Optimisation des réseaux aériens : analyse et sélection de nouveaux marchés

Assia Kamal-Idrissi

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Optimisation du réseau aérien : analyse et sélection de nouveaux marchés

Assia KAMAL-IDRISSI

Laboratoire d'Informatique, Signaux et Systèmes de Sophia Antipolis (I3S)
UMR7271 UNS CNRS

Présentée en vue de l'obtention du grade de docteur en Informatique d'Université Côte d'Azur

Dirigée par :
Jean-Charles REGIN

Co-encadrée par :
Arnaud MALAPERT

Soutenue le : 19 octobre 2020

Devant le jury, composé de :

Marie-José Huguet, PR, Institut National des Sciences Appliquées de Toulouse
Xavier Lorca, PR, Institut Mines-Télécom d'Albi
Jean-Charles Régin, PR, Université Côte d'Azur
Arnaud Malapert, MCF, Université Côte d'Azur

MILANAMOS
Big data insights for the travel industry

cnrs

Optimisation du réseau aérien : analyse et sélection de nouveaux marchés

Composition du jury :

Rapporteurs

Marie-José Huguet, Professeure des Universités, Institut National des Sciences Appliquées
Xavier Lorca, Professeur des Universités, Institut Mines-Télécom

Examinateurs

Jean-charles Régin, Professeur des Universités, Université Côte d'Azur
Arnaud Malapert, Maître de conférences, Université Côte d'Azur

Invité

Carine Fédèle, Maître de conférences, Université Côte d'Azur
Christophe Imbert, Président, Milanamos
Rémi Jolin, Directeur technique, Milanamos

A mes chers

parents.....à mon cher

mari.....à mes petits anges A_2 :

Ahmed-Ali et Aline.....je

dédie ce travail ♡

Remerciements

En premier lieu, je remercie le très miséricordieux et TOUT-PUISSANT "ALLAH" qui m'a donné la force et l'audace pour compléter cette thèse.

Je tiens à remercier Pr Marie-José Huguet et Pr Xavier Lorca, qui m'ont fait l'honneur d'être les rapporteurs de ces travaux de thèse.

Je souhaite exprimer toute ma gratitude à mon directeur de thèse, Pr Jean-Charles Régin pour avoir accepté de diriger ce travail, pour la qualité de l'encadrement et du soutien qu'il m'a accordés pendant cette thèse.

Mes plus grands remerciements vont à mon encadrant, Arnaud Malapert, avec qui j'ai eu la chance de travailler pendant ma thèse. Arnaud m'a appris énormément d'un point de vue scientifique, parfois dans des domaines que je n'aurais jamais pensé explorer. Et plus que des connaissances techniques, il m'a inculqué une véritable démarche scientifique, une manière de penser. Plus personnellement, je le remercie grandement pour ses conseils, ses encouragements, et pour les nombreux conseils et les riches discussions que nous avons eues ensemble avec Carine Fidèle. Vous avez fait de ces trois années et demi une bien belle expérience.

Carine, je ne te remercierai jamais assez pour tout le soutien que tu m'as donné durant les dernières années de cette thèse sur le plan scientifique aussi bien que sur le plan humain. Tu n'es pas uniquement une co-encadrante pour ma thèse, tu es une amie...

Je remercie également Christophe Imbert pour m'avoir accueillie à Milanamos et pour avoir encouragé l'esprit d'innovation à travers ce travail de thèse. Je remercie également Rémi Jolin pour son aide et ses conseils, sans oublier Valentin qui, bien que tôt parti de l'entreprise, m'a beaucoup aidé au début de la thèse. Chez Milanamos, j'ai également côtoyé plusieurs autres personnes auprès desquelles j'ai beaucoup appris sur le secteur aérien. Merci donc à Christophe Ritter, Natalya et Stéphane.

Un grand merci à tous les amis et membres du Laboratoire I3S qui m'ont chaleureusement accueilli et m'ont permis de travailler dans une ambiance aussi conviviale. Merci tout spécialement à Sandra, Rémi, Nico, Lætitia, Lætitia 2, Samvel, Oussama, Guillaume, Heytem, Ophélie, Ingrid, Piotr, Sara et Marie pour son aide et sa bonne volonté. C'était un plaisir pour moi de vous avoir connus.

Le meilleur est pour la fin dit-on. Je suis et je serai toujours reconnaissante à mes parents sans qui je n'en serai pas là aujourd'hui. Maman, papa, je vous remercie pour votre amour et votre soutien inconditionnel tout au long de cette carrière. Je vous aime. Merci à ceux que j'aime : mes beaux-parents, mes sœurs, mes neveux et mes nièces d'avoir toujours cru en moi. Sans vous tous, ce travail n'aurait certainement pas vu le jour.

Enfin, tous les mots ne suffiront pas pour remercier la personne qui m'a le plus encouragé à faire cette thèse, qui a toujours cru en moi, qui a été présente à mes côtés dans les hauts et les bas, dans les joies et les tristesses, qui m'a appris à prendre du recul et à faire la part des choses, mon cher mari Mohamed El Afouani...Merci pour ton soutien inconditionnel, ton aide, tes encouragements, tes conseils et tes critiques constructives tout au long de la thèse....Cette thèse est aussi la tienne...

Je dédie également la thèse à mes adorables enfants Ahmed-Ali et Aline qui ont supporté mon absence et mes sauts d'humeur... Mes trésors, un jour vous lirez peut être la thèse de votre maman et vous en serez fiers...

Résumé

Les problèmes rencontrés dans l'industrie aérienne sont divers et compliqués. Leur résolution réduit les coûts et maximise les revenus tout en améliorant la qualité de service, par exemple, en capturant de nouveaux passagers sur des vols existants ou sur de nouveaux marchés. La sélection des nouveaux marchés permet de définir la structure du réseau à opérer, et d'estimer le flux des passagers, leurs choix d'itinéraires ainsi que les revenus et les coûts impliqués par ces décisions. Nos travaux concernent l'amélioration du calculateur de parts de marché dans l'application `PlanetOptim` de la startup *Milanamos*. Cet outil permet aux décideurs des aéroports et des compagnies aériennes d'analyser l'historique des données et de simuler des marchés afin de trouver une opportunité économique. Ces travaux sont orientés vers les niveaux de décision stratégiques et tactiques. Grâce à une analyse poussée des données, le réseau aérien a pu être modélisé par un graphe indépendant du temps stocké dans une base de données orientée graphe `Neo4j`. Tout d'abord, nous avons défini le Flight Radius Problem, dans le cadre de la sélection des marchés, dont la résolution permet de déterminer un sous-réseau centré autour d'un vol pour lequel les parts de marchés du vol sont non négligeables. Plusieurs méthodes de résolution ont été proposées basées sur des requêtes ou des algorithmes de plus courts chemins couplés à des techniques d'accélération et de parallélisme. Nos algorithmes identifient rapidement un ensemble de marchés prometteurs centré sur un vol. Ensuite, pour accélérer le calcul des parts des marchés, nous avons proposé une nouvelle méthode basée sur le modèle logit. L'intégration de la théorie des graphes dans les bases de données ouvre de nouvelles perspectives pour l'analyse et la compréhension de grands réseaux.

Mots-clés : transport aérien, choix d'itinéraire, graphe indépendant du temps, base de données orientée graphe, plus court chemin, parallélisme, logit.

Abstract

In the airline industry, problems are various and complicated. Solving these problems aims at reducing costs and maximizing revenues. Revenues can be increased while improving the quality of service. For example, one way is to catch new passengers on existing flight connections or on new markets. The selection of new markets consists in determining network structure to operate, and to estimate passengers flow, their choice of itineraries as well as incomes and costs incurred by these decisions. Our research is about improving market planner engine. *Milanamos* develops an application for the analysis and simulation of markets intended for airports and airlines. It offers its customers a decision-making tool to analyze historical data and to simulate markets in order to find an economic opportunity. This project takes place earlier in the decision process. Thanks to a thorough data analysis, the air transport network could be modelized as a time-independent graph and stored in the `Neo4j` graph database. The first step consists in defining the Flight Radius problem which resolution allows to determine a sub-network centered around a flight for which market shares of the flight are meaningful. Several methods have been proposed based on queries or on shortest path algorithms combined with acceleration and parallelism techniques. Our algorithms identify some new markets for a flight. The second step consists in calculating market shares, a method has been proposed based on logit model to reduce a combinatorial on itineraries to evaluate. Combining graph theory with databases offers new opportunities for analyzing and studying large networks.

Keywords: air transportation, itinerary choice, time-independant graph, graph database, shortest-path, parallelism, logit.

Table des matières

Contributions

CHAPITRE 1

Introduction

Le transport aérien connaît depuis plusieurs décennies un très fort taux de croissance. En 2018, le trafic de passagers a augmenté de 6,4% : 8,8 milliards de passagers se sont déplacés en avion [1]. Cette croissance a poussé les compagnies aériennes à améliorer leur qualité de service à travers la maîtrise de plusieurs facteurs sensibles comme la fréquence, le prix, ou la concurrence.

Afin de capturer un flux important de passagers, les compagnies aériennes proposent des vols correspondant aux attentes de leurs passagers ; elles se concentrent principalement sur la planification des vols. Cette étape est l'un des éléments importants et implique des décisions complexes. Elle prend en compte la demande des passagers, les caractéristiques des aéroports et des avions, puis génère une sélection de segments de vol maximisant leur profit en fonction des contraintes de ressources (capacité des avions et des aéroports, heures de travail maximales, temps minimal au sol, *etc.*). Un segment de vol est défini par trois attributs [Belobaba et al., 2015] : une paire origine-destination (OD), une paire heure de départ-heure d'arrivé et le type d'avion.

L'étape de la planification des vols vise à répondre aux questions suivantes :

1. Quelles destinations la compagnie peut-elle desservir ?
2. À quelle fréquence la compagnie peut-elle offrir ?
3. À quelle date peut-elle proposer ?
4. Quelle capacité fournir sur chaque vol ?
5. Quels sont les choix des vols concurrents ?
6. Quels sont les horaires des vols proposés ?

La décision prise est très importante pour une compagnie aérienne, la qualité du service et les prix proposés influencent l'attraction des passagers. Pour cela, la compagnie aérienne doit prendre en considération le choix des passagers. En fait, elle doit tenir compte des critères tels que le prix du billet, le temps de trajet, mais également le type de vol. Par exemple, un homme d'affaires est intéressé par le temps de voyage, un étudiant souhaite minimiser le coût dépensé et un visiteur souhaite éviter le vol avec correspondance. Les passagers ont donc des préférences différentes sur les critères qui sont propres à leurs cultures et besoins.

Ce processus consiste à prendre des décisions à différentes étapes concernant l'ajout de nouveaux vols ou l'ouverture de nouvelles routes. Cette dernière nécessite une prévision de la demande du marché desservi qui n'est pas basée sur des données historiques puisqu'inexistantes. Par conséquent, la demande peut être calculée avec des modèles de part de marché et la meilleure décision est choisie. Quand elle possède un historique des données, en revanche elle se base sur des techniques de l'apprentissage automatique.

1. Rapports annuels de IATA et ICAO.

Le problème de la planification des vols concerne la détermination d'un ensemble de paires origine-destination (OD) puis le choix des heures d'arrivée et de départ d'un avion, compte tenu de certaines contraintes, qui minimisent les coûts d'exploitation et maximisent les revenus d'une compagnie aérienne. Ce problème suscite la sélection des marchés, mais certaines considérations opérationnelles et économiques doivent être prises en compte pour optimiser un réseau d'une compagnie aérienne. Par conséquent, une prévision de la demande est nécessaire pour estimer la demande de passagers pour chaque itinéraire en tenant compte de la demande du marché, puis déterminer le coût prévu et enfin calculer le temps de vol et les revenus. La prévision de la demande est l'élément clé alors qu'une compagnie aérienne prévoit d'ajouter un nouveau vol. En effet, la compagnie aérienne doit estimer le nombre total de passagers qui sont prêts à emprunter ce nouvel itinéraire, en particulier si l'itinéraire est déjà exploité par d'autres compagnies aériennes. Il doit fournir suffisamment de sièges pour satisfaire la demande.

L'ajout d'un nouveau vol au réseau actuel est compliqué et implique plusieurs décisions [Barnhart and Smith, 2012, Garrow, 2016] :

1. Les décisions sur les horaires doivent être prises en fonction des vols existants de la compagnie. Cette dernière doit décider quel marché est concerné par cet ajout après avoir considéré les concurrents.
2. Mesurer la rentabilité de la route, déterminer la rentabilité économique de l'ouverture d'un nouveau vol. S'il s'agit d'une nouvelle destination, les autres coûts doivent alors être pris en compte, y compris le coût supplémentaire de l'aéroport.

Pour l'estimation de la rentabilité des routes, les modèles de planification de réseau (en anglais *Network planning*), également appelés modèles de rentabilité de routes (en anglais *Route profitability*) sont utilisés. Ils aident par exemple les compagnies aériennes à mettre en œuvre des scénarios de fusion et d'acquisition, une analyse des itinéraires, des scénarios de partage de code, des études sur le temps minimum de connexion *MCT* (en anglais *Minimum Connecting Time*), des études sur l'élasticité-prix, des études de localisation des *hubs* et des décisions d'achat d'appareil.

La rentabilité concerne le flux de passagers sur le niveau marché ainsi que sur un niveau plus fin, l'itinéraire. Dans la réalité, lorsqu'un passager veut réserver un vol, plusieurs itinéraires sont proposés. La question qui se pose est de savoir sur quels types de critères de choix le passager se basera pour choisir l'itinéraire le plus attrayant ?

Le choix des passagers peut être ainsi modélisé en tenant compte de ses préférences. Pour cela, il y a des modèles économiques qui estiment la probabilité qu'un passager sélectionne un itinéraire spécifique reliant une paire d'aéroports [Jacobs et al., 2012] (en anglais *Itinerary Market Share Models*) :

— Modèle qualitatif : modèle *QSI* (en anglais *Quality of Service Index*).

— Modèle de choix discret, comme le modèle basé sur la régression logistique (*logit*).

Cependant, la plupart de ces modèles économiques ne tiennent pas compte de la masse énorme de données. Par ailleurs, les chercheurs n'étudient qu'une seule paire OD où il faut énumérer tous les itinéraires possibles reliant la paire origine-destination (OD) sans avoir testé la faisabilité de l'itinéraire.

Aujourd'hui il demeure encore difficile de traiter le problème de la planification des vols en temps réel malgré la résolution séquentielle qui réduit bien évidemment la complexité du problème. Mais elle peut générer des solutions de moindre qualité et même loin des désirs de la compagnie aérienne. En effet, les décisions prises au niveau de la planification des vols ne tiennent pas

compte des dépendances des autres problèmes, notamment, l'estimation des parts de marchés. Vue l'importance de l'interaction entre la sélection des marchés et l'estimation des parts de marchés, il est préférable de résoudre simultanément ces deux problèmes : mais cela demeure un grand défi en raison d'un grand volume de données qui sont non structurées et incomplètes. En effet, lorsque le nombre de marchés augmente, la taille du problème et le temps de calcul augmentent rapidement puisqu'il s'agit d'évaluer simultanément l'ensemble des itinéraires des marchés lors de la résolution, qui est potentiellement exponentiel. C'est un des défis qui nous intéressent dans cette thèse : estimer les parts de marchés sur un grand réseau pour plusieurs marchés en temps réel.

1.1 Analyse du problème industriel et scientifique

1.1.1 Contexte industriel

L'entreprise *Milanamos* développe une application d'analyse et de simulation des marchés aériens, `PlanetOptim`. L'ajout d'un nouveau service par une compagnie consiste à déterminer une origine et une destination à desservir, puis à choisir la fréquence et l'horaire qui maximisent la qualité de service. C'est dans ce contexte que *Milanamos* propose à ses clients les modèles de planification des routes. Elle se base sur le modèle *QSI* pour estimer la part de marché capturée. Le nouveau service peut être :

1. ouvrir une nouvelle destination ;
2. ajouter un nouveau vol.

1.1.2 Problématique

Étant donné une origine et une destination, l'application `PlanetOptim` affiche tous les aéroports reliés par un arc vers l'origine ou issus de la destination. Ce sous-graphe est souvent trop grand et dense pour être visualisé correctement dans l'application. De plus, certains chemins du sous-graphe n'ont aucun intérêt en pratique. Par exemple, un trajet New York-Paris-Los Angeles n'est pas une alternative réaliste pour relier New York à Los Angeles. Ensuite, l'application estime la part de marché capturée par le nouveau vol entre la paire `OD` avec le modèle *QSI*.

Le modèle actuel estime la part de marché en tenant compte des vols avec une seule correspondance. Par ailleurs, il ne prend pas en considération les marchés influencés par l'ajout de ce nouveau vol. En effet, le processus d'estimation est compliqué et long car les formules économiques ne passent pas à l'échelle.

Notre objectif est d'aider le décideur à sélectionner des marchés qui représentent l'ensemble des paires `OD` telles que la part de marché est non négligeable, puis d'estimer la part de marché. Ce calcul prend en compte l'ensemble des itinéraires reliant l'origine à la destination ainsi que l'ensemble des marchés impactés par le nouveau service, et il doit être réalisé en quelques secondes dans la pratique.

C'est dans ce cadre que nous avons défini un nouveau problème, appelé : *Flight Radius Problem* (*FRP*). C'est un problème d'aide à la décision ayant un aspect multicritère qui permet de déterminer un sous-graphe de chemins valides. Un chemin valide passe par l'arc entre l'origine et la destination en limitant, par exemple, la perte de temps ou le surcoût par rapport aux meilleurs chemins. Cela permettra d'améliorer la visualisation et accélérer le calcul des parts de marché.

1.2 Objectifs de la thèse

1.2.1 Objectifs généraux

Ce travail de thèse a pour but de fournir un outil d'aide à la décision à *Milanamos* qui pourra ainsi le mettre en œuvre afin que ses clients puissent prendre de meilleures décisions pour trouver une opportunité économique. La décision peut être :

1. modifier la fréquence ;
2. proposer un nouveau vol ;
3. proposer un vol partagé.

Tout d'abord, la solution du problème *FRP* consiste à écarter les marchés dont la part de marché est nulle et ensuite à évaluer les parts de marchés sur ces marchés.

1.2.2 Objectifs spécifiques

Afin de mener à terme une telle étude, l'élaboration d'objectifs spécifiques est nécessaire. Ainsi, ceux-ci peuvent être décrits comme suit :

— analyser les données aériennes de *Milanamos* ;
— modéliser et analyser le réseau aérien ;
— formuler et résoudre le *FRP* ;
— modéliser et évaluer le calcul de la part de marché.

Il y a deux parties majeures dans cette étude : la première consiste à trouver un sous-graphe de chemins valides, ce qui permet d'améliorer la visualisation et détecter les marchés potentiels notamment en gardant les itinéraires valides ; la seconde partie permet de calculer les parts de marchés sur ce nouveau graphe. Ce qui permettra par la suite d'accélérer les calculs sur `PlanetOptim`.

1.3 Complexité du problème

Si la décision d'ajuster ou non les horaires des vols semble évidente, le choix des vols à modifier n'est pas évident. Il est une chose de décider d'inclure une destination, il en est une autre de prendre en considération les marchés impactés par cette décision. Pour cela, il existe un large choix de marchés à opérer, quand et comment tel que chaque marché à une taille différente. En outre, le vol est destiné aux passagers : il faut donc tenir compte de leurs préférences. On propose de relâcher l'aspect temps, ce qui permet de réduire la complexité, et ne garder que la fréquence pour le calcul des parts de marché. Le problème *FRP* consiste ainsi à pré-sélectionner un ensemble de nouveaux marchés selon les préférences des passagers.

Pour ce type de problématique, il existe plus de facteurs subjectifs que de facteurs objectifs. C'est une problématique qui fait appel à l'évaluation des critères de décideurs vis-à-vis des passagers et l'utilisation des outils d'aide à la décision. D'un côté, aucune compagnie aérienne ne pourrait optimiser parfaitement son réseau. De l'autre, aucun outil d'aide à la décision ne pourrait tenir compte de tous les facteurs qui influencent l'ajout d'un nouveau service et offrir la meilleure solution. Il serait d'ailleurs bizarre de quantifier exactement les conséquences de cet ajout. Par

conséquent, une bonne manière de résoudre le problème est d'utiliser des outils d'aide à la décision qui considèrent un maximum de facteurs objectifs et d'évaluer les facteurs subjectifs en mettant en œuvre les différents scénarios possibles sur la base d'une expérience solide de l'industrie aérienne.

1.4 Méthodologie de la résolution

Nous avons commencé par analyser les données aériennes de *Milanamos*. En effet, les données réelles présentent souvent des soucis et leur qualité joue un rôle important sur le résultat attendu et dans l'expérience utilisateur. Cette étape nous a ainsi permis de mieux estimer la taille et la complexité de notre problématique.

Pour modéliser le réseau aérien par un graphe, nous avons proposé une nouvelle approche basée sur le modèle indépendant du temps qui modélise le temps de transfert. Le graphe généré est un graphe condensé dans lequel les nœuds représentent les aéroports et un arc indique l'existence d'un vol entre les deux aéroports. Le transfert est représenté par un arc.

Le graphe représentant le réseau aérien est stocké dans une base de données orientée graphe : `Neo4j` [Technology, 2017]. Dans notre contexte, une base de données orientée graphe est plus adaptée à la résolution des problèmes auxquels est confronté *Milanamos* que la base de données actuelle.

Nous avons formulé le problème *FRP* de la façon suivante : trouver le sous-graphe maximal, en termes de nœuds dans lequel au moins un chemin valide passe par chaque nœud. Nous avons introduit la notion de regret pour modéliser les préférences des passagers.

Pour résoudre le problème *FRP*, nous avons testé la méthode proposée par `Neo4j`. Suite à ces limites, nous avons proposé quatre méthodes qui se basent sur les algorithmes fondamentaux des plus courts chemins couplés à des techniques de parallélisme et d'accélération.

Pour réduire la combinatoire dans le calcul des parts de marché, nous proposons une méthode qui est basée sur la technique d'accélération de calcul de plus court chemin : *Hub labeling*. L'idée est de calculer les plus courts chemins qui passent par des nœuds intermédiaires dits *hubs*. Or, le réseau aérien est caractérisé par la présence des aéroports de type *hub*. Notre méthode commence d'abord par identifier les hubs selon les critères métiers, et ensuite, par calculer les plus courts chemins passant par ces hubs. Ceci nous permettra de réduire le nombre de chemins sur chaque marché.

1.5 Organisation du manuscrit

Le manuscrit comprend les chapitres suivants (voir la figure 1.1).

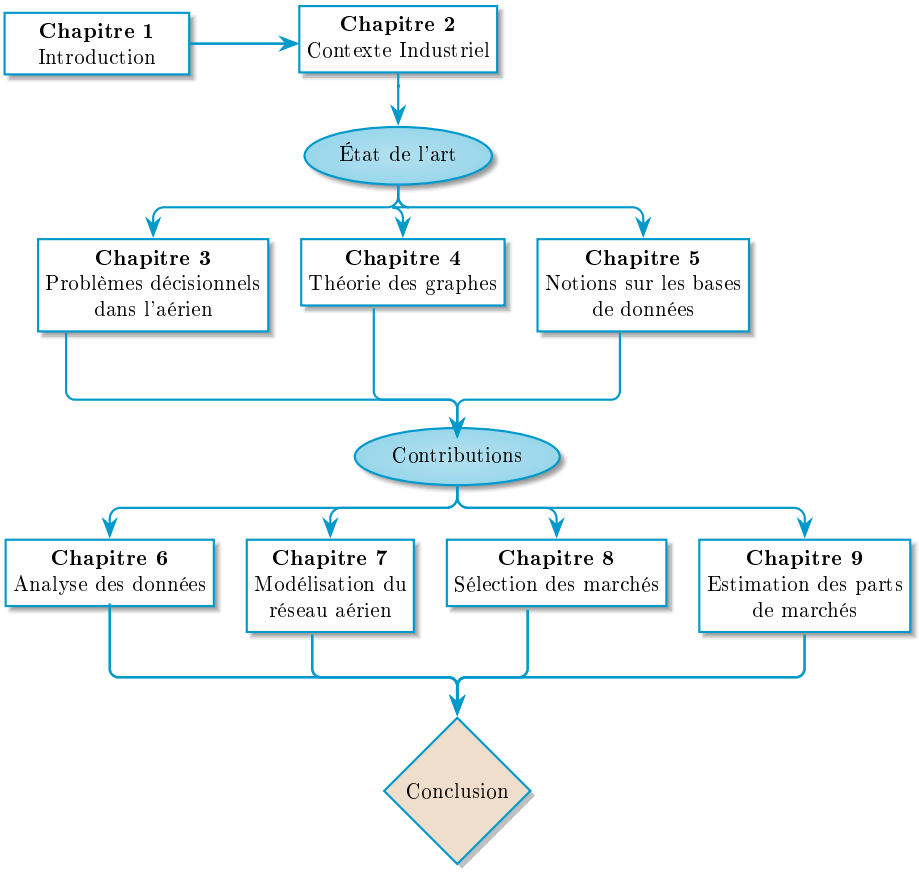


FIGURE 1.1 – Structure du manuscrit.

2. Contexte industriel - dans ce chapitre, nous présentons le contexte industriel de la société *Milanamos*. Nous commençons par présenter son produit `PlanetOptim`. Ensuite, nous décrivons le module le plus pertinent de ce travail de thèse. On retrouve dans ce chapitre, l'analyse des besoins et les motivations pour ce travail de thèse.

3. Problèmes décisionnels dans l'aérien - ce chapitre est consacré à la description du processus décisionnel lors de la planification des vols dans le transport aérien. Nous présentons les différents problèmes auxquels se confrontent les compagnies aériennes et le périmètre du travail de la thèse. Ce chapitre présente également une revue de la littérature dans le domaine de l'aérien qui regroupe les modèles d'optimisation et les approches proposés pour traiter le problème de développement du réseau des routes. Les modèles d'estimation de part de marché à différents niveaux de décision sont également abordés et constituent les principaux axes de recherche proposés pour traiter de manière pragmatique et réaliste le problème de recherche. Parmi ces différents modèles, nous avons retenu pour les travaux de la thèse, le modèle de choix discret : le modèle multinomial (en anglais *Multinomial Logistic Regression*).

4. Théorie des graphes - ce chapitre présente une revue de la littérature sur la théorie des graphes. Nous présentons ainsi les algorithmes fondateurs, `Dijkstra` et `Bellman` ainsi que les différentes techniques d'accélération proposées dans la littérature. Parmi ces techniques, nous avons opté pour l'algorithme de *hub labeling* qui a connu un grand succès [Delling et al., 2014, Bast et al., 2016].

5. Notions sur les bases de données - dans ce chapitre, nous abordons quelques notions sur les bases de données. Nous présentons les différents types qui existent actuellement, à savoir, les bases relationnelles et les bases `NoSQL`. Parmi les bases `NoSQL`, il y a les bases orientées graphe. Nous avons choisi la base orientée graphe `Neo4j` pour stocker notre graphe.

6. Analyse des données aériennes - ce chapitre est consacré principalement à l'analyse des données aériennes. Les données provenant du monde réel présentent souvent des soucis. En fait, résoudre une problématique réelle nécessite de faire une certaine analyse pour proposer des hypothèses, comprendre l'évolution des données dans le temps et bien choisir la méthode adéquate pour résoudre le problème de recherche.

7. Modélisation du réseau aérien - ce chapitre présente la modélisation du réseau aérien. Nous commençons par une revue de la littérature des différentes approches pour modéliser un réseau de transport. Ensuite, nous décrivons le processus de la génération du graphe. Finalement, nous présentons l'analyse du réseau aérien ainsi notre méthode pour identifier les hubs par des indicateurs métiers déterminés à partir de la revue de la littérature et sélectionnés selon les données disponibles. Ce réseau modélisé sera utilisé par la suite pour l'estimation des parts des marchés.

8. Sélection des marchés - ce chapitre est consacré essentiellement à la formulation mathématique du problème *FRP* défini pour répondre au problème de la sélection des marchés. Nous présentons l'exemple de motivation en rappelant les limites de `PlanetOptim`, le positionnement par rapport à l'optimisation multicritère, les méthodes de résolution et les tests expérimentaux réalisés.

9. Modèle d'estimation des parts de marchés - ce chapitre commence par présenter le modèle actuel utilisé par *Milanamos* pour estimer les parts de marchés, qui est basé sur le modèle *QSI* ainsi que ses limites. Ensuite, nous introduisons notre modèle proposé, basé sur le modèle multinomiale *logit*. Ce modèle permet d'estimer les parts des marchés identifiés par le *FRP* sur le graphe condensé. Finalement, nous concluons par les études de cas réalisées pour évaluer notre approche.

Notations

Modèles de choix

X	variables explicative
S	part d'un itinéraire
J	ensemble des itinéraires d'un marché
QSI	score d'un itinéraire
η^{S}_{QSI}	élasticité de la part par rapport au score
ϵ	erreur de l'observation
U	fonction de l'utilité d'un itinéraire
V	utilité observée d'un itinéraire
i	alternative i
j	alternative j
n	individu (passager)
k	attribut (critère) de l'alternative
N	ensemble des individus
K	ensemble des critères d'une alternative
y	variable binaire pour le choix
$G(0, \gamma)$	loi de *Gumbel* avec un mode 0 et un écart-type γ
F	fonction de répartition
f	fonction de densité
D	domaine de définition d'une variable
A	événement

Théorie des graphes

$\mathcal{X}$	ensemble des nœuds
n	nombre de nœuds dans le graphe
$\mathcal{U}$	ensemble des arcs
m	nombre d'arcs dans le graphe
G	graphe
x	exemple de nœud
y	exemple de nœud
u	exemple d'arc
$\mathcal{W}$	ensemble des poids
w	poids (ou coût) d'un arc
ω	extrémité
ω^+	extrémité initiale
ω^-	extrémité finale
Γ	ensemble des successeurs
Γ^{-1}	ensemble des prédécesseurs
d_G	degré d'un nœud
d^+	degré sortant
d^-	degré entrant
D	densité d'un graphe
G'	sous-graphe
$\mathcal{A}$	ensemble des nœuds d'un sous-graphe
$\mathcal{V}$	ensemble d'arcs d'un sous-graphe
$\mathcal{A}$	matrice d'adjacence
$\mathcal{M}$	matrice d'incidence
μ	chaîne
P	chemin
l	longueur d'un chemin
$l^\star$	longueur du plus court chemin
d	borne supérieure sur la longueur du plus court chemin
p	parent d'un nœud
S	statut d'un nœud
T	arbre du plus court chemin
B	ensemble des nœuds pour la recherche en arrière
F	ensemble des nœdus pour la recherche en avant
L	ensemble des étiquettes d'un nœud
L_f	ensemble des étiquettes en avant
L_b	ensemble des étiquettes en arrière
$\mathcal{O}(1)$	complexité constante
$\mathcal{O}(nm)$	complexité polynomiale
h	heuristique pour l'algorithme A^*

Schéma et analyse des données aériennes

d_{se}	distance de la collection `segment`
d_{sc}	distance de la collection `schedule`
$E(d_{se}, d_{sc})$	erreur absolue
δ	écart entre le nombre de passagers de *Milanamos* et celui de la *DGAC*

Modélisation du réseau aérien

o	aéroport origine
d	aéroport destination
$\mathcal{A}$	ensemble des aéroports
$\mathcal{F}$	ensemble des vols
$\mathcal{T}$	périodicité de la table des horaires
$\mathcal{C}$	ensemble des connexions élémentaires
t_s	heure de départ du vol
t_e	heure d'arrivé du vol
$\mathcal{C}_{\wr\lceil}$	ensemble des connexions élémentaires entre les deux aéroports o et d
F_{od}	nombre de connexions élémentaires entre o et d
c_{od}	capacité totale en terme de nombre de passagers
P_{od}	nombre total de passagers
R_{od}	revenu total
$\bar{R}_{od}$	revenu minimum par passager
t_{od}	durée minimale du vol
T_{od}	durée maximale du vol
D_{od}	distance entre les deux aéroports
A_{od}	liste des compagnies aériennes qui opèrent les vols entre deux aéroports
W_{od}	liste des jours quand le vol est opéré
WF_{od}	nombre d'opérations hebdomadaires
ym	year_month
T	période
$N_s(x)$	nombre de paires de nœuds connectés au nœud x avec s sauts
N_s	nombre de paires de nœuds pour s sauts

Flight Radius Problem

(o, d)	arc étudié entre l'origine o et la destination d
$R_{od}(i, j)$	contrainte de regret d'un chemin entre i et j passant par l'arc (o, d)
$\overrightarrow{R_{od}}(j)$	contrainte de regret dans le cas d'une direction sortante
$\overleftarrow{R_{od}}(j)$	contrainte de regret dans le cas d'une direction entrante

Modèles du calcul d'une part de marché

D_f	fréquence directe
I_{df}	fréquence indirecte
E_{df}	nombre de fréquences existantes entre une origine et une destination en vol direct sur la semaine
E_{vf}	nombre de fréquences existantes entre une origine et une destination en correspondance sur la semaine
D	ensemble de destinations desservies par une compagnie aérienne depuis une origine
D_{all}	ensemble de destinations desservies par toutes les compagnies aériennes depuis une origine
A_c	capacité de l'avion
id	distance par correspondance
$\bar{d_v}$	distance moyenne d'un aéroport par rapport à la distance du vol direct
C	ensemble des aéroports en correspondance
C_t	circuitry
A_p	dominance d'une compagnie aérienne
$\mathbb{1}_{H(x)}$	fonction indicatrice sur l'ensemble des hubs définie pour un nœud x
H	liste des hubs
u'	arc ajouté entre le hub et le nœud
$f_{u'}$	fréquence minimale de l'arc u'
$c_{u'}$	capacité minimale de l'arc u'
$t_{u'}$	durée minimale totale de l'arc u'
$R_{u'}$	revenu total de l'arc u'
$D_{u'}$	distance totale de l'arc u'
$e_{u'}$	nombre d'escale pour l'arc u'
$h_{u'}$	nombre de hubs pour l'arc u'
ns	niveau de Service
sbc	deuxième meilleure correspondance
$sbctd$	différence de temps avec la deuxième meilleure correspondance
$bctd$	meilleure correspondance
d	rapport de la distance
f	fréquence
p	rapport du prix
c	capacité

Sigles et Abréviations

IATA	International Air Transport Association
ICAO	International Civil Aviation Organization
MCT	Minimum Connecting Time
MWT	Minimum Waiting Time
QSI	Quality of Service Index
FRP	Flight Radius Problem
NoSQL	Not Only SQL
Pax	Passenger
BKK	Aéroport de Bangkok
NCE	Aéroport de Nice
CDG	Aéroport de Charles De Gaulle
TLS	Aéroport de Toulouse
KPI	Key Performance Indicator
CRS	Computer Reservations Systems
RD	Route Development
FS	Flight Scheduling
OAG	Official Airline Guide
MS	Market Share
MNL	Multinomial logistic regression
IIA	Independence of Irrelevant Alternatives
BFS	Breadth-First Search
DFS	Depth-First Search
PCC	Plus Court Chemin
HL	Hub Labeling
CH	Contraction Hierarchies
SGBD	Système de Gestion de Base de Données
ACID	Atomicité, Cohérence, Isolation, Durabilité
CRUD	Create, Read, Update, Delete
SQL	Structured Query Language
BASE	Basic, Availability, Soft state, Eventual consistency
BSON	Binary JSON
APOC	Awesome Procedures on Cypher
LCC	Low-cost carriers
DGAC	Direction Générale de l'Aviation Civile
CFN	Condensed FLight Network
SCC	Strongly Connected Component
SSSP	Single-Source Shortest Path
DCP	Decomposition en plus courts chemins
SEQ	Algorithmes en séquentiel
RMS	Referenced Market Share

CHAPITRE 2

Contexte industriel

Dans ce chapitre, nous présentons le contexte industriel de la société Milanamos. *Nous commençons par présenter l'entreprise et son produit* `PlanetOptim`. *Ensuite, nous décrivons en détail le module qui nous intéresse dans ce travail de thèse et abordons les motivations de cette thèse.*

2.1 Présentation de *Milanamos*

Milanamos est une start-up qui propose des services en matière de développement de réseau et d'analyse des lignes aériennes afin de développer des stratégies de voyage. Cette start-up dispose d'un outil `PlanetOptim` permettant à ses clients (principalement des compagnies aériennes et des aéroports) de faire des études de marché (rentabilité d'une nouvelle route). Pour cela, cette start-up communique avec des services qui fournissent un ensemble de données concernant le trafic aérien international. Elle propose alors un certain nombre d'outils permettant d'étudier en détail le marché du transport aérien.

2.1.1 Historique de l'entreprise

Christophe Imbert fonde, avec Christophe Ritter, la start-up *Milanamos* en janvier 2014. Trois mois plus tard, son outil `PlanetOptim` a été sélectionné par *Price Waterhouse & Coopers* (*PwC*) et *LeighFisher*, qui est une société de conseil spécialisée dans les infrastructures de transport, comme plate-forme pour leurs offres analytiques aux aéroports, opérateurs de transport et organismes gouvernementaux. Dans le même mois, *Milanamos* a gagné le concours mondial de l'innovation 2030 (2030 *World Innovation Awards*) pour son expertise en *Big Data*. Depuis sa création, *Milanamos* continue à travailler avec les principaux voyagistes, associations et propriétaires d'infrastructures afin de mettre en œuvre la suite du programme des données de voyage `PlanetOptim`.

2.1.2 Organigramme de l'entreprise

Aujourd'hui, *Milanamos* a une équipe composée de cinq personnes qui travaillent ensemble pour développer, améliorer, promouvoir, et optimiser son outil `PlanetOptim` (voir l'organigramme de la figure 2.1).

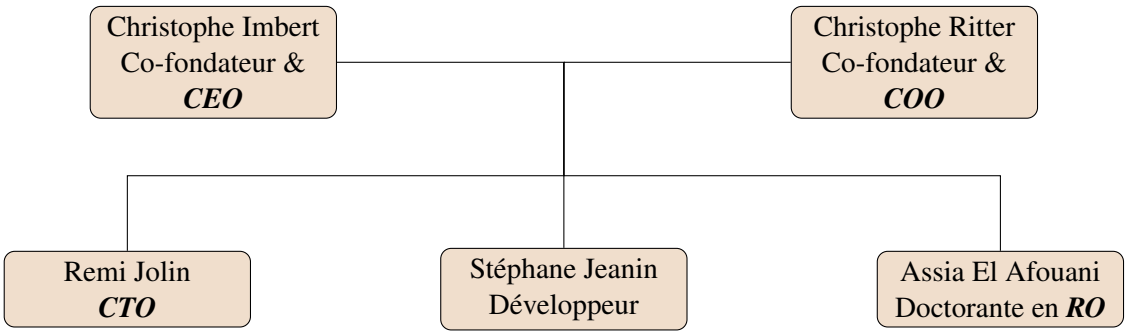


FIGURE 2.1 – Organigramme de *Milanamos*.

2.1.3 Chiffres clés

Milanamos a beaucoup progressé au cours des dernières années. En effet, la start-up a réussi à augmenter son chiffre d'affaire initial de 600%.

L'entreprise *Milanamos* manipule un volume très important des données, qui sont constituées d'informations sur le trafic de passagers. Ces données sont stockées dans la base de données

optimode, contenant plus de 4 milliards de lignes afin de fournir des informations synthétiques et pertinentes à son outil PlanetOptim.

2.1.4 Clients et partenariats

Milanamos offre aux aéroports, aux compagnies aériennes et ferroviaires, ainsi qu'aux autres compagnies de transports un outil pour chercher des moyens plus efficaces pour planifier de nouveaux itinéraires et établir de nouveaux partenariats. En effet, cette start-up a plus de 30 clients :

— 30 petits et grands aéroports : les aéroports de la Côte d'Azur, de Paris, de La Réunion, de Madère, de Limoges, de Manille ;

— des compagnies aériennes : flyDubai, Air France, Nordica et Omni Air ;

— des compagnies ferroviaires : Russian Railways et la SNCF.

Des investisseurs de renom comme Starquest Capital et BPI France ont investi en 2015 dans *Milanamos*. De plus, *Milanamos* a été accompagnée par l'incubateur PACA-Est (recherche publique) et est membre du Pôle SCS ainsi que co-fondateur du Travel Data Alliance. *Milanamos* collabore également avec le laboratoire Ville Mobilité Transport de l'École des Ponts ParisTech, l'Institut Mines-Télécom, l'I3S et le Technion en Israël.

2.2 Présentation du produit de *Milanamos* : PlanetOptim

2.2.1 Généralités

Milanamos a mis en place l'outil PlanetOptim courant de l'année 2014. C'est une application *Web* destinée à des compagnies aériennes et aéroports pour analyser et prédire le trafic aérien (voir la figure 2.2).

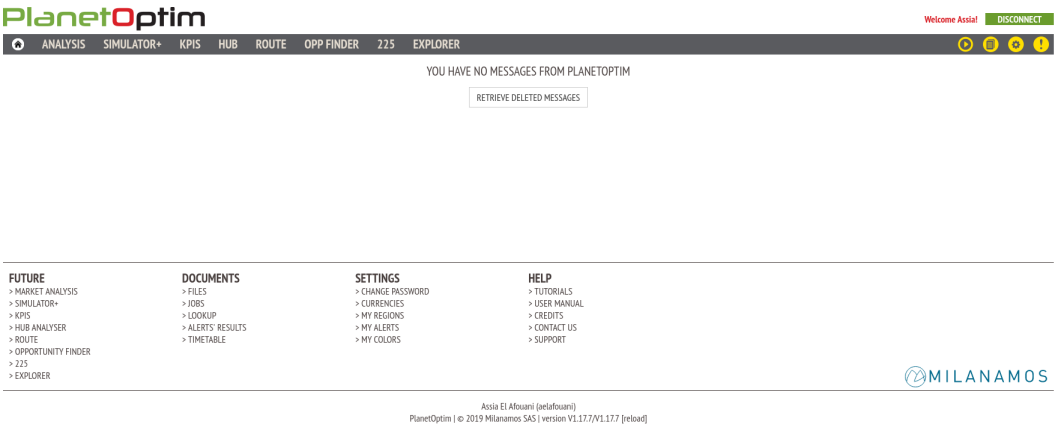


FIGURE 2.2 – Interface de PlanetOptim.

PlanetOptim dispose de huit modules qu'on décrit dans la prochaine section : ***Analysis***, ***Simulator***, ***KPIS***, ***Hub***, ***Route***, ***Opp Finder***, ***225***, ***Explorer***.

PlanetOptim propose un certain nombre d'outils à ses utilisateurs :

1. l'analyse de marché, en fonction d'un trajet donné ;

2. l'analyse de plate-forme aéroportuaire ;
3. l'analyse du réseau d'une compagnie aérienne ;
4. la fonctionnalité phare du logiciel, le *Simulator*.

Le *Simulator* permet de simuler une route en fonction des paramètres choisis par l'utilisateur, et de lui fournir ensuite tous les résultats qu'il pourrait en calculer, comme la rentabilité de la route, le coût engendré et les parts de marché qu'il pourrait capturer. Pour pouvoir fournir tous ces résultats, le logiciel va en réalité se baser sur des routes existantes qui ressemblent le plus au type de trajet simulé : taille des aéroports, compagnie qui opère, fréquence de trajet, prix du carburant, distance de trajet, *etc*.

La figure 2.3 illustre les principales fonctionnalités de PlanetOptim. L'outil a les caractéristiques suivantes (voir le tableau 2.1 pour les abréviations) :

— prise en compte des deux visions : Segment et *O&D* (voir 2.2.2.1) ;
— historique de 17 ans de données ;
— les principaux utilisateurs sont les aéroports et les compagnies aériennes ;
— analyse des données des marchés par mois ou par période ;
— deux métriques principales : Pax et Rev.

FIGURE 2.3 – Fonctionnalités de `PlanetOptim`.

Abréviation	Désignation
comp. aér.	Compagnie aérienne
mk.	Commercialisante (*marketing*)
op.	Opérante (*operating*)
opér.	Opérations
cap.	Capacité
dest.	Destination
Rev.	Revenu
org.	Origine
PAER.	Prix-élasticité
hebd.	Hebdomadaire
app.	Appareil

TABLE 2.1 – Tableau des abréviations.

