordeaux	12:40	13:10	NC	Volotea	V72411	Tarbes 12h40
Rome	14:10	14:50	NC	Volotea	V72572	Montpellier 14h10
Nice	14:15	15:05	NC	Volotea	V72518	Nice 14h15
Bordeaux	16:00	16:35	NC	Volotea	V72611	Nantes 16h
Madrid	17:35	18:05	NC	Iberia	IB8241	Madrid 17h35
Nador	17:50	18:40	NC	Air Arabia Maroc	3O316	Nador 17h50
Barcelone	18:40	19:10	NC	Volotea	V72326	Rome 18:40
Nantes	20:20	20:55	NC	Volotea	V72641	Bordeaux 20h20
Toulouse	20:25	20:55	NC	Volotea	V72092	Toulouse 20h25
Marseille	21:25	21:55	NC	Volotea	V72762	Marseille 21h25
	21:45	23:59	NC	Amelia	8R6775	Amsterdam 21h45
	23:15	23:59	NC	Volotea	V72519	Nice 23h15
	23:30	23:59	NC	Volotea	V72327	Barcelone 23h30

Figure 7 – Activité de l'aéroport **SXB** le 03/01/2024

```
— G.nodes()
— greedy_color()
— spring_layout()
— draw_networkx().
```

Tout d'abord, on se pose la question de la *structure de données* pour représenter notre graphe. Au niveau de la complexité (pire cas), une liste d'adjacence demande, pour l'algorithme glouton, pour chaque sommet, de parcourir tous ses voisins. Cela se fait en une complexité pire cas $\mathcal{O}(|V|^2)$, même si en pratique c'est moins car chaque noeud du graphe n'est pas relié à **tous** les autres noeuds. Si l'on considère que le parcours des voisins se fait en temps $\mathcal{O}(1)$, alors l'algorithme glouton est de complexité $\mathcal{O}(|V|)$. En revanche pour la matrice d'adjacence, la complexité pire cas est bien $\mathcal{O}(|V|^2)$ car pour chaque noeud, il faut parcourir tous les noeuds du graphe pour savoir qui sont ses voisins. Ainsi pour ce problème, étant donné que l'on a des graphes relativement creux, on préfère utiliser une représentation des graphes d'intervalles par liste d'adjacence. C'est à priori la structure de données par défaut utilisée dans NetworkX (selon la documentation : “The graph internal data structures are based on an adjacency list representation and implemented using Python dictionary datastructures⁴.”).

Pour construire le graphe d'intervalles après avoir construit les intervalles, une fonction est déjà pré-implémentée : il suffit de faire

`graph = nx.interval_graph(intervals_strasbourg)` (`intervals_strasbourg` contient les intervalles définissant les arrivées et départs des avions sur la journée d'étude).

De plus, `G.nodes` permet d'accéder aux noeuds (les intervalles) du graphe `G`.

4. source : <https://networkx.org/documentation/stable/reference/introduction.html>

Ensuite, pour le coloriage, on utilise `greedy_color` avec la stratégie Left-Sort évoquée plus haut (Section II.2). Cette fonction correspond à l'algorithme glouton (*greedy algorithm*) précédemment présenté. Si l'algorithme glouton est déjà implémenté dans NetworkX, ça n'est pas le cas de la stratégie Left-Sort. Cependant la fonction `greedy_color` de NetworkX laisse la possibilité à l'utilisateur de définir sa propre stratégie. Nous avons donc écrit une stratégie permettant de trier les noeuds par heure d'arrivée croissante pour le coloriage :

```

1 def strategy_left_sort(G,colors):
2     """stratégie de coloriage: par ordre croissant d'heure d'arrivée ! Primordial
       → pour assurer l'optimalité dans le cas général.
    """
3
4     return sorted(list(G.nodes), key = lambda l : l[0])
5
6 # on peut ensuite appliquer greedy_color avec cette strategie:
7 coloriage = nx.greedy_color(G,strategy=strategy_left_sort)

```

Enfin, on peut représenter le graphe (colorié ou non) afin de visualiser les résultats obtenus. On utilise pour cela les fonctions `spring_layout()` (*Position nodes using Fruchterman-Reingold force-directed algorithm*, voir [2]) pour placer les noeuds dans l'espace, et `nx.draw_networkx` pour afficher le graphe. Voir le code Python fourni en parallèle de ce rapport pour l'application de ces fonctions en pratique. Un exemple des résultats obtenus pour l'aéroport de Strasbourg (avec coloration) est visible Figure 8.