PlanetOptim a pour but également d'aider ses clients à chercher des routes les plus rentables en faisant des analyses de marché, de fournir des simulations et des prévisions lors d'une ouverture de nouvelles routes, de maximiser les connexions de vols jusqu'à 12 mois dans le futur, et d'aider des compagnies à choisir les meilleurs réseaux de partenaires. Les résultats sont présentés de manière simple à travers plusieurs options d'affichage comme des tables (*Dashboard*), des graphiques, des diagrammes et des cartes. Par ailleurs, les données livrées par cet outil sont exportables dans n'importe quel format et sont réutilisables par les clients.

2.2.2 Description des modules de PlanetOptim

2.2.2.1 Analysis

Le module ***Analysis*** de l'application PlanetOptim a pour but d'analyser le marché du trafic aérien sur une période donnée. Cette analyse porte sur deux métriques importantes dans l'aérien : revenu et nombre de passagers, Pax. L'analyse tient compte des deux visions de marché : *Segment* et *O&D* (voir la figure 2.5).

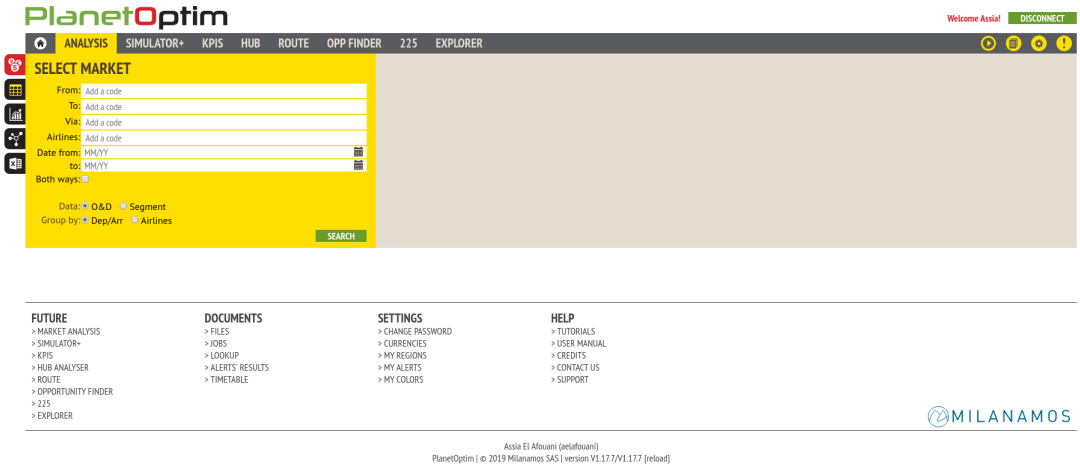
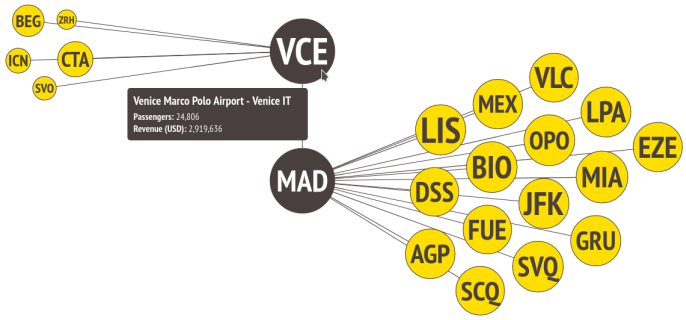


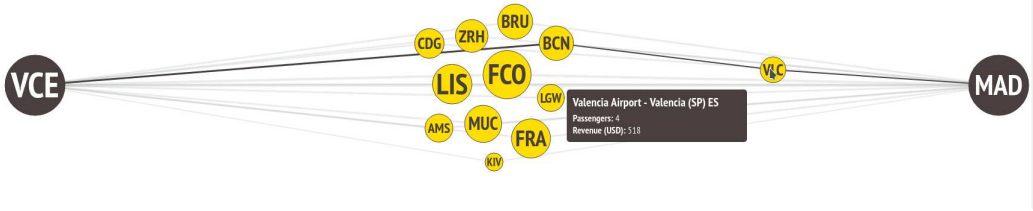
FIGURE 2.4 – Module ***Analysis*** de PlanetOptim.

En effet, la gestion du transport aérien peut être vue de deux façons : Une vision *O&D* basée sur le marché des *O&D* et une vision *Segment* basée sur le marché des *segments*. Un marché *O&D* est défini par le point d'entrée et sortie du passager dans le système de réservation de la compagnie aérienne. Cette vision est importante pour la compagnie aérienne, car cela lui permet de connaître combien de passagers voyagent entre deux villes pendant une certaine période de temps. Toutefois, l'information sur le marché *Segment* concerne plutôt une route spécifique opérée par un avion sur un vol sans escale entre une origine et une destination. Plus concrètement, le marché *O&D* est une vision de passagers alors que le marché *Segment* est une vision d'équipement. C'est la raison pour laquelle les données sur le marché *O&D* nous informent sur l'ensemble des itinéraires possibles pour rejoindre une destination depuis une origine quelconque. Donc, on retrouve toutes les données sur les aéroports de correspondance, ce qui permet de suivre le flux des passagers et savoir par où ils passent. D'ailleurs, nous trouvons également des informations sur la compagnie qui ne fait que commercialiser le vol dans le cas d'un partage de codes, puisque ce type de marché se base sur une vision marketing. Contrairement au marché des *segments*, ce type de marché se

base sur les informations collectées pour une route spécifique entre deux aéroports donnés. Cela correspond plutôt à des vols directs (voir les figures 2.5a et 2.5b).



(a) Vision `Segment` (extrait de l'application `PlanetOptim`).



(b) Vision *O&D* (extrait de l'application `PlanetOptim`).

FIGURE 2.5 – Vision des marchés.

L'affiche des résultats se fait de toutes façons sur plusieurs options : tables et graphiques (voir la figure 2.6).

Exemple 2.2.1 (Module *Analysis*)**.**
Supposons que nous souhaitions analyser les données du marché *Charles De Gaulle*/France (`CDG`)-*Nice*/France (`NCE`) pour la période *02/19-08/19*. La figure 2.6 renvoie les résultats groupés par `Dep/Arr` (Départs/Arrivées). Nous pouvons également choisir le groupage par compagnie aérienne (`Airlines`). Le camembert à droite affiche le top 10 des routes en termes de nombre de passagers.

2.2.2.2 Simulator

Le module ***Simulator*** est le module principal de l'application `PlanetOptim`. Il a été appelé par *Milanamos*, `PlanetOptim.Futur`, car il permet aux utilisateurs de prédire l'avenir à travers la simulation. L'objectif du simulateur (en anglais *Simulator*) est d'étudier la rentabilité d'une route en comparant des routes existantes. Ainsi, une compagnie aérienne qui voudrait créer une nouvelle route pourrait estimer les parts de marchés attendues.

Le travail de cette thèse est basé sur ce module qui a connu plusieurs évolutions depuis 2016. Nous décrivons plus en détail l'évolution de ce module dans les sections à venir (voir la figure 2.7).

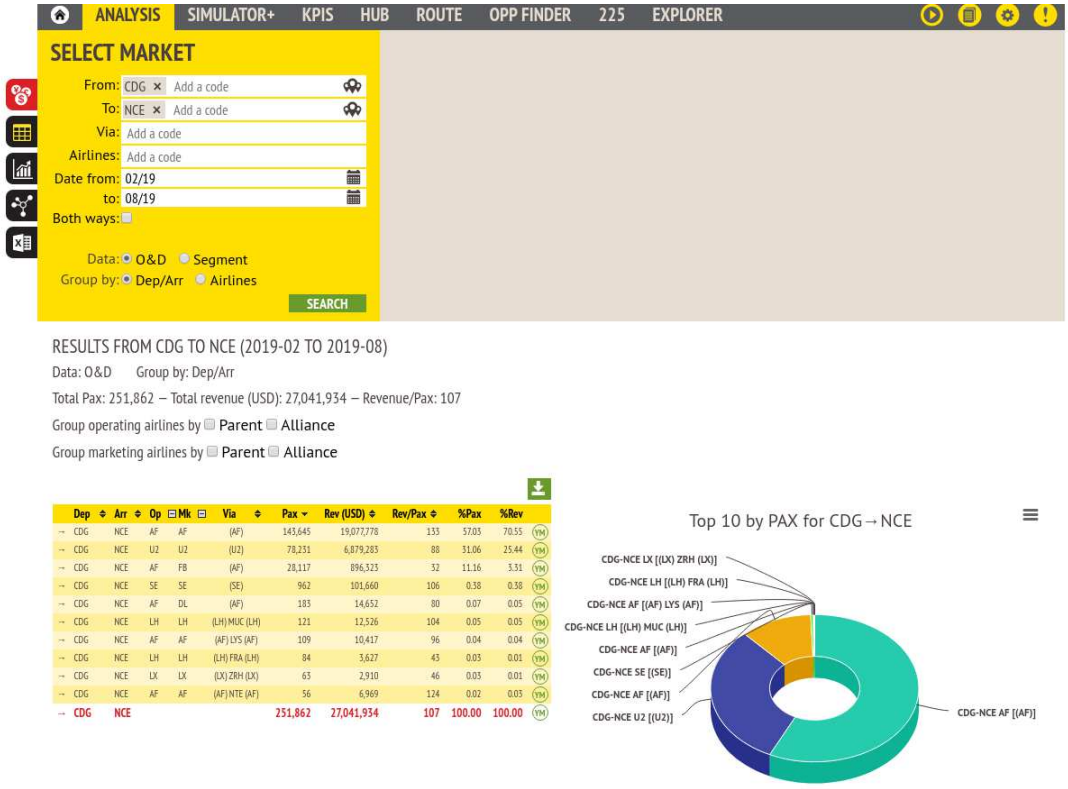


FIGURE 2.6 – Résultat du module *Analysis* de PlanetOptim.

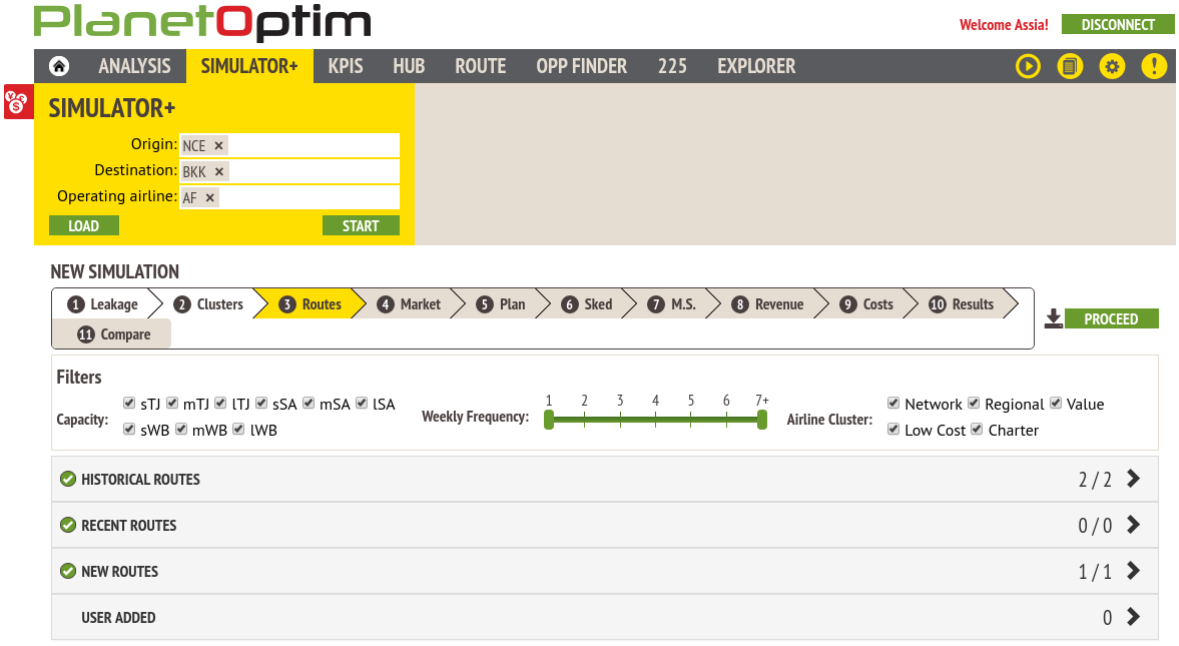


FIGURE 2.7 – Module *Simulator* de PlanetOptim.

2.2.2.3 KPIS

`PlanetOptim` fournit pour ses utilisateurs les indicateurs clés de performance (***KPI***), pour les deux perspectives de marchés : `Segment` et *O&D*. Ces clés permettent de mesurer la rentabilité d'une route. Parmi ces clés, on trouve : le nombre total de passagers transportés, le revenu total et la répartition des compagnies aériennes opérant cette route. Dans ce module, la comparaison se fait avec la période de l'année précédente.

Exemple 2.2.2 (Module *KPIS*)**.**
L'exemple suivant nous donne les indicateurs des performances *KPI* pour le marché *Nice*/France (`NCE`)-*Bangkok*/Thaïlande (`BKK`) pour la période *04/17-10/17*. Par exemple, Il y a 40 passagers par jour contre 35 l'année précédente et une baisse de revenu total de 8,7% (voir la figure 2.8).

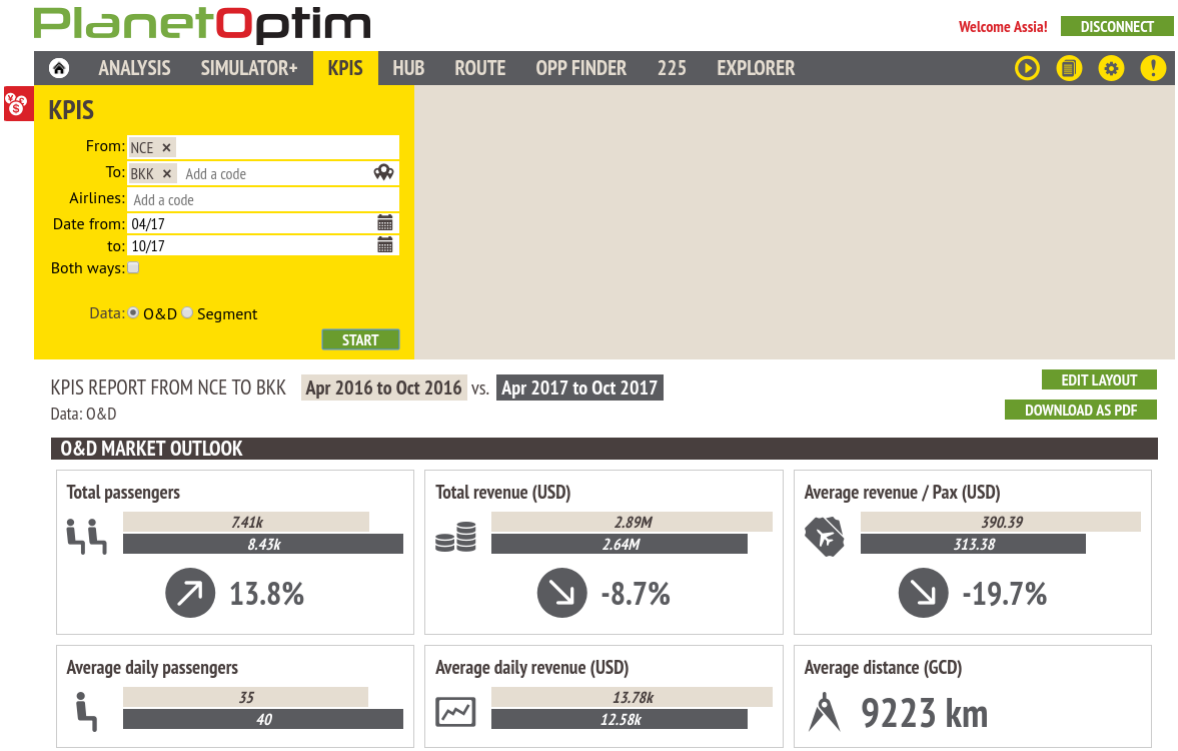


FIGURE 2.8 – Module *KPI*.

2.2.2.4 Hub

Le module ***Hub*** aide les utilisateurs à évaluer les connexions potentielles sur un aéroport donné. Il fournit des informations opérationnelles et commerciales détaillées pour trouver l'heure optimale d'arrivée et de départ à n'importe quel endroit, pour augmenter le potentiel de circulation avant et après le trafic.

Exemple 2.2.3 (Module *Hub*)**.**
L'exemple suivant nous affiche le plan des vols pour les vols entre *Nice*/France (`NCE`)-*Paris Charles De Gaulle*/France (`CDG`) pour la semaine 36 de l'année 2019 (voir la figure 2.9).

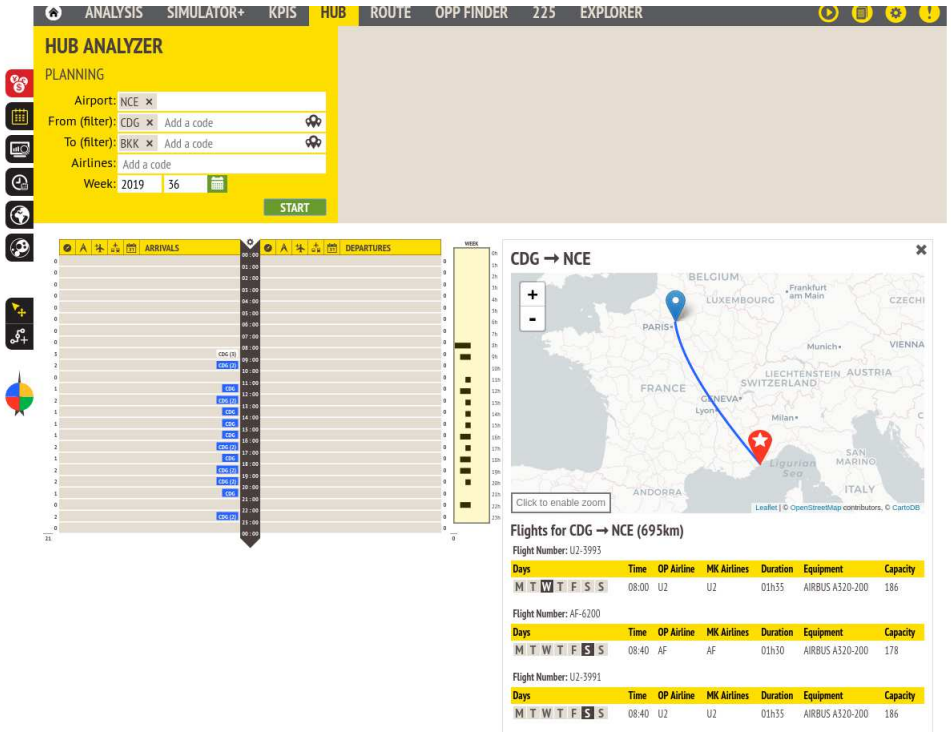


FIGURE 2.9 – Module *Hub*.

2.2.2.5 Route

Le module ***Route*** fournit une vue détaillée du calendrier des routes pour chaque semaine d'opération de vol disponible dans la base de données, soit dans le passé (à partir de la semaine 1 de 2002) jusqu'au moins 12 mois en avance, si ces informations sont communiquées par les compagnies aériennes exploitantes. Deux vues différentes sont disponibles :

— Planification de route : fournit les horaires pour un segment spécifique et une semaine de l'année ;

— Programme de route : fournit le programme de vol pour un segment et une période spécifiques et met en évidence les changements d'horaire par rapport à une semaine de base.

Exemple 2.2.4 (Module *Route*)**.**
L'exemple suivant concerne le vol entre *Nice*/France (NCE)-*Paris Charles De Gaulle*/France (CDG) de la semaine 36 de l'année 2019 (voir la figure 2.10). Quatre métriques sont calculées :

1. répartition de la capacité : *AirFrance* (AF) domine *easyJet* (U2) ;
2. répartition de la fréquence ;
3. distribution de la flotte ;
4. distribution de la capacité.

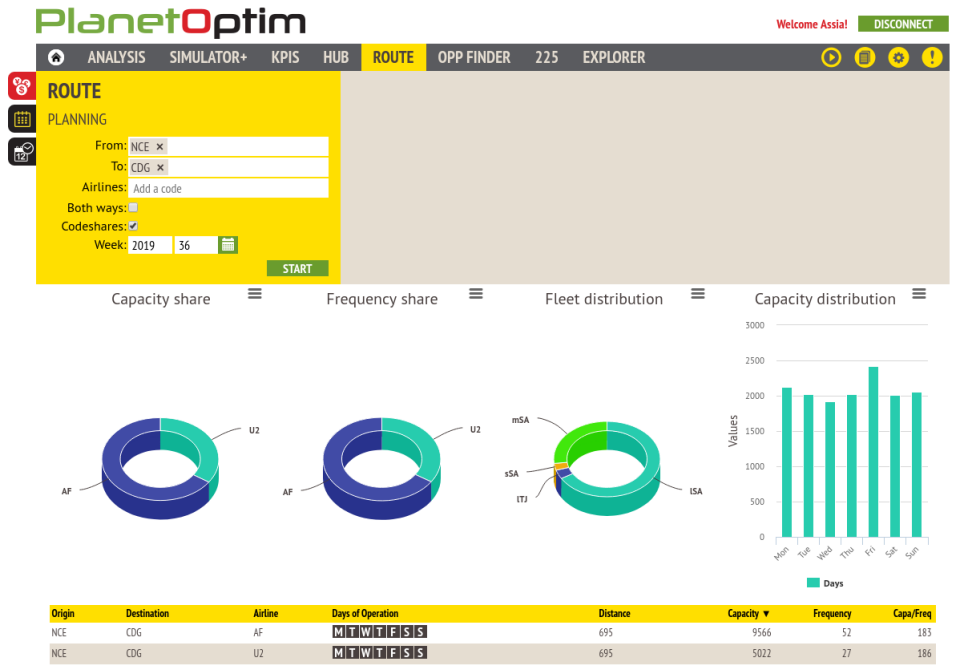


FIGURE 2.10 – Module *Route*.

2.2.2.6 Opp Finder

Le module ***Opp Finder*** (en anglais *Opportunity Finder*) aide les utilisateurs à évaluer rapidement le réseau des transporteurs aériens ou des destinations aéroportuaires, à la fois par une carte et par une table.

Exemple 2.2.5 (Module *Opp Finder*)**.**
L'exemple suivant compare les aéroports *Paris Charles De Gaulle*/France (CDG) et *Francfort*/Allemagne (FRA). Cinq métriques sont comparées :

1. nombre de destinations ;
2. jours d'opérations ;
3. nombre de compagnies aériennes ;
4. capacité ;
5. capacité moyenne.

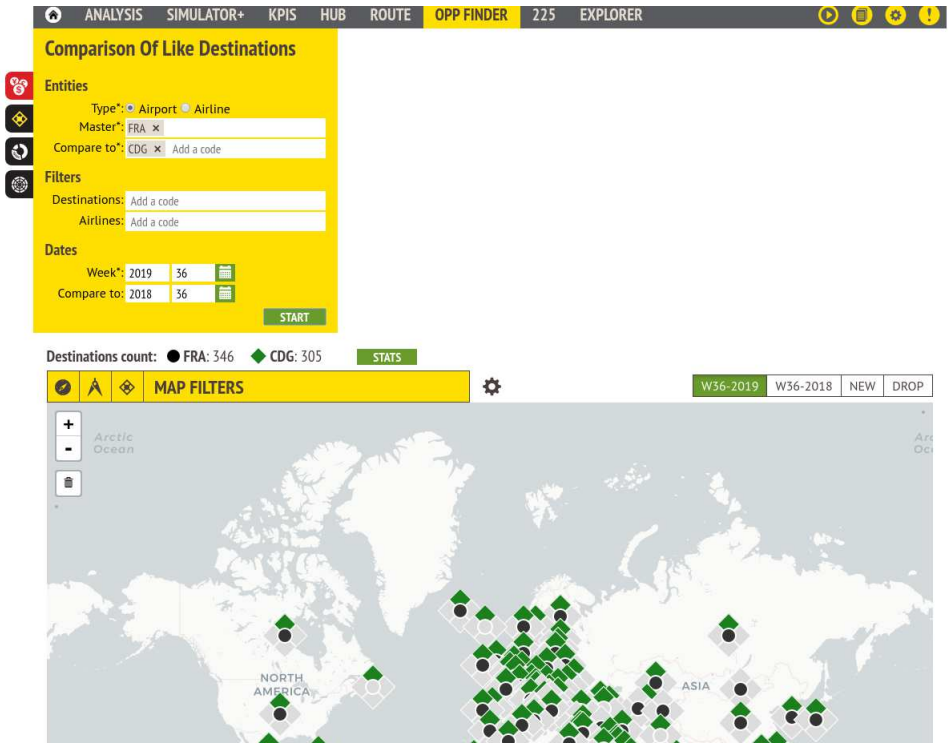


FIGURE 2.11 – Module *Opp Finder*. Sur l'exemple, l'aéroport CDG a 305 destinations et FRA 346 destinations.

2.2.2.7 225

Le module ***225*** a été développé pour répondre aux besoins d'un client. Ce module permet de comparer plusieurs vols selon les informations suivantes :

1. compagnie qui opère ;
2. compagnie qui commercialise le vol ;
3. jour de la semaine d'opération ;
4. capacité ;
5. fréquence.

Exemple 2.2.6 (Module *Opp Finder*)**.**
L'exemple suivant compare les vols entre *Paris Charles De Gaulle*/France (CDG)-*Nice*/France (NCE) pour la période *04/19-10/19* (voir la figure 2.12).

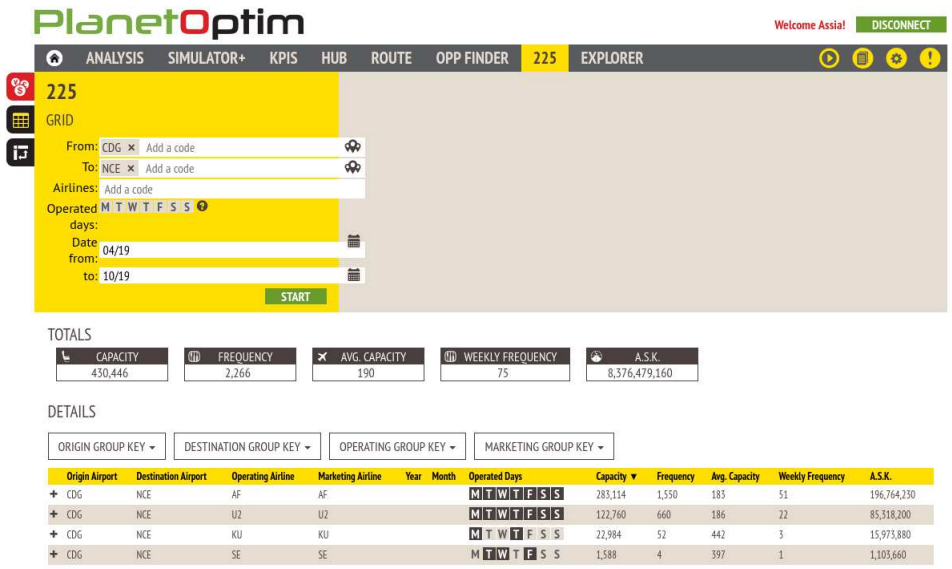


FIGURE 2.12 – Module *225*.

2.2.2.8 Explorer

Ce module consiste à afficher le réseau d'une compagnie aérienne à travers le monde pour un mois donné. Par exemple, la compagnie *AirFrance* opère la route *Paris Charles De Gaulle*/France (CDG)-*Bangkok*/Thaïlande (BKK) (voir la figure 2.13).

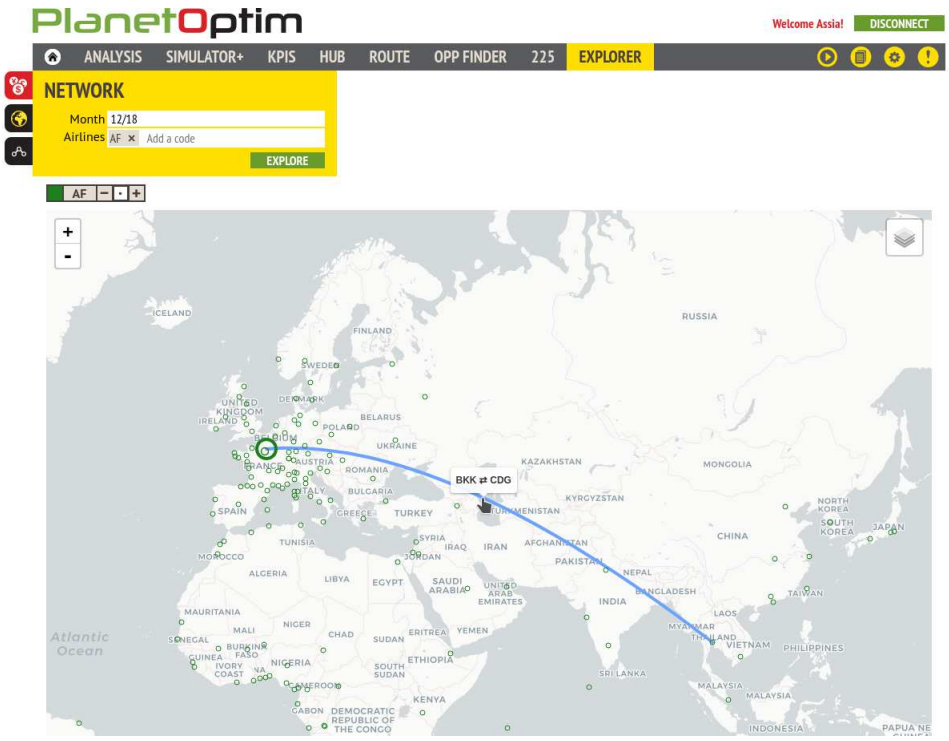


FIGURE 2.13 – Module *Explorer*.

2.2.3 Architecture logicielle de `PlanetOptim`

L'architecture logicielle de `PlanetOptim` porte sur un ensemble de fonctions. En effet, chaque fonction fait appel au même mécanisme : d'un côté en *Backend*, en interrogeant la base puis en formatant les données ; d'un autre côté en *Frontend*, où les données sont récupérées, traitées puis affichées. On peut donc facilement dégager un schéma d'exécution (voir la figure 2.14) :

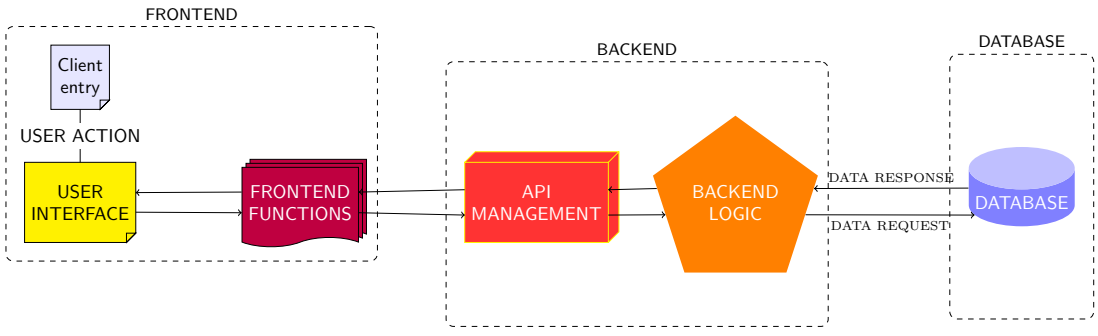


FIGURE 2.14 – Architecture logicielle de `PlanetOptim`.

Lorsque l'utilisateur effectue une action sur l'interface, la partie « Vue » du *Frontend* appelle la fonction du « contrôleur » qui lui est associée. En général, l'objectif de cette fonction est de générer un appel pour récupérer des données de la base, et enfin les traiter. On trouve donc dans ces fonctions un appel à un service qui va se charger d'interroger le *Backend*. C'est donc *via* une

API propre que la fonction adéquate va être appelée dans le *Backend*. Cette dernière n'aura plus qu'à effectuer la requête sur la base, et les données pourront remonter toute la chaîne jusqu'au *Frontend*, qui affichera les éléments demandés par l'utilisateur.

2.3 Évolution du module *Simulator*

Au début de ce projet de thèse, le module *simulator* de `PlanetOptim` consistait à afficher les correspondances les plus pertinentes vis-à-vis de la création d'une nouvelle route (voir la section 2.3.1). À la fin de l'année 2017, le calcul des parts de marché a été intégré (voir la section 2.3.2).

2.3.1 Au début de la thèse en 2016

Le module avait pour but de répondre aux questions suivantes :

1. Est-ce qu'il existe des passagers qui seraient potentiellement intéressés par ce nouveau service ?
2. Quelles sont les origines/destinations de ces passagers ?
3. Quel est l'horaire de vol le plus optimal qui maximise les correspondances ?

La *première étape* consistait à saisir les informations obligatoires concernant la nouvelle route à ajouter, à savoir (voir la figure 2.15) :

— **From** : aéroport d'origine ;
— **To** : aéroport destination ;
— **Airlines** : compagnie aérienne qui opère le nouveau vol ;
— **Departure time** : heure de départ du nouveau vol ;
— **Arrival time** : heure d'arrivée du nouveau vol ;
— **Week** : semaine d'opération du vol.

FIGURE 2.15 – Ancienne version du *Simulator*.

La *deuxième étape* consistait à afficher le résultat de la simulation qui sont les aéroports en amont de l'aéroport d'origine et en aval de l'aéroport de destination. Par exemple dans la figure 2.16, le passager qui vient de CDG et qui souhaite rejoindre ICN, peut faire une connexion à NCE pour prendre le nouveau vol de NCE à BKK, ensuite une deuxième connexion à BKK pour atteindre sa destination finale.

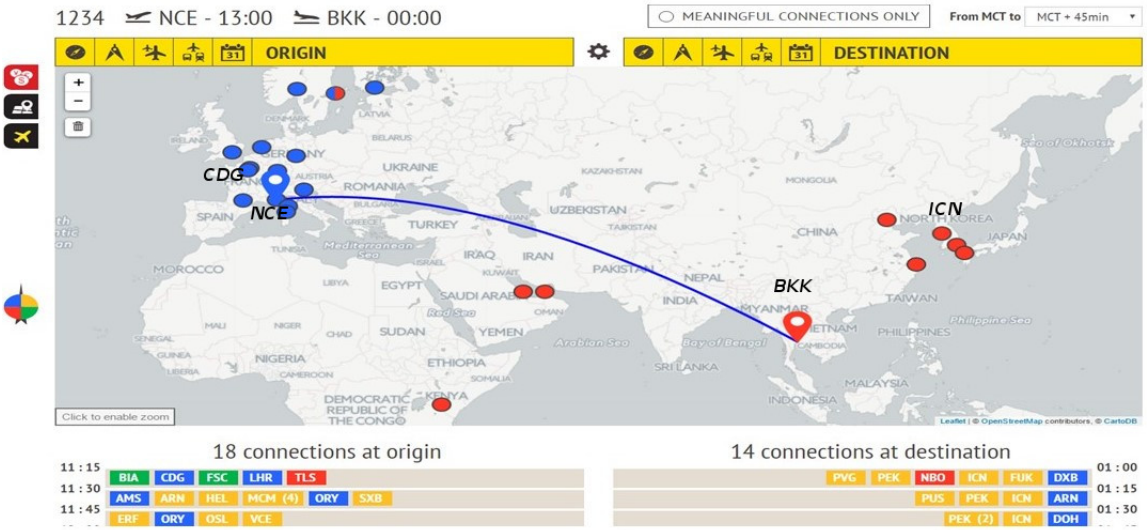


FIGURE 2.16 – Correspondances du nouveau vol NCE–BKK. L'arc bleu représente le nouveau vol, les couleurs des aéroports référent leurs positions géographiques selon la boussole.

Le processus pour déterminer ces aéroports consiste à chercher tous les aéroports qui sont similaires aux aéroports origine et destination en termes de revenu et nombre de passagers (Pax). Ensuite, on regarde toutes les routes qui existent entre les deux ensembles amont et aval.

2.3.2 À la fin de la thèse en 2019

À la fin de l'année 2017, *Milanamos* a mis à la disposition de ses utilisateurs une nouvelle version du module *Simulator*, offrant ainsi une solution unique permettant d'analyser de manière polyvalente le développement des routes de bout en bout. Par rapport à la version précédente du module *Simulator*, *Milanamos* voulait donner aux utilisateurs un contrôle complet et une visibilité sur les hypothèses, les données d'entrée et les modifications apportées par l'utilisateur autorisées par l'outil. En fin de compte, ses utilisateurs devront présenter et justifier leurs hypothèses et leurs choix à leurs partenaires commerciaux. Il était donc primordial de développer une méthodologie complète, avec, à chaque étape, un contrôle et une validation par les utilisateurs.

Ce nouveau calcul des routes développé tient compte des calculs des parts de marché (voir la section 2.2.2.2) pour combler les besoins des clients. En effet, l'ouverture d'une nouvelle route sur un nouveau marché nécessite le calcul des parts de marché pour pouvoir estimer combien la compagnie aérienne peut capturer de la demande de ce marché (voir la section 3.2).

2.3.2.1 Objectifs du *Simulator*

Le module *Simulator* a pour objectif de fournir une méthodologie et un processus de bout en bout pour toutes les activités liées au développement des routes, appelées dans la littérature, les méthodes de planification des routes (en anglais *Route planning*) (voir la section 3.2.2).

Son objectif est de permettre à l'utilisateur d'évaluer le potentiel d'une nouvelle route en termes de :

1. potentiel de demande ;
2. trafic à bord attendu aussi bien en local, qu'en correspondance ;
3. revenu par passager et des recettes totales ;
4. des coûts d'exploitation de l'horaire prévu.

Et finalement, déterminer la rentabilité de la route et des indicateurs clés de performance au cours des trois premières années d'exploitation. Le module *Simulator* a été décomposé en 11 étapes permettant à chaque utilisateur de choisir les hypothèses à utiliser, des ajustements de données peuvent être apportés et de valider le résultat de chaque étape avant de passer à la suivante.

2.3.2.2 Description des étapes

La figure 2.7 du paragraphe 2.2.2.2 illustre la version améliorée du *Simulator*. La nouvelle version du module comprend les onze phases suivantes.

Leakage

Une fois que la route étudiée est saisie, la première étape consiste à chercher d'autres aéroports qui sont proches des aéroports origine-destination pour pouvoir proposer un vol à partir de ces aéroports.

Clusters

Cette étape consiste à rechercher d'autres aéroports similaires à l'aéroport d'origine et à l'aéroport de destination afin de se concentrer spécifiquement sur les similitudes du marché desservi par chacun d'eux. La sélection des aéroports similaires est basée sur une analyse multicritères afin de garantir que les paramètres de volume de trafic et de rendement soient pris en compte. L'objectif de cette phase est de fournir une liste d'aéroports comparables à l'origine et à la destination.

Routes

En utilisant la sélection d'aéroports d'origine et de destination similaires, le système recherchera des routes exploitées entre ces aéroports au cours des cinq dernières années. Toutes les routes seront sélectionnées et la compagnie aérienne, le marché, le trafic, les revenus, la fréquence et la capacité sont fournis. L'objectif pour les utilisateurs est de sélectionner les routes les plus similaires, afin de déterminer le coefficient de stimulation de la demande après l'introduction ou le changement de la capacité.

Market

Les informations de la phase précédente permettent au système de capturer les coefficients de stimulation de la demande afin de créer une courbe de la demande du marché appliquée aux données historiques d'origine et de destination du marché demandé. Sur la base de cette courbe utilisant des données historiques sur les cinq dernières années, les prévisions sont calculées pour déterminer le potentiel de demande de marché aller-retour pour les trois prochaines années d'exploitation.

Plan

Cette phase consiste à modifier le programme de vol, à savoir, la fréquence d'opération hebdomadaire ainsi que mensuelle et la capacité vendue.

Connections

Ce créateur de connexions fournit des connexions entrantes et sortantes pour le vol régulier, pour toutes les compagnies aériennes potentielles ou uniquement pour certains transporteurs limités identifiés par l'utilisateur. Ces informations seront ajoutées au trafic local point à point.

Market Share (*M.S.*)

Pendant cette phase, le système détermine la part de marché potentielle de la compagnie aérienne en activité. Il utilise à la fois des paramètres quantitatifs et qualitatifs, ainsi que la méthodologie de l'indice de qualité de service (*QSI*) et la part de marché historique de la compagnie aérienne (ou des compagnies aériennes appartenant au même groupe de compagnies aériennes) sur des marchés comparables. Différents paramètres *QSI* doivent être définis par l'utilisateur pour les éléments qualitatifs ou calculés automatiquement par le système en fonction des données disponibles.

Revenue

Sur la base de la part de marché pour le trafic point à point et du potentiel de flux de passagers estimé, le système calcule le revenu total attendu en utilisant le prix moyen actuel du marché pour l'origine et la destination. En outre, les frais de billets, les recettes annexes sont estimés à partir d'un ensemble de données du transporteurs.

Costs

Cette phase estime le coût d'exploitation des opérations en fonction de divers paramètres et variables de coûts.

Results

Cette étape fournit un rapport sur les profits et les dépenses pour les trois premières années à partir du lancement du service proposé, ainsi que tous les indicateurs clés de performance qui doivent être disponibles pour être discutés avec un aéroport ou une compagnie aérienne partenaire.

Compare

Cette étape consiste à comparer le résultat avec d'autres simulations.

2.4 Analyse des besoins

2.4.1 Cadre de la thèse

Le travail de la thèse s'inscrit dans le cadre du module *Simulator*, en particulier, les deux étapes *Connections* et *Market Share* (*M.S.*). L'objectif est de détecter puis évaluer une opportunité économique basée sur les décisions suivantes :

1. modifier la fréquence ;
2. proposer un nouveau vol ;
3. proposer un vol partagé.

Nous allons décrire les besoins à travers l'exemple de motivation que l'on reprendra plus tard tout au long de la thèse.

2.4.2 Exemple de motivation

Considérons l'exemple pour lequel une nouvelle route sera créée entre les aéroports de `NCE` et `BKK` (voir la figure 2.17). Cette nouvelle route permet aux passagers venant en amont de `NCE` d'atteindre d'autres aéroports *via* `BKK`. Prendre le nouveau vol peut être un peu plus long mais moins cher que d'aller directement à la destination finale. Le choix dépend des passagers car ils ont des préférences différentes. L'application renvoie en fait des aéroports sans intérêt. Par exemple, passer de `NCE` à `DXB` *via* `BKK` n'est pas un itinéraire réaliste pour un passager (voir la figure 2.18). L'itinéraire est trop long et coûteux que le vol direct entre `NCE` et `DXB`. Ainsi, il est nécessaire d'éliminer les marchés qui ne correspondent pas à des itinéraires réalistes. Cela permet également d'améliorer la visualisation sur l'outil `PlanetOptim`.

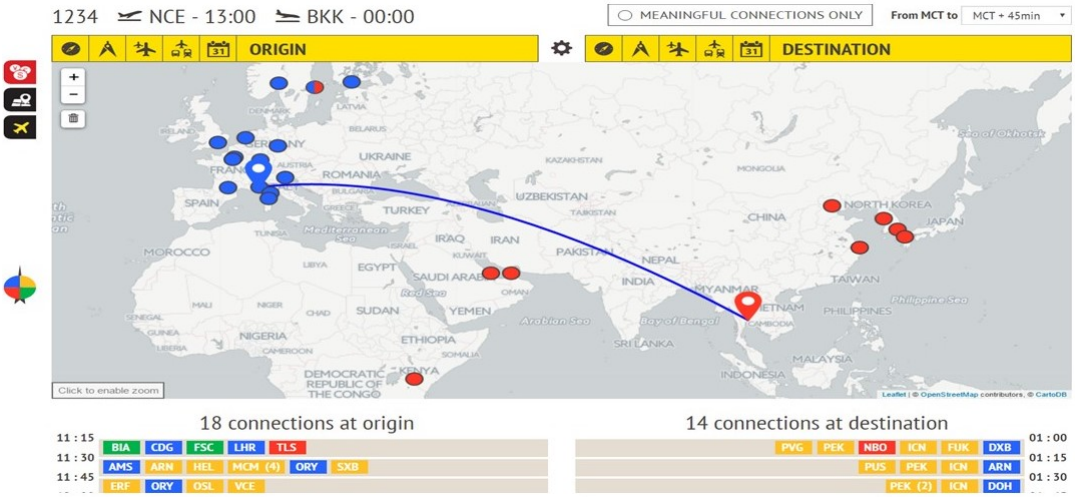


FIGURE 2.17 – Correspondances pour le vol entre NCE et BKK.

La méthode actuelle pour afficher les correspondances consiste à récupérer tous les aéroports dont les vols ont comme destination NCE et les vols arrivant à BKK sans tenir compte de la faisabilité de l'itinéraire. De plus, le calcul des parts de marché se fait uniquement sur l'ensemble des itinéraires reliant l'origine à la destination avec une seule correspondance. Par ailleurs, l'étape *Connections* est indépendante de l'étape *M.S.*.

Les éléments principaux ressortent donc de l'étude de l'existant :

1. la nécessité de filtrer l'ensemble des aéroports à afficher (voir la figure 2.18) ;
2. le calcul des parts de marché uniquement sur l'ensemble des routes intéressantes ;
3. la nécessité d'inclure les marchés impactés par ce nouveau vol.

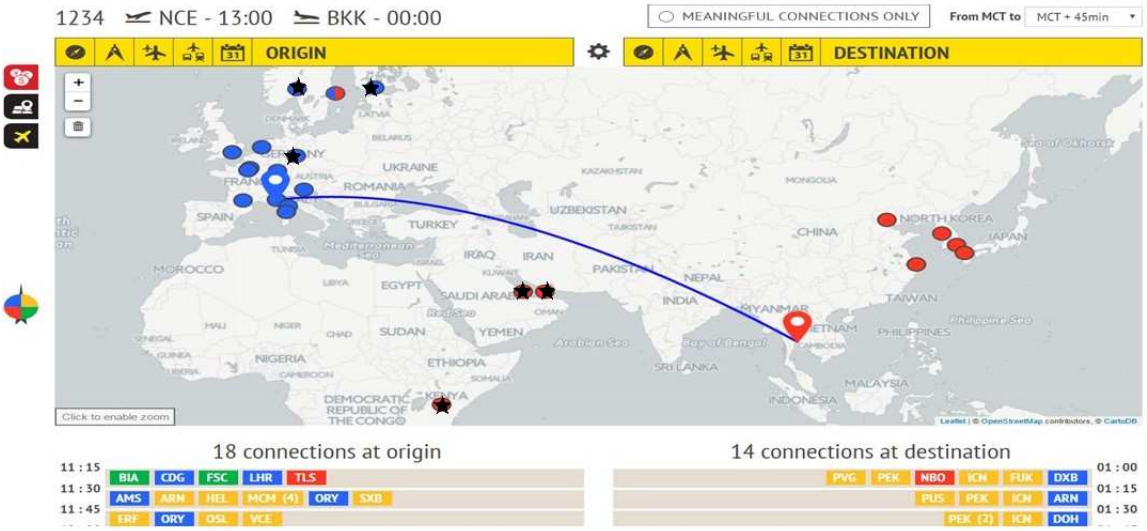


FIGURE 2.18 – Correspondances non réalistes. Les aéroports barrés représentent des aéroports dont les routes ne correspondent pas à des itinéraires réalistes.

2.5 Motivations

La start-up *Milanamos* a trois objectifs principaux : améliorer la visualisation, sélectionner les marchés potentiels à desservir et estimer les parts de marché attendues.

Le calcul des parts de marché est une étape longue, car il faut regarder tous les itinéraires possibles pour aller de l'origine à la destination. Ce qui donne une explosion exponentielle. Pour l'instant, *Milanamos* se contente des routes à une seule correspondance sans tenir compte de l'ensemble des marchés impactés par ce nouveau service. Actuellement, la start-up a choisi le modèle *QSI* pour le calcul des parts de marché. En effet, *Milanamos* utilise une moyenne pondérée dont les poids sont fixés par l'utilisateur. Ce qui néglige l'interdépendance des critères. On reviendra plus tard au chapitre 9 pour présenter les limites du modèle existant (voir la section 9.1.3).

À la lumière des objectifs cités, on propose d'intégrer des algorithmes d'aide à la décision dans le processus de calcul des parts de marché qui tiennent compte des impacts sur la répartition des passagers dans le réseau et de l'interaction des marchés impactés par cette prise de décision. Les performances des algorithmes doivent respecter les contraintes sur le temps de réponse de l'application.

Pour calculer efficacement les parts de marché, on propose de sélectionner les marchés potentiels vis-à-vis des préférences des passagers pour éliminer ceux avec une part de marché négligeable. En se basant sur le nouveau réseau qui contient uniquement les routes avec des parts de marché non négligeable, une méthode sera développée pour améliorer le modèle de calcul des parts de marché de *Milanamos*. On propose le modèle le plus récent, *logit*, qui a connu un grand sucés selon la littérature grâce à ses performances. Pour réduire la complexité combinatoire sur les itinéraires de chaque marché, on regarde uniquement ceux passant spécifiquement par des aéroports de type *hub*. Pour cela, une approche sera mise en place pour identifier les *hubs*.

Ce projet de thèse est à l'intersection des cinq disciplines : l'aérien, l'aide à la décision, les modèles économiques, la théorie des graphes et les bases de données.

Il s'agit d'appliquer des méthodes d'aide à la décision à l'aérien en gérant une masse énorme de données et en modifiant des modèles économiques qui sont souvent proposés sans tenir compte de la combinatoire.

2.6 Problématique de recherche

Á la lumière des motivations et des problèmes constatés par les limites de l'application industrielle `PlanetOptim`. Nous avons défini notre problématique générale autour du problème de la sélection des marchés et du développement du réseau.

En partant d'un réseau aérien qu'on suppose complet, c'est-à-dire que tous les aéroports sont desservies par des vols directs. Le résultat actuel du module *Flight Simulator* représente les aéroports en amont, qui sont présentés dans la figure 2.19 par la couleur bleu et ceux en aval par la couleur rouge. Pour répondre à notre problématique de recherche, la première étape consiste en deux étapes (voir la figure 2.19) :

1. Sélection des marchés : résoudre ce problème revient à identifier les marchés prometteurs centré sur le vol étudié. Ce qui nous amène à réduire le réseau de départ. Cette étape permet d'écarter les paires d'aéroports qui ne sont pas pertinents selon ce vol.
2. Estimation des parts de marchés : estimer sur le réseau réduit en tenant compte des marchés en correspondances et celui au niveau local.

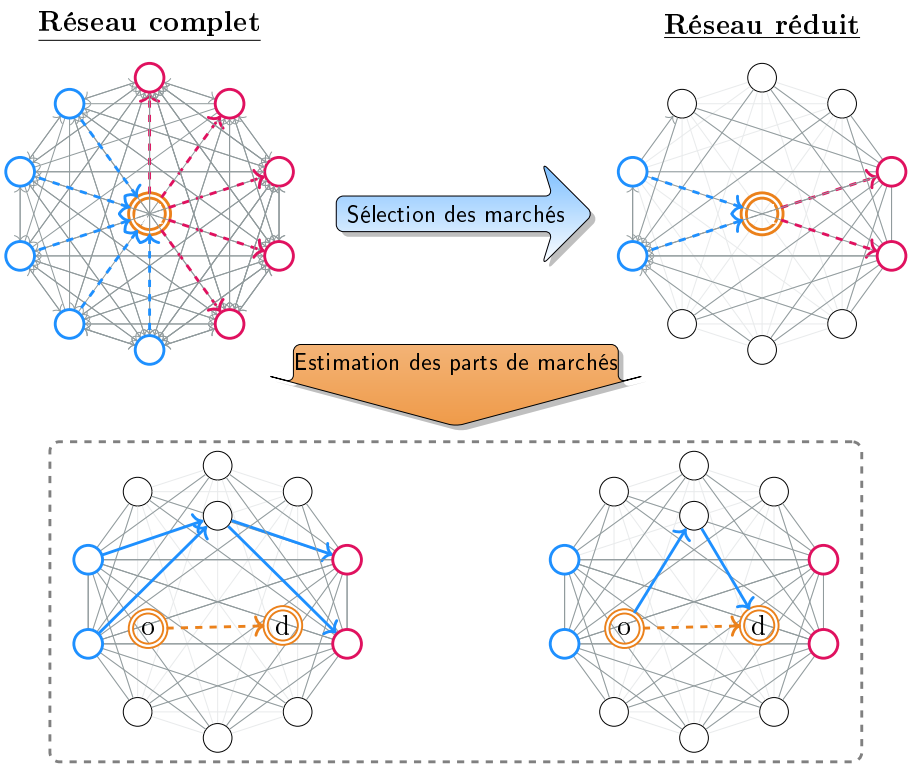


FIGURE 2.19 – Problème de recherche.

État de l’art

CHAPITRE 3

Les problèmes décisionnels dans l'aérien

Ce chapitre couvre la littérature qui fait état des problèmes décisionnels dans l'aérien. Tout d'abord, il décrit le processus décisionnel ainsi que les problèmes d'optimisation qui en découlent. Ensuite, le chapitre situe le cadre de la thèse dans la littérature, qui est le développement des routes et leurs rentabilités et aborde les différentes méthodes de résolution de ce problème. Dans ce contexte, nous présentons les modèles de choix d'itinéraires proposés dans la littérature pour estimer la demande d'un marché.

3.1 Processus de planification dans une compagnie aérienne

Les problèmes décisionnels rencontrés par une compagnie aérienne s'avèrent divers et compliqués. Il n'est pas possible de proposer une démarche globale et optimale pour les résoudre. Il faut plutôt envisager une approche structurée en tenant compte des interdépendances. Ainsi, les compagnies aériennes affrontent un ensemble de problèmes interdépendants de gestion des opérations sur différents horizons de temps [Barnhart and Cohn, 2004]. À long terme, la gestion de la capacité de production doit être planifiée comme tout processus industriel. Pour cela, on trouve la gestion de la flotte qui nécessite de déterminer les dimensions et sa composition ainsi que le type d'appropriation (achat ou location). La conception du réseau à opérer nécessite quant à elle de déterminer l'ensemble des routes à desservir. À moyen terme, une compagnie doit planifier la fréquence et les horaires des vols. Finalement, à court terme, il faut affecter les avions et les équipages. En pratique, le processus de planification fait partie des problèmes complexes en raison de la nature des problèmes abordés. Il a donc été décomposé en quatre sous-problèmes d'optimisation [Belobaba et al., 2015, Eltoukhy et al., 2017, Grosche, 2009b, Abdelghany and Abdelghany, 2018a, Garrow, 2016] (voir la figure 3.1) :

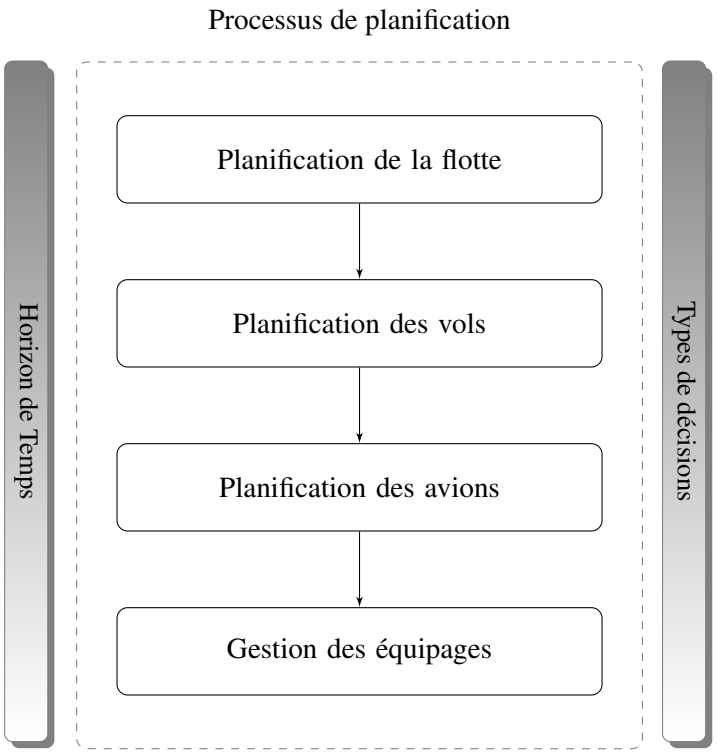


FIGURE 3.1 – Processus de planification pour une compagnie aérienne.

Par la suite, nous donnons brièvement une description de chaque étape. Pour commencer la description, nous présentons quelques définitions (dont la plupart ont été tirées de [Belobaba et al., 2015]) du secteur aérien qui seront utilisées par la suite.

3.1.1 Définitions et terminologie

Définition 3.1.1 (Segment de vol)**.** Un *segment de vol* (en anglais *flight leg*) est une desserte assurée par un vol. Cette liaison correspond à un vol direct. Il est défini par une heure de départ, une heure d'arrivée, un aéroport de départ, un aéroport d'arrivée et un numéro de vol. Il caractérise un mouvement de déplacement d'un avion constitué d'un décollage survenant à l'heure de départ suivi d'un seul atterrissage survenant à l'heure d'arrivée, reliant ainsi l'aéroport de départ à l'aéroport d'arrivée.

Le segment de vol est une entité insécable du transport aérien puisqu'il s'agit du déplacement de l'avion associé à ce segment.

Définition 3.1.2 (Marché)**.** Un *marché* (en anglais *market*) est défini par une demande entre un aéroport d'origine et un aéroport de destination pendant une période donnée. Il représente la demande des passagers qui voyagent d'une origine à une destination pendant une période donnée.

Par exemple, les personnes voulant se rendre à Paris depuis Nice au mois de juin représentent le marché de Nice & Paris pour juin.

Définition 3.1.3 (Taille de marché)**.** Une *taille de marché* (en anglais *market size*) indique le nombre total de passagers (volume de passagers) qui ont l'intention de voyager par avion sur ce marché.

Définition 3.1.4 (Escale)**.** Une *escale* est un arrêt dans un aéroport intermédiaire où le passager ne change pas d'avion. Ce type de vol est appelé *vol avec escale* (en anglais *direct flight*) où le même vol assure plusieurs dessertes, mais uniquement avec le même avion. Quand il s'agit d'une seule *escale*, le vol comporte deux segments de vol.

Définition 3.1.5 (Correspondance)**.** Une *correspondance* consiste en un arrêt intermédiaire où le passager change d'avion. Ce type de vol est appelé un *vol avec correspondance* (en anglais *connecting flight*). Quand il s'agit d'une seule *correspondance*, le vol comporte deux segments de vol.

Définition 3.1.6 (Vol direct)**.** Un *vol direct* (en anglais *non-stop flight*) est un vol sans escale et sans correspondance, le passager se rend directement à l'aéroport de destination depuis l'aéroport d'origine.

Les vols directs et avec escale se caractérisent par un numéro de vol unique.

Définition 3.1.7 (Horaire de vols)**.** Un ensemble de vols définis par une paire d'origine et de destination (paire `OD`) constitue un *horaire de vols* (en anglais *schedule*) : chaque vol est caractérisé par un ensemble d'informations incluant l'heure de départ et l'heure d'arrivée. La gestion des horaires de vols fait partie des décisions opérationnelles où les compagnies aériennes proposent de construire un horaire de vols qui est cyclique sur une semaine.

Définition 3.1.8 (Itinéraire de passager)**.** Un *itinéraire de passager* est une séquence finie dans le temps de segments de vols que le passager emprunte pour rejoindre l'aéroport de sa destination finale depuis l'aéroport d'origine de son point de départ.

Un itinéraire de passager peut être :

— direct : l'itinéraire ne comporte qu'un seul segment de vol ;

— avec une ou plusieurs correspondances : l'itinéraire connaît plusieurs décollages, celui à l'aéroport d'origine ou à l'aéroport intermédiaire, et plusieurs atterrissages, à l'aéroport de destination, ou aux aéroports intermédiaires. L'itinéraire est caractérisé par plusieurs changements d'appareils ;

— avec une ou plusieurs escales : c'est le même type d'itinéraire avec une ou plusieurs correspondances sauf que celui-là est opéré par un seul appareil.

Définition 3.1.9 (Route)**.** Une *route* d'avion est une séquence finie de segments de vols consécutifs réalisés par un type spécifique d'avion partant d'un aéroport origine et allant à un aéroport destination.

Définition 3.1.10 (Partage de code)**.** Un *partage de code* (en anglais *code sharing*) est une action commerciale qui consiste à établir un partenariat entre les compagnies aériennes pour le partage de tronçons de vol. On parle de cette action quand une compagnie aérienne A vend des sièges sur le vol opéré par une compagnie aérienne B. On distingue ainsi deux types de compagnies aériennes pour ce vol : compagnie opérante (B) (en anglais *operating airline*) et compagnie commercialisant le billet de vol (A) (en anglais *marketing airline*).

Définition 3.1.11 (Système de réservation informatique)**.** Un système de réservation informatique (en anglais *Computer Reservations Systems*), *CRS* est utilisé par les agences de voyage pour réserver des billets d'avions. Ce type de système existe aussi dans d'autres domaines.

3.1.2 Description des sous-problèmes

3.1.2.1 Planification de la flotte

La planification de la flotte (en anglais *Fleet planning*) fait partie de la gestion des ressources d'une compagnie aérienne. Une flotte est un ensemble d'avions caractérisé par le nombre total des avions que la compagnie possède, ainsi que le type d'avion. Cette décision concerne également l'achat, la location ou bien simplement l'utilisation d'appareils déjà existants.

3.1.2.2 Planification des vols

Étant donné une flotte, le problème de la planification des vols *FS* (en anglais *Flight Scheduling*) a pour objectif de générer une table de segments de vols. Chaque segment de vol comporte les informations concernant l'origine, la destination, l'heure de départ et l'heure d'arrivée correspondantes, tout en tenant compte des enjeux économiques tels que la demande des passagers et le prix des billets [Barnhart and Cohn, 2004]. Ce problème se compose de trois étapes :

1. développement des routes ;
2. élaboration des horaires des vols ;
3. rentabilité des routes.

En premier lieu, il y a le développement des routes *RD* (en anglais *Route Development*) qui a pour objectif de décider la liste des marchés à desservir. En second lieu, il y a la confection des plans de fréquences et des horaires de vols pour déterminer le nombre de fois à opérer une route proposée et les horaires des vols sur cette route. En dernier lieu, il y a la rentabilité des routes pour évaluer le plan des routes prévu.

3.1.2.3 Planification des avions

La planification des avions vient en troisième étape. Elle tient compte de la table des horaires proposée dans l'étape précédente. Il existe deux parties à traiter pour planifier les avions de la flotte disponible :

1. affectation des types d'avions aux vols ;
2. construction des routes d'avion.

Affectation des types d'avions aux vols

Compte tenu de la table des vols sélectionnés ainsi que les horaires établis, cette étape a pour objectif de présenter une offre qui répond mieux à la demande. En effet, le problème d'affecter les types d'avions aux vols (en anglais *Fleet assignment*) vise à déterminer pour chaque segment de vol du réseau opéré quel type d'avion est le plus approprié de façon à maximiser le profit de la compagnie pour satisfaire les contraintes de la demande des passagers, de la composition de la flotte, et de la balance [Barnhart and Cohn, 2004]. Cette dernière contrainte permet la conservation de flot à chaque point du réseau pour chaque flotte. Il existe généralement trois types de contraintes à satisfaire dans ce problème :

1. couverture des vols ;
2. conservation de flot ;
3. disponibilité des avions.

Construction des routes d'avion

Une fois l'affectation des types d'avions aux segments de vol établie, le problème de construction des routes d'avion (en anglais *Aircraft routing*) a pour but de trouver pour chaque avion une séquence de segments de vols (*routage*) de façon à couvrir chaque segment de vol exactement une seule fois tout en respectant les contraintes d'entretien et de maintenance.

3.1.2.4 Planification d'équipages

Le problème d'affectation d'équipages (en anglais *Crew scheduling*) est ensuite résolu pour déterminer l'affectation des équipages, typiquement les pilotes et le personnel de cabine de façon à minimiser les coûts. En considérant les routes d'avion construites dans l'étape précédente du processus, les compagnies aériennes établissent un calendrier d'horaires des membres d'équipage qui spécifie la séquence des segments de vols en tenant compte des contraintes du personnel. En raison de sa complexité, ce problème est divisé en deux sous-problèmes (voir la figure 3.2) :

Le problème de construction des rotations d'équipages (en anglais *Crew pairing*) consiste à générer des mini-calendriers, appelés rotations (en anglais *pairing*), couvrant une période de 1 à 5 jours. Une rotation est une séquence de services de vols (segments de vol, attente, repos, *etc*.) effectuée par un équipage durant une période donnée. Cette suite débute et se termine au même aéroport. L'objectif de ce problème est de minimiser les coûts d'exploitation d'un équipage associé à couvrir tous les segments de vol dans le calendrier des vols proposé auparavant.

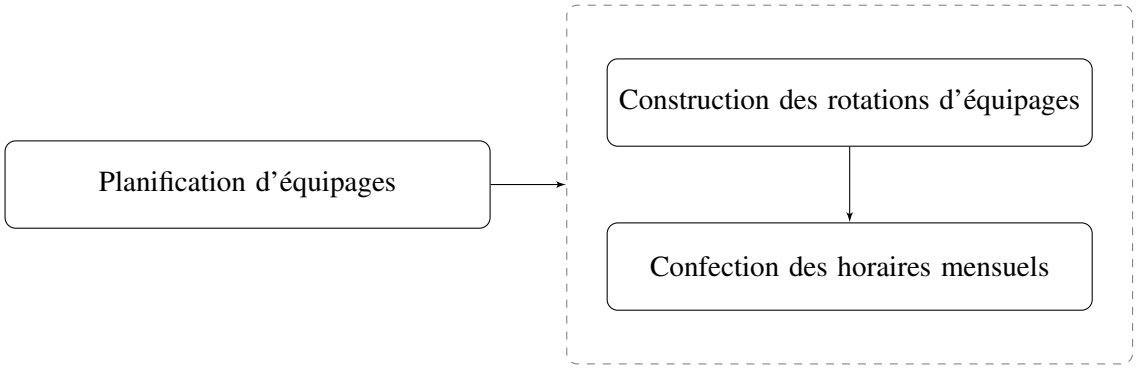


FIGURE 3.2 – Sous-problèmes de la planification d'équipages.

Ensuite, une fois le problème de construction des rotations d'équipages résolu, on construit les horaires mensuels pour chaque membre de l'équipage (en anglais *Crew assignment*) tout en prenant en considération ses contraintes de travail (vacances, formation, congés, *etc.*).

Ces deux sous-problèmes ne sont pas indépendants. En effet, le problème de construction des rotations d'équipages a pour objectif d'assembler les rotations générées précédemment aux horaires des membres d'équipages de sorte à maximiser le niveau de satisfaction du personnel [Belobaba et al., 2015].

3.1.3 Cadre de la thèse

Le cadre de la thèse se situe dans l'étape 2 du processus : planification des vols. Le problème *FS* réfère à une collection de modèles utilisés pour déterminer la structure du réseau à opérer et la sélection des marchés potentiels à servir, le flux des passagers ainsi que leurs choix d'itinéraires et le revenu et le coût impliqués par ces décisions. *Milanamos* propose une partie de ces modèles (voir la section 2.3 du chapitre 2).

Plus précisément, notre travail concerne la sélection des marchés *O&D* et la rentabilité des routes générées. Cette partie représente la clé du processus décisionnel de la compagnie aérienne.

Tout d'abord, nous proposons le problème *FRP* qui consiste à sélectionner de nouveaux marchés en tenant compte des préférences des passagers. Il permet de donner une vue centrée sur un marché et déterminer ainsi les itinéraires réalistes passant par ce nouveau vol. Par conséquent, le résultat du *FRP* permet de mieux visualiser le réseau de la compagnie aérienne en se focalisant que sur les itinéraires intéressants reliant cet ensemble de marchés potentiels sur lesquels la compagnie aérienne peut attendre une part de marché quand elle décide d'ajouter un nouveau vol ou d'augmenter la fréquence. Par ailleurs, ce résultat permettra d'accélérer les calculs des parts de marché. Ensuite, nous développons une approche travaillant sur ce réseau pour calculer efficacement ces parts de marchés en réduisant la complexité combinatoire sur l'ensemble des itinéraires reliant chaque paire *O&D*.

Le reste du chapitre est organisé comme suit. La section 3.2 présente en détail le problème de la planification des vols, à savoir, les modèles utilisés et les méthodes de résolution. Ensuite, la section 3.3 décrit les modèles des parts de marché utilisés dans la littérature.

3.2 Planification des vols

3.2.1 Présentation du problème

Compte tenu du choix de la compagnie des avions et de la composition de la flotte qui détermine la disponibilité et la capacité de chaque avion ainsi que les différentes caractéristiques, la prochaine étape dans le processus de la planification est la planification des vols *FS*. La décision concernant les marchés à desservir fait partie de cette étape. Il est à noter que dans certains cas, cette démarche peut être inversée. C'est le cas où une route rentable est identifiée. Ainsi, pour trouver les marchés à desservir, une prévision de la demande et du revenu (qui comprend les coûts de chaque opération) est nécessaire pour la période concernée.

Le problème *FS* comprend les étapes suivantes (voir la figure 3.3) [Eltoukhy et al., 2017, Grosche, 2009b] :

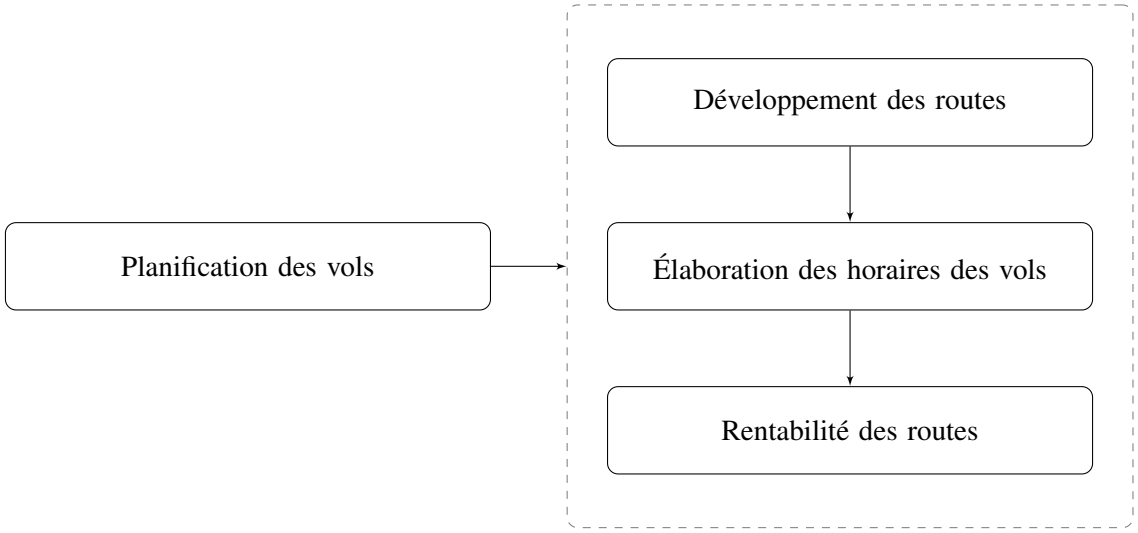


FIGURE 3.3 – Étapes du problème *FS*.

Le problème *FS* est généralement résolu en trois étapes. La première étape de la planification est le développement des routes, qui prend en compte l'identification des paires de villes d'origine et de destination (les marchés), créant ainsi le réseau de routes que la compagnie aérienne souhaite desservir. La conception est basée sur des décisions stratégiques ainsi que sur des prévisions de trafic incluant les variations de la demande. Le développement des routes a été expliqué de différentes manières. Halpern and Graham [2015] le définit comme *"les activités de marketing entreprises par les compagnies aériennes dans le but d'attirer de nouvelles routes. Cela indique qu'il inclut l'attraction, l'initiation, l'expansion, la rétention, ou toute modification éventuelle des prix, de la fréquence, de la capacité ou du nombre de destinations desservies sans escale"*. L'objectif le plus évident du développement des routes est d'encourager les compagnies aériennes à exploiter de nouvelles routes [Barnhart et al., 2003b].

La deuxième étape, qui est l'élaboration des horaires des vols, est consacrée pour l'attribution du nombre et des horaires de vols pour chaque paire origine-destination [Reiners et al., 2012]. Tandis que l'évaluation de la rentabilité des routes est effectuée dans la troisième étape. L'évalua-

tion de la rentabilité des routes se fait en vérifiant la faisabilité de chaque segment de vol et en veillant à minimiser les coûts de chaque vol. Afin d'améliorer le calendrier des vols construit, ces deux dernières étapes sont répétées jusqu'à ce qu'aucune amélioration ne puisse être apportée. Les compagnies aériennes doivent déterminer certains aspects tels que les marchés prévus desservis, le type de vols (vols sans escale ou vols avec correspondance) et la fréquence des vols sur chaque marché.

3.2.1.1 Développement des routes

L'objectif du problème de développement des routes *RD* ou la conception du réseau (en anglais *Network Design*) est d'identifier les paires de villes origine-destination (*O&D* ou marchés) que la compagnie aérienne a l'intention de desservir. Une fois les marchés sélectionnés, la compagnie aérienne doit décider son réseau de routes. Chaque *O&D* peut être effectué soit par un vol direct (vol simple), ou par un vol avec escale ou correspondance (vol multi-segments).

Deux sous-problématiques interviennent dans le problème *RD* : la sélection des marchés à desservir et la sélection de la fréquence. Les deux sous-problématiques partagent les mêmes objectifs : la rentabilité de la décision à prendre. Ainsi, la compagnie aérienne doit prendre deux types de décisions :

— Augmenter ou réduire la fréquence sur des vols existants ?

— Sélectionner des nouveaux marchés et décider quelle fréquence à offrir ?

La décision à prendre concernant la sélection des marchés à desservir est plutôt stratégique et fait appel à plusieurs enjeux économiques notamment, les coûts d'ajout d'une nouvelle destination [Belobaba et al., 2015].

D'une part, il faut estimer la demande et le revenu pour évaluer la rentabilité de la route. D'autre part, il faut regarder le type de réseau opéré.

En effet, il existe trois types de réseaux : *Point à point*, *Hub & Spoke* et *Hybride* [Abdelghany and Abdelghany, 2018a].

— ***Point à point*** : ce modèle permet de minimiser les temps de voyage et les escales en proposant du direct. L'inconvénient est que la compagnie desservira moins de routes depuis le même point de départ.

— ***Hub & Spoke*** : la structure du réseau de *Hub & Spoke* (étoile) permet aux compagnies aériennes de desservir plus de marchés et d'augmenter la fréquence des vols. Cela permet de réduire les coûts. Ce modèle a émergé de l'industrie aérienne suite à la politique de déréglementation mise en pratique par l'état. Il devient plus simple d'entrer ou de sortir d'un marché en ouvrant ou fermant des routes. Sur des destinations à forte demande, la compagnie continue à proposer des lignes directes. En revanche, sur des liaisons à faible demande, ce modèle permet de regrouper la demande sur des points de transfert appelés *hubs* avant de les redistribuer localement vers des points terminaux appelés *spokes*. En effet, la compagnie peut bien jouer sur l'aspect temps en réduisant la fréquence. Toutefois, les passagers peuvent être attirés par des compagnies concurrentes. Ce qui va jouer un effet négatif sur la demande et par conséquent le revenu de la compagnie. Cette stratégie de regroupement permet aux compagnies de maximiser le taux de remplissage.

— ***Hybride*** : dans certains cas, pour tirer parti de la structure en étoile, les compagnies aériennes qui adoptent la structure point à point ont tendance à définir certaines villes comme

des points de concentration ou focus. Il est prévu que les vols se rencontrent dans ces villes cibles dans un laps de temps prédéfini pour permettre des correspondances. Cette structure est connue sous le nom de structure de réseau hybride. La structure du réseau de la compagnie aérienne hybride inclut à la fois les caractéristiques de la structure de réseau en étoile et point à point. Ainsi, ces compagnies aériennes ont des destinations principales qui fonctionnent comme des *hubs*, où elles assurent des vols à destination et en provenance de ces *hubs*. Ils relient également les destinations liées directement avec des vols sans escale.

Exemple 3.2.1 (Types de réseau aérien)**.** La figure ci-dessous représente les deux types de modèles : le modèle *Point à point* (voir la figure 3.4) et le modèle *Hub & Spoke* (voir la figure 3.5).

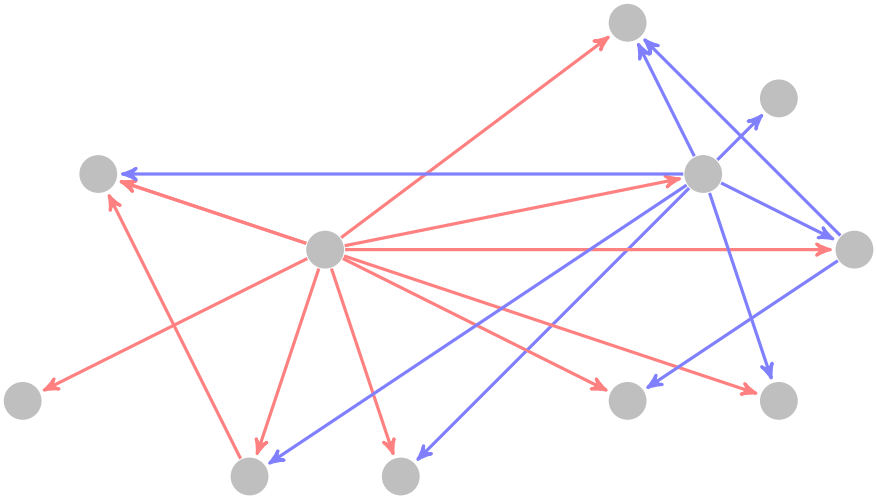


FIGURE 3.4 – Réseau **Point à point**.

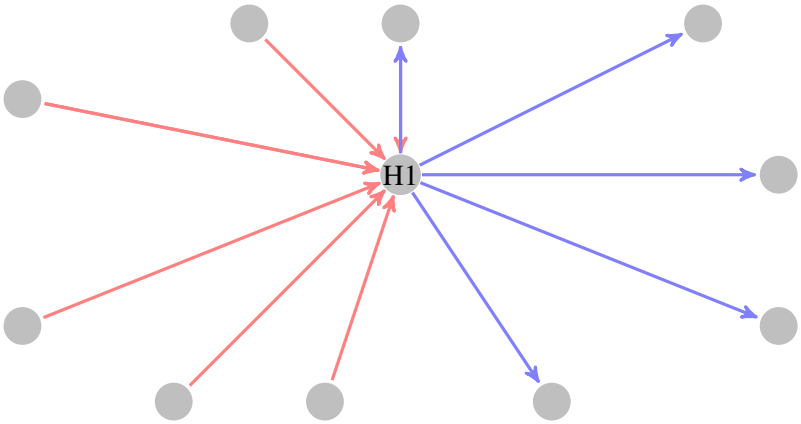


FIGURE 3.5 – Réseau **Hub & Spoke**.

Dans l'exemple de la figure 3.5, il y a 5 vols qui arrivent (arcs de couleur rouge) et 5 vols qui partent (arcs de couleur bleue). Chaque segment de vol arrivant ou partant du *hub* dessert six marchés *O&D* y compris le marché local entre le *hub* et le *spoke*, ainsi que 5 marchés « en correspondance » supplémentaires. Ce qui permet de desservir au total 35 marchés avec simplement 10

segments de vols et seulement 5 avions traversant le *hub*, sous l'hypothèse que l'avion effectue le retour pour revenir à la case de départ. Alors que dans le modèle *Point à point* (voir la figure 3.4) qui fournit des services directs, il faudra 35 segments de vols et des centaines d'avions en fonction des contraintes de planification.

D'une manière générale pour n segments de vols et 1 *hub* connectant ses segments de vols, il y a : $n^2 + 2 \times n$ marchés desservis dans le modèle *Hub & Spoke*. Ainsi, les passagers qui transitent par un *hub* sont plus rentables pour les compagnies aériennes.

3.2.1.2 Élaboration des horaires des vols

La prochaine étape dans le processus du *FS* est l'élaboration des horaires des vols. Étant donné une liste des routes à opérer dans le réseau de la compagnie aérienne, l'élaboration ou la confection des horaires a pour objectif de répondre aux questions suivantes :

— Combien de vols ?

— À quelle heure la compagnie peut-elle proposer le vol ?

— Quels jours peut-elle opérer ce vol ?

En général, cette étape commence un an ou plus avant le départ du vol. L'étape d'élaboration des horaires de vols est une étape cruciale permettant de concrétiser l'offre de service que la compagnie aérienne souhaite proposer pour cerner la demande du marché. En général, le plan de fréquence est établi en premier en se basant sur la planification des routes qui évalue la disponibilité des avions résultant du plan de la flotte. En second, les tables horaires sont élaborées en tenant compte de la fréquence de service choisi pour chaque route.

3.2.2 Rentabilité des routes

La prochaine étape dans le processus *FS* est d'évaluer la rentabilité des routes en se basant sur les tables des horaires établies dans l'étape précédente. Il est à noter que dans certains cas, ces démarches peuvent être inversées.

Les modèles de planification de réseau (en anglais *Network planning*), également appelés modèles de rentabilité des routes (en anglais *Route profitability*) sont utilisés pour évaluer la rentabilité des routes en se basant sur la structure du réseau, le plan des routes et les tables des horaires des vols [Barnhart and Smith, 2012, Belobaba et al., 2015, Garrow, 2016]. Ils aident par exemple les compagnies aériennes à mettre en œuvre des scénarios de fusion et d'acquisition, une analyse des itinéraires, des scénarios de partage de code, des études sur le temps minimum de connexion *MCT* (en anglais *Minimum Connecting Time*), des études sur l'élasticité-prix, des études de localisation des *hubs* et des décisions d'achat d'appareil.

Sur le plan conceptuel, les modèles de planification de réseau désignent un ensemble de modèles utilisés pour déterminer le nombre de passagers désirant voler, les itinéraires (définis comme un vol ou une séquence de vols) choisis, ainsi que les implications en termes de revenus et de coûts sur les vols choisis. Ces modèles correspondent à la partie *Flight Simulator* de l'application `PlanetOptim` (voir la section 2.3 du chapitre 2).

L'objectif de ces modèles est de permettre aux compagnies aériennes de choisir des itinéraires afin de maximiser leurs profits, en fonction d'un ensemble d'itinéraires candidats, de la demande estimée et des contraintes de flotte et de capacité [Belobaba et al., 2015].

3.2.2.1 Étapes du processus

Les modèles de planification de réseau se réfèrent à une collection des sous modèles [Jacobs et al., 2012] (voir la figure 3.6) :

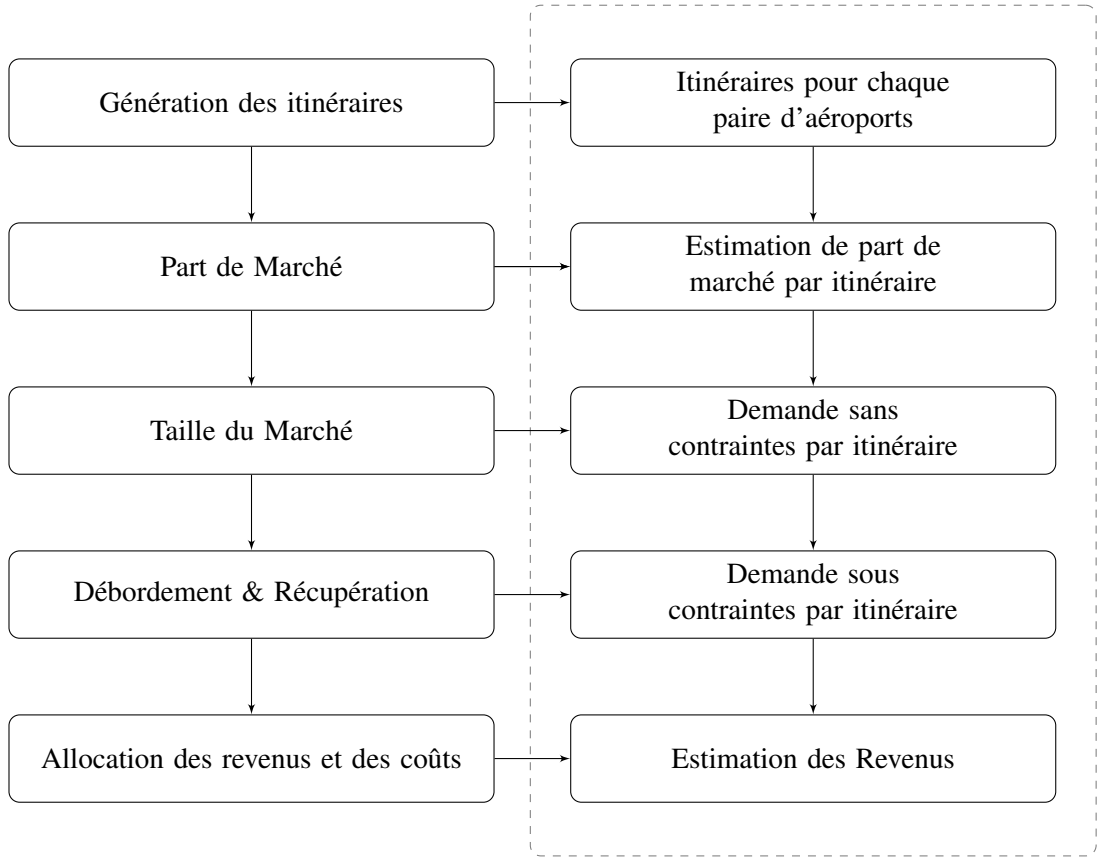


FIGURE 3.6 – Étapes de planification de réseau.

3.2.2.2 Description des étapes

Génération des itinéraires

La première étape consiste à générer des itinéraires réalistes entre l'ensemble des paires origine-destination (*O&D*) choisies, la génération des itinéraires se fait en connectant les segments de vols. Les données sont obtenues à partir des données de l'*OAG* (*Official Airline Guide*), une entreprise qui analyse les horaires et les fréquences des compagnies aériennes du monde entier. Elles contiennent des informations pour chaque vol incluant la compagnie opérante, l'origine, la destination, le numéro de vol, l'heure de départ et d'arrivée, l'équipement, les jours d'opération et le temps de vol. Les itinéraires sont définis comme un vol ou une séquence de vols utilisé pour voyager entre la paire d'aéroports. En général, lors de la génération des itinéraires, les algorithmes vérifient deux points :

1. la distance en se basant sur la logique de circuit pour éliminer les itinéraires non raisonnables.

2. les temps minimum et maximum de connexion pour assurer que les correspondances irréalistes ne sont pas autorisées.

Estimation de la part de marché

Cette étape consiste à calculer la part de marché pour chaque itinéraire généré. En effet, il existe différents types de modèles de part de marché qui sont utilisés en pratique et peuvent être caractérisés de manière générale selon la méthode sur laquelle ils s'appuient. Les deux types de modèles de part de marché sont abordés dans la section 3.3 :

— des méthodes qualitatives utilisant le modèle *QSI* ;

— des méthodes de choix discrets basées sur la fonction *logit*.

Détermination de la taille du marché

Une fois que les parts des marchés sont calculées, la demande des passagers pour chaque itinéraire est obtenue en multipliant la part de marché estimée, qui représente le pourcentage des passagers attendus, par la demande totale prédite, ou bien le nombre des passagers voyageant entre la paire d'aéroports si la compagnie aérienne possède un historique de données sur ce marché.

Débordement & Récupération et Allocation des revenus et des coûts

Pour mieux optimiser le processus et mieux se projeter dans le cas réel, les modèles *spill&recapture* sont appliqués pour faire une réallocation des passagers. En effet, un débordement (en anglais *spill*) se produit quand la demande excède la capacité disponible sur certains vols. Pour ne pas perdre ces passagers, la compagnie cherche à proposer d'autres alternatives pour les récupérer, nous parlons alors de récupération (en anglais *recapture*). À la fin de cette étape, la compagnie aérienne estime deux types de demandes [Lohatepanont and Barnhart, 2004, Abdelghany and Abdelghany, 2018b]. :

— demande contrainte, soumise à des contraintes de capacité ;

— demande non contrainte, c'est-à-dire la demande maximale que la compagnie aérienne est capable de capturer.

Finalement, des modèles d'allocation des revenus et coûts sont utilisés pour déterminer la rentabilité de l'ensemble des itinéraires choisis.