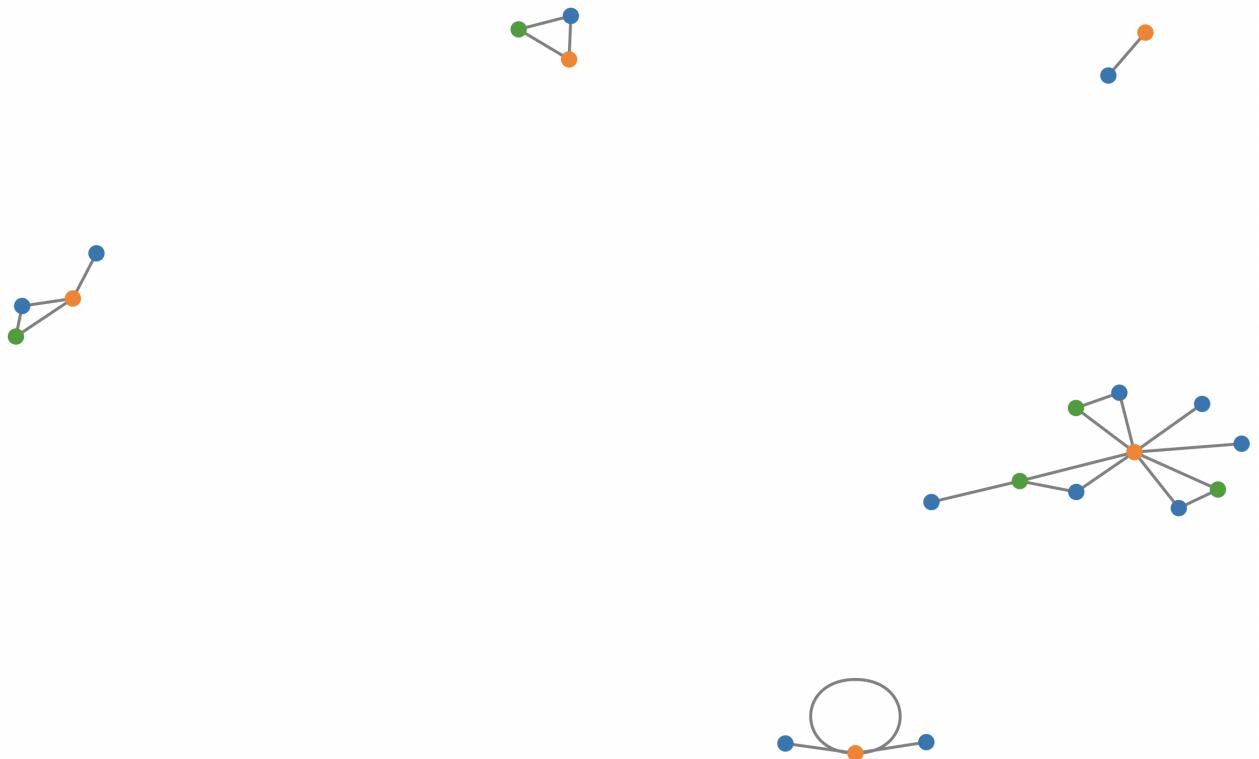


Figure 8 – Graphe colorié renvoyé par la méthode `nx.draw_networkx`

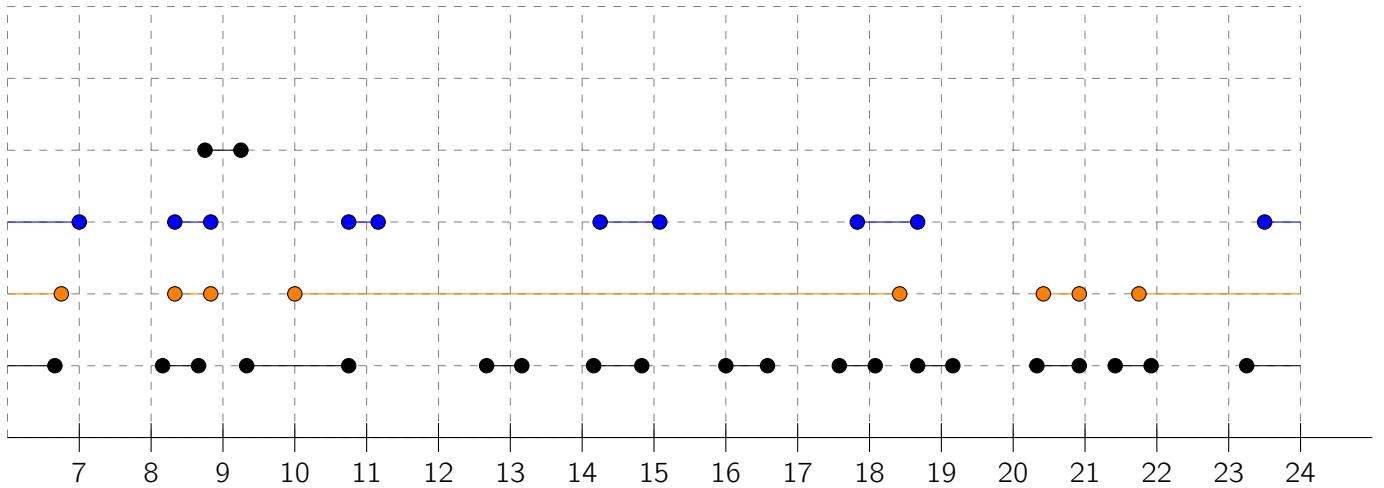


Figure 9 – 3-Coloration donnée par l’algorithme glouton (méthode LS) sur l’exemple de l’aéroport de Strasbourg (heure en abscisse).