3.2.3 Études menées dans la littérature

3.2.3.1 Types de modélisation

Yu and Yang [1998] reformulent mathématiquement le problème de développement du réseau dans l'industrie aérienne. En effet, il peut être présenté comme suit : trouver la structure optimale du réseau avec les routes transportant le flux des passagers avec un coût de transport total minimum. Ce problème a été démontré comme un problème critique car il impacte directement la part de marché de la compagnie aérienne.

Les différentes approches pour résoudre les problèmes de planification dans les compagnies aériennes se basent principalement sur un mélange entre la théorie des réseaux, la programmation

mathématique et la théorie de l'utilité [Barnhart et al., 2003a, Grosche, 2009b, Eltoukhy et al., 2017].

L'objectif principal reste de trouver une solution qui minimise les coûts ou maximise les profits selon des contraintes opérationnelles. La tâche est compliquée et voici une liste non exhaustive de divers éléments à prendre en cause :

1. Plusieurs aéroports avec des contraintes diverses.
2. Plusieurs types d'avions avec différentes caractéristiques : coût, capacité et maintenance.
3. Nombreux personnels avec des contraintes de disponibilités et de préférences à gérer.
4. Un large choix de marchés *O&D* à opérer, quand et comment. Chaque marché a une taille différente.

Les modèles proposés dans la littérature tiennent compte de ces éléments et proposent de résoudre une combinaison des sous-problèmes. La plupart proposent la modélisation du problème *RD* avec le problème d'affectation des vols. En effet, ces deux problèmes partagent les mêmes contraintes : la structure du réseau opéré, les origines, les destinations et la fréquence. Le marché à servir ne peut pas être déterminé sans avoir pris en considération la fréquence et la demande. D'autres proposent des modèles pour la localisation du *hub* pour une compagnie aérienne. Certains se basent sur des modèles économiques et d'autres proposent des modèles mathématiques. Nous distinguons, ainsi, quatre types d'études suivant :

1. Des études portent sur la modélisation de la structure du réseau à opérer. Dans ce genre d'études, on distingue deux objectifs :
   (a) la conception du réseau, y compris la sélection des routes à supprimer ou ajouter ;
   (b) la localisation du *hub*.
2. Des études portent sur l'étude d'une opportunité économique. Dans ce genre d'études, les auteurs utilisent des modèles économiques pour modéliser le choix de la sélection des itinéraires.
3. Une approche hybride entre les modèles de conception du réseau et les modèles de choix ;
4. Des études ont pour objectif la conception du réseau couplée à l'affectation des vols.

Catégorie 1

Dans la littérature, la plupart des modèles impliquent le choix à partir d'un ensemble de marchés proposés. Typiquement, les vols sont représentés par des arcs. Un passager allant de Paris à Londres est représenté différemment d'un passager qui va de Nice à Paris. Ce modèle contient deux types de variables, une variable discrète pour modéliser la décision sur le choix et une variable continue pour modéliser la décision sur le flux [Magnanti and Wong, 1984].

Kim and Barnhart [1999] présente trois modèles pour le problème de la conception du réseau et propose une méthode de la génération des colonnes pour le résoudre.

Certains chercheurs comme [Jaillet et al., 1996] ont proposé des programmes mathématiques en nombres entiers pour modéliser des réseaux aériens avec capacité. La matrice des demandes sur les villes origine-destination est fournie. Ensuite, le réseau conçu en *Hub & Spoke* est renvoyé.

Catégorie 2

Sha et al. [2015] ont proposé un modèle de choix discret utilisant la théorie de l'utilité. La fonction d'utilité est modélisée comme une combinaison linaire de plusieurs variables corrélées. Ainsi, les préférences de chaque variable sont estimées en utilisant des données historiques. Les auteurs modélisent la décision de la sélection des routes comme une variable binaire.

Catégorie 3

Dobson and Lederer [1993], Lederer and Nambimadom [1998] ont étudié ce problème couplé avec la planification des vols en tenant compte de la concurrence. Ainsi, les chercheurs calculent la demande pour chaque route comme une fonction de la qualité de service et le prix de toutes les routes concurrentes. En outre, ils proposent une heuristique pour trouver les prix des routes avec les horaires qui maximisent le profit de la compagnie aérienne. La plupart des études s'adressent généralement à des problèmes de minimisation des coûts. Cette étude tient en compte deux visions : le passager et la compagnie aérienne. La demande est calculée pour chaque route avec la fonction de la régression logistique (*logit*). Par ailleurs, Adler [2005] ont utilisé la théorie des jeux pour modéliser l'aspect de la concurrence.

Catégorie 4

Budhiraja et al. [2006], Zhang et al. [2016] ont proposé un modèle qui résout le problème de développement des routes avec le problème d'affectation d'avions aux vols. Le modèle intègre les préférences des passagers pour tenir compte de la concurrence. Afin de modéliser ces préférences, les auteurs utilisent la fonction de la régression logistique pour le calcul de la part de marché. Les chercheurs traitent la demande dynamiquement en fonction de l'offre et considèrent les deux types de demandes, contrainte et non contrainte. Le problème défini par les auteurs considère un ensemble d'itinéraires possibles sur un marché donné et la décision porte sur quels itinéraires la compagnie aérienne va opérer en tenant compte de la flotte disponible (le nombre d'avions, et le type de chaque avion). Ainsi, les auteurs ne traitent que le développement local c'est-à-dire au niveau d'un marché quelconque et non pas le réseau global de la compagnie aérienne.

Lohatepanont and Barnhart [2004] proposent une version du problème d'affectation d'avions aux vols permettant de sélectionner des segments de vols à inclure dans le but de maximiser le profit. Ils ont prouvé qu'il y a un gain potentiel dans le revenu. Généralement, la génération des horaires se fait manuellement au début dû à la complexité du problème. Ensuite, des techniques d'optimisation viennent pour optimiser, c'est-à-dire, supprimer des vols non rentables et les remplacer par de nouveaux rentables.

Schön [2007] a proposé un modèle orienté marché qui résout le problème *RD* avec le problème de planification des vols. Le problème a été modélisé sous forme d'un programme mixte binaire tel que la fonction objectif est concave avec des contraintes linéaires. De plus, le modèle intègre des informations sur la tarification suivant les classes des passagers.

Di Wang et al. [2014] étudient l'effet du *spill&recapture* dans le cas de la planification de la capacité. En effet, le *spill* est un phénomène qui survient quand la demande dépasse la capacité ou bien quand l'itinéraire en question n'est plus disponible et donc cela revient à voir les autres itinéraires disponibles (*recapture*). Les auteurs proposent un nouveau modèle d'affectation d'avions

aux vols. Comme mentionné par les auteurs, la décision concernant la sélection des marchés est plus compliquée car cela engendre un coût supplémentaire lié à l'entrée de ce marché. Les auteurs proposent, ainsi, un ensemble des marchés potentiels qui présentent des bénéfices pour l'ensemble des vols du réseau. La sélection des marchés à entrer s'appuie sur la sélection de la fréquence. En fait, l'identification d'itinéraires à marquer détermine naturellement si un marché doit être ouvert.

3.2.3.2 Méthodes de résolution

D'un autre côté, on trouve des modèles de conception de réseau, qui ont fait l'objet de plusieurs études dans la littérature. D'autres ont utilisé de l'optimisation mathématique linéaire pour résoudre ce problème couplé avec le problème d'affectation des vols. La résolution de ce modèle utilise les techniques de génération des colonnes et lignes, avec comme fonction objectif, la maximisation du profit. Les entrées du modèle incluent une liste des marchés, les taux de récupération, les données sur la demande et les caractéristiques de la flotte. Quant aux sorties, c'est une liste des vols à ajouter ou supprimer à travers l'affectation des vols. Puis, il y a ceux qui ont formulé le problème de la sélection des routes sous forme d'un programme à variable mixte (voir la figure 3.7).

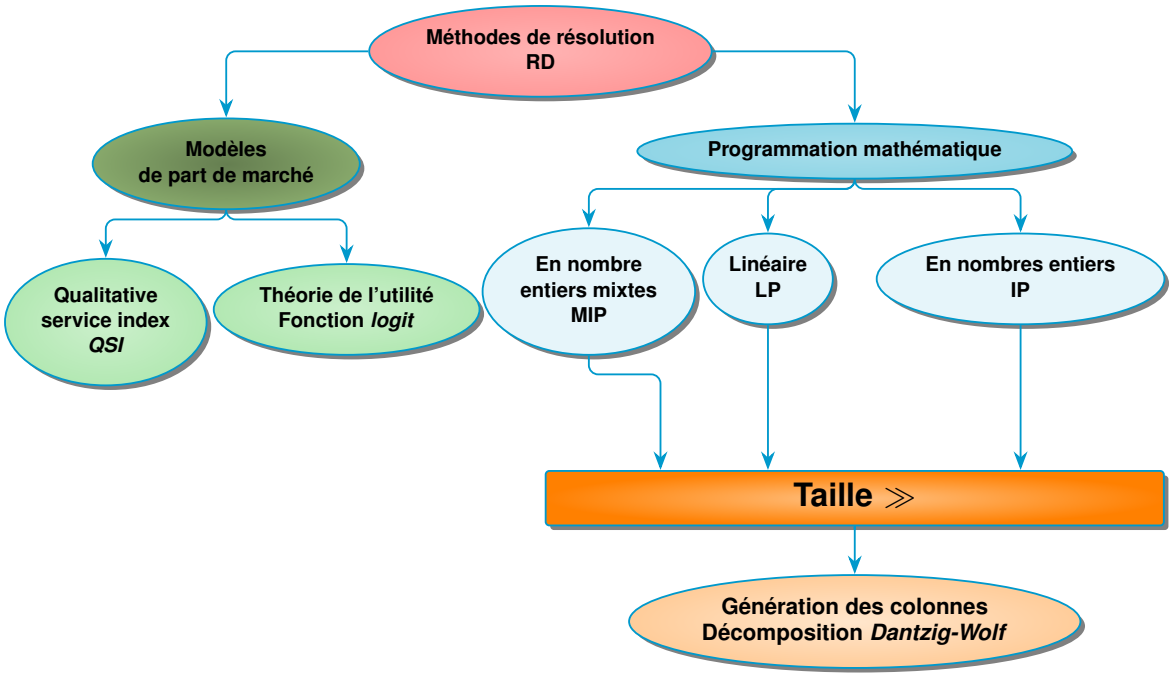


FIGURE 3.7 – Schéma des méthodes de résolution du problème *FS*.

Des modèles mathématiques ont été largement utilisés pour modéliser les problèmes de la planification dans une compagnie aérienne. Trois problèmes d'optimisation combinatoire principaux sont utilisés pour la modélisation :

— problème de partition ;
— problème de flot multi-commodités ;
— problème d'*Euler*.

Le problème d'*Euler* est résolu en temps polynomial tandis que les autres sont des problèmes NP-complet. Des méthodes exactes comme l'algorithme de séparation et évaluation (en anglais *Branch & Bound*) ont été utilisées. Quand il s'agit des problèmes de grandes tailles, d'autres techniques de résolutions sont appelées comme la génération des colonnes. Le problème pour trouver l'affectation d'avion ou l'équipage aux vols consiste à trouver le flot à coût minimum. Quand il y a plusieurs types d'avions, des versions de problème flot comme multi-commodités sont utilisées. Quand le problème devient grand, des techniques spécifiques comme la relaxation Lagrangienne ou la décomposition de *Dantzig-Wolf* sont utilisées pour les LP relaxations [Eltoukhy et al., 2017].

3.2.3.3 Tableau récapitulatif

Le tableau 3.1 présente un récapitulatif des études menées dans la littérature sur le problème de développement du réseau (RD). Les colonnes de la table présentent les informations suivantes :

— Auteurs ;

— problème étudié ;

— objectifs ;

— formulation mathématique ;

— méthodes de résolution.

3.2.3.4 Résultats

Bien que les considérations concernant les aéroports, les avions, le nombre personnel et les marchés constituent à elles seules un énorme défi pour les responsables des compagnies aériennes, la tâche de planification des horaires des vols est rendue encore plus difficile par deux facteurs principaux :

— l'ampleur du problème (les grandes compagnies aériennes exploitent des milliers de vols avec des centaines d'équipages et d'avions) ;

— le fait que le système est sujet à des incertitudes affectant la demande de passagers, les prix et les opérations des compagnies aériennes, entre autres.

À notre connaissance, tous les travaux de recherche menés sur le problème de planification des horaires ne se résolvent pas en temps réel, car les temps de calcul sont très importants. Ainsi, ils partent d'une solution de départ établie manuellement. Ensuite, ils essayent de l'améliorer avec les modèles d'optimisations couplés à des outils de simulation. Par exemple, [Lohatepanont and Barnhart, 2004] résolvent les problèmes d'une grande compagnie aérienne qui contient environ 800 segments de vol potentiels pour l'étude, 65 000 itinéraires et 165 avions, résultant en des formulations avec 30 000 à 60 000 lignes et 50 000 à 65 000 colonnes, et des temps de solution allant de 12 heures à plus de 3 jours.

Néanmoins, en raison de l'incapacité de ces modèles d'optimisation à résoudre le problème en temps réel, même pour une résolution partielle qui réduit sans doute la complexité du problème mais génère des solutions de moindre qualité. Il est encore difficile à ce jour de résoudre un sous-problème en temps réel indépendamment des autres. Les chercheurs ainsi se contentent juste d'évaluer le gain généré.

Auteurs	Problème étudié	Objectifs	Formulation	Méthodes de Résolution
[Magnanti and Wong, 1984]	*network design*	concevoir la topologie du réseau aérien	MIP	*B & B*
[Jaillet et al., 1996]	*network design*	trouver le réseau optimal	PLNE	heuristique
[Kim and Barnhart, 1999]	problème de flot multi-commodités	analysez la concurrence vis-à-vis du prix, fréquence et taille des avions	*logit*	génération des colonnes
[Sha et al., 2015]	planification des routes	sélectionner des routes à ajouter ou à supprimer	théorie de l'utilité choix discrets	données historiques estimation
[Dobson and Lederer, 1993]	*network design*, *schedule design*	maximiser le profit en tenant compte de la concurrence	*logit*	heuristique
[Lederer and Nambimadom, 1998]	*network design*, *schedule design*	maximiser le profit en tenant compte de la concurrence	*logit*	heuristique
[Adler, 2005]	*network design*	analysez la concurrence vis-à-vis du prix, fréquence et taille des avions	*logit*	théorie des jeux
[Lohatepanont and Barnhart, 2004]	*schedule design*, *fleet assignement*	sélectionner des segments de vols maximisant le profit et minimisant les coûts de débordement	MIP	génération des colonnes décomposition de *Benders*
[Budhiraja et al., 2006]	*schedule design*, *fleet assignement*	sélectionner les itinéraires suivant la demande de chaque marché	LP *logit*	CPLEX
[Schön, 2007]	*schedule design*, *fleet assignement*, *market share*	affecter les vols en se basant sur la part de marché de l'itinéraire modéliser le débordement et la récupération	PLNE	décomposition de *Benders*
[Di Wang et al., 2014]	*schedule design*, *fleet assignement*, *pricing*	sélectionner les segments de vols avec les prix correspondants	MIP *logit*	CPLEX
[Zhang et al., 2016]	*schedule design*, *fleet assignement*	minimiser les coûts d'opérations d'avion	MIP	heuristique

schedule design : élaboration des horaires de vols
network design : conception du réseau
fleet assignement : affectation des vols
logit : régression logistique

TABLE 3.1 – Récapitulatif des travaux traitant le problème *RD* dans la littérature.

3.3 Modèles de part de marché : *MS*

Après avoir introduit les principales composantes des modèles de développement des routes, cette section fournit une description détaillée des modèles de parts de marché utilisés pour l'estimation de la part de marché (voir l'étape 2 de la figure 3.6).

3.3.1 Utilités

Les modèles d'estimation de parts de marché *MS* (en anglais *Itinerary Market Share Models*) sont utilisés pour estimer la probabilité qu'un passager sélectionne un itinéraire spécifique reliant une paire d'aéroports [Garrow, 2016]. Les itinéraires sont les produits finalement achetés par les passagers. Un itinéraire peut être simplement un vol direct ou bien un vol avec escale ou correspondance. Les vols directs, avec escale ou avec correspondance représentent l'ensemble des itinéraires que le passager peut choisir. L'objectif de ces modèles est de mesurer l'attractivité de chaque itinéraire pour un seul passager. Cette attractivité peut être interprétée comme la part de marché de l'itinéraire. La part de marché d'une compagnie aérienne A est définie par la proportion de la demande totale du marché que la compagnie aérienne peut capturer [Belobaba et al., 2015].

La compagnie aérienne utilise ce type de modèles pour estimer la part de marché attendue d'un vol donné sur un marché *O&D* donné. Ces modèles mesurent la profitabilité de ce vol. Un moyen éprouvé est de refléter le processus de décision des passagers, sur la base des données empiriques des passagers réels. Prenons l'exemple 3.3.1.

Exemple 3.3.1 (Cas d'étude de part de marché)**.** Supposons qu'un passager veuille acheter un billet pour aller de *Nice*/France (`NCE`) à *Bangkok*/Thaïlande (`BKK`). En pratique, le passager se rend dans une agence de voyage à *Nice* et demande s'il existe une possibilité pour se rendre à *Bangkok*. Le système de réservation de l'agence de voyage affichera toutes les possibilités disponibles, notamment les vols sans escale et avec correspondance pour faire le transfert. Généralement, le transfert se fait *via* `CDG`, `FRA`, `AMS`, `LHR`, et d'autres *hub*s de correspondance. La question qui se pose est sur quels types de critères de choix le passager se basera pour choisir l'itinéraire le plus attrayant ? Probablement, le passager comparera la durée totale du voyage, la réputation de la compagnie aérienne concernée, l'heure de départ et d'arrivée et le tarif proposé (voir la figure 3.8). En fonction de l'importance de ces critères, le passager décidera finalement l'option qui correspond le plus à ses critères.

Dans la figure 3.8, la formule pour estimer la part de marché correspond à la formule utilisée dans le cas des modèles basés sur la fonction *logit* (voir la section 3.3.2.2).

<table>
<tr><th>Étapes</th><th>Descriptions</th></tr>
<tr><td>1- Étude du profil de la demande</td><td>Trajet demandé : NCE-BKK</td></tr>
<tr><td>2- Simulation de la demande sur le CRS</td><td>
<table>
<tr><th>Compagnie</th><th>Départ</th><th>Arrivée</th></tr>
<tr><td>AF</td><td>14 :00</td><td>23 :00</td></tr>
<tr><td>AF</td><td>07 :00</td><td>20 :00</td></tr>
<tr><td>LH</td><td>05 :00</td><td>21 :00</td></tr>
</table>
</td></tr>
<tr><td>3 - Estimation de l'utilité du choix</td><td>"Utilité" attendue de l'itinéraire i ($util(i)$) =</td><td>durée de trajet * β_1 + arrêts * β_2 + prix* β_3 + ...</td></tr>
<tr><td>4 - Estimation de la part de marché probable</td><td>Part de Marché capturée sur l'itinéraire i =</td><td>$\frac{exp(util(i))}{\sum_{j \in J} exp(util(j))}$</td></tr>
</table>

FIGURE 3.8 – Schéma du fonctionnement des modèles de *MS* basés sur les modèles *logit*

Les modèles *MS* sont similaires à ceux de *revenue management* dans le sens où ils estiment la demande pour le transport aérien. Ces modèles sont utilisés pour aider à prendre des décisions concernant les destinations à servir, le type d'équipement à utiliser et prédire le comportement de l'évolution de la demande en fonction de la fréquence de vol, du niveau de service et du type d'appareil [Garrow, 2016]. Ces modèles se rapportent aux modèles de la prévision de la demande puisque la demande totale pour un itinéraire spécifique est représentée par l'ensemble des choix faits par les passagers.

Cette approche se distingue des techniques statistiques traditionnellement utilisées par les compagnies aériennes pour la prévision de la demande. La prévision de la demande se présente sous deux formes : prévision du marché total et prévision de la part de marché capturée par la compagnie aérienne sur un itinéraire. Ces approches classiques se basent essentiellement sur l'historique ou bien sur des facteurs démographiques et économiques en l'absence de l'historique. Cependant, elles sont limitées pour trois raisons :

— la plupart se base principalement sur l'historique qui n'est pas toujours disponible ;

— Les facteurs utilisés n'expliquent pas le choix des passagers ;

— Elles ne permettent pas d'estimer la demande sur un niveau plus détaillé que la demande globale du marché.

Par ailleurs, les modèles *MS* peuvent bien être couplés avec ces modèles classiques pour bien estimer une demande spécifique. Multiplier la demande totale par la part de marché estimée donne alors la demande estimée pour un itinéraire spécifique.Ainsi, le calcul de la demande pour un itinéraire se fait en deux étapes :

— estimer la part de marché (l'étape 2 de la figure 3.6) ;

— estimer la demande totale sur le marché (l'étape 3 de la figure 3.6).

Aujourd'hui, il y a une forte demande de ces modèles dans le secteur aérien. Comme le choix des passagers est l'itinéraire, il est donc indispensable de se focaliser sur les caractéristiques de ces

itinéraires. En faisant leur choix d'itinéraire, les passagers font des compromis entre les critères qui définissent chaque itinéraire, à savoir : durée de vol, prix de vol, nombre de correspondance, type d'appareil, *etc.* Ainsi, la modélisation de ces compromis est cruciale pour bien estimer la demande.

Les anciennes études ont identifié deux catégories principales des modèles de calcul de part de marché [Barnhart and Smith, 2012] que l'on peut classer selon la nature de l'approche :

— Méthodes quantitatives : basées sur les modèles de choix discret (ou *logit*) ;

— Méthodes qualitatives : basées sur le modèle de *QSI*.

Les modèles de choix discret sont également appliqués dans l'aérien pour modéliser le choix de passager, ce choix porte sur l'itinéraire. Cela permettra d'estimer la part de marché pour la compagnie aérienne [Coldren, 2005, Koppelman et al., 2008, Grosche, 2009a, Goedeking, 2010, Barnhart and Smith, 2012, Garrow, 2016, Barnhart and Smith, 2012, Busquets et al., 2018].

3.3.2 Catégories des modèles *MS*

3.3.2.1 Modèle *QSI*

Présentation du modèle

Les premiers modèles de part de marché sont les modèles de *QSI* (en anglais *Quality of Service Index*). Les modèles de *QSI*, développés par le gouvernement américain en 1957 dans le cadre de la réglementation des compagnies aériennes, associent la part des passagers d'un itinéraire à sa qualité vis-à-vis de la qualité des autres itinéraires. C'est un modèle qualitatif qui mesure la qualité d'un itinéraire, où la qualité est exprimée en fonction des différents critères suivant leurs poids de préférences correspondants. Pour un modèle de *QSI* donné, ces poids de préférences peuvent être obtenus à l'aide des méthodes statistiques et/ou d'une intuition d'analyste [Coldren, 2005]. Le modèle de *QSI* est un modèle subjectif qui laisse plus de liberté à l'utilisateur de définir le modèle qui colle le mieux à son positionnement sur le marché considéré.

Une fois que les poids des critères sont fixés, le résultat obtenu est un score de l'itinéraire observé vis-à-vis des autres itinéraires du marché considéré. Le score *QSI* est exprimé sous forme d'une équation linéaire ou multiplicative.

Formule de calcul

Le *QSI* est calculé pour chaque itinéraire possible comme une somme ou un produit pondéré de critères. Ainsi, la probabilité qu'un passager utilise un itinéraire i est exprimée comme suit [Coldren, 2005, Garrow, 2016, Jacobs et al., 2012].

Exemple 3.3.2 (Modèle *QSI*)**.** Supposons qu'on ait un modèle *QSI* mesurant la qualité de l'itinéraire en fonction de quatre critères de service :

1. nombre de correspondances ;
2. tarif ;
3. type de compagnie aérienne ;
4. type d'appareil.

Ces critères sont représentés par des variables indépendantes X_1, X_2, X_3, X_4 et leurs poids correspondants β_1, β_2, β_3, β_4 :

$$QSI_i = (\beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4)$$

$$QSI_i = (\beta_1 X_1)(\beta_2 X_2)(\beta_3 X_3)(\beta_4 X_4)$$

Ainsi, pour un itinéraire i, la probabilité qu'un passager choisisse cet itinéraire parmi les itinéraires de l'ensemble J, appelé part des passagers S_i est :

$$S_i = \frac{QSI_i}{\sum_{j \in J} QSI_j}$$

où J est l'ensemble de tous les itinéraires possibles sur le marché *O&D*.

Après avoir calculé la part des passagers pour chaque itinéraire connectant l'origine à la destination, la part de marché totale est calculée en faisant la somme des passagers capturés sur chaque itinéraire de l'ensemble J.

Limites des modèles *QSI*

Le modèle de *QSI* pose des problèmes pour deux raisons [Barnhart and Smith, 2012, Garrow, 2016] :

— l'interdépendance des critères ;

— la concurrence entre les itinéraires.

D'une part, les poids sont fixés d'une manière indépendante des autres poids. Ainsi, les modèles de *QSI* ne prennent pas en compte les interactions qui existent entre les critères d'un itinéraire. À titre d'exemple, le temps de voyage est fortement corrélé avec le nombre de correspondance.

D'autre part, les modèles de *QSI* ne permettent pas de mesurer la concurrence entre les itinéraires qui existent. Cette problématique peut être constatée en analysant l'équation d'élasticité croisée 3.6 par la variation de la part des passagers pour l'itinéraire j, S_j, en raison de la modification du score de *QSI* de l'itinéraire i, QSI_i. En économétrie, l'élasticité croisée est calculée pour le critère de prix et permet ainsi de mesurer la variation relative du prix d'un bien à la suite d'une augmentation relative du prix d'un autre bien [Louviere et al., 2000] :

$$\eta_{QSI_i}^{S_j} = \frac{\partial Sj}{\partial QSIi} \frac{QSI_i}{Sj} = -S_i$$

Pour calculer l'élasticité pour le modèle *QSI*, on commence par appliquer la règle de la dérivée d'un quotient ($\frac{\partial}{\partial x}(\frac{f(x)}{g(x)}) = \frac{\frac{\partial f(x)}{\partial x}(g(x)) - (f(x))\frac{\partial g(x)}{\partial x}}{(g(x))^2}$). On obtient :

$$\frac{\partial S_j}{\partial QSI_i} = \frac{\partial}{\partial QSI_i}(\frac{QSI_j}{\sum_{k \in J} QSI_k}) \tag{3.1}$$

$$\frac{\partial S_j}{\partial QSI_i} = \frac{(0)(\sum_{k \in J} QSI_k) - QSI_j}{(\sum_{k \in J} QSI_k)^2} \tag{3.2}$$

$$\frac{\partial S_j}{\partial QSI_i} = -\frac{QSI_j}{(\sum_{k \in J} QSI_k)^2} \tag{3.3}$$

En multipliant l'équation 3.3 par QSI_i/S_j, on obtient :

$$\frac{\partial S_j}{\partial QSI_i}\frac{QSI_i}{S_j} = -\frac{QSI_j}{(\sum_{k\in J} QSI_k)^2}\frac{QSI_i}{S_j} \tag{3.4}$$

$$\frac{\partial S_j}{\partial QSI_i}\frac{QSI_i}{S_j} = -\frac{QSI_j}{(\sum_{k\in J} QSI_k)^2} QSI_i \frac{\sum_{k\in J} QSI_k}{QSI_j} \tag{3.5}$$

D'où :

$$\boxed{\eta^{S_j}_{QSI_i} = \frac{\partial S_j}{\partial QSI_i}\frac{QSI_i}{S_j} = -S_i} \tag{3.6}$$

Tel que :

$$S_i = \frac{QSI_i}{\sum_{k\in J} QSI_k}$$

D'après l'équation 3.6, on constate que le résultat de l'élasticité ne dépend pas de l'indice j. On peut traduire ça par le fait que varier la qualité du *QSI* de l'itinéraire i n'impacte pas la proportion des passagers capturés des autres itinéraires reliant la paire d'aéroports. Ce qui n'est pas réaliste. En effet, si le prix d'un itinéraire reliant la paire d'aéroports a baissé, il est probable qu'il y aura plus de passagers empruntant cet itinéraire. Donc, cela va impacter la proportion des passagers restant sur les autres itinéraires du marché considéré.

En résumé, les modèles de *QSI* ne peuvent pas capturer les interactions entre les itinéraires du marché.

Pour l'équation 3.6 de l'élasticité croisée du modèle *QSI*, les deux livres cités dans cette section [Barnhart and Smith, 2012, Garrow, 2016] présentent l'erreur suivante :

$$\eta^{S_j}_{QSI_i} = \frac{\partial Sj}{\partial QSIi}\frac{QSI_i}{Sj} = -S_i QSI_i$$

Il faut se référer à la publication originale de l'auteur [Coldren, 2005].

Modèle *S-curve*

Dans la littérature, il existe un autre modèle pour le calcul d'une part de marché : *S-curve*. Ce modèle établit la relation entre la part du marché d'une compagnie aérienne et la part de la fréquence (en anglais *Frequency Share*) [Belobaba et al., 2015]. Ce modèle suppose que la fréquence de la compagnie aérienne est variable tandis que toutes les autres variables (prix du billet, type de service) restent les mêmes pour toutes les compagnies aériennes.

Le modèle ressemble à celui du *QSI*, ce dernier inclut plus de critères qui permettent de se rapprocher au maximum de la réalité.

3.3.2.2 Modèle de choix discret

Introduction à la théorie des choix discrets

Les modèles de choix discrets sont utilisés pour modéliser le comportement des clients et *« prédire la probabilité qu'un décideur choisisse une alternative parmi un ensemble fini d'alternatives mutuellement exclusives et collectivement exhaustives »* [Barnhart and Smith, 2012, Coldren,

2005, Garrow, 2016]. Ces modèles peuvent être caractérisés par quatre éléments [Barnhart and Smith, 2012, Coldren, 2005] : un décideur, un ensemble d'alternatives disponibles, les attributs des alternatives et une règle de décision.

Premièrement, le décideur est une personne ou groupe de personnes qui prend la décision de choisir l'une des alternatives possibles. Ces alternatives possibles constituent un ensemble fini d'alternatives mutuellement exclusives et collectivement exhaustives. Les attributs sont les caractéristiques des alternatives que les décideurs considèrent lors du processus de choix. Enfin, la règle de décision considérée par la plupart des décideurs est de choisir l'alternative qui maximise leur utilité. L'utilité représente la valeur que le décideur attribue à différents attributs et illustre comment le décideur fait des compromis entre différentes alternatives [Barnhart and Smith, 2012, Coldren, 2005].

Formulation de la fonction d'utilité

Comme expliqué avant, le décideur doit choisir l'alternative qui maximise leur utilité. Dans le cas de notre domaine d'application, on considère un ensemble de passagers confrontés au même ensemble d'alternatives, qui sont les itinéraires reliant une paire d'aéroports sur un marché.

Comme l'analyste ne connaît pas l'utilité réelle. L'utilité de chaque alternative pour une décision donnée est décomposée en deux composantes : une composante déterministe qui représente les attributs observables des alternatives, et une composante aléatoire ϵ représentant ce qui n'a pas été observé. En d'autres termes, ϵ représente l'erreur de l'observation [Coldren, 2005].

$$U_i = V_i + \epsilon_i \tag{3.7}$$

$$V_i = \sum_{k=1}^{k=n} \beta_k X_{ki} \tag{3.8}$$

V_i est l'utilité observée de l'alternative i qui est supposée une combinaison linéaire des variables explicatives X_{ki} ($k^{\text{ième}}$ attribut de l'alternative i) et des paramètres estimés β_k.

Notation

i : alternative i (itinéraire i), $i \in J$
j : alternative j (itinéraire j), $j \in J$
n : individu (passager) n, $n \in N$
k : attribut (critère) de l'alternative, $k \in K$
N : ensemble des individus
J : ensemble des alternatives possibles (les itinéraires du marché)
K : ensemble des critères d'une alternative

La règle de décision est alors la suivante : l'individu n choisit une des deux alternatives si l'utilité qu'il en retire est supérieure à l'utilité attendue de l'autre alternative. Reprenons notre exemple 3.3.1. Le passager n peut choisir l'itinéraire qui est un vol direct entre *Nice* et *Bangkok* opéré par la compagnie aérienne `LH`, si l'utilité de cet itinéraire est plus grande que les autres itinéraires, typiquement, l'itinéraire opéré par la compagnie `AF` (voir la figure 3.8). Soit y une variable binaire modélisant l'alternative choisie :

$$y_{ni} = \begin{cases} 1, & \text{si l'individu } n \text{ choisit l'alternative } i \\ 0, & \text{sinon} \end{cases} \tag{3.9}$$

Dans la logique de cette approche, on suppose que le décideur sélectionne l'alternative parmi les différentes alternatives qui maximise son utilité, et que l'analyste considère que l'utilité est aléatoire. Cette hypothèse de choix aléatoire se traduit par la nécessité de calculer la probabilité que le décideur choisisse une alternative i parmi l'ensemble des alternatives possibles dans J, $P(i : J)$ [Coldren, 2005] :

$$\begin{aligned} P(y_{ni} = 1) &= P(U_i \geq U_j \quad \forall j) \\ &= P(V_i + \epsilon_i \geq V_j + \epsilon_j \quad \forall j) \\ &= P(\epsilon_j - \epsilon_i \leq V_i - V_j \quad \forall j) \end{aligned} \tag{3.10}$$

En considérant différentes hypothèses sur la distribution de l'erreur ϵ, on trouvera différents modèles de choix discrets. Par exemple, en supposant que l'erreur est indépendante et identiquement distribuée (iid) selon la loi de *Gumbel* avec un mode 0 et un écart-type γ, iid $G(0, \gamma)$, on obtiendra le fameux modèle : **régression logistique multinomiale** *MNL* (en anglais *multinomial logit*). Ce modèle a été développé pour la première fois par [McFadden et al., 1974].

La figure 3.9 formalise le processus de la prise de décision dans la théorie de l'utilité aléatoire sous forme d'une série de processus interdépendants, reliant chaque processus à une étape formelle du processus de prise de décision. À titre d'exemple, l'étape de la fonction utilité inclut la prise en compte de l'erreur qui modélise l'esprit du décideur.

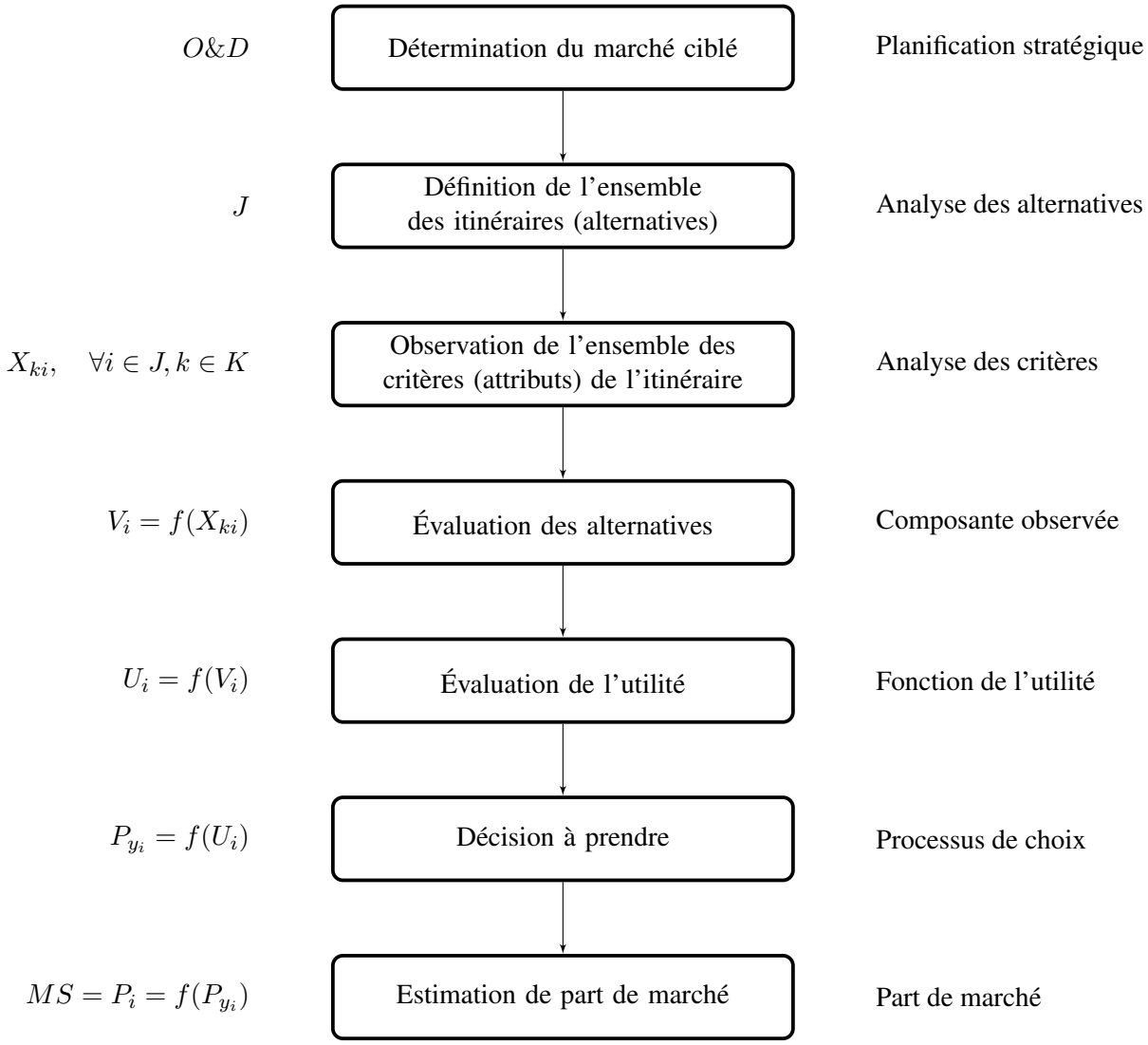


FIGURE 3.9 – Schéma du processus des modèles de choix discrets.

Présentation du modèle *MNL*

Comme présenté dans le paragraphe 3.3.2.2, les modèles de choix discrets tels que le modèle *MNL* est un modèle basé sur la théorie de l'utilité aléatoire. Il est considéré comme un modèle économique de choix discrets [Garrow, 2016]. C'est une généralisation du modèle de la régression logistique binomiale qui est utile dans le cas où on souhaite classer des objets en fonction des valeurs d'un groupe de variables à prédire. Ainsi, on distingue des variables explicatives et des variables à expliquer.

Dans la logique de cette approche, les passagers choisissent un itinéraire parmi un ensemble fini d'itinéraires qui sont mutuellement exclusifs et collectivement exhaustifs (c'est-à-dire décrivant toutes les éventualités possibles). L'individu choisit l'alternative maximisant son utilité. Dans

le cas de ce modèle, l'attraction mesurée qui est la part des passagers capturée sur l'itinéraire i, S_i est :

$$S_i = P_i = \frac{e^{V_i}}{\sum_{j \in J} e^{V_j}} \tag{3.11}$$

Tel que :

S_i est la probabilité de choisir l'itinéraire i, V_i est la valeur de l'itinéraire i et j représente tout itinéraire reliant la paire d'aéroports du marché considéré.

Preuve 3.3.1. Tout d'abord, on commence par rappeler la distribution de la loi standard de *Gumbel*, qui est une loi continue. Soit X une variable aléatoire continue, $X \sim \mathcal{G}(0,1)$ [Louviere et al., 2000] :

$$\begin{cases} F(x) = \exp(-\exp(-x)) & \text{(3.12a)} \\ f(x) = \exp(-x).\exp(-\exp(-x)) & \text{(3.12b)} \end{cases}$$

Tel que :

$F(x)$ est la fonction de répartition et $f(x)$ est la fonction de densité $\forall x \in D(X)$. $D(X)$ est le domaine de définition de la variable X, qui est définie sur tout l'ensemble des réels : $\mathbb{R}$.

D'après l'équation 3.10, on a pour un certain ϵ_i :

$$P(y_{ni} = 1|\epsilon_i) = P(y_{ni} = 1) = P(\epsilon_j \leq \epsilon_i + V_i - V_j \quad \forall j \neq i) \tag{3.13}$$

En supposant que $U_i \neq U_j$, l'inégalité devient stricte dans l'équation 3.13. On notera $P(y_{ni} = 1|\epsilon_i)$ par $P_{ni|\epsilon_i}$ pour des raisons de simplicité d'écriture :

$$P_{ni|\epsilon_i} = P(\epsilon_j < \epsilon_i + V_i - V_j \quad \forall j \neq i) \tag{3.14}$$

On rappelle que $P_{ni|\epsilon_i}$ est la probabilité que l'individu n choisisse l'alternative i pour un certain ϵ_i, qui est l'itinéraire i dans notre cas, et que y_{ni} est une variable aléatoire booléenne définie de la façon suivante :

$$y_{ni} = \begin{cases} 1, & \text{si l'individu } n \text{ choisit l'alternative } i \\ 0, & \text{sinon} \end{cases} \tag{3.15}$$

Comme ϵ_j sont disjoints deux à deux, c'est-à-dire tels que $\epsilon_i \bigcap \epsilon_j = \emptyset$ pour tous $j \neq i$.

$$\begin{aligned} P_{ni|\epsilon_i} &= P(\bigcap_{j \neq i} \epsilon_j < \epsilon_i + V_i - V_j) \\ &= \prod_{j \neq i} P(\epsilon_j < \epsilon_i + V_i - V_j) \\ &= \prod_{j \neq i} F_{\epsilon_j}(\epsilon_i + V_i - V_j) \end{aligned} \tag{3.16}$$

L'équation 3.16 représente la fonction de répartition de la variable ϵ_j au point $\epsilon_i + V_i - V_j$, qui est, selon l'équation 3.12a, est $exp(-exp(-(\epsilon_i + V_i - V_j)))$ car ϵ_i est distribué indépendamment suivant la loi de *Gumbel*, $\epsilon_i \sim \mathcal{G}(0,1)$ [Louviere et al., 2000, Gubner, 2006] :

$$P_{ni|\epsilon_i} = \prod_{j \neq i} \exp(-\exp(-(\epsilon_i + V_i - V_j))) \tag{3.17}$$

En appliquant la loi de probabilité totale pour le cas des variables aléatoires mixtes [Gubner, 2006] :

$$P(Y \in A) = \int_{-\infty}^{+\infty} P(Y \in A | X = x) dF_x(x) \tag{3.18}$$

Comme $f(x) = dF_x(x)/dx \quad \forall x \in D(X)$ et A est un événement. En substituant cette loi dans l'équation 3.17 :

$$P_i = \int_{-\infty}^{+\infty} P_{ni|\epsilon_i} f(\epsilon_i) d\epsilon_i \tag{3.19}$$

Bien sûr ϵ_i n'est pas donné, alors la probabilité de choix de P_i est l'intégrale de $P_{ni|\epsilon_i}$ sur toutes les valeurs possibles de ϵ_i pondérées par leurs densités 3.12b, rappelons que ϵ_i est une variable continue qui représente l'erreur et y_{ni} est une variable aléatoire discrète définie dans l'équation 3.15 :

$$P_i = \int_{-\infty}^{+\infty} (\prod_{j \neq i} \exp(-\exp(-(\epsilon_i + V_i - V_j)))) f(\epsilon_i) d\epsilon_i \tag{3.20}$$

En remplaçant la densité f_{ϵ_i} par son expression définie dans l'équation 3.12b :

$$P_i = \int_{-\infty}^{+\infty} (\prod_{j \neq i} e^{-e^{-(\epsilon_i + V_i - V_j)}})(e^{-\epsilon_i}.e^{-e^{-\epsilon_i}}) d\epsilon_i \tag{3.21}$$

On note une nouvelle variable $s = \epsilon_i$. L'équation 3.21 devient :

$$P_i = \int_{-\infty}^{+\infty} (\prod_{j \neq i} e^{-e^{-(s + V_i - V_j)}})(e^{-s}.e^{-e^{-s}}) ds \tag{3.22}$$

Rappelons que le produit des exponentiels est simplement l'exponentiel de la somme de ses exposants : $\prod_j exp(x_j) = exp(\sum_j x_j))$ et que : $\exp(-s) = exp(-(s + V_i - V_i))$ car $V_i - V_i = 0$, cela nous permet d'inclure le cas où $j = i$. Donc, on peut l'inclure dans le produit :

$$P_i = \int_{-\infty}^{+\infty} (\prod_j e^{-e^{-(s+V_i-V_j)}}) e^{-s} ds \tag{3.23}$$

$$P_i = \int_{-\infty}^{+\infty} \exp(-\sum_j e^{-(s+V_i-V_j)}) e^{-s} ds \tag{3.24}$$

$$P_i = \int_{-\infty}^{+\infty} (\exp(-e^{-s} \sum_j e^{-(V_i-V_j)} e^{-s} ds \tag{3.25}$$

$$\tag{3.26}$$

Pour calculer la primitive, on passe par un changement de variable.
Posons $t = \exp(-s) \Longleftrightarrow dt = -\exp(-s)ds$

Et $\begin{cases} t \xrightarrow[s \to -\infty]{} +\infty \\ t \xrightarrow[s \to +\infty]{} 0 \end{cases}$

$$P_i = \int_{+\infty}^{0} (\exp(-t \sum_j e^{-(V_i-V_j)}))(-dt) \tag{3.27}$$

$$= \int_{0}^{+\infty} (\exp(-t \sum_j e^{-(V_i-V_j)}))dt \tag{3.28}$$

$$\tag{3.29}$$

On sait que la primitive de la fonction exponentielle est $\int exp(ax+b)dx = \frac{1}{a} exp(ax+b) + C$. soit $a = -\sum_j e^{-(V_i-V_j)}$:

$$P_i = \frac{\exp(-t \sum_j e^{-(V_i-V_j)})}{-\sum_j e^{-(V_i-V_j)}} \Big|_{t=0}^{t=+\infty} + C \tag{3.30}$$

$$\tag{3.31}$$

La constante $C = 0$ car P_i est nulle quand t tend vers $+\infty$.

$$P_i = \frac{\exp(-t \sum_j e^{-(V_i-V_j)})}{-\sum_j e^{-(V_i-V_j)}} \Big|_{t=0}^{t=+\infty} \tag{3.32}$$

$$P_i = \frac{\exp(-\infty) - exp(0)}{-\sum_j e^{-(V_i-V_j)}} \tag{3.33}$$

$$P_i = \frac{-1}{-\sum_j e^{-(V_i-V_j)}} \tag{3.34}$$

$$\tag{3.35}$$

En multipliant le numérateur de l'équation par $exp(V_i)$ alors on trouve la formule de la part de marché pour le modèle *MNL* présentée dans l'équation 3.11 :

$$P_i = \frac{-e^{V_i}}{-e^{V_i} \sum_j e^{-(V_i-V_j)}}$$

$$\boxed{P_i = \frac{e^{V_i}}{\sum_j e^{V_j}}} \tag{3.36}$$

Résultats du modèle MNL dans la littérature

Dans la littérature, différents types de modèles de choix discrets ont été utilisés dans plusieurs domaines d'application y compris le transport aérien [Garrow, 2016]. Le modèle *MNL* constitue un cadre de référence incontournable pour la modélisation des choix discrets individuels au sein d'une catégorie de produits. Le modèle a été appliqué pour la première fois dans le domaine du marketing et a été appliqué par la suite à la modélisation des choix des itinéraires des passagers pour une paire d'aéroports. L'étude publiée par [Coldren et al., 2003] est parmi les premières études publiées sur les modèles *MNL*. Les auteurs ont utilisé une approche agrégée au niveau régional pour estimer la part de marché de tout itinéraire reliant chaque paire de villes aux États-Unis.

Les données utilisées dans ce travail proviennent de trois sources différentes. Les données de réservation (CRS : Système de réservation centralisé (en anglais *Computer Reservation System*)) contiennent des enregistrements détaillés sur les itinéraires réservés [Jacobs et al., 2012]. Les données de CRS contiennent les informations suivantes pour chaque itinéraire : origine/destination du voyage, segments de vols, numéro de vol, heure départ/arrivée, date de départ/arrivée et la compagnie aérienne. La deuxième source est les données de l'OAG (*Official Airline Guide*). Cette source fournit les informations suivantes : la compagnie qui opère, le partage de code (s'il y a eu un partage de code), jour d'opération, distance, heure de départ/d'arrivée, numéro de vol et durée de vol. La dernière source qui est un super-ensemble des données provenant de la base de données (DB1A : *Origin and Destination Data Bank 1A*) [Jacobs et al., 2012, Guide, JULY 2018], ce sont les données collectées avant 1998 sur les marchés *O&D*. Ces données sont basées sur un échantillon de 10% de billets d'avion achetés par les passagers qui sont à bord de l'avion exploités par les compagnies aériennes américaines. Par ailleurs, ces données fournissent des informations sur le nombre de passagers qui voyagent entre la paire d'aéroports origine-destination, des informations sur l'itinéraire (compagnie vendeur, compagnie opérante, classe de service : économique, affaire) et les informations tarifaires (tarif moyen trimestriel sur toutes les classes défini par chaque compagnie aérienne pour une paire origine/destination).

Même si les données brutes de ces jeux de données sont populairement utilisées dans la recherche académique, les compagnies aériennes achètent généralement un super-ensemble de données de la compagnie Data Base Products[Garrow, 2016, Jacobs et al., 2012]. C'est une version qui a été nettoyée après avoir été validée par croisement avec d'autres sources de données. Cela permet de fournir une estimation plus précise de la taille du marché.

Les jeux de données constituent un seul mois des départs (janvier 2000) représentant toutes les paires d'aéroports définies par deux entités régionales qui sont définies par fuseau horaire [Coldren and Koppelman, 2005]. Ainsi, l'étude contient 5 entités. Les résultats montrent que le modèle *MNL* dépasse le modèle *QSI* car l'erreur est de l'ordre de 10-15%. Le modèle *QSI* utilisé par la compagnie aérienne américaine a été remplacé par le modèle *MNL* développé par [Coldren et al., 2003]. Les auteurs ont statistiquement montré l'impact de la part estimée pour chaque itinéraire sur l'ensemble des critères caractérisant un itinéraire donné. Les critères pris en compte dans cette étude sont décrits dans le tableau 3.2.

Variable	Description
Niveau de service	variable binaire représentant le niveau de service de l'itinéraire (direct, avec escale, avec correspondance)
Type de compagnie	représente les compagnies ayant plus que 0.5% des départs
Tarif	pourcentage du tarif moyen de la compagnie par rapport au tarif moyen sur le marché
Ratio de distance	ratio de la distance d'itinéraire par rapport à la distance du plus court itinéraire
Deuxième meilleure connexion	indique quel'itinéraire n'est pas la meilleure connexion pour les itinéraires de correspondance
Partage du code	indique si le segment du vol de l'itinéraire a été réservé en tant que partage de code
Avion avec hélice	indique s'il existe un segment de vol de l'itinéraire qui a été opéré par un avion avec hélice
Avion régional	indique s'il existe un segment de vol de l'itinéraire qui a été opéré par un avion régional
Sièges d'hélice	mesure le nombre de sièges du petit avion avec hélice dans le cas où l'itinéraire inclut un segment de vol avec ce type d'avion
Sièges d'avion régional	mesure le nombre de sièges du petit avion régional dans le cas où l'itinéraire inclut un segment de vol avec ce type d'avion
Moment de la journée	variable pour chaque heure de la journée

TABLE 3.2 – liste des critères pris en compte dans l'étude [Coldren et al., 2003]

[Koppelman et al., 2008] ont repris le même modèle mais cette fois-ci pour inclure les retards. Ceci a été modélisé par le fait d'imposer une fonction de pénalité.

[Busquets et al., 2018] ont repris le modèle proposé par [Coldren et al., 2003] mais sur un niveau plus agrégé sans tenir compte des préférences des itinéraires.

Une autre approche d'estimation de part de marché se base sur les modèles des réseaux neurones. Ces modèles fonctionnent comme des boîtes noires : on donne de grands échantillons de données réelles (données de réservation) pour apprendre les interrelations entre les variables, appliquer par la suite ce qu'ils ont appris pour estimer les préférences des passagers pour des marchés inconnus. Les réseaux neurones donnent des résultats souvent supérieurs aux modèles de choix discrets, typiquement, le modèle *MNL* [Grosche and Rothlauf, 2007, Goedeking, 2010]. Toutefois, ces modèles ne donnent pas un aperçu sur les règles qu'ils ont apprises et appliquées. Ainsi, de nombreux utilisateurs hésitent d'accepter le résultat sans comprendre la raison pourquoi le modèle a donné ce résultat. En revanche, les modèles de choix discrets offrent des résultats sensés avec une transparence et visibilité maximale concernant le processus de calcul.

Comparaison entre le modèle *MNL* et *QSI*

La principale limite du modèle multinomial provient de l'hypothèse d'indépendance des alternatives non pertinentes *IIA*. Cette hypothèse atteste que le choix entre deux alternatives est indépendant des autres alternatives possibles. D'une manière formelle, le rapport des probabilités de choix P_i/P_j pour $i, j \in J$ est indépendant des critères des autres alternatives. La propriété

IIA peut être interprétée comme la probabilité de choisir l'alternative i en changeant la $k^{éme}$ variable qui décrit l'utilité de la $i^{éme}$ alternative pour l'individu n, elle est équivalente à l'équation d'élasticité croisée du modèle *QSI* (voir l'équation 3.37).

$$\eta_{X_{ik}}^{P_j} = \frac{\partial Pj}{\partial X_{ik}} \frac{X_{ik}}{Pj} \tag{3.37}$$

Preuve 3.3.2. Cette équation de l'élasticité nécessite une évaluation de la dérivée partielle, qui peut être obtenue en utilisant la règle du quotient pour les dérivées (c'est-à-dire, $\partial e^{aY}/\partial Y = ae^{aY}$) :

D'une part, on a :

$$\frac{\partial P_j}{\partial X_{ik}} = \frac{\partial}{\partial X_{ik}} \left(\frac{e^{(V_j)}}{\sum_{k \in J} e^{(V_j)}} \right)$$

En appliquant la règle de la dérivée d'un quotient, on obtient :

$$\frac{\partial P_j}{\partial X_{ik}} = \frac{(\sum_l e_l^V) \frac{\partial (e_j^V)}{\partial X_{ik}} - (e_j^V) \frac{\partial (\sum_{l \in J} e_l^V)}{\partial X_{ik}}}{(\sum_{l \in J} e_l^V)^2}$$

Comme le terme e_j^V ne dépend pas de la variable X_{ik}, alors $\partial e_j^V = 0$

$$\frac{\partial P_j}{\partial X_{ik}} = \frac{(\sum_l e_l^V)(0) - (e_j^V)(\beta_k e_i^V)}{(\sum_{l \in J} e_l^V)^2} = -P_i P_j \beta_k$$

Tel que :

$$P_i = \frac{e^{(V_j)}}{\sum_{k \in J} e^{(V_j)}}$$

D'une autre part :

$$\frac{\partial P_j}{\partial X_{ik}} \frac{X_{ik}}{P_j} = -P_i X_{ik} \beta_k$$

Par conséquent, l'élasticité croisée peut être évaluée pour le cas du modèle *MNL* comme suit [Louviere et al., 2000] :

$$\boxed{\eta_{X_{ik}}^{P_j} = \frac{\partial Pj}{\partial X_{ik}} \frac{X_{ik}}{Pj} = -P_i X_{ik} \beta_k} \tag{3.38}$$

Le résultat de l'équation de l'élasticité croisée pour le modèle *MNL* (voir l'équation 3.38) n'est pas en fonction de j. Ceci est la même chose pour le modèle de *QSI* ; cela veut dire que l'élasticité croisée est invariante d'une alternative à une autre. En d'autres termes, un changement au niveau d'une alternative n'engendra pas une modification de l'utilité d'une autre alternative. Cette hypothèse ne peut pas être réaliste. Par exemple, choisir un itinéraire avec correspondance peut être fortement corrélé avec la perte des bagages [Barnhart and Smith, 2012].

Les modèles de *QSI* suivent le même raisonnement que les modèles *MNL* pour refléter le processus de décision des passagers. Ces modèles calculent un indice de qualité de service, *QSI*, pour chaque opportunité de connexion, et prennent la part du score, *QSI*, d'une opportunité de connexion particulière parmi toutes les opportunités de connexion sur l'*O&D* sélectionné comme

part de marché. Les modèles *MNL* sont une variante des modèles *QSI* dans le sens que ces modèles supposent une relation particulière entre les facteurs qui déterminent la prise de décision. Les différentes implémentations de *QSI* diffèrent en ce qui concerne le calcul du nombre et de la quantité de critères utilisés, la manière dont les facteurs de pondération sont déterminés et la manière dont les facteurs pondérés sont combinés en un seul *QSI* global.

L'une des raisons pour lesquelles les modèles *MNL* sont couramment utilisés dans les prévisions réside dans leur structure bien définie, facilitant la calibration des paramètres à estimer (les paramètres β_k) et les temps de calcul. En outre, ces modèles modélisent bien l'interaction entre les différents critères d'une alternative, ce qui n'est pas le cas pour le modèle de *QSI* (voir le paragraphe 3.3.2.1).

Bien que le modèle *MNL* puisse être critiqué pour l'indépendance des alternatives qu'il impose, des comparaisons récentes de modèles de choix d'itinéraires basés sur les méthodologies *MNL* et *QSI* chez une grande compagnie aérienne américaine (*Americain Airlines*) ont clairement montré que les modèles *MNL* sont plus fiables que les modèles de *QSI*. Par conséquent, la propriété de l'indépendance des alternatives non pertinentes ne doit pas être considérée comme une limite des modèles de choix discrets, car de nombreux autres modèles (décrits en détail dans le livre de [Garrow, 2016]) peuvent être utilisés pour relâcher cette propriété. Notamment, le modèle logit emboîté basé sur le modèle multinomial (en anglais *nested logit*). Néanmoins, il est intéressant de noter que, dans le contexte des modèles de choix d'itinéraire, même le modèle *MNL* a considérablement dépassé le modèle *QSI* [Barnhart and Smith, 2012, Garrow, 2016, Goedeking, 2010, Busquets et al., 2018, Grosche and Rothlauf, 2007, Koppelman et al., 2008].

3.4 Lacunes de la littérature

Dans tous les articles cités dans la revue de littérature, les auteurs utilisent un modèle mathématique qui calcule le nombre de passagers attendu pour chaque itinéraire liant le marché. Le modèle prend compte de plusieurs paramètres notamment, le niveau de service (maximum 2 correspondances). Cette approche procède en deux phases. La première phase estime la part de marché de chaque itinéraire grâce aux modèles de calcul de part de marché. Dans la seconde phase, on applique le pourcentage obtenu sur la demande totale du marché estimée ou récupérée de l'historique.

La plupart de ces études reposent sur le modèle *MNL* qui est un modèle de choix discret. Les autres regardent que l'ensemble des itinéraires du marché considéré, mais néglige les autres marchés influencés par l'ajout ou la suppression d'un vol. En effet, le calcul de part de marché permet d'estimer la part des passagers sur un nouveau vol. Ainsi, on peut capturer d'autres passagers provenant d'autres marchés, notamment ceux qui font des correspondances. Par exemple, si un vol NCE-BKK est ajouté, désormais il peut attirer les passagers faisant TLS-BKK en direct. Ainsi, le fait d'ajouter un nouveau vol peut modifier le flot de passagers dans ce marché et probablement dans plusieurs autres marchés voisins. La profitabilité de la route dépend donc également des marchés influencés par l'ajout de ce nouveau vol.

De plus, les auteurs [Coldren et al., 2003], [Busquets et al., 2018], présentés dans la section 3.3.2.2 regardent un niveau opérationnel puisqu'ils incluent l'aspect horaire, et considèrent le partage d'itinéraire du point de vue plus concurrentiel. Mais si on focalise que sur le meilleur suivant le flux de passagers qui peut être plus avantageux dans notre cas. Ceci dit, négliger l'aspect

horaire mais tenir compte des meilleures connexions qu'on peut proposer en termes de revenue, distance et de temps.

Deux éléments principaux ressortent donc de l'étude des modèles de part de marché proposés dans la littérature :

1. la nécessité d'inclure les marchés impactés par le changement d'un vol ; ceci dit regarder plusieurs $O\&D$;
2. la nécessité de calculer la part de marché mais suivant le meilleur en terme de flux de passagers.

CHAPITRE 4

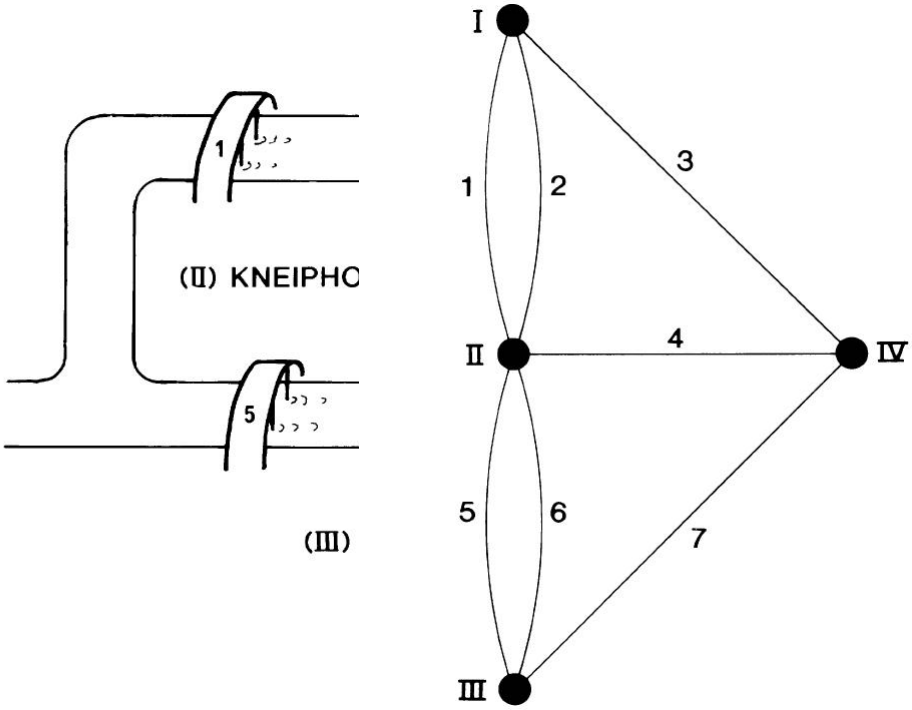
Théorie des graphes

Dans ce chapitre, on rappelle quelques notions de la théorie des graphes ainsi que les différents algorithmes utilisés de recherche de chemins dans un graphe.

4.1 Généralités sur les graphes

La théorie des graphes a été mentionnée pour la première fois par *Euler* en *1741*. [Euler, 1741] a publié sur le problème des septs ponts de Königsber (en anglais *the Königsberg Bridge Problem*) où il essayait de démontrer s'il était possible de se promener dans la ville en ne traversant chaque pont qu'une seule fois. La formulation de ce problème de manière abstraite a donné naissance à la notion de *graphe* (voir la figure 4.1).

Euler a modélisé ce problème sous la forme d'un graphe (voir la figure 4.1b) où les quatre masses terrestres (en anglais *Landmasses*) sont connectées par les sept ponts. [Euler, 1741] démontra qu'il n'y a pas de solution à son problème sur ce graphe, c'est-à-dire qu'il est impossible de traverser une et une seule fois tous les ponts.



(a) Les sept ponts de Königsberg. (b) Modélisation du problème par un graphe.

FIGURE 4.1 – Problème des septs ponts de Königsber [Foulds, 2012].

D'une façon intuitive, un graphe est une structure de données permettant de représenter des entités, des relations et les dépendances entre ses éléments. Il est utilisé dans plusieurs domaines d'application [Foulds, 2012] :

— Réseau social : relations d'amitié entre des personnes.

— Réseau de transport aérien : vols entre des aéroports.

— Réseau biologique : interactions physiques entre des protéines.

4.1.1 Notations

Cette section définit l'ensemble de notations utilisées tout au long de ce manuscrit de thèse. Nous nous sommes basés sur le livre [Lacomme et al., 2003] pour ces définitions.

Nœuds et arcs

D'une façon formelle, un graphe G est un couple $G = (\mathcal{X}, \mathcal{U})$ constitué de deux ensembles :

1. $\mathcal{X}$ est un ensemble $(x_1, x_2, \ldots, x_n)$ de n **sommets** (en anglais *vertices*), appelés également des **nœuds** (en anglais *nodes*) dans le contexte des réseaux.
2. $\mathcal{U} = (u_1, u_2, \ldots, u_m)$ est une **famille** de couple de sommets telle que $\mathcal{U} \subseteq \mathcal{X} \times \mathcal{X}$.
   Les éléments de cette famille sont appelés **arcs** (en anglais *arcs* ou *directed edges*) ou **arêtes**, cela dépend du type de graphe. Dans cette famille, les répétitions sont autorisées.

On définit $|\mathcal{X}| = n$, l'ordre du graphe G comme le nombre de nœuds et $|\mathcal{U}| = m$ sa taille.

Un arc u est de la forme (x, y) avec deux extrémités : x est **l'extrémité initiale** et y **l'extrémité finale**. La notion d'extrémité nous permet de définir les notions suivantes :

— $\omega(x)$: l'ensemble des arcs ayant x comme extrémité ;

— $\omega^+(x)$: l'ensemble des arcs ayant x comme extrémité initiale, en fait l'ensemble des arcs sortants de x ;

— $\omega^-(x)$: l'ensemble des arcs ayant x comme extrémité finale, en fait l'ensemble des arcs entrants dans x.

Définition 4.1.1 (Arc incident)**.** Soit un arc $u = (x, y)$. On dit que l'arc u est *incident* intérieurement à y (car il part de x) et *incident* extérieurement à x (car il arrive en y).

Définition 4.1.2 (Arc adjacent)**.** On dit que deux arcs sont *adjacents* s'ils ont au moins une extrémité commune.

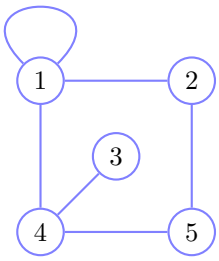
Définition 4.1.3 (Boucle)**.** Une *boucle* (en anglais *loop*) est un arc reliant un sommet à lui-même. L'extrémité initiale de cet arc est la même que l'extrémité finale.

Orienté ou pas

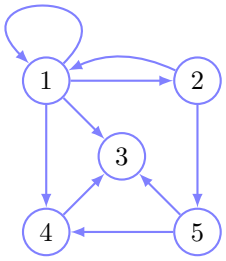
On distingue deux types de graphes : les graphes non orientés et les graphes orientés. Le graphe non orienté se caractérise par le fait que les relations sont symétriques : $\forall (x, y) \in \mathcal{U}, (y, x) \in \mathcal{U}$ Dans ce cas-là, on parle d'**arêtes** (en anglais *edges*). Un réseau social sera généralement non orienté puisque la relation amitié est réciproque. Contrairement à un réseau de transport, qui doit être orienté pour modéliser les rues à sens unique. Généralement, on utilise les termes *nœud* et *arc* pour les graphes orientés et *sommet* et *arête* pour les graphes non orientés.

Dans un graphe orienté, l'ensemble $\mathcal{U}$ est un ensemble de couples ordonnés de sommets. On parle alors d'**arcs**.

La figure 4.2 sera ensuite utilisée comme exemple pour expliquer l'ensemble des notations présentées par la suite.



(a) graphe non-orienté.



(b) graphe orienté

FIGURE 4.2 – Types de graphe.

Poids sur les arcs

Un graphe est dit *valué* ou *pondéré* (en anglais *weighted*) si on lui associe une fonction de poids qui permet d'attribuer une valeur (un poids) à chaque arc du graphe afin de le qualifier. Le poids peut représenter le temps de vol dans un réseau aérien ou bien la distance kilométrique dans un réseau routier. Parfois le terme *coût* est utilisé. On notera $w(x,y)$ le poids de l'arc (x,y). Le graphe est alors représenté par un triplet $G = (\mathcal{X}, \mathcal{U}, \mathcal{W})$ où $\mathcal{W}$ l'ensemble des poids.

Remarquons que, dans un contexte du transport urbain, les poids des arcs ne peuvent pas être négatifs. Ainsi dans ce cas là $\forall (x,y) \in \mathcal{U}, w(x,y) \geq 0$.

Dans certains types de graphe, les poids peuvent ne pas être constants. Dans ce cas, il est possible d'utiliser des fonctions dynamiques pour calculer le poids.

Successeurs et prédécesseurs

Dans un graphe orienté $G(\mathcal{X},\mathcal{U})$, la notion de couples ordonnés nous permet de définir deux nouveaux ensembles pour chaque nœud $x \in \mathcal{X}$:

— $\Gamma(x)$: ensemble des successeurs de x ;

— $\Gamma^{-1}(x)$: ensemble des prédécesseurs de x.

Formellement, les ensembles $\Gamma(x)$ et $\Gamma^{-1}(x)$ sont définis comme suit :
$\Gamma(x) = \{y \in \mathcal{X} \mid (x,y) \in \mathcal{U}\}$
$\Gamma^{-1}(x) = \{y \in \mathcal{X} \mid (y,x) \in \mathcal{U}\}$

Définition 4.1.4 (sommet voisin)**.** Soit un arc de la forme (x,y), y est successeur de x, et x est prédécesseur de y. On dit que les sommets x et y sont *voisins* (ou *adjacents*).

Degré d'un nœud

Le degré d'un nœud x, $d_G(x)$, est le nombre d'arcs incidents à x. Dans un graphe orienté G, on parle de degré sortant $d^+(x)$ (en anglais *out-degree*) et degré entrant $d^-(x)$ (en anglais *in-degree*) définis de la façon suivante :

— $d^+(x) = |\omega^+(x)|$, le nombre d'arcs partant de x.

— $d^-(x) = |\omega^-(x)|$, le nombre d'arcs arrivant en x.

Le degré $d_G(x)$ représente la somme des deux degrés $d^+(x)$ et $d^-(x)$.

Densité d'un graphe

La *densité* d'un graphe G, D, est définie comme le rapport entre le nombre d'arcs et le nombre de nœuds, soit m/n^2. Ce nombre peut varier de 0 à 1, et souvent exprimé en pourcentage. Cette notion nous permet de définir deux types de graphes.

Définition 4.1.5 (Dense)**.** Un graphe *dense* est un graphe dont le nombre d'arcs est de l'ordre n^2 : le degré moyen de ses nœuds est de l'ordre de n.

Définition 4.1.6 (Creux)**.** Un graphe *creux* est un graphe dont le nombre d'arcs est de l'ordre n : le degré moyen de ses nœuds est une constante.

Parties d'un graphe

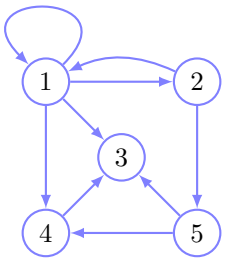
Un graphe $G' = (\mathcal{A}, \mathcal{V})$ est un *sous-graphe* (en anglais *subgraph*) de G si $\mathcal{A} \subseteq \mathcal{X}$ et $\mathcal{V} \subseteq \mathcal{U}$. On distingue trois types [Berge, 1973, Ahuja et al., 1993] :

— **sous-graphe induit** (engendré) par $\mathcal{A}$ avec $\mathcal{A} \subset \mathcal{X}$ si $\mathcal{V}$ contient tout arc de $\mathcal{U}$ avec les deux extrémités appartenant à $\mathcal{A}$ (voir la figure 4.3b).

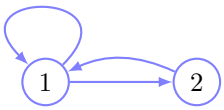
— **sous-graphe couvrant** (en anglais *spanning*) est un graphe partiel engendré par $\mathcal{V}$ si $\mathcal{V} \subseteq \mathcal{U}$ et $\mathcal{A} = \mathcal{X}$, c'est-à-dire qu'il contient tous les sommets de G (voir la figure 4.3c).

— **sous-graphe partiel** est un sous-graphe engendré par $\mathcal{A}$ et $\mathcal{V}$ si $\mathcal{A} \subseteq \mathcal{X}$ et $\mathcal{V} \subseteq \mathcal{U}$ (voir la figure 4.3d).

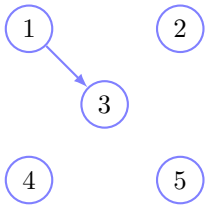
La figure 4.3 sera ensuite utilisée comme exemple pour expliquer les types des sous-graphes.



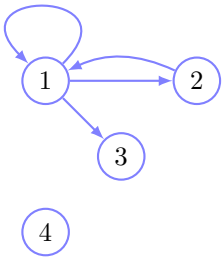
(a) graphe G.



(b) sous-graphe induit par $A = \{x_1, x_2\}$.



(c) graphe partiel engendré par $V = \{(x_1, x_3)\}$.



(d) sous-graphe partiel.

FIGURE 4.3 – Types de sous-graphe.

4.1.2 Représentation d'un graphe

Après avoir présenté l'ensemble des notations de la théorie des graphes, il faut décider comment stocker le graphe. Au cours de l'histoire de la théorie des graphes, trois principales modélisations ont été proposées et peuvent être choisies. Dans les sous-sections suivantes, nous examinons de plus près les trois modèles :

— matrice d'adjacence ;

— liste d'adjacence ;

— matrice d'incidence.

4.1.2.1 Matrice d'adjacence

Une matrice d'adjacence $\mathcal{A} = a_{ij}$ (en anglais *adjacency matrix*) est définie comme une matrice de taille $n \times n$ où n est l'ordre du graphe G. La matrice a une ligne et une colonne correspondant à chaque nœud. Les valeurs de la matrice dépendent du type de graphe représenté. Généralement, une matrice d'adjacence pour un graphe non pondéré est définie comme $\mathcal{A} = |a_{ij}|$, tel que a_{ij} est le nombre d'arêtes entre les sommets x_i et x_j. Pour le graphe orienté, les valeurs de $\mathcal{A}$ sont des valeurs binaires telles que a_{ij} égal à 1 veut dire qu'il existe un arc entre les nœuds i et j et 0 sinon.

La matrice d'adjacence à n^2 éléments, dont seulement m qui sont non nuls. Par conséquent, cette représentation est plus efficace en terme de mémoire si le graphe est suffisamment dense ; pour les graphes creux, cette représentation engendre une perte de mémoire assez importante. Néanmoins, la représentation sous forme de matrice d'adjacence nous permet de déterminer facilement l'existence d'un arc entre deux nœuds i et j ; cela correspond à l'élément de la i-ème ligne et de la j-ème colonne : a_{ij}. La complexité de cette opération est en $\mathcal{O}(1)$. On peut obtenir la liste des successeurs d'un nœud i en parcourant la ligne i de la matrice $\mathcal{A}$ en $\mathcal{O}(n)$. Finalement, la mémoire nécessaire pour stocker le graphe sous forme d'une matrice d'adjacence est en $\mathcal{O}(n^2)$.

Exemple 4.1.1 (Matrice d'adjacence)**.** On peut représenter le graphe de la figure 4.2b de la façon suivante :

$$\mathcal{A} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

4.1.2.2 Liste d'adjacence

Une liste d'adjacence (en anglais *adjacency list*) est une structure de données qui consiste à stocker le graphe sous forme d'une liste de sommets [Berge, 1973]. Chaque sommet contient alors une information sur ses successeurs sous la forme d'une liste chaînée.

Informatiquement, la liste d'adjacence est facile à implémenter et à utiliser. Tous les sommets du graphe sont stockés dans un vecteur de pointeurs de n éléments faisant référence au premier nœud d'une liste simplement chaînée. Si un sommet n'a pas de successeurs, alors son pointeur est défini à *null*. Ce type de présentation est préférée pour les graphes grands et peu denses car elle nécessite moins de mémoire qu'une matrice d'adjacence. De plus, obtenir la liste de successeurs d'un nœud n, $\Gamma(n)$, est en $\mathcal{O}(1)$.

Exemple 4.1.2 (Liste d'adjacence)**.** Le graphe de la figure 4.2b peut être représenté sous forme de liste d'adjacence (voir la figure 4.4).

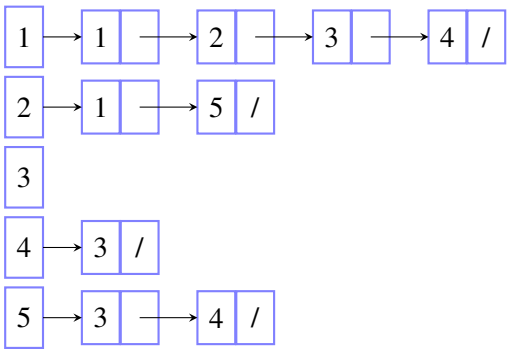


FIGURE 4.4 – Représentation du graphe de la figure 4.2b avec une liste d'adjacence.

4.1.2.3 Matrice d'incidence

La matrice d'incidence (en anglais *incidence matrix*) est très similaire à la matrice d'adjacence, mais au lieu de représenter les relations entre les sommets eux-mêmes, elle représente les relations entre les sommets et les arêtes. Ce qui signifie que la taille de la matrice est $n \times m$ tel qu'il y a une ligne pour chaque nœud du graphe et une colonne pour chaque arc. La matrice d'incidence du graphe G est donc $\mathcal{M} = m_{ij}$, où m_{ij} est le nombre de fois (0,1 ou 2 dans le cas d'une boucle) que le nœud x_i et l'arc u_j sont incidents.

Dans un graphe orienté, la matrice d'incidence est caractérisée par la présence des valeurs négatives [Bondy et al., 1976] ; le signe de ces valeurs décrit l'orientation de l'arc. Étant donné un arc $u = (x, y)$, alors l'entrée m_{xu} qui correspond à la ligne du nœud x et la colonne de l'arc u est positive. L'entrée de la ligne du nœud y et de la colonne de l'arc u est négative.

Exemple 4.1.3 (Matrice d'incidence)**.** Pour le graphe G de l'exemple de la figure 4.2b, la matrice d'incidence $\mathcal{M}$ se présente comme suit :

$$\mathcal{M}(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccccc} (1,2) & (1,3) & (1,4) & (2,5) & (4,3) & (5,3) & (5,3) \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

4.2 Approches sur les graphes

Il existe différentes approches sur le graphe selon le type de question posée :

— Approche analytique : analyser et comprendre le comportement et les propriétés d'un graphe (voir le chapitre 7).

— Approche algorithmique : élaborer des algorithmes pour résoudre un problème sur les graphes.

Dans ce chapitre, nous allons focaliser sur l'approche algorithmique. Nous commençons par présenter les notations utilisées dans la théorie des graphes pour ce genre d'approche.

Définition 4.2.1 (Chaîne)**.** Une *chaîne* $\mu = (u_1, u_2, \ldots, u_n)$ est une séquence finie de sommets et d'arêtes, débutant et finissant par des sommets. L'extrémité initiale de u_i doit être l'extrémité terminale de u_{i-1} si $i \geq 1$. La longueur de la chaîne μ est égale au nombre d'arêtes qui la compose.

Une *chaîne* est dite *simple* si aucune des arêtes n'apparaît plus d'une fois dans la séquence. Une *chaîne* est dite *élémentaire* si aucun des sommets n'apparaît plus d'une fois dans la séquence.

Définition 4.2.2 (Chemin)**.** Un *chemin* (en anglais *path*) est une séquence de nœuds $P = \{x_1, x_2, ..., x_k\}$ telle que pour chaque $1 \leq i < k, (x_i, x_{i+1}) \in \mathcal{U}$ (la notion d'un *chemin* est équivalent à celle de chaîne dans un graphe orienté.).

La longueur d'un chemin P est la somme des poids de ses arcs et est notée par :

$$\text{ł}(P) := \sum_{i=1}^{k-1} w(x_i, x_{i+1})$$

$l^\star(s,t)$ pour une paire de nœuds donnée s et t, est la longueur du plus court chemin commençant à s et se terminant à t. Un *chemin* en G est appelé *simple* si aucun arc ne se produit plus d'une fois. Un *chemin* est dit *élémentaire* si aucun nœud n'apparaît plus d'une fois dans la séquence.

Définition 4.2.3 (Cycle)**.** Un *cycle* est une chaîne fermée, c'est-à-dire, dont les deux extrémités coïncident.

Définition 4.2.4 (Circuit)**.** Un *circuit* est un chemin fermé, c'est-à-dire, $x_1 = x_k$.

Définition 4.2.5 (Connexe)**.** Un graphe *connexe* est un graphe tel que pour toute paire (x, y) de sommets distincts, il existe une chaîne reliant x à y.

Définition 4.2.6 (Parcours)**.** Un *parcours* (en anglais *walk*) dans un graphe G regroupe les chemins, les circuits, les chaînes et les cycles.

Plusieurs types de problèmes sont considérés dans l'approche algorithmique [Lacomme et al., 2003]. Par la suite, nous présentons brièvement le parcours de graphes et les problèmes de connexité. Ensuite, nous passons au problème de cheminement dans les graphes. En effet, nous avons travaillé dans cette thèse sur ce type de problème.

4.2.1 Parcours de graphes et problèmes de connexité

De nombreux problèmes en théorie des graphes concernent la connexité. Par exemple :

— l'accessibilité d'un sommet à partir d'un autre sommet dans le graphe ;

— la connexité d'un graphe ;

— la présence des composantes fortement connexes ;

— la recherche d'un chemin entre deux sommets du graphe ;

— la détection des cycles.

Pour résoudre ces problèmes, Tarjan [1972] a proposé un ensemble d'algorithmes en complexité linéaire basés sur une méthode d'exploration du graphe.

Définition 4.2.7 (Exploration d'un graphe)**.** Une *exploration* d'un graphe (en anglais *graph search*) est une opération fondamentale utilisée dans plusieurs algorithmes de graphes. Cette méthode consiste à explorer un graphe à partir d'un sommet, c'est-à-dire déterminer l'ensemble des sommets appartenant aux chemins d'origine de ce sommet.

Il existe deux types d'exploration ; exploration en largeur et exploration en profondeur [Ahuja et al., 1993, Lacomme et al., 2003].

4.2.1.1 Exploration en largeur

L'exploration ou parcours en largeur `(BFS)` (en anglais *Breadth-First Search*) consiste à visiter les sommets par ordre de distance croissante, en nombre d'arcs. Partant d'un sommet de départ s, on visite d'abord l'ensemble des successeurs de s, $\Gamma(s)$, puis les successeurs de ses successeurs, *etc*. Un exemple d'application de ce type de parcours permet le calcul du plus court chemin, en termes de nombre d'arcs, dans un graphe orienté.

4.2.1.2 Exploration en profondeur

L'exploration ou parcours en profondeur `(DFS)` (en anglais *Depth-First Search*) consiste à visiter un à un tous les sommets du graphe accessibles depuis le sommet de départ d'une façon récursive. Partant d'un sommet de départ s, on visite un sommet voisin x, puis le sommet voisin du sommet x jusqu'à ce qu'il n'existe plus de sommet, après on remonte progressivement dans le graphe. Le parcours en profondeur permet de tester la connexité du graphe, la recherche d'un chemin mais également la détection des cycles dans un graphe.

4.2.2 Cheminement dans les graphes

Le problème de cheminement dans les graphes fait partie des problèmes les plus classiques dans la théorie des graphes. Le problème du plus court chemin (PCC) (en anglais *Shortest Path Problem*) compte parmi ces problèmes qu'on peut définir de la façon suivante :

Entrée *Un graphe* $G = (\mathcal{X}, \mathcal{U}, \mathcal{W})$.
Sortie *Déterminer le plus court chemin dans le graphe* G.

Dans la littérature, il existe trois variantes du problème de plus court chemin [Lacomme et al., 2003].

Problème A : Étant donné deux sommets s et t, trouver un plus court chemin entre les deux nœuds s et t. Ce problème est connu sous le nom de *plus court chemin entre une origine et une destination* (en anglais *single-pair shortest path problem*).

Problème B : Étant donné un sommet de départ s, trouver un plus court chemin de s vers tous les autres sommets. Ce problème est connu sous le nom de *plus court chemin depuis la source* (en anglais *single-source shortest path problem*). Le problème A est un cas particulier de ce problème. De même pour le problème de plus court chemin depuis la destination où il faut inverser les arcs du graphe orienté (en anglais *single-destination shortest path problem*).

Problème C : Trouver un plus court chemin entre toutes les paires de sommets du graphe (en anglais *all-pairs shortest path problem*).

Par la suite, nous focalisons sur les deux premiers problèmes : A et B. Nous présentons les algorithmes proposés dans la littérature pour résoudre ce type de problème.

4.3 Algorithmes de plus court chemin

4.3.1 Méthode d'étiquetage

Les algorithmes de plus court chemin se basent sur la méthode d'étiquetage (en anglais *labeling*). La méthode d'étiquetage est définie comme suit [Cherkassky et al., 1996]. Pour chaque nœud x, la méthode maintient une étiquette (en anglais *label*) :

— distance $d(x)$, qui est une borne supérieure sur la longueur du plus court chemin à x ;

— parent $p(x)$;

— statut $S(x)$.

On a trois statut :

— ouvert : candidat pour la prochaine étape ;

— fermé : sommet déjà choisi ;

— indéfini : qu'on ne peut pas atteindre.

Initialement pour chaque nœud x, $d(x) = \inf$, $p(x) = nil$, et $S(x) = indéfini$. Pour un nœud de départ, la méthode initialise $d(s)$ et $S(s) = ouvert$. Ensuite, l'algorithme commence par scanner tous les nœuds ouverts jusqu'à ce qu'il n'existe aucun nœud ouvert. L'opération de `SCAN` pour un nœud ouvert consiste à vérifier pour tous les arcs sortants (x, y) appartient à $\mathcal{U}$, si $d(x)+w(x, y) < d(y)$. Alors, si l'inégalité est vérifiée, $d(y)$ est mise à jour. S'il n'y a pas de cycle négatif, les arcs $(p(x), x)$ forment un arbre T avec comme racine s. Quand l'algorithme s'arrête, T est un arbre de plus courts chemins.

Function 1 : SCAN(x)

```
foreach (x, y) ∈ U do
    if d(x) + w(x, y) < d(y) then
        d(y) ← d(x) + w(x, y);
        S(y) ← ouvert;
        p(y) ← x;
    end
end
S(x) ← fermé;
```

Il existe deux catégories d'algorithmes de plus court chemin : algorithmes à fixation d'étiquettes (en anglais *setting algorithms*) et ceux à correction d'étiquettes (en anglais *correcting algorithms*). Les deux types d'algorithmes diffèrent dans la stratégie de sélection des nœuds ouverts à fermer.

4.3.2 Algorithmes à fixation d'étiquettes

4.3.2.1 Algorithme Dijkstra

L'algorithme de `Dijkstra` est l'algorithme à fixation d'étiquettes le plus connu dans le cas des graphes avec des arcs de poids positifs ou nuls. Dans l'algorithme de `Dijkstra`, le principe

est de sélectionner un nœud avec le poids minimal à chaque itération. Deux ensembles sont conservés, un ensemble permanent représentant les nœuds fermés et un ensemble temporaire désignant les nœuds non encore sélectionnés comprenant les nœuds ayant les statut ouvert et indéfini. Le pseudo-code de l'algorithme `Dijkstra` est présenté par l'algorithme 2.

Algorithme 2 : Dijkstra Algorithm(DA)

initialization : $T_p \leftarrow \emptyset, T_t \leftarrow \mathcal{X}, \forall i \in \mathcal{X}; d[i] \leftarrow \infty, d[s] \leftarrow 0, p[s] \leftarrow nil$

while $T_t \neq \emptyset$ **do**
- $d[i] \leftarrow \min_{j \in T_t} d[j]$;
- $T_t \leftarrow T_t \setminus \{i\}$;
- $T_p \leftarrow T_p \bigcup \{i\}$;
- $SCAN(i)$;

end

4.3.2.2 Complexité

L'algorithme de `Dijkstra` scanne chaque nœud au plus une fois, cela donne une complexité de $\mathcal{O}(n^2)$ dans le pire des cas [Ahuja et al., 1993] où n est le nombre de nœuds. En effet, l'algorithme effectue l'opération de sélection de nœud n fois. Chaque opération nécessite de scanner chaque nœud étiqueté temporairement, ce qui conduit à $\mathcal{O}(n^2)$. En outre, l'algorithme effectue l'opération de mise à jour de distance pour tous les arcs sortants d'un nœud v. Globalement, l'algorithme nécessite $\mathcal{O}(m)$ puisque chaque opération nécessite un temps constant, $\mathcal{O}(1)$. Finalement, l'algorithme de `Dijkstra` résout le problème du plus court chemin en $\mathcal{O}(n^2)$.