III.1.2 Prise en compte des retards

Nous avons implémenté les deux approches pour les retards. Pour la première, il suffit d’ajouter à toute la colonne “Départ” du *dataframe* Pandas le retard moyen, nous ne nous attarderons pas sur ce sujet.

En revanche pour la modélisation d’un retard exceptionnel, on regarde pour chaque nœud, l’impact d’un potentiel retard (entre 0 et 200 minutes de retard). Nous l’avons implémenté avec une boucle `for`. Le reste est similaire au traitement du retard moyen, les mêmes fonctions sont utilisées pour colorier le graphe et vérifier si oui ou non le retard augmente le nombre chromatique par rapport au graphe d’origine.

III.2 Analyse des résultats

III.2.1 Problème initial (sans retards)

Dans le cadre du problème initial, on obtient avec l’algorithme glouton un nombre chromatique de 3 pour notre graphe d’intervalles. Ce qui signifie qu’on pourrait n’utiliser que 3 emplacements (+2 emplacements d’urgence) pour gérer les flux de cette journée (si tous les vols sont à l’heure). La Figure 9 montre le graphe de la Figure 3, colorié par l’algorithme glouton LS.

III.2.2 Problème avec retards

L’étude du problème avec la prise en compte des retards nous donne plusieurs résultats intéressants pour la prise de décision finale à savoir le nombre d’emplacements à avoir. Tout d’abord on observe que l’ajout d’un retard de 0 à 3h peut augmenter le nombre chromatique χ jusqu’à 4 au maximum.

De plus cette augmentation de χ ne s’effectue qu’à partir d’un retard de 65 minutes, ainsi on sait que pour un retard peu important il n’est pas nécessaire d’aménager un nouvel emplacement.

Enfin on est capable de connaître les vols critiques qui par leurs retards risquent de surcharger l’aéroport : ce sont dans l’ordre de la journée les vols 3, 4, 5, 6, 9, 10, 11, 12, 13, 17, 18 et 19 sur les 22 vols de la journée.

III.2.3 Etude de l’ajout de vols

L’étude sur la possibilité d’ajouts de vols nous montre quels seraient les intervalles où des ajouts de vols seraient possible sans augmenter le nombre chromatique. Par exemple si une compagnie aérienne

souhaite ouvrir une nouvelle ligne au départ de Strasbourg. Pour la configuration initiale sans les retards, l'algorithme nous fournit les intervalles suivants : (400, 420) et (1220, 1225)) (intervalles donnés en minutes) comme candidats.

Si on considère maintenant qu'un avion doit passer un minimum de 30 min au sol pour pouvoir débarquer et embarquer ses passagers, (intervalles de longueur minimum 30), on en conclut qu'il n'est possible d'utiliser cette méthode pour ajouter des vols. Cette approche ne prend également pas en compte les éventuels retards de ces avions ou d'autres avions qui pourraient interférer. Il faudrait alors pousser l'étude plus loin pour chercher éventuellement les intervalles qui garantissent de ne pas surcharger l'aéroport même en cas de retards. Pour cela, il faudrait améliorer l'algorithme en prenant en compte les cas où A est vide (comme précisé plus haut, voir Section II.3).

Conclusion



Figure 10 – Aéroport de Strasbourg, vue satellite (©Google)

Cette étude nous pousse à la conclusion suivante avec les hypothèses choisies : il faudrait 4 emplacements d'avions pour satisfaire le flux et gérer les imprévus (plus les 2 emplacements pour les atterrissages d'urgence). Evidemment cette étude n'est basée que sur le planning de l'aéroport sur une seule journée choisie aléatoirement, et il faudrait aussi prendre en compte les journées où le flux est le plus important, ou encore faire une étude sur une durée plus longue.

Finalement, lorsque nous examinons l'aéroport, il est en réalité composé d'environ 10 emplacements (en rouge sur la figure 10), ce qui peut s'expliquer d'abord par le fait qu'il doit y avoir 2 emplacements pour gérer les atterrissages d'urgence. Cela s'explique aussi par les hypothèses simplificatrices de notre modélisation (on omet en effet tout ce qui est aviation de tourisme, militaire, le fret...), et met en exergue les limites du modèle.

Cependant, on est tout de même amenés à se questionner sur l'utilité et le besoin réel derrière ce grand nombre de places. En effet, à travers cette étude il est possible de voir que l'aéroport (sur une journée classique) est capable de n'utiliser que seulement 40% de sa capacité totale d'accueil de vols.

Références

- [1] Documentation NetworkX, [accès en ligne]
- [2] Fonction greedy_color de NetworkX, [accès en ligne]
- [3] Kosowski, Adrian and Krzysztof Manuszewski, "Classical Coloring of Graphs.", (2008), page 8, [accès en ligne]
- [4] Exercice d'entraînement sur le coloriage de graphes, Paul Melotti, 2014, [accès en ligne]
- [5] Kierstead, H. A. ; Trotter, W. T. (1981), "An extremal problem in recursive combinatorics"
- [6] Stephan Olariu, An optimal greedy heuristic to color interval graphs, [accès en ligne]