Il existe de nombreuses variantes de l'algorithme de `Dijkstra` dans le but d'améliorer cette complexité temporelle. Ces variantes essayent différentes structures de données et plusieurs implémentations de l'algorithme [Ahuja et al., 1993]. L'utilisation d'une liste d'adjacence pour représenter le graphe peut réduire la complexité à $\mathcal{O}((n+m) \times \log n)$. L'idée est de parcourir tous les sommets du graphe en utilisant l'algorithme de `BFS` avec une complexité de $\mathcal{O}(n+m)$ et un tas comme structure de données pour stocker les nœuds ouverts. Cela permet d'extraire le minimum en $\mathcal{O}(\log n)$. De plus, utiliser un `tas de Fibonacci` pour extraire le poids minimum peut réduire le temps d'exécution à $\mathcal{O}(m + n \times \log n)$ [Fredman and Tarjan, 1987]. La raison est que ce type de structure nécessite un temps constant, $\mathcal{O}(1)$, pour extraire le minimum alors que la complexité dans un tas est $\mathcal{O}(\log n)$.

4.3.3 Algorithmes à correction d'étiquettes

4.3.3.1 Algorithme Bellman

L'algorithme `Bellman-Ford-Moore` est le plus connu dans les algorithmes à correction d'étiquettes pour trouver le plus court chemin. Il atteint la meilleure borne connue à ce jour avec des arcs de poids négatifs $\mathcal{O}(nm)$ où m est le nombre d'arcs. L'algorithme conserve l'ensemble des nœuds ouverts dans une file d'attente `FIFO` et permet de détecter un cycle négatif dans un graphe orienté pondéré. Dans `Bellman`, les arcs sont considérés un par un. Le prochain nœud à

scanner est retiré de la tête de la file d'attente ; un nœud qui devient fermé est ajouté à la queue de la file d'attente. Le pseudo-code de l'algorithme `Bellman` est présenté par l'algorithme 3.

Algorithme 3 : Bellman Algorithm(BA)

initialization : $\forall i \in V; d[i] \leftarrow \infty, d[s] \leftarrow 0, p[s] \leftarrow nil$

while $\exists (i,j) \mid d[j] > d[i] + w(i,j)$ **do**
  $SCAN(i)$;
end

4.3.3.2 Complexité

L'algorithme effectue au plus $n-1$ passages sur des arcs. Puisque chaque passe s'effectue en $\mathcal{O}(1)$ pour chaque arc, cela implique que l'algorithme se fait en $\mathcal{O}(nm)$. L'algorithme `Bellman-Ford-Moore` est qualifié comme un algorithme robuste car il n'y a pas de file d'attente prioritaire.

4.4 Optimisations du calcul du plus court chemin

4.4.1 Dijkstra bidirectionnel

Dans certaines applications du problème du plus court chemin, nous cherchons uniquement à déterminer le plus court chemin entre deux nœuds s et t. L'algorithme `Bidirectionnel` (en anglais *Bidirectional*) de `Dijkstra` résout ce problème rapidement, car il élimine certains calculs inutiles en réduisant le nombre de sommets ouverts dans la pratique.

Comme le nœud destination est connu, il est possible d'utiliser l'algorithme `Dijkstra` depuis le nœud destination et en parcourant les arcs dans le sens inverse. Le principe de la recherche bidirectionnelle est d'appliquer simultanément l'algorithme de `Dijkstra` entre le nœud origine (recherche en avant) et le nœud destination (recherche en arrière). Notons F_s (respectivement F_t), l'ensemble des nœuds vers lesquels le plus court chemin depuis s (respectivement t) est connu. B_s (respectivement B_t), l'ensemble des nœuds depuis lesquels le plus court chemin vers s (respectivement t) est connu. Lorsque un nœud k est marqué simultanément par les deux algorithmes, c'est-à-dire $k \in F_s \bigcap B_t$. Soit $P(k)$ le plus court chemin depuis s vers k et $P(t)$ le plus court chemin depuis k vers t, On sait alors que les deux plus courts chemins ont été trouvés, mais cela ne garantit pas l'optimalité de leur combinaison. Pour cela, il faut le retenir et mettre à jour le minimum, pour tout $k \in F_s \bigcap B_t$, de la somme des longueurs de ces deux chemins.

La condition d'arrêt de l'algorithme est lorsque le plus court chemin est trouvé, c'est-à-dire que certains nœuds x ont été traités, *i.e.*, sont devenus fermés dans les deux recherches [Ahuja et al., 1993].

L'avantage de cette méthode est que l'espace de recherche est réduit à deux cercles autour de la source s et la destination t, avec un rayon d'environ la moitié de la longueur de $l^\star(s,t)$. Alors que dans l'algorithme de `Dijkstra`, l'espace de recherche serait un cercle de rayon $l^\star(s,t)$ (voir la figure 4.5).

4.4.2 Algorithme A*

L'algorithme `Dijkstra` parcourt tous les nœuds ayant une étiquette plus petite que l'étiquette du nœud destination, $d(t)$. Cependant, il existe des techniques de recherche-guidée (en anglais *goal-directed*) qui visent à guider la recherche vers la destination en évitant les visites des nœuds qui ne sont pas dans la direction de la destination t. Cela permet d'explorer les nœuds les plus proches de la destination lors de la recherche du plus court chemin entre deux nœuds. Ces techniques se basent sur la topologie géométrique du réseau ou les propriétés du graphe.

La recherche A* (en anglais *A* search*) est la méthode la plus classique de la recherche-guidée [Bast et al., 2016]. L'algorithme est basé sur une fonction heuristique $h(x)$ qui estime le poids du chemin le moins coûteux de x à la destination. Ensuite, à chaque étape, il sélectionne un nœud ouvert x avec le plus petit poids, défini comme $f(x) = d(x) + h(x)$. Dans les réseaux de transport, la manière classique d'utiliser A* estime $h(x)$ en fonction de la distance euclidienne entre ces deux nœuds [Delling et al., 2009b]. Il est facile de voir que la recherche A* est équivalente à l'algorithme de `Dijkstra` quand $h(x)$ est égale à zéro.

L'espace de recherche de cette approche est sous forme d'une ellipse. La figure 4.5 est une extraction de la revue littérature sur les algorithmes de plus court chemin [Bast et al., 2016]. Elle illustre l'espace de recherche schématisé pour les méthodes : `Dijkstra`, `Dijkstra bidirectionnel` et A*.

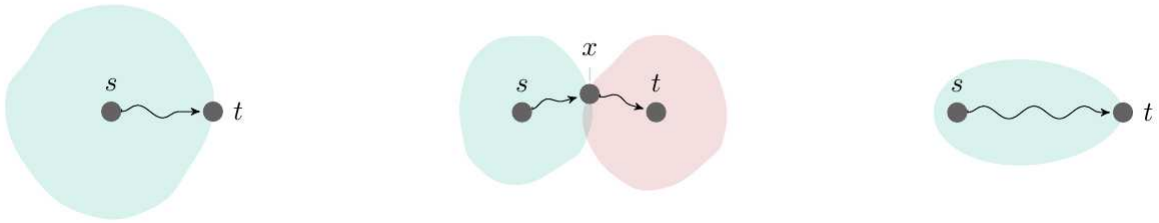


FIGURE 4.5 – À gauche Dijkstra, au milieu Bidirectionnel, à droite A*.

4.4.3 *Parent Checking*

Certaines heuristiques ont été introduites pour améliorer les performances pratiques de l'algorithme. Par exemple, [Cherkassky et al., 1996] ont introduit une heuristique de vérification de parent (*parent checking*) qui scanne uniquement le nœud x si son parent $p(x)$ n'a pas été à jour. Intuitivement, si $p(x)$ n'est pas dans la file d'attente alors l'étiquette $d(x)$ sera à nouveau mise à jour, et donc inutile de scanner les arcs sortants de x jusqu'à ce que cette mise à jour ait lieu.

[Cherkassky et al., 1996] fournissent une étude approfondie sur les algorithmes de plus court chemin avec des explications théoriques et des résultats expérimentaux en fonction des différentes instances de divers problèmes.

4.4.4 *Hub Labeling*

Parmi les techniques d'accélération, il y a celles qui n'utilisent que des plus courts chemins déjà calculés lors d'un prétraitement, et pas le graphe lui-même. L'idée derrière ces techniques est de précalculer les distances entre des paires de nœuds, en ajoutant implicitement des raccourcis virtuels au graphe. Le plus court chemin peut être directement consulté en parcourant la table des

distances. Parmi celles-ci, nous présentons en détail l'algorithme *Hub Labeling* (HL) (ou *2-hop labeling*) que nous avons utilisé dans ce travail de thèse [Cohen et al., 2003, Abraham et al., 2011, Delling et al., 2014].

L'algorithme HL est un cas particulier de la méthode d'étiquetage (voir la section 4.3.1) qui a été développée spécifiquement pour résoudre le problème A sur les réseaux routiers, c'est-à-dire le calcul du plus court chemin entre deux points (voir les variantes des problèmes PCC dans la section 4.2.2). L'algorithme HL a plusieurs propriétés intéressantes [Delling et al., 2013] :

1. C'est l'algorithme le plus rapide pour ce genre de problème, aussi bien pour les requêtes à longue distance que pour les requêtes les plus courantes.
2. La requête est plus simple : l'algorithme n'a même pas besoin de la structure de données du graphe.
3. Le concept d'étiquettes (et de hubs) est intuitif et extrêmement puissant, se prêtant naturellement à la mise en œuvre de requêtes beaucoup plus sophistiquées, telles que la recherche de points d'intérêt à proximité, l'optimisation des schémas de covoiturage ou la construction de tableaux de distance.

4.4.4.1 Description de la méthode

L'algorithme HL se base sur deux étapes. Une étape qui précalcule les étiquettes où l'étiquette $L(x)$ associée au nœud x se compose d'un ensemble de nœuds appelés *hubs* avec leurs distances depuis x, c'est-à-dire de plus courts chemins de x vers les nœuds *hubs*. Ces nœuds sont choisis de façon à ce qu'il satisfassent la propriété de couverture.

Propriété 4.4.1 (Propriété de couverture)**.**
Pour chaque paire (s, t) de nœuds distincts, $L(s) \bigcap L(t)$ doit contenir au moins un nœud appartenant au plus court chemin entre s et t (s'il existe).

Il découle de la propriété que $L(s) \bigcap L(t) \neq \emptyset$ si est seulement si t est accessible depuis s.

L'étape pour trouver le plus court chemin est plus simple, étant donné la propriété de couverture. Étant donné s et t, il s'agit de trouver le nœud $x \in L(s) \bigcap L(t)$ qui minimise $l^\star(s, x) + l^\star(x, t)$. On peut imaginer l'étiquette d'un nœud x comme un ensemble des hubs auxquels u est connecté directement. La propriété de couverture assure que les deux sommets partagent au moins un *hub* sur le plus court chemin entre eux.

Pour les graphes orientés, l'étiquette associée au nœud x consiste en deux : l'ensemble des étiquettes en avant $L_f(x)$ a les distances *de* x vers tous les *hubs*, quant à l'ensemble en arrière $L_b(x)$ a les distances de tous les *hubs vers* le nœud x ; le plus court chemin entre s et t a un *hub* dans $L_f(s) \bigcap L_b(t)$ (voir la figure 4.6).

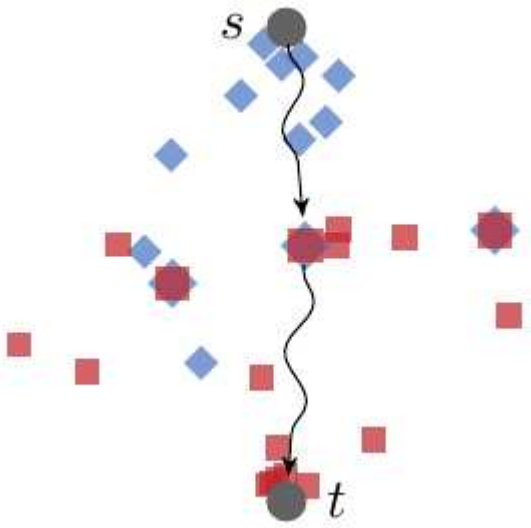


FIGURE 4.6 – Illustration du marquage des étiquettes des nœuds s (losanges bleu) et t (carrés rouges) (Bast et al. [2016]).

4.4.4.2 Complexité

Bien que la distance du plus court chemin $l^\star(s,t)$ peut être trouvée en temps polynomial $\mathcal{O}(n)$ en évaluant $l^\star(s,t) = \min\{l^\star(s,x) + l^\star(x,t) \mid x \in L(s), x \in L(t)\}$ Delling et al. [2014], elle peut être considérablement plus petite pour certains types de graphes.

Pour les réseaux routiers, Abraham et al. [2011] ont montré que l'on peut obtenir de bons résultats en définissant l'étiquette du sommet x comme l'espace de recherche avec la technique de la contraction hiérarchique (voir la section 4.4.4.3). En général, tout ordre de sommet définit complètement un étiquetage Abraham et al. [2012], et un ordre peut être converti efficacement en étiquetage correspondant. Pour des étiquettes encore plus petites, on peut choisir les sommets les plus importants, en fonction du nombre des plus courts chemins qui passent par ce nœud.

D'ailleurs, si les étiquettes sont triées par les identifiants des *hubs* (ID), une requête consiste en un balayage linéaire sur deux tableaux, comme dans le tri fusion. Ainsi, il n'est même pas nécessaire de regarder tous les *hubs* d'une étiquette. En conséquence, HL dispose des requêtes connues les plus rapides pour les réseaux routiers, prenant à peu près une microseconde pour le calcul. En revanche, la méthode souffre du temps de prétraitement qui est extrêmement long par rapport aux autres méthodes et nécessite un espace de mémoire plus important pour stocker les grandes étiquettes.

4.4.4.3 Génération des étiquettes

D'un point de vue théorique, l'ensemble des *hubs* optimal devrait fournir les meilleures performances, en termes de temps de réponse. Cependant générer un tel ensemble est un problème non trivial NP-difficile [Cohen et al., 2003]. Beaucoup de travaux ultérieurs [Abraham et al., 2011, 2012] tentent de générer l'étiquetage du *hub* en se basant sur des heuristiques appelées, *méthodes*

hiérarchiques. La structure des réseaux routiers, qui est basée sur des hiérarchies, a donné naissance à la technique de la Contraction Hiérarchique (en anglais *Contraction Hierarchies*).

Contraction Hierarchies

La technique de *Contraction Hiérarchique* (CH) est une approche importante pour exploiter la hiérarchie du réseau en créant des arcs raccourcis (en anglais *shortcuts*) pour ignorer les nœuds « sans importance » [Geisberger et al., 2008]. Le graphe contracté représente ainsi la structure compacte du graphe de départ. La méthode consiste à exécuter de manière répétée une opération de contraction de sommet. Pour contracter un sommet $x \in \mathcal{X}$, il est (temporairement) supprimé de G et remplacé par des arcs entre ses voisins u et v, c'est-à-dire des *raccourcis*, de telle sorte qu'il existe un plus court chemin qui passe par le nœud x entre ses voisins. L'arc ajouté, qui est le raccourci, a une distance égale à la distance du plus court chemin entre ces voisins.

La figure 4.7 montre un exemple de contraction d'un nœud x.

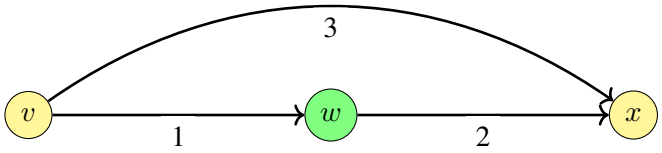


FIGURE 4.7 – Exemple de contraction.

Au cours du prétraitement, CH classe d'une manière heuristique les sommets par «importance» et les contracte du moins important au plus important. Ce classement est défini de manière à minimiser le nombre de raccourcis ajoutés.

CHAPITRE 5

Notions sur les bases de données

Dans ce chapitre, nous commençons par présenter les systèmes de gestion de bases de données. Ensuite, nous introduisons les bases de données relationnelles qui sont connues historiquement, à savoir leurs architectures, syntaxe du langage de requêtage et leurs limites. Face à ses limites, nous abordons les bases de données `NoSQL`*. Nous présentons la base de données* `MongoDB` *utilisée par* Milanamos *dans notre contexte industriel et puis la base alternative proposée, la base orientée graphe* `Neo4j`*, pour répondre aux besoins de* Milanamos*. Comme exemple fil conducteur, nous avons opté pour le problème* friends of friends *présenté dans le domaine aérien par* 2-hops*.*

5.1 Système de gestion de bases de données

5.1.1 Présentation d'un système de gestion de bases de données

Un système de gestion de base de données (*SGBD*) est un ensemble de logiciels informatiques permettant aux utilisateurs de manipuler une base de données. Ce système à usage général facilite les processus de définition, de construction, de manipulation et de partage de bases de données entre plusieurs utilisateurs et applications [Elmasri and Navathe, 2010].

Le *SGBD* sert également à manipuler les données stockées en effectuant des opérations ordinaires telles que :

— interroger la base de données pour extraire des données spécifiques ;

— mettre à jour la base de données pour intégrer les modifications ;

— générer des rapports.

Le *SGBD* joue un rôle intermédiaire entre les utilisateurs et la base de données. Il fonctionne comme une interface qui veille à garder les données stockées de manière permanente et sûre sur de longues périodes, indépendamment des programmes qui y ont accès [Silberschatz et al., 1997].

5.1.2 Architecture d'un *SGBD*

Le *SGBD* utilise un catalogue pour stocker la description de la base de données. On appelle ce catalogue un `schéma`, qui prend en charge plusieurs vues d'utilisateurs. Ainsi, nous distinguons trois niveaux dans un *SGBD* [Gardarin, 2003] :

— **Niveau physique** (*SGBD* interne : le système de gestion de fichiers) : il gère l'organisation et le stockage physique des données. Ce niveau utilise le modèle physique des données et décrit les détails pour accéder aux données ainsi que les chemins d'accès à la base de données dans le système.

— **Niveau conceptuel** : il décrit la structure de toute la base de données. Ainsi, on trouve les détails sur le schéma des tables, les propriétés des colonnes des tables, les procédures stockées et les contraintes.

— **Niveau vue** (*SGBD* externe) : il explique la mise en forme et la présentation des données aux utilisateurs ayant accès à la base de données. Ce niveau décrit simplement la partie de données présentant un intérêt pour un utilisateur. Cependant dans les autres niveaux, les schémas décrivent toute une base de données.

5.1.3 Propriétés communes

Dans cette section, nous allons présenter les propriétés communes à tous les *SGBD* : propriétés *ACID*, opérations *CRUD* et indexation.

Pour cela, nous avons besoin de définir quelques termes [Silberschatz et al., 1997].

Définition 5.1.1 (Transaction)**.** Une *transaction* est une opération atomique sur les données. Elle peut entraîner la lecture de certaines données depuis la base de données ainsi que leur écriture dans la base de données.

Définition 5.1.1 (Requête)**.** Une *requête* est un ensemble d'instructions qui se fait d'une manière déclarative pour manipuler des données de la base.

Définition 5.1.2 (Langage de requête)**.** Un *langage de requête* est un langage déclaratif utilisé pour récupérer les informations à travers une requête. Ce type de langage nécessite qu'un utilisateur spécifie quelles données demander sans avoir besoin de spécifier comment obtenir ces données.

Définition 5.1.3 (Jointure)**.** Une opération de *jointure* est une opération binaire permettant de fusionner les données à partir de deux ensembles de données.

Définition 5.1.4 (Contrainte d'intégrité)**.** Une *contrainte d'intégrité* est une règle définie dans la base de données pour garantir la cohérence d'une donnée ou d'un ensemble de données. Dans une base de données, on peut définir plusieurs contraintes.

Propriétés ACID

Les propriétés *ACID* sont les quatre principaux composants permettant de garantir qu'une transaction a été exécutée de façon fiable. Ces propriétés sont très utiles pour réaliser des transactions ou des applications financières : par exemple, un transfert de fonds d'un compte bancaire à un autre. En effet, les systèmes bancaires se basent sur des transactions et les propriétés *ACID*. *ACID* signifie **Atomicité, Cohérence, Isolation et Durabilité**. Ces quatre propriétés sont la base des applications transactionnelles [Silberschatz et al., 1997].

— ***Atomicité*** (en anglais *Atomicity*) signifie que si une transaction exécute certaines tâches, celles-ci doivent être exécutées complètement ou pas du tout. Si jamais, une tâche ne peut pas être faite, alors il faut effacer toute trace de la transaction et remettre les données dans l'état initial.

— ***Cohérence*** (en anglais *Consistency*) assure que les données restent cohérentes avant et après la transaction. Cela veut dire que n'importe quelle transaction transformera le système d'un état valide à un autre état valide. Un système à état valide est un système respectant les contraintes d'intégrité.

— ***Isolation*** (en anglais *Isolation*) veut dire que l'exécution d'une transaction se fait indépendamment des autres transactions, c'est comme si elle était seule sur le système. Cette propriété garantit que l'exécution simultanée de plusieurs transactions produit le même état que leur exécution en série. Ainsi, les opérations d'écritures et lectures des transactions réussies ne seront pas affectées par celles d'autres transactions quels que soient leurs résultats.

— ***Durabilité*** (en anglais *Durability*) assure la sauvegarde des données. Il est possible d'annuler une transaction ou de rétablir l'état du système après une défaillance. Ceci est réalisé grâce au journal des transactions. En effet, chaque *SGBD* possède un journal des transactions qui enregistre toutes les transactions et les modifications apportées par chacune d'entre elles.

Opérations CRUD

Dans un *SGBD*, la manipulation des données consiste à effectuer les opérations *CRUD*, qui signifie **Create, Read, Update et Delete**. Les opérations *CRUD* sont implémentées par quatre procédures stockées [Gardarin, 2003] :

— ***Create*** crée des nouvelles données (par exemple, créer de nouveaux aéroports) ;

— ***Read*** lit les données pour les afficher d'une façon simple ;

— ***Update*** met à jour l'ensemble des champs modifiés ;

— ***Delete*** supprime les informations demandées (par exemple, la suppression d'un vol).

Indexation

La technique d'indexation est une technique utilisée dans les systèmes des fichiers et des bases de données. Cette technique a pour but d'améliorer les performances de recherche des informations qui se fait par un `index`. Un `index` est une structure de données qui reprend la liste des valeurs auxquelles il se rapporte : c'est une copie "intelligente" des données de la table [Kroenke and Auer, 2013].

Il existe différents types d'index, les plus fréquents étant les *arbres B* (en anglais *B-Tree*) [Gardarin, 2003, Silberschatz et al., 1997]. Un arbre B est un arbre équilibré, dont les données sont triées. C'est une généralisation d'un arbre binaire de recherche où les nœuds parents peuvent posséder plus de deux nœuds enfants. Ce type de structure permet de stocker les données sous une forme triée ce qui permet d'exécuter des opérations de recherche et d'insertion en temps logarithmique [Comer, 1979].

La technique d'indexation est très utile pour certaines requêtes de filtre ou tri sur le champ ayant l'index. Néanmoins, elle a des inconvénients notamment quand il s'agit de mettre à jour les données. En fait, il faut mettre à jour les données ainsi que les index. Un autre inconvénient est la volumétrie des index car cela ajoute du volume à la base de données.

5.1.4 Exemple fil conducteur : *2-hops*

Dans cette section, nous présentons un problème de recherche de correspondance dans un réseau aérien inspiré par le problème de la recherche d'amis de mes amis (en anglais *friends of friends*). La recherche d'amis de mes amis est très connu dans le domaine des systèmes de recommandations. Un tel système a pour but de fournir à un utilisateur des ressources pertinentes en fonction de ses préférences. Ainsi, ce concept permet à l'utilisateur de réduire son temps de recherche. Par exemple dans le domaine du e-commerce, *Amazon* a un système qui recommande à l'utilisateur des produits qui répondent le plus à ses préférences [Béchet, 2012].

Le problème de *friends of friends* consiste à déterminer quels sont les amis possibles d'un utilisateur à un utilisateur dans un réseau social. Pour répondre à cette question, les bases de données relationnelles demandent un nombre coûteux d'opérations de groupage [Miller, 2013]. Il en est de même, pour la plupart des bases de données `NoSQL` [Robinson et al., 2015] car elles stockent les données sous une forme déconnectée, ce qui complique la recherche des amis.

Pour illustrer ce problème, nous allons prendre un exemple dans le domaine aérien que nous appelons : problème de deux sauts (en anglais *2-hops problem*). Supposons que nous ayons un réseau aérien et que nous souhaitions trouver les itinéraires à une escale seulement partant d'un aéroport. Dans un graphe, cela consiste à chercher les chemins ayant deux arcs.

Prenons l'exemple d'un réseau aérien qui contient 16 aéroports et 21 vols présenté dans la figure 5.1. Les nœuds du graphe représente les aéroports tandis que la présence d'un arc entre une paire de nœuds correspond au vol entre cette paire d'aéroports. Le poids sur les arcs correspond au temps de vol moyen. Le réseau est modélisé par une approche indépendante du temps qui est le modèle condensé (voir la section 7.1).

Nous voyons par la suite comment mettre en œuvre cet exemple dans différents types de systèmes de bases de données que nous allons présenter. Pour cela, nous commençons par présenter l'ensemble des requêtes nécessaires pour répondre aux questions suivantes, ces exemples permettant de mieux illustrer les opérations *CRUD* (voir la section 5.1.3) :

1. Comment créer les données du graphe 5.1 dans un *SGBD* ?

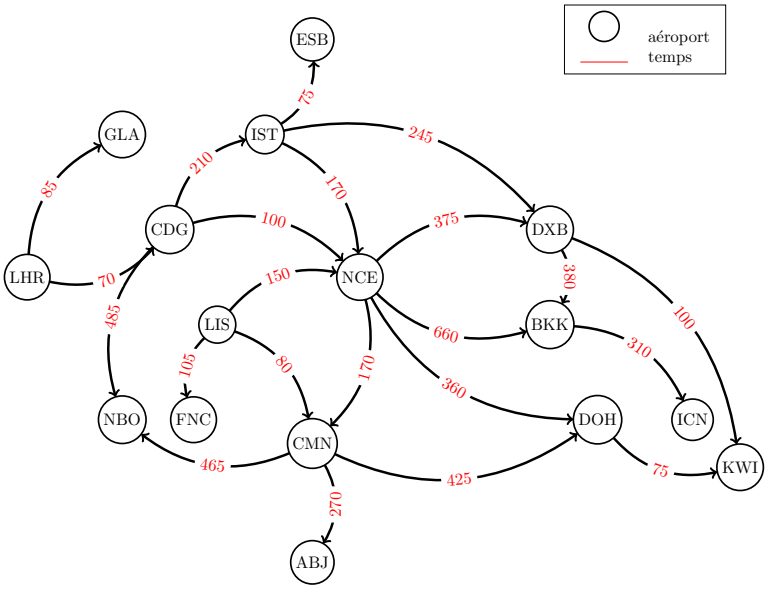


FIGURE 5.1 – Réseau aérien de vols. Le graphe contient 16 nœuds (aéroports) et 21 arcs (vols) .

2. Comment exprimer les contraintes d'intégrité dans un *SGBD* ?
3. Comment afficher les informations sur les aéroports ?
4. Comment peut-on trouver la liste des destinations joignables à partir de l'aéroport `NCE` ?
5. Quelles sont les destinations joignables en moins de 2h en partant de l'aéroport `LIS` ?
6. Comment peut-on trouver la destination la moins chère en partant de l'aéroport `NCE` ?
7. Comment récupérer les noms des aéroports ayant des vols ?
8. Comment peut-on trouver les itinéraires à une escale seulement partant de l'aéroport `NCE` ?

La dernière question correspond à la question que nous avons posé dans le problème *2-hops*.

5.2 Base de données relationnelle

5.2.1 Présentation d'un système de bases de données relationnelle

Un *SGBDR* est un *SGBD* basé sur le modèle relationnel présenté par [Codd, 1981]. C'est un modèle dans lequel les données sont organisées sous la forme de tables. Chaque table contient plusieurs lignes, chaque ligne représente un enregistrement. Définir une base de données nécessite la spécification des types de données, structures et contraintes des données à stocker. Le *SGBDR* supporte les propriétés *ACID*.

Le *SGBDR* permet de résoudre les problèmes de dépendance de données. Cette dépendance n'est pas gérée dans les systèmes de gestion de données par fichiers. Comme tout *SGBD*, le *SGBDR* se décompose en trois niveaux. Dans l'exemple 5.2.1, nous illustrons le niveau conceptuel et le niveau physique.

Exemple 5.2.1 (Structure de *SGBDR*)**.** L'exemple suivant illustre deux tables, dont la table (a) est celle qui stocke les informations des aéroports du graphe 5.1 : `code` et `nom`. Supposons que nous

souhaitions afficher uniquement le code aux utilisateurs. Alors, le niveau physique du *SGBDR* décrit le stockage du schéma. Le niveau conceptuel contient le nom des champs et le niveau vue décrit le champ code comme une vue.

Code	Name
ESB	Ankara Esenboga Airport
CDG	Charles De Gaulle Airport

(a) niveau conceptuel.

Code
ESB
CDG

(b) niveau vue.

TABLE 5.1 – Exemple d'architecture d'un *SGBDR*.

Le `SGBDR` doit limiter les redondances. En effet, stocker des informations redondantes provoque une perte en espace mémoire mais également un risque d'erreurs lors de la mise à jour des données. Ces données doivent être cohérentes entre elles. Quand une donnée fait référence à une autre donnée dans la base de données, alors celle-ci doit être présente dans le système : les données sont dépendantes dans un tel système. Effectivement, le `SGBDR` contrôle que les contraintes d'intégrité sont respectées après chaque mise à jour. Cette intégrité logique est traduite par des contraintes d'intégrité. Ainsi, il existe deux types de contraintes :

— ***l'intégrité d'entité*** porte sur une colonne unique ;

— ***l'intégrité référentielle*** définit une table lorsque la contrainte porte sur une ou plusieurs colonnes.

Exemple 5.2.2 (Contraintes d'intégrité)**.** Considérons l'exemple 5.2.1. Si on spécifie que la valeur du champ `code` doit être *unique* et non *nulle*, alors c'est une contrainte d'intégrité d'entité. Supposons qu'on ait une deuxième table sur les vols qui contient les champs suivants : `num_vol`, `origine`, `destination` et `durée`. Si on spécifie que chaque `origine` dans la table des *vols* doit être liée à un `code` dans la table des *aéroports* (voir la table 5.1a) alors ce type de contrainte s'inscrit dans la catégorie des contraintes référentielles. Ce type de contrainte est plus complexe, puisqu'il implique plusieurs tables (voir la figure 5.2).

id	origin	destination	duration	cost
1	NCE	BKK	660	2100
2	NCE	DXB	375	1000

TABLE 5.2 – Table des vols en *SGBDR*.

5.2.2 Contraintes d'intégrité

Les contraintes d'intégrité sont définies au moment de la création des tables de la façon suivante [Elmasri and Navathe, 2010] :

— *Unique* assure que la valeur d'une colonne est unique.

— *Primary key* (*Clé primaire*) spécifie la (ou les) colonnes dont la connaissance permet de désigner précisément une et une seule ligne.

— *Foreign key* (*Clé étrangère*) permet de définir les colonnes d'une table garantissant la validité d'une autre table.

— *Not Null* assure que l'absence de valeur n'est pas permise.

— *Check* contrôle la validité de la valeur des propriétés d'une ou plusieurs colonnes spécifiées à un domaine (par exemple, l'âge d'un client doit être toujours > 18).

5.2.3 Langage de requête SQL

Le langage de requête structuré SQL (en anglais *Structured Query Language*) permet de manipuler les données dans un système SGBDR.

La syntaxe globale de la requête en SQL est donnée dans le listing 5.1 :

```
1  SELECT <liste des colonnes>
2  FROM <nom de la table>
3  WHERE <conditions>
```

Listing 5.1 – syntaxe globale d'une requête en SQL

Dans la requête du listing 5.1, les colonnes du résultat sont spécifiées dans la clause SELECT. La clause FROM est une clause obligatoire qui désigne les tables d'extraction. Enfin, WHERE est une clause optionnelle pour filtrer le résultat renvoyé.

Pour insérer des nouvelles données dans un SGBDR, on utilise la clause INSERT. La jointure se traduit par la clause joins. La jointure permet de rechercher les correspondances en combinant deux tables et renvoyer le résultat en une seule table.

Par la suite, nous allons illustrer ces opérations sur l'exemple de *2-hops* présenté dans la section 5.1.4.

5.2.4 *2-hops* dans le modèle relationnel

5.2.4.1 Représentation et manipulation des données en SQL

Tout d'abord, nous commençons par créer les données dans le SGBDR. Les données du graphe 5.1 sont stockées dans deux fichiers *CSV*, elles sont décrites en annexe A :

— airports.csv contient les informations sur les nœuds du graphe : code et name des aéroports (voir l'extrait de la table 5.1a).

— flights.csv stocke les données sur les arcs : origin, destination, duration et cost des vols (voir l'extrait de la table 5.2).

Exemple 5.2.3 (Création en SQL)**.** La procédure de création correspond au schéma de la table à savoir, noms des colonnes, types de données et contraintes d'intégrité.

```
1  CREATE TABLE "airports"(
2      "code" varchar(50) PRIMARY KEY NOT NULL,
3      "name" varchar(200) NOT NULL UNIQUE
4  );
```

Listing 5.2 – requête pour créer la table airports en SQL

La clé primaire de la table `airports` porte sur la colonne `code` de type `varchar(50)` (ligne 2 du listing 5.2). La création de la table des vols (`flights`) se fait de la même manière (voir la figure 5.2).

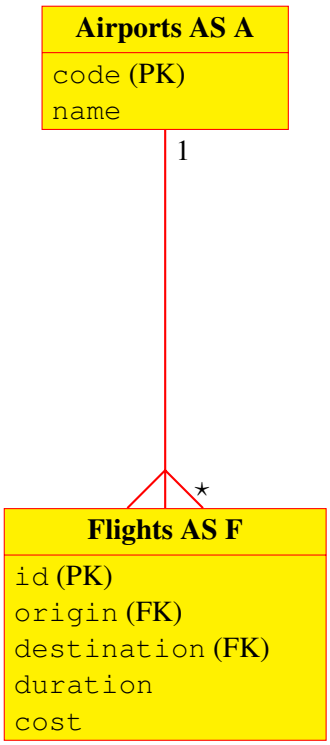


FIGURE 5.2 – Diagramme entité-relation du *2-hops*. `PK` (respectivement `FK`) correspond à la clé primaire (respectivement secondaire).

Exemple 5.2.4 (Insertion en `SQL`)**.** Pour insérer les données, nous utilisons la commande `INSERT`. Le `SGBDR` vérifie lors de toute opération d'insertion que l'utilisateur a bien fourni les valeurs des deux champs : `code` et `name`. Sinon, l'opération est rejetée par le système.

```
1 INSERT INTO airports
2 VALUES ('NCE', 'Nice Côte d''Azur Airport');
```

Listing 5.3 – requête pour insérer des données en `SQL` dans la table `airports`

En `SQL`, il est possible d'importer des données stockées dans un fichier *CSV*. Pour cela, nous utilisons la commande `BULK INSERT`. Cela permet d'insérer en masse des données dans la table.

La commande possède des paramètres pour spécifier le séparateur de lignes et de colonnes (lignes 6 et 7 du listing 5.4). Le paramètre `FIRSTROW` permet d'ignorer l'en-tête du fichier.

```
1 BULK INSERT airports
2 FROM 'airports.csv'
3 WITH
4 (
5     FIRSTROW = 2,
```

```
6       FIELDTERMINATOR= ',',
7       ROWTERMINATOR = '\n'
8   );
```

Listing 5.4 – requête pour importer des données à partir d'un fichier *CSV* en SQL

Après l'exécution de la requête du listing 5.4, le système renvoie une erreur liée à la violation de la contrainte qui porte sur le champ `code`. En effet, nous avons déjà inséré ce champ avec la requête du listing 5.3. Comme cet enregistrement correspond à la deuxième ligne du fichier, nous changeons le paramètre `FIRSTROW` à 3 pour l'ignorer. Pour les vols, on procède de la même façon pour insérer les données.

Exemple 5.2.5 (Affichage en SQL)**.** Supposons que nous souhaitions afficher la table des aéroports. Cette opération correspond à la procédure *READ* des opérations *CRUD*. Le tableau 5.1a présente un extrait de cette table.

```
1  SELECT *
2  FROM airports;
```

Listing 5.5 – requête pour afficher la table `airports` en SQL

Exemple 5.2.6 (Liste des destinations en SQL)**.** Supposons que nous souhaitions renvoyer les destinations des vols partant de l'aéroport `NCE`, alors la requête exprimée est dans le listing 5.6.

```
1  SELECT destination
2  FROM flights
3  WHERE origin = 'NCE';
```

Listing 5.6 – requête pour récupérer la liste des destinations en SQL

destination
BKK
DXB
DOH
CMN

TABLE 5.3 – Résultat de la requête du listing 5.6.

Exemple 5.2.7 (Destinations en moins de 2 heures en SQL)**.** Pour trouver les destinations qu'on peut rejoindre depuis l'aéroport de *Lisbonne* (`LIS`) en moins de deux heures, nous employons un filtre avec la clause `WHERE` (voir la requête du listing 5.7).

```
1  SELECT destination
2  FROM flights
3  WHERE origin = 'LIS'
4  AND duration <=120;
```

Listing 5.7 – requête pour récupérer les destinations en moins de 2h en SQL

destination
FNC
CMN

TABLE 5.4 – Résultat de la requête du listing 5.7.

Exemple 5.2.8 (Destination la moins chère en SQL)**.** Pour chercher la destination la moins chère, nous utilisons la clause `ORDER BY` pour trier la table des vols partant de l'aéroport dont le code est `NCE`. C'est un tri ascendant sur les coûts (ligne 4 du listing 5.8). La destination la moins chère correspond au premier résultat renvoyé par l'ajout du `TOP 1` (voir la requête du listing 5.8). Le résultat de cette requête est présenté dans la table 5.5.

```
1 SELECT TOP 1 destination
2 FROM flights
3 WHERE origin='NCE'
4 ORDER BY cost ASC;
```

Listing 5.8 – requête pour trouver la destination la moins chère en SQL

destination
CMN

TABLE 5.5 – Résultat de la requête du listing 5.8.

Exemple 5.2.9 (Récupérer les noms en SQL)**.** Un exemple de jointure est illustré par la récupération des noms des aéroports stockés dans la table `airports`. Cette opération intervient quand on a besoin de chercher les données dans d'autres tables. Ainsi, nous utilisons la commande `INNER JOIN` pour chercher les noms des aéroports dont le code figure dans la table `flights` (voir la requête du listing 5.9 et son résultat dans la table 5.6).

```
1 SELECT DISTINCT TOP (2) a.name
2 FROM flights AS F
3 INNER JOIN airports AS A
4 ON F.origin=A.code
5 OR F.destination=A.code;
```

Listing 5.9 – requête pour récupérer les noms en SQL

name
Ankara Esenboga Airport
Chales De Gaulle Airport

TABLE 5.6 – Résultat de la requête du listing 5.9.

5.2.4.2 Complexité algorithmique des requêtes

En SQL, la complexité des requêtes nécessitant de faire une opération de sélection est linéaire. En effet, pour récupérer la liste des destinations depuis l'aéroport, il faut parcourir toute la table

`flights` [Ben-Gan et al., 2015]. En revanche, l'opération la plus coûteuse est la jointure. Sa complexité dépend fortement de la taille de la table et de la présence d'un index sur la colonne concernée.

Généralement, il existe trois types de jointures [Taniar et al., 2008] :

1. Jointure par boucle imbriquée (en anglais *Nested loop join*) : énumérer toutes les correspondances possibles.
2. Jointure par hachage (en anglais *hash-based join*) : construire une table de hachage sur une des tables.
3. Jointure par tri-fusion (en anglais *Sort-Merge join*).

Supposons que nous avions deux tables M et N de taille m (respectivement n). Alors, on a une complexité quadratique pour le premier cas, $\mathcal{O}(mn)$. Pour le deuxième cas, une complexité linéaire, $\mathcal{O}(m+n)$. Cependant, la complexité de la jointure par tri-fusion dépend également de la présence d'un index sur la colonne concernée :

— Le pire des cas, c'est dans l'absence des index sur les colonnes de jointure : $\mathcal{O}(m \times \log m + n \times \log n)$.

— Dans le cas où il y un index sur la colonne de la table M, alors on passe d'une complexité linéarithmique à une complexité linéaire, c'est-à-dire $\mathcal{O}(m + n \times \log n)$.

— Si les deux colonnes sont indexées alors : $\mathcal{O}(m+n)$.

Par défaut, la jointure en *SGBDR* est une jointure par tri-fusion. Toutefois, le système peut amener à changer le type de jointure pour optimiser les requêtes. Ce choix se fait selon les cas suivants :

1. Jointure par boucle imbriquée : si une table tient en mémoire ou au moins un index est utilisable.
2. Jointure par hachage : si une des deux tables beaucoup plus petite que l'autre ou elle tient en mémoire.

5.2.4.3 Requête de *2-hops* en `SQL`

La requête du listing 5.10 présente la solution en `SQL` pour la recherche des destinations. La requête est un peu compliquée et nécessite deux jointures pour chercher les données des deux tables : `airports` et `flights`. Tout d'abord, nous récupérons les `code` des aéroports qui sont connectés à l'aéroport *NCE* avec la première jointure. Ensuite, nous cherchons les `code` des aéroports avec qui ils sont connectés en utilisant une deuxième jointure. La jointure `LEFT JOIN` consiste à exclure les aéroports qui sont déjà connectés à l'aéroport *NCE*.

```
1 SELECT DISTINCT F2.destination
2 FROM flights AS F1 JOIN airports AS A
3 ON F1.origin = A.code
4 JOIN flights AS F2
5 ON F2.origin = F1.destination
6 LEFT JOIN flights AS F3
7 ON F3.origin = A.code and F2.destination = F3.destination
8 WHERE A.code= 'NCE' AND F3.origin is NULL;
```

Listing 5.10 – requête pour chercher les aéroports avec 2 sauts en `SQL`

La requête fait deux opérations de jointure pour avoir le résultat final dans la table 5.7.

destination
ABJ
ICN
KWI
NBO

TABLE 5.7 – résultat de la requête du listing 5.10.

La requête en *SQL* nous renvoie uniquement le `code` des aéroports recherchés. En fait, pour renvoyer le nom également, il faudrait faire une troisième jointure pour récupérer les noms de la table `airports`.

5.2.5 Limites des bases de données relationnelles

Les systèmes de gestion de bases de données relationnelles ont connu un grand succès pendant plusieurs décennies [Kroenke and Auer, 2013]. Ce modèle est la solution adoptée par plusieurs utilisateurs pour gérer leurs données dans plusieurs domaines d'application. Toutefois, ces architectures ont atteint leurs limites pour manipuler les données, notamment dans le domaine des réseaux sociaux ou bien les systèmes de recommandation [Chodorow, 2013].

— Nature des données : possibilité de manipuler des données non structurées telles que les images, les vidéos, *etc.*

— Volumétrie des données : possibilité de manipuler de grandes masses de données. Par exemple, le *SGBDR mySQL* a une capacité maximale de 256 To, 64 To pour une table et 4 Go pour un enregistrement [Oracle, 2019].

— Flexibilité : possibilité de ne pas respecter le schéma des données incluant ses contraintes.

5.3 Bases de données NoSQL

La technologie `NoSQL` a émergé en réponse à la complexité croissante des données, notamment en raison de leurs volumes, variétés et vélocités. Dans ce contexte, de nouvelles technologies (Not-Only SQL : `NoSQL`) ont été développées pour faire face aux limites des bases de données relationnelles. `NoSQL` peut être vu comme *non plus SQL* ou bien *pas seulement SQL* [Vaish, 2013]. Contrairement aux bases de données relationnelles où il faut définir le schéma du modèle lors de la création des données (l'ensemble des tables la composant et l'ensemble des champs de ces tables), les bases de données `NoSQL` sont plus flexibles dans le sens où le schéma peut ne pas être fourni. En fait, le schéma peut être complété et modifié au fur et à mesure de l'insertion des données.

Les bases de données `NoSQL` permettent de manipuler des données hétérogènes provenant de sources de données de différents types. La plupart ont abandonné certaines fonctionnalités proposées par les systèmes de bases de données relationnelles pour obtenir des meilleures performances.

Voici certaines contraintes que les bases de données ont relâché pour favoriser l'efficacité :

— Le schéma des données : les modèles `NoSQL` permettent de manipuler les données sans définir auparavant le schéma. Ils exigent des modèles de schéma suffisamment flexibles où les champs peuvent être optionnels ou leur valeur nulle.

— Les propriétés *ACID* : certaines bases de données NoSQL ont abandonné ce type de propriétés, En l'occurrence, elles proposent de se baser sur les propriétés *BASE*.

5.3.1 Propriétés BASE

Certaines bases de données NoSQL se focalisent sur les propriétés *BASE*, **Basic Availability, Soft state, Eventual consistency** [Vaish, 2013]. On peut résumer les propriétés *BASE* de la manière suivante [Lotfy et al., 2016] :

— ***Basic Availability*** (disponibilité de base) : la base de données fonctionne pratiquement tout le temps. Cette propriété garantit la disponibilité des données même en cas de défaillances.

— ***Soft State*** (état souple) : la base de données peut ne pas être cohérente à tout instant. Elle peut changer avec le temps, même sans nouvelles entrées, et c'est à cause du principe de cohérence finale.

— ***Eventual consistency*** (finalement cohérent) : la base de données atteint un état cohérent à terme. En effet, les modifications finiront par arriver à tous les serveurs au bout du temps de réplication, le système finit par se stabiliser.

Contrairement aux propriétés *ACID* qui offrent une cohérence assez stricte sur les données, les propriétés *BASE* ne garantissent pas que toutes les opérations de lecture renvoient les mêmes données à partir de la dernière opération d'écriture terminée. L'utilisateur peut donc être confronté à des données incohérentes quand les mises à jour sont en cours, mais le système finira par renvoyer la bonne version des données. Pour ce faire, les bases de données NoSQL utilisent une approche distribuée de la gestion des données. Au lieu de gérer un grand magasin de données, les systèmes de bases de données NoSQL répartissent les données sur des nombreux systèmes de stockage (en anglais *sharding*) [Chandra, 2015].

5.3.2 Catégories des bases de données NoSQL

Il existe plusieurs systèmes de bases de données NoSQL classés suivant les modèles de données, c'est-à-dire la manière dont les données brutes sont modélisées sur ces systèmes. Chaque modèle a ses avantages et inconvénients [Bhamra, 2017].

— Les bases de données orientées *document* [Chodorow, 2013] stockent les données sous forme de document. Les formats du document les plus utilisés sont : *JSON*, *XML*. Les deux solutions les plus populaires sont : *MongoDB* et *CoucheDB*.

— Les bases de données orientées *Clé-valeur* [Dey et al., 2013] stockent les données de la même façon qu'un dictionnaire. Les valeurs portent les données alors que les clés font référence au nom des champs. L'idée est presque la même que celle d'une table de hachage.

— Les bases de données orientées *colonne* [Stonebraker and Cattell, 2011] stockent les données sous forme de colonnes. Le modèle orienté colonne est totalement l'inverse de celui du *SGBDR*. Les plus connues sont : *Cassandra* et *HBASE* [Nayak et al., 2013].

— Les bases de données orientées *Graphe* [Sylvain et al., 2016] représentent entièrement les données sous forme des graphes. *Neo4j* est l'une des plus utilisées [Holzschuher and Peinl, 2013, Vukotic et al., 2015].

5.4 Base de données orientée document

Dans cette section, nous présentons la base de données orientée *document* MongoDB utilisée dans notre contexte industriel à *Milanamos* pour stocker ses données aériennes. *Milanamos* utilise la version **3.0** qui date de 2015.

5.4.1 Présentation de la base de données MongoDB

MongoDB est le leader des bases de données NoSQL. Le système de stockage est écrit en langage *C++* et a été développé par la compagnie *10gen*, connue actuellement sous MongoDB. C'est une base de données *open-source* distribuée sous licence *SSPL* (en anglais *Service Side Public Licence*). Ce type de base de données est basé sur les concepts suivants (voir Figure 5.3) [MongoDB, 2016] :

— Chaque enregistrement correspond à un document.

— Une table est représentée par une collection.

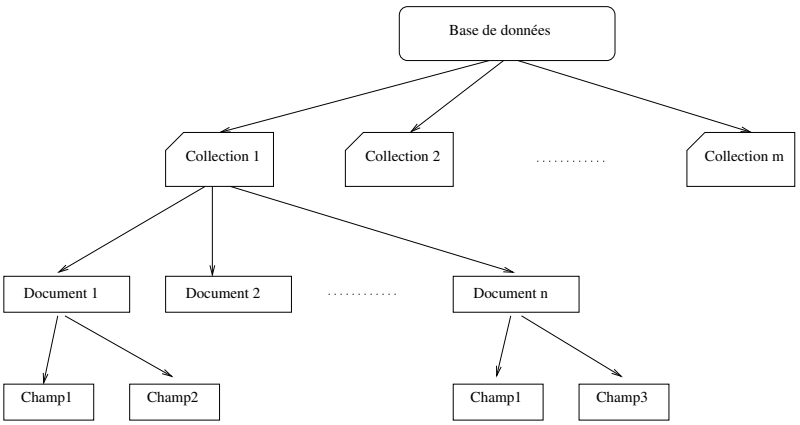


FIGURE 5.3 – Schéma d'une base de données MongoDB. Les documents d'une même collection peuvent avoir des champs différents.

Ainsi, la base de données est constituée de plusieurs collections. Une collection regroupe plusieurs documents qui n'ont pas forcément les mêmes métadonnées.

Pour la représentation des documents en MongoDB, le format *BSON* (*JSON* binaire, *JavaScript Object Notation*) est utilisé. Ce format se base sur un ensemble de paires `champ:valeur`. Comme tout système de base de données NoSQL, MongoDB stocke des données structurées et non structurées. Ainsi, tout type de données peut être stocké dans ces documents et même le type objet est possible. Un document possède un identifiant unique. La taille maximale des documents ne doit pas dépasser 16 Mo.

MongoDB est un *SGBD* sans jointure ni transaction. Cela permet d'avoir un résultat simple et plus rapide au prix d'une possible redondance des données. En effet, il est possible de stocker toutes les informations recueillies par une requête dans un document. Comme tout *SGBD*, MongoDB utilise aussi l'indexation des champs pour pouvoir facilement trouver le résultat cherché.

Stockage des fichiers en MongoDB

Pour stocker des documents de plus de 16 Mo, notamment les images, les vidéos, *etc*, MongoDB propose l'API GridFS [Dasadia and Nayak, 2016]. GridFS est un système de fichier virtuel pour le stockage des fichiers en MongoDB. Il est capable de diviser automatiquement les données en segments et stocke chaque segment en tant que document séparé. GridFS utilise deux collections pour stocker ce genre de fichier. Une collection stocke les fragments de fichier et l'autre stocke les métadonnées de fichier [MongoDB, 2017, Chodorow, 2013].

Aujourd'hui, il existe un nombre important d'application *Web* qui permettent aux utilisateurs de télécharger des fichiers. En *SGBDR*, le problème ne se posent plus, il est souvent plus efficace de stocker séparément les fichiers sur un système de fichiers et de référencer seulement les liens dans la base de données. Les *SGBDR* comportent souvent un type de données nommé DATALINK ou FILESTREAM [Bruchez, 2015]. Toutefois, utiliser cette technique en MongoDB créera probablement un certain nombre de problèmes :

1. Comment répliquer les fichiers sur tous les serveurs nécessaires ?
2. Comment supprimer toutes les copies lorsque le fichier est supprimé ?
3. Comment sauvegarder les fichiers pour la sécurité et la récupération après une panne ?

GridFS résout ce problème pour l'utilisateur en stockant les fichiers avec la base de données. Ainsi, la sauvegarde de votre base de données est utilisée pour sauvegarder les fichiers. De plus, en raison de la réplication en MongoDB, une copie des fichiers est stockée dans chaque serveur. La suppression du fichier est aussi simple que la suppression d'un objet dans la base de données.

5.4.2 Propriétés BASE

La base de données MongoDB supporte les propriétés *BASE* [Nayak et al., 2013]. Par exemple, retirer un élément d'une collection pour l'ajouter à une autre collection dans une seule requête n'est pas possible.

5.4.3 Langage de requête

Comme tout système de gestion de base de données, un langage de traitement des données est utilisé. MongoDB est basé sur le langage *JavaScript*. La structure générale d'une requête en MongoDB se décompose de la manière suivante :

— Le nom de la collection, dans laquelle il faudra chercher.
— La méthode de la collection à utiliser. Par exemple, `find()` pour trouver des documents suivant certains critères.
— Le filtre à appliquer sur les champs du document est optionnel (RequeteDocument).
— La projection du document est optionnelle. Cet élément permet de décrire le format désiré du résultat ainsi que les champs à inclure et à exclure.

La syntaxe globale de la requête en MongoDB se trouve dans le listing 5.11 :

```
1 NomBase.NomCollection.MethodeCollection(RequeteDocument,
      ProjectionDocument)
```

Listing 5.11 – syntaxe globale d'une requête en MongoDB

La méthode `find` consiste à chercher les informations. C'est l'équivalent de *select* en SQL (voir la requête du listing 5.12). Dans cette méthode, il faut préciser le nom du champ cherché. `ProjectionDocument` est ignoré comme c'est un composant facultatif de la requête.

```
1 NomBase.NomCollection.find({champ:valeur})
```

Listing 5.12 – exemple de la méthode `find()`

Un autre exemple de méthode qui est la plus souvent utilisée pour agréger les documents est la méthode `aggregate`. Cette méthode consiste à agréger les documents suivant les champs demandés. C'est l'équivalent à la méthode `GROUPBY` en `SQL`.

En `MongoDB`, chaque document correspond à une ligne de table dans *SQL*. Le bloc `$Match` sert à sélectionner les données et `$group` pour les regrouper et à faire des opérations arithmétiques telles que la somme. La projection présentée par la méthode `$project` permet de préciser les champs à renvoyer et sous quel format.

5.4.4 Opérations CRUD en `MongoDB`

On trouve également les opérations principales CRUD (Create, Read, Update et Delete) en `MongoDB` (voir la page 96).

Une méthode en `MongoDB` renvoie le résultat sous forme d'un curseur. Un curseur s'applique sur un ensemble d'enregistrements ramenés par une requête. Il se base sur deux notions :

— pointeur : un curseur est un pointeur sur un document des résultats.

— itérateur : un curseur utilise un itérateur pour itérer sur ces résultats.

Ce concept existe aussi en *SGBDR*. En fait, dans un tel système, le résultat est sous forme d'une table [MongoDB, 2016].

5.4.5 Interaction de `MongoDB` avec `Python`

Pour manipuler les données de la base de données `MongoDB`, le *SGBD* est livré avec des pilotes pour les principaux langages de programmation [MongoDB, 2016], notamment, le langage de programmation *Python* utilisé par *Milanamos*. `MongoDB` fonctionne en symbiose avec ce langage. Il est appelé avec un pilote *Python* natif, *PyMongo*, qui fournit toutes les fonctions pour interagir avec la base de données `MongoDB`. Pour pouvoir travailler sur plusieurs collections, il faudra utiliser un script en *Python* qui permet de fusionner le résultat de plusieurs requêtes. Nous allons voir par la suite un exemple de jointure en `MongoDB`.

5.4.6 *2-hops* en `MongoDB`

5.4.6.1 Manipulation des données en `MongoDB`

On reprend l'exemple présenté dans la section 5.1.4. Comme `MongoDB` est sans schéma, la création de la collection se fait en même temps que l'insertion des données. Toutefois, on peut créer des index avant l'insertion des données (voir la requête du listing 5.13).

Exemple 5.4.1 (Création en `MongoDB`)**.** La requête du listing 5.13 crée un index sur le champ `Code`. Cette contrainte impose l'unicité du champ *code*.

```
1 db.airports.createIndex({Code:1},{unique:true})
```

Listing 5.13 – requête pour créer un index en `MongoDB`

Exemple 5.4.2 (Insertion en MongoDB)**.** Pour insérer les données, nous utilisons la méthode `insert`. Un document sera créé contenant les champs : `Code` et `Name`.

```
1 db.airports.insert({"Code":"NCE", "Name": "Aéroport Nice Côte d'Azur"})
```

Listing 5.14 – requête pour insérer des données en MongoDB

Pour importer des données stockées dans un fichier *csv*, MongoDB utilise la commande `mongoimport` qui comporte les paramètres suivants (voir la requête du listing 5.15) :

- `-c` : nom de la collection ;
- `-d` : nom de la base de données ;
- `-file` : chemin du fichier ;
- `-type` : type du fichier en entrée ;
- `-headerline` : pour inclure l'en-tête.

```
1 mongoimport -c airports -d 2HOPS --file these/bd/data/airports.
    csv --type csv --headerline
```

Listing 5.15 – commande pour importer des données à partir d'un fichier *CSV* en MongoDB

Le système crée une nouvelle base de données si aucune base de données ne correspond au nom passé dans le paramètre `-d`.

Exemple 5.4.3 (Affichage en MongoDB)**.** La méthode `find` permet d'afficher une collection.

```
1 db.airports.find({})
```

Listing 5.16 – requête pour afficher une collection en MongoDB

Exemple 5.4.4 (Liste des destinations en MongoDB)**.** La requête du listing 5.17 renvoie la liste des destinations à partir de l'aéroport `NCE`. La méthode `distinct` permet de filtrer les doublons sur un champ. Dans cet exemple, `_id:0` signifie que l'`id` ne sera pas dans le résultat.

```
1 db.flights.find({"Origin":"NCE"},{"_id":0,"Destination":1})
```

Listing 5.17 – requête pour récupérer la liste des destinations en MongoDB

```
1  {
2  "Destination" : "CMN"
3  }
4  {
5  "Destination" : "DXB"
6  }
7  {
8  "Destination" : "BKK"
9  }
10 {
11 "Destination" : "DOH"
12 }
```

Listing 5.18 – résultat de la requête du listing 5.17

Exemple 5.4.5 (Destinations en moins de 2 heures en MongoDB)**.** Pour trouver les destinations qu'on peut rejoindre depuis l'aéroport de *Lisbonne* (LIS) en moins de deux heures, nous employons l'opérateur `$lt` pour spécifier que la durée est moins de 120 minutes. Pour renvoyer uniquement la destination, nous utilisons une projection (voir le listing 5.19).

```
db.flights.find({"Origin":"LIS","Duration":{$lt:120}},{"_id":0,"Destination
    ":1})
```

Listing 5.19 – requête pour récupérer les destinations à moins de 2h en MongoDB

```
/* 1 */
{
        "Destination" : "FNC"
}

/* 2 */
{
        "Destination" : "CMN"
}
```

Listing 5.20 – résultat de la requête du listing 5.19

Exemple 5.4.6 (Destination la moins chère en MongoDB)**.** Pour chercher la destination la moins chère, nous utilisons la méthode `sort` pour trier les vols partant de l'aéroport dont le code est NCE. C'est un tri ascendant sur les coûts. La destination la moins chère correspond au premier résultat renvoyé par l'ajout de la méthode `limit()` (voir la requête du listing 5.21).

```
db.flights.find({"Origin":"NCE"},{'_id':0,'Destination':1}).sort({Cost:1}).
    limit(1)
```

Listing 5.21 – requête pour trouver la destination la moins chère en MongoDB

```
/* 1 */
{
        "Destination" : "CMN"
}
```

Listing 5.22 – résultat de la requête du listing 5.21

Exemple 5.4.7 (Récupérer les noms en MongoDB)**.** La récupération des noms nécessite l'utilisation des deux collections `airports` et `flights`. Comme il n'y a pas de jointure en MongoDB, nous avons besoin de passer par un script en Python. Tout d'abord, la première requête du listing 5.23 récupère les codes de tous les aéroports de la collection `flights` (voir le résultat dans le listing 5.25). Ensuite, la deuxième requête du listing 5.24 cherche les noms de ces aéroports dans la collection `airports`. Ce genre d'opération nécessite souvent un script en *Python* pour manipuler les deux requêtes (passer la liste des aéroports retournée par la requête 5.23) en paramètre à la requête du listing 5.24

```
db.flights.aggregate(
[
{
      '$group': {
```

```
            '_id': null,
            origins: {'$addToSet': '$Origin'},
            destinations: {'$addToSet': '$Destination'}
      }
},
{
      '$project': {
            '_id': 0,
            airports: {'$setUnion': ['$origins', '$destinations']}
      }
}
])
```

Listing 5.23 – requête pour récupérer les codes des aéroports de la collection `flights` en `MongoDB`

```
db.airports.find({ "Code": { $in: airportsCodes}} ,{"_id":0,"Name":1} ).
    limit(2)
```

Listing 5.24 – requête pour récupérer les noms en `MongoDB`

```
/* 1 */
{
"Name" : "Charles De Gaulle Airport"
}

/* 2 */
{
"Name" : "Ankara Esenboga Airport"
}
```

Listing 5.25 – résultat de la requête du listing 5.24

5.4.6.2 Requête de *2-hops* en `MongoDB`

Nous présentons deux façons pour résoudre le problème de *2-hops*. Ces deux méthodes se distinguent par la redondance des données qui permet d'avoir un résultat plus rapidement. La première méthode consiste à stocker les données de la même façon qu'en relationnelle, cela veut dire, stocker les aéroports dans une collection et les vols dans une autre collections. Quant à la deuxième méthode, les données sont stockées de sorte à privilégier un seul sens, c'est-à-dire, origine → destination. Ainsi, trouver les destinations d'une origine se fait rapidement avec les index par rapport aux relationnelles. Si on veut les origines d'une destination, alors on est obligé de parcourir tous les éléments. On est alors sur les mêmes performances que le relationnel.

Méthode 1.

Pour résoudre le problème de *2-hops*, nous avons besoin de faire deux requêtes :

1. la liste des destinations de l'aéroport en question ;
2. la liste des destinations des aéroports récupérés de la première requête.

Pour réaliser cette suite de requêtes, il faut utiliser un script en *Python*. Nous commençons par construire la liste des destinations de l'aéroport en question (ligne 2 à 9 du programme B.1), en utilisant la méthode `aggregate`.

Ensuite, il faut répéter la même requête pour les destinations retournées par la première requête du listing (ligne 2 à 9 du programme B.1). Cette requête nous retourne la liste des destinations de l'aéroport *NCE*. Puis, il faut exclure les aéroports qui figurent à la fois dans le résultat de la première requête et qui sont connectés à l'aéroport de *NCE* (ligne 11 à 18 du programme B.1).

Le programme (code en annexe B) nous retourne le résultat suivant (voir le résultat du listing 5.26).

```
liste des destinations: ['ABJ', 'NBO', 'KWI', 'ICN']
```

Listing 5.26 – résultat de la requête du listing B.1

Comme en `SQL`, pour avoir plus d'informations sur ces aéroports notamment le nom, il faudra faire une troisième requête sur la collection `airport`.

Méthode 2.

Si nous nous arrêtons à la question de trouver la liste des destinations à partir d'un aéroport avec exactement un seul vol, alors, la manière la plus simple est de créer une seule collection qui contient des documents incorporés où chaque document contient l'`id` de l'aéroport, son `code` et la liste des `code` de ces aéroports destinations.

Par défaut, `MongoDB` nous propose de générer lui-même des identifiants (`_id`) grâce aux `ObjectId`, la classe que `MongoDB` utilise pour générer les identifiants. En pratique, on utilise le `code` qu'on a défini comme index, comme identifiant (voir le listing 5.13).

On trouvera dans le listing 5.27 un exemple de document :

```
/* 1 */
{
        _id : ObjectId("4e77bb3b8a3e000000004f7a"),
        code : "NCE",
        destinations : ["CMN", "DXB", "BKK", "DOH"]
}
```

Listing 5.27 – deuxième implémentation en `MongoDB`

Pour retrouver la liste des aéroports destinations de l'aéroport en question (l'aéroport *NCE*), on utilise la requête du listing 5.17 appelée sur la collection `destinations`. Cela permet de trouver rapidement la liste des destinations d'un aéroport au lieu d'appliquer une agrégation à chaque aéroport. Ensuite, pour répondre à la question du problème *2-hops*, il faudra faire une deuxième requête qui prend en paramètre la liste des destinations renvoyée par la requête du listing 5.17 et chercher leurs listes de destinations en excluant le cas des aéroports qui sont déjà en liaison avec l'aéroport *NCE*. C'est bien la question que nous cherchons à résoudre.

La requête du listing 5.28 commence par selectionner les aéroports dont le code figure dans la liste des destinations en entrée (ligne 2). Ensuite, on utilise la méthode `unwind` pour créer des documents pour chaque élément de la liste des destinations, cela nous permettra de chercher pour chaque élément qui le code de destination, leurs aéroports destinations. Finalement, pour la récupération des destinations finales joignables depuis *NCE* avec deux sauts, il faudra exclure les

aéroports qui figure dans la liste des destinations de l'aéroport de *NCE* y compris ce dernier. Ceci nous permettra d'éviter les cycles dans le graphe.

```
db.destinations.aggregate([
   {$match:{"code":{'$in':["CMN","DXB","BKK","DOH"]}}},
   {$unwind:"$destinations"},
   {$match:{"destinations":{'$nin':["NCE","CMN","DXB","BKK","DOH"]}}},
   {$group:{
      '_id':null,
      mesDest:{$addToSet:"$destinations"}}}
])
```

Listing 5.28 – requête pour récupérer les liste des destinations d'une liste de destinations en MongoDB

Le problème de ce modèle est qu'il y a beaucoup de redondances comme on l'avait dit au début. En effet, pour avoir un résultat plus rapidement, on passe à la redondance des données. Ce qui n'est pas avantageux notamment pour répondre à ce genre de question.

Conclusion. Jusqu'à présent, la conclusion que nous pouvons tirer de cet exemple est que la modélisation du problème *2-hops* est plus compliqué quand il s'agit des bases de données relationnelles ou orientées document comme la base de données MongoDB.

5.4.7 Limites des bases de données orientées document

La base de données orientée document offre la disponibilité, en termes de grands volumes de données, distribution des données et de hautes disponibilités comme toute base NoSQL. Toutefois, cette flexibilité entraîne des coûts sur les requêtes. En effet, ce modèle est souvent long pour exécuter de grandes requêtes, comme le modèle relationnel, ainsi qu'à supporter la jointure de plusieurs requêtes (voir l'exemple 5.4.7).

5.4.8 Comparaison des terminologies avec le relationnel

Le tableau 5.8 résume l'équivalence entre les termes utilisés en MongoDB avec ceux connus traditionnellement en *SGBDR*.

Relationnel	**MongoDB**
Base de données	Base de données
Table	Collection
Ligne	Document
Colonne	Champ
Index	Index
Curseur	Curseur
Clé primaire	`_id`
Clé secondaire	n'importe quel champ
Schéma	-
Select	Find
Update	Update
Insert	Insert

TABLE 5.8 – terminologie relationnel/orienté document.

Évolution de MongoDB.

— La version **3.2** de `MongoDB` offre la possibilité de faire une "quasi jointure". Cette dernière consiste à compléter les données d'un document avec les données d'une collection « de référence » (la méthode `$lookup`).

— Depuis la version **3.4**, il y a un moyen de parcourir un arbre directement dans la requête avec la méthode `$graphLookup`.

— Il existe depuis la version **4.0** de 2018 (et améliorée en **4.2**) une notion de transaction.

— Il existe aussi le moyen de décrire comment valider les documents stockées (depuis **3.2**).

5.5 Base de données orientée graphe

Dans cette section, nous introduisons le modèle de base des données orientée graphe `Neo4j`. Tout d'abord, nous introduisons les concepts de base, puis le langage de requête `Cypher`. Ensuite, nous illustrons les avantages des bases de données orientées graphe en comparaison des autres bases de données plus particulièrement `MongoDB`. Pour finir, nous décrivons la structure de `Neo4j` et les différents algorithmes de graphe testés sur la base de données `Neo4j`.

5.5.1 Présentation de la base de données orientée graphe

Le modèle de base de données orientée graphe structure les données sous forme de nœuds, relations et propriétés formant un graphe. Ce modèle inspiré de la théorie des graphes conserve certaines caractéristiques des *SGBDR* (les transactions, les propriétés *ACID* et les opérations *CRUD*) pour certaines bases de données orienté graphe [Rai and Chettri, 2018]. En effet, les bases de données ne sont pas toutes *ACID*, notamment la base de données orientée graphe `DEX` [Pokornỳ, 2015, Robinson et al., 2015, Angles and Gutierrez, 2008] (voir la section 5.5.2).

D'une manière moins formelle, un modèle de base de données orientée graphe est un ensemble d'entités connectées par un ou plusieurs types de relations. La modélisation des données dans les

bases de données orientées graphe est un processus relativement simple par rapport au schéma entité relation mais il reste à concevoir le graphe, une tâche plus compliquée en raison du haut niveau de connectivité fourni par les graphes, ce qui permet d'avoir un modèle proche de la réalité [Braimniotis, 2017]. Contrairement à la modélisation de données relationnelles qui requiert l'utilisation de plusieurs clés primaires et étrangères pour connecter des entités, les bases de données orientées graphe sont très expressives et réduisent le déséquilibre entre l'analyse et la mise en œuvre, le graphe étant conçu selon ce qu'on veut analyser. On trouve ce problème dans les implémentations de bases de données relationnelles qui sont conçues comme un outil de stockage plus que comme une représentation des données [Robinson et al., 2015].

5.5.2 Bases de données orientées graphe

Il existe plusieurs solutions commerciales de systèmes de gestions de bases de données orientés graphe qui sont utilisées dans plusieurs domaines notamment, les systèmes de recommandation et les systèmes de transport [Ciglan et al., 2012, Shimpi and Chaudhari, 2012]. Des études ont été réalisées dans la recherche académique pour tester les différentes solutions suivant différents critères, notamment, la flexibilité du schéma, les propriétés *ACID*, la sauvegarde des données, la fragmentation (en anglais *sharding*). Nous avons ainsi choisi `Neo4j` qui représente la meilleure solution pour notre étude, seule la fonctionnalité de la fragmentation qui n'est pas implémentée [Fernandes and Bernardino, 2018, Angles, 2012, Agrawal and Patel, 2016, Shimpi and Chaudhari, 2012, Jadhav and Oberoi, 2015].

5.5.3 Base de données orientée graphe `Neo4j`

`Neo4j` est une base de données orientée graphe open source implémentée en **Java**. Elle utilise le modèle de graphe étiqueté (voir la section 5.5.4). La base de données dispose d'une interface Web pour visualiser le graphe interactivement. `Neo4j` est considérée comme la base de données orientée graphe la plus populaire et la plus utilisée dans le monde. Il présente de nombreux avantages concurrentiels [Buerli, 2012, Fernandes and Bernardino, 2018, Angles, 2012, Agrawal and Patel, 2016]. [Fernandes and Bernardino, 2018].

La première version a été créé en 2007. Il existe deux éditions de `Neo4j` :

— édition communautaire (*Community Edition*), disponible sous la licence *GPLv3*. Cette édition est parfaitement adaptée pour des projets personnels sur une instance unique.

— édition entreprise (*Entreprise Edition*), disponible sous une licence par abonnement, ou une licence gratuite sous l'*AGPLv3* pour des projets *open-source*.

Les scénarios d'application typiques de `Neo4j` sont les recommandations, la détection de la fraude, les graphes de connaissances, l'analyse et les algorithmes de graphes.

5.5.4 Modèle de données orienté graphe étiqueté

5.5.4.1 Présentation du modèle

`Neo4j` utilise le modèle orienté graphe étiqueté pour modéliser les graphes dans la base de données. Ce modèle a été choisi pour des raisons de flexibilité et de polyvalence. Ainsi un graphe en `Neo4j` contient des nœuds et des relations. Chaque nœud et chaque relation possèdent des propriétés. Ainsi, le modèle de graphe de données nous fournit les composantes structurelles suivantes [Grosche and Rothlauf, 2007] (illustré par la figure 5.4) :

— *Nœuds* représentent une entité d'information. Les nœuds sont similaires aux documents en `NoSQL` ;

— *Relations* connectent les nœuds. Une relation a un nom unique, un nœud de départ, un nœud terminal et une direction. Les relations sont les arcs en théorie des graphes ;

— *Propriétés* sont représentées sous forme de paires clés-valeurs ;

— *Étiquettes* permettent de spécifier le type de nœuds puisqu'un nœud peut jouer plusieurs rôles dans un graphe. C'est une composante facultative mais en utilisant ce concept, nous arrivons facilement à regrouper les nœuds dans des ensembles et cela permet d'exécuter efficacement des requêtes. En `Neo4j`, un nœud peut avoir de zéro à plusieurs étiquettes.

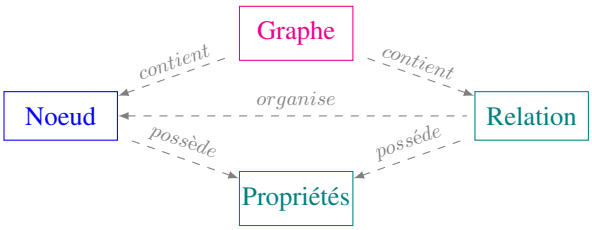


FIGURE 5.4 – Modèle de base de données graphe.

Les données sont stockées dans des propriétés sous forme de paires clés-valeurs. Les propriétés peuvent être ajoutées aux nœuds et relations. Chaque nœud ou chaque relation est porteur de ses propres propriétés. En `Neo4j`, les propriétés peuvent être définies d'une façon dynamique dans le sens où leurs modifications ne dépendent pas du modèle : c'est-à-dire, on peut créer des nouvelles propriétés sans modifier le modèle actuel du graphe. De la même façon, les relations peuvent être modifiées dynamiquement. Autrement dit, les entités du modèle peuvent changer et évoluer complètement ou partiellement.

Sur la figure 5.5, nous pouvons voir un exemple de l'interface `Neo4j` s'exécutant en mode serveur. Le graphe est affiché sur le navigateur *web* où la barre à gauche contient les informations sur la base de données, les étiquettes de nœuds, les types de relations et les clés des propriétés. Au centre, nous avons un exemple de graphe avec plusieurs nœuds (des aéroports), les relations (arcs). Comme la visualisation est interactive, il est possible de voir les liens d'un nœud du graphe résultant. Au-dessus, nous avons la section du texte pour écrire la requête.

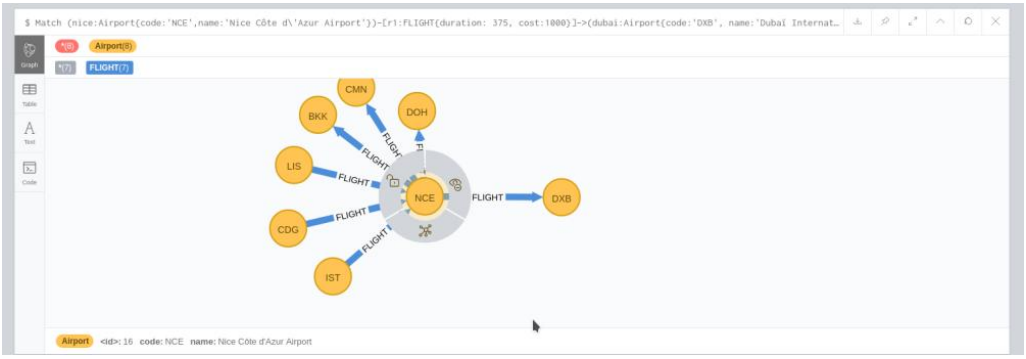


FIGURE 5.5 – Interface `Neo4j`.

Pour l'affichage des résultats, il existe trois modes :

— graphe (voir la figure 5.5) ;
— tableau (voir la figure 5.6) ;
— texte (voir la figure 5.7).

FIGURE 5.6 – Résultat en mode tableau.

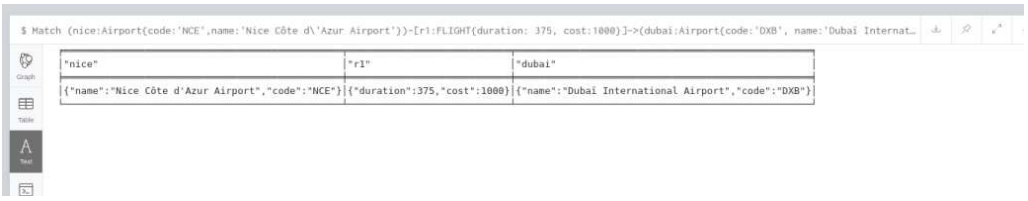


FIGURE 5.7 – Résultat en mode texte.

5.5.4.2 Les contraintes d'intégrité

Dans le modèle orienté graphe, les index et les contraintes d'intégrité sont optionnels mais il est recommandé de les définir pour obtenir des meilleures performances et surtout assurer l'intégrité et la cohérence des données. En `Neo4j`, le schéma est défini par les index et les contraintes.

Les index permettent de trouver rapidement les informations. En `Neo4j`, il est possible de créer des index d'une propriété de nœuds. La création des index en modèle orienté graphe se base principalement sur le même concept que le modèle relationnel ou document.

Les contraintes d'intégrité assure la cohérence de la base de données. Par exemple : "Si un nœud a une étiquette d'aéroport et une propriété *code*, la valeur du code doit être unique parmi tous les nœuds ayant l'étiquette *aéroport*". L'ajout de la contrainte unique ajoutera implicitement un index sur cette propriété. Contrairement aux *SGBDR*, les contraintes et les index peuvent être ajoutés à n'importe quel moment même avant la création des données. En revanche, si la base de données contient déjà les données, l'ajout des contraintes et des index prendra plus de temps.

Ces contraintes portent sur les nœuds et les relations [Technology, 2017]. Les contraintes sur les relations sont disponibles seulement dans l'édition entreprise.

Pour les contraintes, nous distinguons les contraintes d'unicité et d'existence. Cette dernière est disponible uniquement dans l'édition entreprise.

Elles sont traduites en `Neo4j` de la façon suivante :

— l'unicité de l'identifiant pour chaque nœud (intégrité d'entité) ;
— une propriété unique pour un seul nœud ;

— l'existence d'une propriété.

Nous verrons par la suite la création des contraintes pour notre exemple *2-hops* (voir la section 5.5.6)

5.5.5 Langage de requête *Cypher*

Dans cette section, nous allons présenter le langage *Cypher*, le langage de requête jumelé à `Neo4j`. *Cypher* est un langage fourni par `Neo4j` afin d'accéder, faire des requêtes et mettre à jour les données du graphe facilement.

La structure du langage *Cypher* est inspirée du *SQL* utilisé avec le modèle relationnel. Le langage *Cypher* est structuré à partir de différentes clauses. Deux clauses principales sont utilisées : *MATCH* et *WHERE* :

— *MATCH* est utilisée pour établir la structure du motif cherché.

— *WHERE* permet de filtrer les résultats de la sélection faite par *MATCH*.

Dans une requête *Cypher*, les nœuds sont représentés par une paire de parenthèses. Il y a deux types de relations : des relations orientées et non orientées. Les relations non orientées sont représentées par une paire de tirets (`--`). Quand c'est un graphe orienté, on ajoute une flèche à la représentation précédente pour spécifier l'orientation (`<--`, `-->`). Les propriétés et les étiquettes (en anglais *label*) sont incluses directement dans les parenthèses quand il s'agit des nœuds. Tandis que dans le cas des relations, on utilise une paire de crochets (*e.g.* -[identifiant : nom de la relation {nom de la propriété : valeur de la propriété}]-). L'identifiant est facultatif.

Exemple 5.5.1 (Syntaxe du langage *Cypher*)**.** Prenons l'exemple du listing 5.29 qui illustre d'une façon plus détaillée la syntaxe du langage *Cypher*. Supposons qu'on ait deux nœuds représentant l'aéroport de *Nice* et de *Dubai*.

Dans l'exemple du listing 5.29, la variable 'nice' est utilisée pour identifier le label 'Airport'. Ce dernier a deux propriétés `code` et `name`, qui correspondent respectivement aux valeurs 'NCE' et 'Nice Airport'.

```
1 (nice:Airport{code:'NCE' name:'Nice Airport'})
2 (dubai:Airport{code:'DXB' name:'Dubai Airport'})
```

Listing 5.29 – exemple de nœuds

Nous avons également un seul type de relations : `Flight` qui connecte les nœuds de cette façon `-[:Flight]->`. En combinant les éléments décrits précédemment, nous obtenons les motifs du listing 5.30.

```
1 (nice:Airport{code:'NCE' name:'Nice Airport'})-[:FLIGHT]-> (
    dubai:Airport{code:'DXB' name:'Dubai Airport'})
```

Listing 5.30 – exemple de relations

La figure 5.8 illustre le graphe des motifs décrits avant, les motifs qui représentent le vol entre *Nice* & *Dubai* d'une durée de 375 min et d'un coût 1000$:

En *Cypher*, il y a différentes clauses à utiliser quand il s'agit de la création ou bien de la recherche dans le graphe. Pour créer les entités du graphe, nous utilisons la clause *CREATE*. Une requête en *Cypher* se termine toujours par la clause *RETURN* pour renvoyer le résultat.

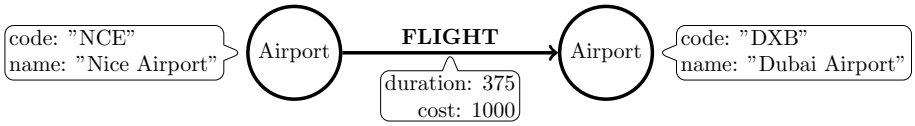


FIGURE 5.8 – Exemple de graphe en Neo4j.

Exemple 5.5.2 (Requête *Cypher*)**.** L'exemple suivant crée les motifs décrits ci-dessus et renvoie les éléments créés avec la requête du listing 5.31.

```
1 MATCH (nice:Airport{code:'NCE',name:'Nice Airport'})-[r1:FLIGHT{duration: 375,
      cost:1000}]->(dubai:Airport{code:'DXB', name:'Dubai Airport'})
2 RETURN nice,r1,dubai
```

Listing 5.31 – requête de création en Neo4j

Dans le résultat illustré dans la figure 5.9, les nœuds sont affichés par la propriété code.

FIGURE 5.9 – Résultat de la requête du listing 5.31.

5.5.6 *2-hops* dans le modèle orienté graphe

5.5.6.1 Manipulation des données en Neo4j

Tout d'abord, nous commençons par créer les données en Neo4j. Les figures illustrant les résultats sont présentées en annexe C. On retrouve les mêmes résultats obtenus avant en MongoDB et en SQL.

Exemple 5.5.3 (Création en *Cypher*)**.** La requête du listing 5.32 assure l'unicité du champ. Comme le code et le nom d'un aéroport sont uniques, si on veut créer un autre aéroport avec les mêmes informations sur le code et le nom, alors le système nous renvoie une erreur.

```
1 CREATE CONSTRAINT ON (airport:Airport) ASSERT airport.code IS UNIQUE;
2 CREATE CONSTRAINT ON (airport:Airport) ASSERT airport.name IS UNIQUE;
```

Listing 5.32 – requête pour créer une contrainte d'unicité en Neo4j

La requête du listing 5.33 assure que le champ code existe lors de la création d'un nœud de type aéroport. C'est valable uniquement dans la version *entreprise*.

```
1 CREATE CONSTRAINT ON (a:Airport) ASSERT exists(a.code)
```

Listing 5.33 – requête *Cypher* pour créer une contrainte d'existence

Exemple 5.5.4 (Insertion en *Cypher*)**.** Nous avons vu comment on crée les données en *Cypher* (voir la requête du listing 5.31). Toutefois, il est possible d'importer des données stockées dans un fichier *CSV*. Pour cela, nous utilisons la commande LOAD CSV, qui nous permet d'insérer en masse des données dans le graphe.

La commande possède des paramètres pour spécifier le séparateur de colonnes (ligne 2 du listing 5.34). La ligne 3 crée pour chaque ligne du fichier, un nœud de type Airport avec des propriétés code et name.

```
1 LOAD CSV WITH HEADERS FROM "file:///airports.csv"
2 AS line FIELDTERMINATOR ';'
3 CREATE (a:Airport{code:line.Code, name:line.Name})
4 RETURN a.code AS code, a.name AS name
```

Listing 5.34 – requête pour créer les nœuds à partir d'un fichier csv en Neo4j

De même pour insérer les vols représentés par des arcs dans le graphe de la figure 5.1, nous créons des relations en Neo4j contenant les informations de ces vols (voir la requête du listing 5.35). Pour cela, on commence par chercher, pour chaque vol du fichier flights dans le graphe, les nœuds de type Airport dont le code correspond au code origine et destination de ce vol (ligne 3 de la requête du listing 5.35). La clause MATCH dans la requête du listing 5.35 permet d'identifier cette paire d'aéroports. Ensuite, on crée la relation de type FLIGHT entre cette paire de nœuds avec les propriétés qui contiennent les informations du vol en termes de durée et coût (ligne 4 de la requête du listing 5.35).

```
1 LOAD CSV WITH HEADERS FROM "file:///flights.csv"
2   AS line FIELDTERMINATOR ','
3 MATCH (a:Airport{code:line.Origin}),(b:Airport{code:line.Destination})
4 CREATE r=(a)-[:FLIGHT{duration:toInt(line.Duration),cost:toInt(line.Cost)}]->(b
      )
5 RETURN r as relation
```

Listing 5.35 – requête pour créer une relation en Neo4j à partir d'un fichier csv

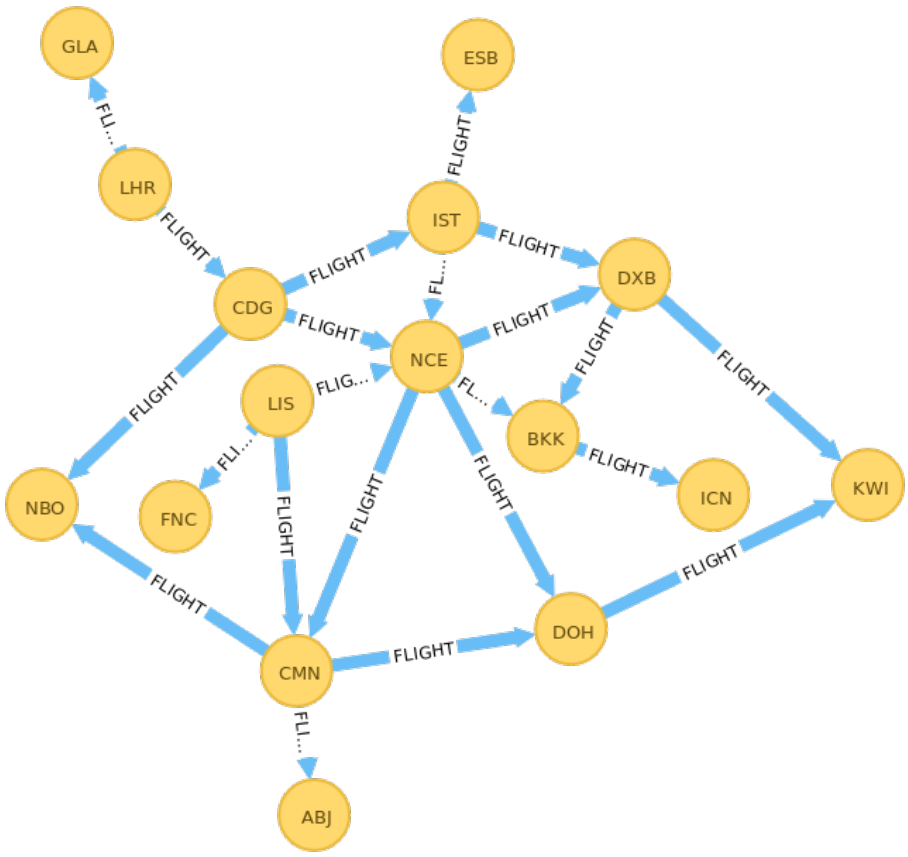


FIGURE 5.10 – Réseau aérien de la figure 5.1 sur `Neo4j`.

Exemple 5.5.5 (Affichage en *Cypher*)**.** Une fois que les données ont été créées et le graphe formé, la prochaine étape concerne le requêtage de ces données pour les exploiter. Nous utilisons donc la clause MATCH décrivant ce que nous cherchons. Le résultat est renvoyé sous forme d'une ligne qui correspond à chaque motif trouvé.

```
1 MATCH (a:Airport)
2 RETURN a AS airport
```

Listing 5.36 – requête pour l'affichage en *Cypher*

Exemple 5.5.6 (Liste des destinations en *Cypher*)**.** La requête du listing 5.37 renvoie la liste des destinations des vols partant de l'aéroport `NCE`. La variable `a` est utilisée pour identifier le label `Airport`. Ce dernier a la propriété `code` qui correspond à la valeur cherchée *NCE*. Les nœuds `Airport` sont liés entre eux par la relation `FLIGHT`. La clause `RETURN` permet de renvoyer le résultat.

```
1 MATCH (a:Airport{code:"NCE"})-[:FLIGHT]->(m)
2 RETURN m.code as destination
```

Listing 5.37 – requête pour récupérer la liste des destinations en *Cypher*

En `Neo4j`, il est possible de renvoyer plusieurs formats de résultats : table, texte et graphe.

Si nous souhaitons renvoyer le résultat sous forme d'un graphe, alors la requête du listing 5.37 sera modifiée comme dans le listing 5.38 (voir la figure 5.11).

```
1 MATCH p=(a:Airport{code:"NCE"})-[:FLIGHT]->(m)
2 RETURN p
```

Listing 5.38 – requête du listing 5.37 sous forme d'un graphe

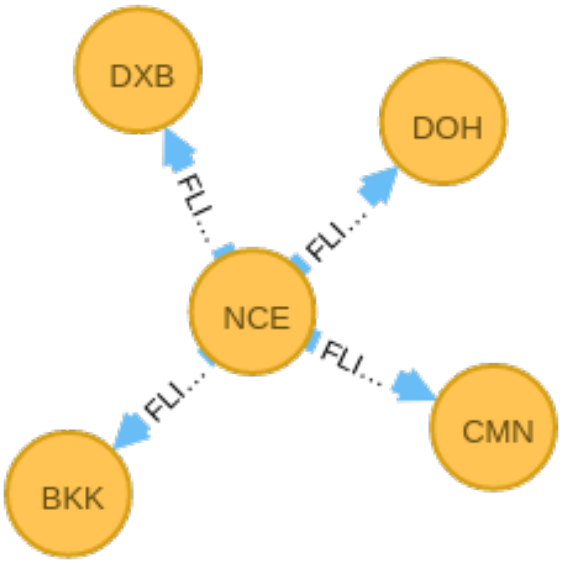


FIGURE 5.11 – Liste des destinations depuis l'aéroport NCE.

Exemple 5.5.7 (Destinations en moins de 2 heures en *Cypher*)**.** Nous employons un filtre avec la clause WHERE (voir le listing 5.39).

```
1 MATCH (a:Airport{code:"LIS"})-[r:FLIGHT]->(m)
2 WHERE r.duration<120
3 RETURN m.code as destination
```

Listing 5.39 – requête pour récupérer les destinations en moins de 2h en *Cypher*

Exemple 5.5.8 (Destination la moins chère en *Cypher*)**.** En *Cypher*, on utilise la même syntaxe qu'en SQL.

```
1 MATCH (a:Airport{code:"NCE"})-[r:FLIGHT]->(m)
2 RETURN m AS destination
3 ORDER BY r.cost ASC
4 LIMIT 1
```

Listing 5.40 – requête pour chercher la destination la moins chère en *Cypher*

airportName
Hamad International Airport
Mohammed V International Airport

FIGURE 5.12 – Résultat de la requête du listing 5.41.

Exemple 5.5.9 (Récupérer les noms en *Cypher*)**.** Contrairement à SQL et MongoDB, la jointure se fait d'une manière plus simple. En effet, les données sont stockées sous forme d'un graphe, donc la jointure est déjà fournie dans ce type de structure de données.

```
1 MATCH (o:Airport)-[r:FLIGHT]->(d:Airport)
2 WITH collect(o.name)+collect(d.name) AS listAirports
3 UNWIND listAirports AS airportName
4 RETURN DISTINCT airportName
5 LIMIT 2
```

Listing 5.41 – requête pour récupérer les noms en *Cypher*

La ligne 2 de la requête du listing 5.41 crée une liste de tous les codes d'aéroports ayant des vols (une relation de type `FLIGHT`). Ensuite, la clause `UNWIND` sépare les éléments de la liste pour récupérer le nom de chaque élément.

5.5.6.2 Requête de *2-hops* en *Cypher*

Nous présentons la requête consistant à répondre à la question posée par le problème *2-hops*. Ceci explique clairement les avantages de notre choix de base de données de stockage, qui est la base de données orientée graphe.

En `Neo4j`, le modèle de données de l'exemple du réseau aérien sera stocké sous cette forme. Grâce à ce graphe, nous pouvons déjà identifier des relations. Par exemple : quelles sont les destinations possibles en partant de l'aéroport *NCE* ?

`(NCE)-FLIGHT-(destinations)`

Le résultat obtenu sera : `DXB, DOH, BKK`, et `CMN`.

Très souvent dans les réseaux aériens, nous sommes intéressés par la recherche des itinéraires avec un nombre de sauts prédéfini. Prenons l'exemple ci-dessus, et supposons que nous souhaitions la liste des destinations à atteindre depuis un aéroport avec une seule escale. La question est exprimée sous cette forme : quels sont les aéroports voisins de mes aéroports voisins ?

`(NCE)-FLIGHT-(mesDest)-FLIGHT-(dest_de_mes_Dest)`

Bien évidemment, il faut exclure les aéroports dont il existe déjà un vol venant de l'aéroport *NCE*. C'est le cas de l'aéroport *DOH*.

Pour récupérer les aéroports atteignables depuis *NCE*, la requête en *Cypher* sera (voir le listing 5.42).

```
1 MATCH p=(o:Airport)-[:FLIGHT]->(d:Airport)-[:FLIGHT]->(dd:Airport)
2 WHERE NOT (o)-[:FLIGHT]->(dd) AND o.code="NCE"
3 RETURN distinct dd
```

Listing 5.42 – requête pour répondre au problème de *2-hops* en `Neo4j`

La variable `d` dans le motif de `MATCH` désigne la liste des destinations de l'aéroport d'origine. La variable `dd` est pour les destinations finales (voir ligne 1 de la requête du listing 5.42).

En langage *Cypher*, le nom est fourni directement avec l'accès à l'entité aéroport qui est stockée sous forme de nœud dans le graphe. Alors qu'en `MongoDB`, il fallait appeler une troisième méthode de recherche `find`, et dans le cas du `SQL`, une troisième jointure sur la table `airport`.

En résumé, les bases de données orientées graphe dépassent les *SGBDR* en termes de performances des requêtes et d'évolution des schémas dans le cas de données fortement interconnectées ou de mises à jour régulières des schémas. `Neo4j` s'adresse davantage à des systèmes vivants qu'à des systèmes d'archivages.

5.5.7 Terminologie avec le modèle relationnel et orienté document

La table 5.9 résume l'équivalence entre les termes utilisés en `Neo4j` avec ceux connus dans le modèle orienté document, notamment en `MongoDB` :

relationnel	orienté graphe	MongoDB
-	Graphe	-
Ligne	Nœud	Document
Ligne	Relation	Document
Colonne	Propriété	Champ
Table	Étiquette	Collection
Index	Index	Index
Curseur	Curseur	Curseur
Schéma	Schéma	-
Select	Match	Find
Update	Set	Update
Insert	Create	Insert

TABLE 5.9 – équivalence entre la terminologie en `Neo4j` et celle de `MongoDB`.

5.5.8 Limites de la base de données orientée graphe

D'un côté, Les *SGBDR* sont plus performants que les bases de données orientées graphe quand il s'agit de manipuler des données sur une seule table [ShefaliPatil and Bhatia, 2014]. Cela est dû à leur structure physique. Par contre, ce qui est coûteux en base de données orientée graphe, typiquement `Neo4j`, c'est la récupération des propriétés qui sont stockées séparément des fichiers des entités nœuds et relations. D'un autre côté, la volumétrie des données à traiter empêche la maintenance des propriétés *ACID*. Il faudra probablement passer aux propriétés *BASE*. En effet, Les propriétés *BASE* sont plus flexibles et s'occupent surtout de la disponibilité des données. Plutôt que de vérifier la cohérence des données après chaque transaction, il suffit que la base de données finisse par devenir cohérente (*Eventual consistency*).

En revanche, les propriétés *ACID* s'occupent surtout de la sécurité des données alors que le domaine ne l'exige pas forcément.

D'autres types de bases de données ne fournissent pas un langage déclaratif pour manipuler les données du graphe. [Pokornỳ, 2015] ont cité plus de limites.

5.5.9 Avantages de la base de données orientée graphe

Les *SGBDR* sont plus performants comme système de stockage. Or, quand il s'agit de faire des opérations sur ces données notamment les jointures et les filtres, cela devient plus coûteux en temps. En revanche, les graphes sont conçus pour parcourir des modèles plus complexes. L'utilisation des graphes comme moyen de représentation nous permet de modéliser aisément une très grande quantité des données qui sont très connectées tout en offrant une structuration flexible, des opérations de modification et de parcours performantes. De plus, les graphes ont un intérêt à identifier des relations et à travailler par sous-graphes mais également à identifier des chemins afin de traverser les données du graphes.

Comme l'exemple du problème *2-hops*, l'exécution d'une requête en `Neo4j` commence par une recherche par index pour trouver le point de départ du parcours. Ensuite, les relations sont traversées. L'algorithme de `DFS` est utilisé pour ce genre de parcours. Comme il existe un nombre exponentiel de chemins dans le graphe, un grand volume de données sera parcouru ce qui rend `Neo4j` plus performant que les *SGBDR*. Cela montre bien que, les bases de données relationnelles sont moins adéquates pour faire des requêtes à travers les relations. Ce parcours de graphe est tout simplement traduit en *SGBDR* par la réalisation d'un ensemble de requêtes sur différentes tables, suivant les clés étrangères et les autres index. Ce qui devient plus coûteux en terme de temps.

5.5.10 Niveau physique de la base de données `Neo4j`

Dans la plupart des systèmes de gestion des bases de données, il existe quatre types d'opérations basiques pour traiter les données : insertion, mise à jour, suppression et recherche. Les coûts de ces opérations dépendent de la façon dont ces données sont stockées, comment l'accès se fait ainsi que le volume de ces données.

Structure de la base de données `Neo4j`

Nœuds et relations. La structure de la base de données graphe est décrite comme suit [Czerepicki, 2016] : les nœuds (les éléments A, B et C sur la Figure 5.13 de la page 127), les relations (les éléments X, Y et Z sur la Figure 5.13 de la page 127) et les propriétés sont stockées dans des fichiers de données non ordonnés séparés. Ce type de fichiers stocke les enregistrements dans l'ordre dans lequel ils sont insérés. Ainsi, l'adresse d'un objet est définie comme un produit de la taille de l'enregistrement et un nombre identifiant qui est un nombre entier. Par conséquent, la complexité de l'opération de la recherche par un identifiant (ID) est en $\mathcal{O}(1)$, la taille de l'enregistrement étant fixe. Cela veut dire que cette complexité ne dépend pas de la taille de la base de données. Une complexité similaire est obtenue pour l'opération d'ajout. Cette dernière consiste à créer un nouveau champ à la fin du fichier de la base de données graphe. Sur le disque, les données sont stockées sous forme d'une liste chaînée. Les nœuds pointent vers le premier champ du nœud de la relation en question (voir la figure 5.13). Chaque relation pointe vers ses champs : le premier et second nœud mais également le champ de la relation précédente (resp. prochaine) pour le premier (resp. second) nœud. Donc, traverser une relation coûte $\mathcal{O}(1)$ quelle que soit la taille du graphe, qu'il soit en mémoire ou sur disque [Robinson et al., 2015].

Propriétés. Les propriétés d'un objet (un nœud ou une relation) sont obtenues à partir de ces objets [Raj, 2015]. Par exemple, sur la Figure 5.13, les propriétés *duration* et *cost* sont les propriétés de l'objet A et sont accessibles à partir de cet objet. Ce dernier contient une structure de données qui pointe vers l'objet de la première propriété. Cet objet contient le type de la propriété et sa valeur mais également un pointeur vers la prochaine propriété. Dans le cas où c'est la dernière propriété alors le pointeur sera nul. Sinon, la valeur sera l'adresse du fichier dans lequel la prochaine propriété est stockée car `Neo4j` stocke les propriétés comme un type de chaîne de caractères (*String*) quel que soit leur type (*entier, booléen, etc.*) [Xia et al., 2014]. Ainsi, l'ensemble des propriétés constitue une structure de données du type liste unidirectionnelle : on la parcourt dans un seul sens. Par conséquent, l'opération de la recherche d'une propriété par *numéro* consiste à chercher tout d'abord l'objet associé au nœud. Ensuite, il faut parcourir la liste des propriétés de ce nœud. La complexité de cette opération dépend du nombre de propriétés du nœud [Robinson

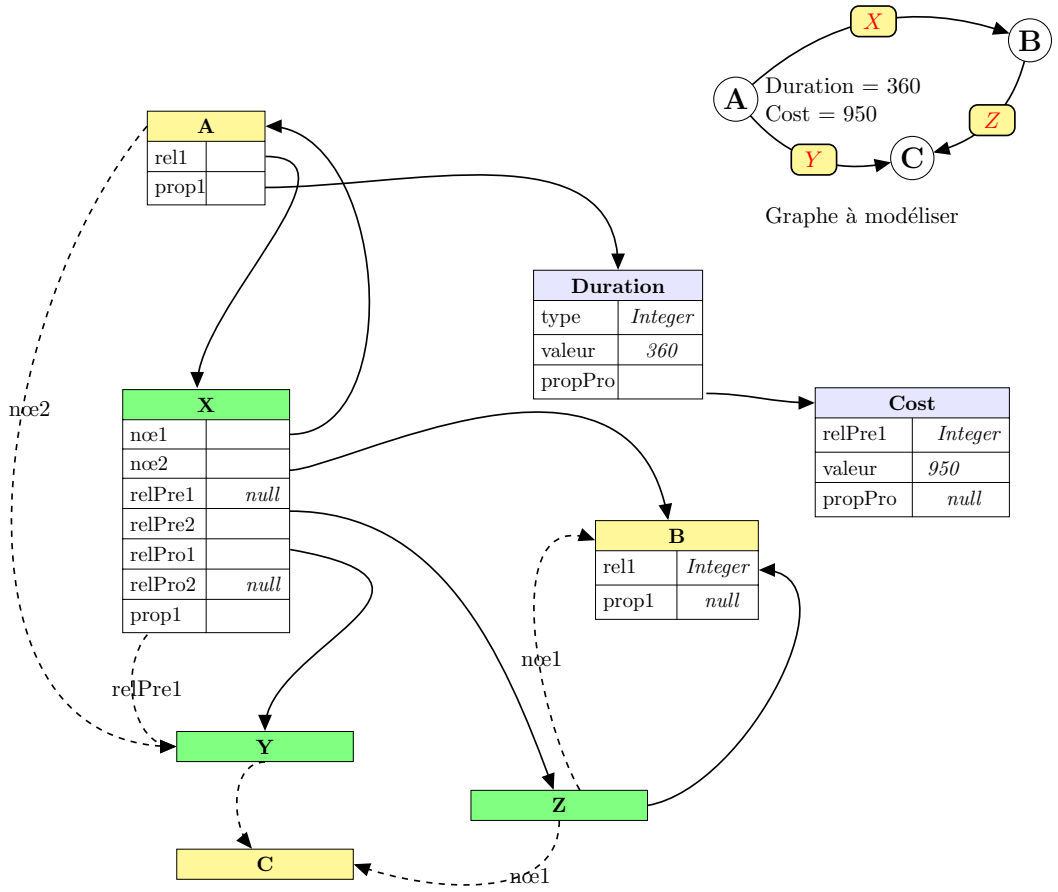


FIGURE 5.13 – Structure du fichier de la base de données orientée graphe. Le graphe modélise un réseau de vol tel que les nœuds sont les aéroports et les relations représentent les vols.

et al., 2015]. On trouve la même complexité pour les autres opérations telles que la suppression et l'ajout d'une propriété.

Inconvénients. La structure du fichier de données de la base de données orientée graphe est organisée pour faire des recherches dans les graphes : le temps d'exécution du parcours entre deux nœuds connectés par un arc ne dépend pas de la taille de la base de données graphe [Miler et al., 2014]. Toutefois, le vrai challenge est l'accès aux propriétés de ces entités qui se fait d'une façon dynamique contrairement à la théorie des graphes où cet accès est caractérisé comme un accès statique (les propriétés se trouvent au même endroit que les arcs et les nœuds). En effet, pour chaque opération incluant la propriété, il faut faire appel à des fichiers qui sont stockés séparément des fichiers des nœuds et des relations ce qui est coûteux dans une base de données orientée graphe. Contrairement aux bases de données relationnelles, la requête nécessitant de renvoyer les champs d'une table n'est plus coûteuse en terme de temps. Quand il s'agit de plusieurs tables interrogées, la requête devient coûteuse en *SGBDR* (opération de jointure).

Conclusion. Pour résumer, les données du graphe sont stockées en `Neo4j` sous forme des listes chaînées d'enregistrements de taille fixe. Les propriétés sont stockées sous forme d'une liste simplement chaînée. Chaque enregistrement de la liste contient une paire clé-valeur et un pointeur vers la propriété suivante. Chaque nœud et relation font référence à son premier enregistrement de propriété. Un nœud fait également référence à sa première relation de la liste de ses relations. Chaque élément de cette liste fait référence à ses deux extrémités. Il fait également référence à l'enregistrement de la relation précédente et suivante.

Utilisation du système cache

Concernant le système du cache, `Neo4j` met sur le disque la plupart des informations dans des champs de relations, ainsi les nœuds se réfèrent juste à leur première relation. Quant au système de cache, les nœuds portent des pointeurs vers toutes ses relations. Les relations, quant à elles, portent seulement leurs propriétés.

L'avantage du système de cache peut être illustré par les opérations qui consistent à trouver un chemin. Comme toutes les références se font par des ID, les parcours se font indirectement à travers le cache [Robinson et al., 2015]. Les relations pour chaque nœud sont regroupées par type de relations ce qui permet de parcourir rapidement un type spécifique de relation. En revanche, le temps d'exécution d'un parcours ne dépend pas de la taille du fichier de stockage de la base de données [Raj, 2015]. Cette approche surpasse les approches des autres bases de données (voir la figure 5.14).

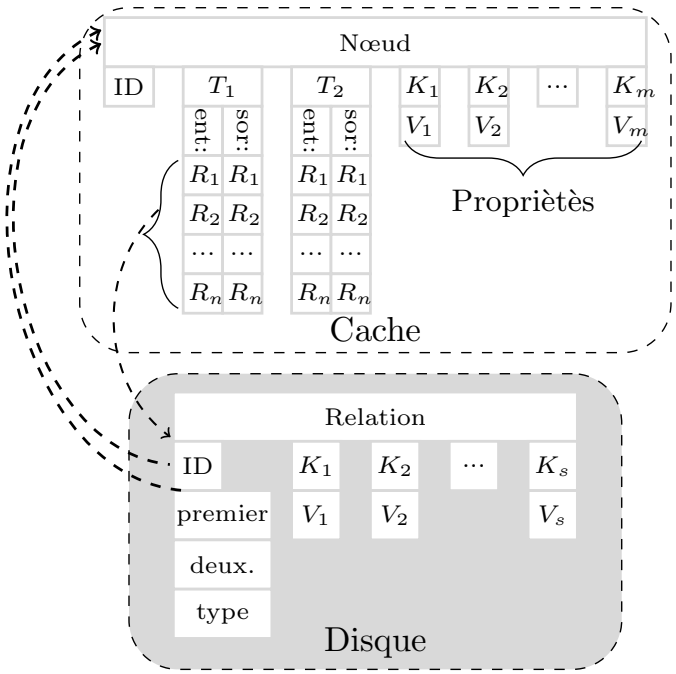


FIGURE 5.14 – Stockage cache/disque. Les flèches en pointillés représentent les références. Toutes les références se font par ID. Les relations sont regroupées par type. $\{T_1, T_2\}$ sont les types des relations de ce nœud.

En l'occurrence, sans fournir un index, chercher une propriété particulière nécessitera de parcourir toute la liste des propriétés avec un coût de $\mathcal{O}(n)$, n étant le nombre d'éléments de la liste des propriétés. Alternativement, le coût d'une recherche par un index est inférieur à $\mathcal{O}(\log_2 n)$ [Rodriguez and Neubauer, 2010].

Neo4j est représenté par une liste d'adjacence, donc visiter un voisin d'un nœud se fait en $\mathcal{O}(1)$ [Robinson et al., 2015].

5.6 Neo4j et Algorithmes de graphe

5.6.1 Algorithmes de parcours de graphe

Les algorithmes de parcours de graphes représentent les opérations les plus compliquées sur les bases de données [Czerepicki, 2016]. En effet, le parcours d'un graphe peut rapidement dépasser la mémoire maximale alors que le parcours n'est pas encore achevé. Neo4j est une base de données orientée graphe efficace pour effectuer ce type d'opérations par rapport aux autres bases de données. Un ensemble de tests réalisé pour la partie analytique du graphe et la recherche des chemins dans les graphes [Hölsch et al., 2017] montrent que la base de données orientée graphe Neo4j dépasse la base de données relationnelle en terme de temps d'exécution.

Neo4j est utilisé pour la partie analytique d'un réseau réel mais également pour faire des cheminements dans le graphe. Pour le calcul du plus court chemin, on trouve l'algorithme de *Dijkstra* [Holzschuher and Peinl, 2014]. [Miler et al., 2014] comparent la performance de l'algorithme Dijkstra dans les réseaux de transport stockés dans une base de données relationnelles et une base de données graphe comme Neo4j. Les résultats montrent que Neo4j surpasse les autres bases de données. Toutefois, la base de données graphe consomme beaucoup de mémoire lors du traitement des grands graphes.

Les opérations présentées dans la section 5.5.10 sont liées à des entités uniques de la base de données orientée graphe, elles peuvent êtres nommées en tant qu'*opérations simples*. Cependant, la recherche des chemins dans un graphe fait partie de la catégorie des *opérations complexes*, demandant des exécutions d'opérations simples (les opérations présentées dans la section 5.5.10) d'une manière itérative pour obtenir le résultat attendu. Dans Neo4j, le chemin renvoyé est une liste chaînée des nœuds connectés par des relations avec leurs ensembles de propriétés. Il est possible de renvoyer des chemins avec contraintes sous forme de conditions. Cette condition porte sur le type de relation, le nœud ou bien les propriétés.

Dans ce contexte, plusieurs études comparent Neo4j avec d'autres bases de données orinetée graphe ou d'autres types de bases de données telles que : NoSQL ou les bases de données relationnelles. Cette comparaison concerne la partie analytique et la partie algorithmique du graphe [Lal, 2015]. Les résultats montrent que Neo4j surpasse ces bases de données.

La table 5.10 de la page 132 présente quelques travaux menés dans la littérature sur Neo4j.

5.6.2 Liste des algorithmes

Voici la liste des algorithmes implémentés en Neo4j [Raj, 2015], ces algorithmes sont accessibles depuis l'API Java de Neo4j :

— allPaths : cette méthode renvoie tous les chemins possibles entre deux nœuds.

— allSimplePaths : cette méthode renvoie tous les chemins élémentaires possibles entre deux nœuds.

— `aStar` : cette méthode implémente l'algorithme A (voir le paragraphe 4.4.2).

— `Dijkstra` : cette méthode implémente l'algorithme de `Dijkstra` (voir la section 4.3.2.1).

— `pathsWithLength` : elle renvoie un algorithme qui calcule un chemin avec une longueur donnée entre deux nœuds en termes de nombre d'arcs.

— `shortestPath` : cette méthode implémente l'algorithme de `BFS` (voir la section 4.2.1.1).

[Larsson, 2008] fournit un rapport bien détaillé des premières versions d'implémentation des algorithmes l'analyse et le cheminement sur un graphe.

5.6.3 Évolution de la partie algorithmique

En 2017, la liste des algorithmes (voir la section précédente) est devenue accessible en langage *Cypher* grâce aux procédures *APOC* que nous présentons dans la section 5.6.4.

Depuis 2018, `Neo4j` propose davantage d'algorithmes accessibles depuis le langage *Cypher* [Allen et al., 2019] qu'on peut regrouper suivant les trois catégories :

— Problème de cheminement : k plus courts chemins [Yen, 1971], arbre couvrant de poids minimal [Nesetril et al., 2001], plus court chemin depuis la source [Ahuja et al., 1993], k arbres couvrants minimaux [Prim, 1957] (algorithme de *Prim*).

— Calcul de centralité, qui consiste à mesurer l'importance d'un nœud. Comme exemple, l'algorithme pour le calcul de la centralité intermédiaire [Freeman, 1977].

— Détection de communautés comme l'algorithme de *Tarjan* pour le calcul des composantes fortement connexes [Tarjan, 1972].

5.6.4 Procédures *APOC*

`Neo4j` propose la librairie *APOC* (en anglais *Awesome Procedures on Cypher*) comme des procédures stockées (450 procédures). Cette librairie regroupe une liste des procédures comme les algorithmes de calcul du plus court chemin dans les graphes [Neo4j and Cypher, 2020]. L'algorithme `Bidirectionnel` de `Dijkstra` est l'un de ces algorithmes pour trouver le plus court chemin entre deux points.

Par ailleurs, la base de données graphe `Neo4j` offre à ses utilisateurs la possibilité d'implémenter les algorithmes comme des procédures définies par l'utilisateur. En effet, avant la publication d'*APOC*, les développeurs devaient écrire leurs propres codes pour des fonctionnalités communes en *Cypher* ce qui entraînent beaucoup de redondances. Les développeurs de `Neo4j` ont créé une librairie standard *APOC* pour les procédures et les fonctions les plus courantes. Cela permet aux utilisateurs d'utiliser cette librairie et d'écrire uniquement les procédures qui répondent à leurs besoins spécifiques à chaque cas d'utilisation.

Voici la liste des principales catégories des procédures *APOC* [Sylvain et al., 2016] :

— opérations sur les fichiers : exportation et importation des données ;

— fonctions spatiales ;

— algorithmes de graphe : `Bidirectionnel`, `A*`, `Arbre couvrant à coût min` ;

— interaction avec des bases `NoSQL` comme `MongoDB`.

Pour résoudre le problème de plus court chemin depuis une source vers tous les nœuds, on ne peut pas utiliser la librairie *APOC*. En effet, les algorithmes de graphes possibles avec cette librairie sont tous pour le calcul du plus court chemin entre deux nœuds, notamment les deux variantes de l'algorithme de Dijkstra : Bidirectionnelle ou A*.

5.6.5 Études menées dans la littérature

Le tableau 5.10 présente un extrait des études menées dans la littérature sur la base de données orientée graphe Neo4j. Les colonnes de la table présentent les informations suivantes :

— objectif de l'étude ;

— type d'algorithme utilisé : algorithmes de graphe ou requêtes en *Cypher* ;

— base de données utilisée : certaines études proposent de comparer Neo4j avec d'autres bases de données qui peuvent être de la même catégorie ou bien d'une autre catégorie du *SGBD*.

— domaine de l'application ;

— taille du jeu de données considéré ;

— résultats de l'étude.

Dans la littérature, nous distinguons trois types d'études que nous abordons dans ce travail de thèse :

— Des études portent sur la modélisation du graphe en Neo4j. Dans ce genre d'études, on cherche à modéliser un réseau réel sur la base de données Neo4j.

— Des études portent sur l'analyse des métriques du graphe. Dans ce genre d'études, les auteurs utilisent des requêtes pour calculer le diamètre, le degré, *etc.*

— Des études ont pour objectif le développement des algorithmes de graphe ou le test des algorithmes qui sont déjà implémentés en Neo4j.

Auteurs	Objectif de l'étude	Algorithme	BD	Domaine	Taille (jeu de données)	Résultats
[Czerepicki, 2016]	modélisation en modèle graphe recherche des informations dans le graphe	requêtes	Neo4j	transport public	/	Neo4j est plus adapté pour ce genre d'études
[Miler et al., 2014]	comparaison du plus court chemin entre deux nœuds	Dijkstra en *Cypher*	Neo4j, *PostgreSQL*	transport	630 000 nœuds 750 000 arcs	Neo4j est plus performant
[Mpinda et al., 2015]	évaluer la technique d'indexation bases de données de graphe	BFS en *Cypher*	Neo4j, *OrientDB*	réseau social	5 000 nœuds 49 000 arcs	Neo4j est plus performant
[Partl et al., 2016]	développer un outil d'analyse visuelle des chemins graphe statique non pondéré	BFS	Neo4j	biologie	10 000 nœuds et arcs	/
[Hölsch et al., 2017]	comparer avec le modèle relationnel recherche d'informations	requêtes	Neo4j, anonyme	enseignement	/	*SGBDR* est plus performant pour des requêtes sur une seule table
[Maduako et al., 2018]	modéliser un réseau de bus analyser des métriques	requêtes analytiques développer des algorithmes comme procédures stockées	Neo4j	réseau de bus canadien	table horaire 2 semaines	Neo4j permet de mieux analyser le comportement du réseau (fluide, congestion) facilité d'inclure les informations temporelles
[Vukotic et al., 2015]	évaluer les performances d'un parcours	BFS	Neo4j,anonyme	réseau social	1 000 000 personnes 50 relations d'amitié	Neo4j est plus rapide pour les algorithmes de graphe
[Morishima and Matsutani, 2014]	implémenter des algorithmes de plus court chemin pour une paire de nœuds technique de parallélisme	Dijkstra, A*	Neo4j	réseau social	100 000 nœuds	Dans le cas du parallélisme, les meilleures performances sont obtenues quand le nombre de requêtes de recherche augmente
[Martínez Porras et al., 2016]	manipulation des données	requêtes	Neo4j *MySQL*	médical	22 types de nœuds 23 types de relations 28 tables 100/ 10 000/100 000 entrées	Neo4j dépasse *MySQL* quand il s'agit de 100 000 entrées par table ou bien des requêtes avec plusieurs jointures
[Batra and Tyagi, 2012]	manipulation des données	requêtes	Neo4j *MySQL*	cinéma	100/ 500/	Neo4j dépasse *MySQL* quand on augmente la taille des données

TABLE 5.10 – Comparaison des travaux Neo4j dans la littérature.

Contributions

CHAPITRE 6

Schéma et analyse des données aériennes

Dans ce chapitre, nous analysons les données existantes de Milanamos*. Nous commençons par définir les termes ainsi que la base de données aérienne* `optimode`*. Ensuite, nous présentons en détail les collections qui nous intéressent de la base de données* `optimode`*. Nous présentons en dernier l'analyse de ces données, les données réelles présentant souvent des soucis.*

6.1 Base de données aérienne

6.1.1 Base de données `optimode`

L'application `PlanetOptim` est basée sur la base de données `optimode` qui contient des données aériennes et quelques données ferroviaires voire de transports en commun [Milanamos, 2016]. Cette base contient six collections principales, le reste des collections étant des collections pré-calculées que *Milanamos* utilise comme un moyen de stockage pour accélérer les calculs.

Dans `optimode`, les données sont collectées mensuellement avec un index sur le champ `year_month` (mois de l'année) que nous allons présenter dans la prochaine section de la description des collections de la base. Les principales collections de la base de données `optimode` sont [Milanamos, 2016] :

— `aircraft` regroupe les informations relatives aux avions.

— `airport` fournit toutes les informations sur les aéroports : code de l'aéroport, pays, ville, coordonnées géographiques, *etc.*

— `company` donne les informations sur les compagnies aériennes : nom, code, base de la compagnie aérienne, alliance, *etc*.

— `o&d` contient toutes les informations sur les itinéraires des passagers : origine, destination, les aéroports de correspondance, `pax`, le revenu, *etc.*

— `schedule` contient toutes les informations sur les vols programmés : la durée, le type de compagnie aérienne, l'heure de départ, l'heure d'arrivée, le terminal, la fréquence, les jours où le vol est opéré, *etc.*

— `segment` stocke toutes les données sur les segments de vol : origine, destination, `pax`, revenu, distance, type d'avion, *etc.*

Milanamos collecte les données depuis plusieurs sources. Les données sur les horaires des vols sont fournies par *OAG* (*Official Airline Guide*), une entreprise qui analyse les horaires et fréquences des compagnies aériennes du monde entier. Les données sur le trafic stockées dans les collections `o&d` et `segment` sont fournies par l'entreprise *Sabre*, des clients de *Milanamos* tels que : les aéroports et les compagnies aériennes.

Dans `optimode`, le processus d'alimentation des données collectées peut prendre plusieurs jours et *Milanamos* garde toutes les données, avec la possibilité de mise à jour mais jamais de suppression. Nous utilisons alors deux indicateurs :

— `record_ok` : un indicateur booléen indique que les données sont en cours d'intégration. Cet indicateur est utilisé dans les collections : `o&d`, `segment` et `schedule`.

— `active_rec` : cet indicateur est utilisé dans la collection `schedule` pour spécifier la disponibilité du programme du vol pour le mois considéré.

6.1.2 Schéma de la base de données

6.1.2.1 Diagramme de la base de données aérienne

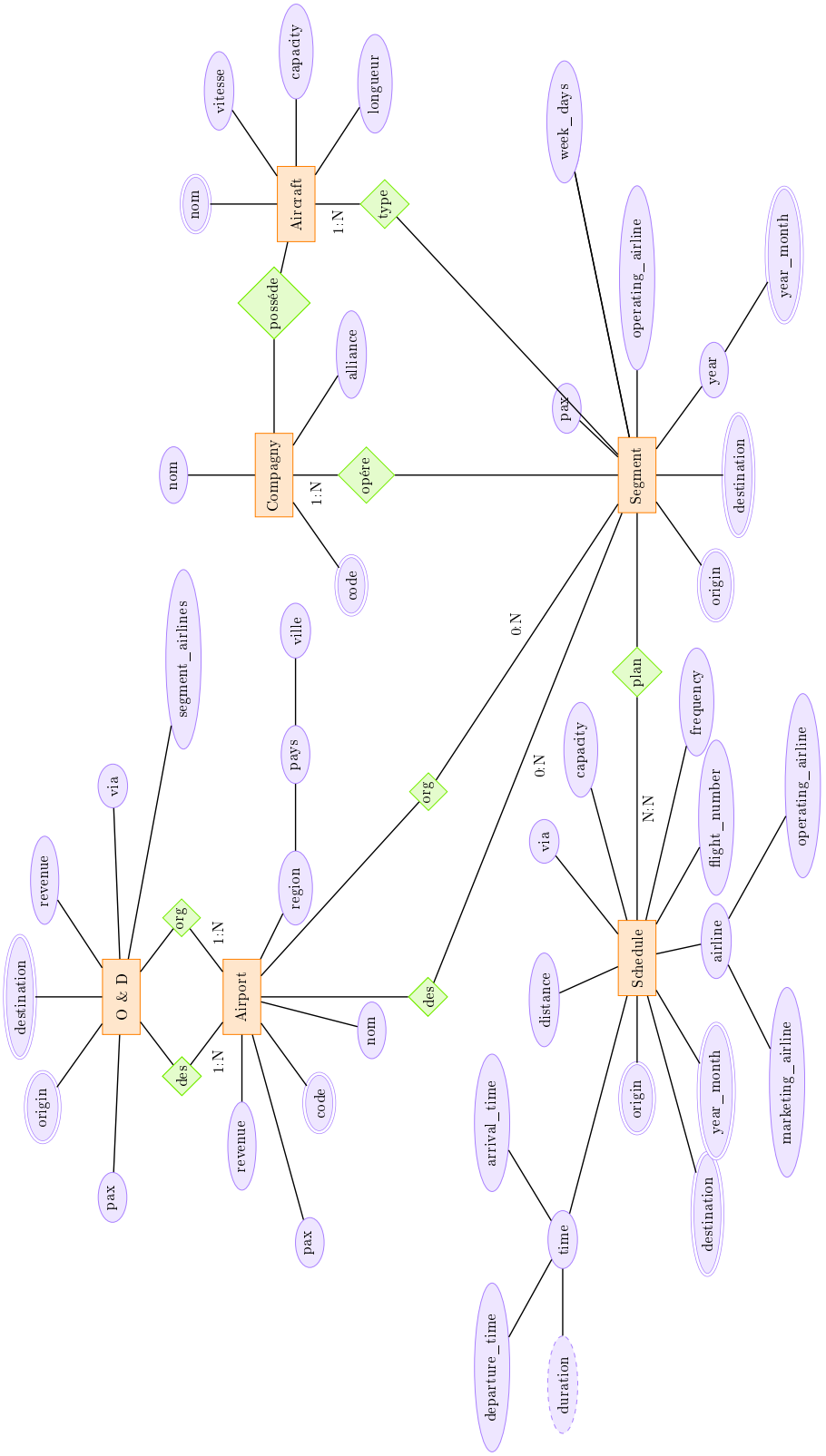


FIGURE 6.1 – Diagramme entité-association de la base de données `optimode`.

La figure 6.1 illustre le diagramme informel entité-association de la base de données `optimode`. Les collections listées dans la section précédente sont représentées par des rectangles ; les ellipses sont les champs dans chaque collection ; les liens décrivent la relation entre le document d'une collection et le document d'une autre. Les champs doublement cerclés représentent les champs en commun avec les autres collections. Selon ce diagramme, on a les caractéristiques suivantes :

— Une compagnie peut opérer plusieurs segments.
— Un segment de vol a plus d'un programme de vol.
— Plusieurs segments peuvent partir du même aéroport.
— Plusieurs segments peuvent arriver au même aéroport.
— Une compagnie aérienne possède plusieurs avions.
— Chaque segment de vol est associé à un modèle d'avion spécifique.
— Plusieurs segments représentent la même paire `OD`.
— Plusieurs horaires par segment de vol.

6.1.2.2 Visions représentées sur la base de données aérienne

La figure 6.2 illustre la vision concernée par chaque collection. La collection `o&d` représente les itinéraires des passagers. La collection `segment` représente les routes réalisées par les avions. La collection `schedule` est le produit vendu par la compagnie aérienne, c'est-à-dire les vols proposés pour les passagers et opérés par les compagnies aériennes.

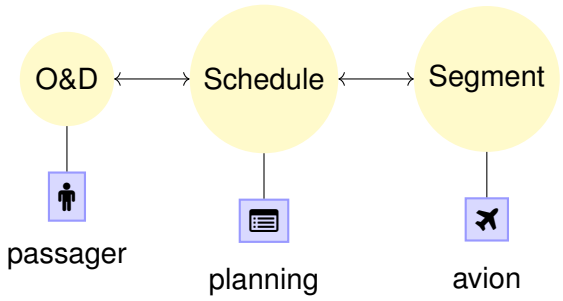


FIGURE 6.2 – Visions représentées dans `optimode`.

6.1.2.3 Statistiques de la base de données aérienne

Nous allons voir par la suite, l'ensemble des informations en détail de la base de données `optimode`. La requête du listing 6.1 nous renvoie les statistiques de la base de données `optimode` (voir les résultats dans le listing 6.2) [1].

```
1 db.stats()
```

Listing 6.1 – requête pour les statistiques de la base de données optimode

1. Les requêtes de chapitre ont été faites en mois d'avril de l'année 2019.

```
{
        "db" : "optimode",
        "collections" : 172,
        "datasize" : NumberLong(5853971454592),
        "storageSize" : NumberLong(1541438582784),
        "nindexes" : 1079
}
```

Listing 6.2 – résultats de la requête 6.1

La base de données `optimode` contient un grand volume de données environ 6 To, information donnée par le champ `datasize`. On peut également avoir l'information sur la taille de la base de données compressée (`storageSize`). La base de données `optimode` contient 1 079 index sur les 172 collections. La plupart des collections de la base `optimode` stockent les résultats des pré-calculs [2]. Dans la suite de l'étude, nous nous baserons sur le marché *segment* pour la facilité des calculs.

6.2 Description du modèle aérien

Dans cette section, nous allons présenter les principales collections que nous avons manipulées dans cette étude : `airport`, `schedule` et `segment`.

6.2.1 Modélisation des aéroports

Dans `optimode`, les aéroports sont stockés dans la collection `airport`. En plus des aéroports, la collection `airport` contient des données des gares de trains et des stations des bus. Un code `AITA` (en anglais *IATA*) [3] ou `OACI` (en anglais *ICAO*) [4] permet d'identifier ces structures géographiques. Le code `IATA` est celui qui figure sur le billet acheté par le passager ; il concerne la partie commerciale. Quant au code `OACI` est plutôt pour la partie opérationnelle, telle que le plan de vol.

En pratique, *Milanamos* se base sur le code `AITA` et utilise le code `OACI` en absence du code `AITA`.

Les documents de la collection `airport` sont modélisés de la façon suivante :

— Un document pour chaque code identifiant. Il identifie une localisation qui peut être : un aéroport, une gare de train, une station de bus, une ville, un pays ou bien une région.

— Chaque document possède les champs : code, nom, ville, pays, région. En plus de ces champs, il y a des données calculées telles que : revenu, `pax`, *etc*.

La collection `airport` comporte 11 999 documents (voir la requête du listing 6.3) dont 11 518 (voir la requête du listing 6.4) ne représentent que des données sur les documents représentant des aéroports ou gares (soit des 95% des données de la collection).

2. En `MongoDB`, les nombres entiers de 64 bits sont représentés par *NumberLong*.
3. Le code `AITA` est un code de trois lettres, attribué par l' *Association internationale du transport aérien (*AITA*)*
4. Le code `OACI` est un code de quatre lettres, attribué par l'*Organisation de l'Aviation Civile Internationale (*OACI*)*

```
db.getCollection('airport').find({}).count()
```

Listing 6.3 – nombre de documents de la collection `airport`

Dans la requête du listing 6.4, Le champ `code_type` sert à filtrer les documents représentant les structures géographiques : aéroport, gare de trains ou station des bus.

```
db.getCollection("airport").find({"code_type":"airport"}).count()
```

Listing 6.4 – nombre de documents correspondant à des structures géographiques

```
db.airport.find({"code_type": "airport","country":"FR"}, {"ville":1,"code": 1,"
    _id": 0}).sort({"pax": -1 }).limit(5)
```

Listing 6.5 – Top 5 des aéroports français

La requête du listing 6.5 permet d'afficher le top 5 des aéroports français en terme de volume de passagers. La donnée `pax` a été calculée par *Milanamos* pour un besoin. Nous recalculons la valeur exacte dans la section 7.6.5.

Nous trouvons deux blocs dans la méthode `find` : le premier précise les données cherchées, par exemple le champ `country` pour spécifier le code du pays dans lequel nous effectuons la recherche. Le deuxième bloc spécifie les colonnes que nous souhaitons renvoyer dans le résultat.

Le lecteur trouvera un exemple de résultat de la requête du listing 6.5 dans la table 6.1 :

Ville	Code	Pax
Paris	*CDG*	`NumberLong`(9 546 164 793)
Paris	ORY	`NumberLong`(2 613 249 322)
Nice	NCE	1 184 835 069
Toulouse	TLS	845 808 774
Lyon	LYS	764 317 612

TABLE 6.1 – résultats de la requête du listing 6.5.

Pour accéder directement à l'ensemble des informations d'une collection, comme la collection `airport`, nous utilisons la même méthode `stats()` :

```
db.airport.stats()
```

Listing 6.6 – requête des statistiques de la collection `airport`

```
{
    "ns" : "optimode.airport",
    "count" : 11999,
    "size" : 27311698,
    "storageSize" : 29691904,
    "nindexes" : 7,

```

```
8 }
```

Listing 6.7 – statistiques de la collection `airport`

L'unité utilisée par défaut est Byte mais il est possible de passer en paramètre l'unité de conversion. Par exemple, si on veut le résultat en *Kilobyte* (*kb*), alors la requête du listing 6.6 devient celle du listing 6.8 :

```
1 db.airport.stats(1024)
```

Listing 6.8 – statistiques de la collection `airport` en kilobyte

Si on veut récupérer directement une information, comme par exemple la taille des données (`dataSize`), il suffit d'appeler directement la méthode `dataSize()`, voir la requête du listing 6.9.

```
1 db.airport.dataSize()
```

Listing 6.9 – taille de la collection `airport`

6.2.2 Modélisation du trafic aérien

6.2.2.1 Présentation de la collection `segment`

Le trafic aérien est représenté sous forme d'un ensemble de segments de vols tel que chaque segment de vol est stocké dans un document dans la collection `segment`.

La collection `segment` est la plus grosse collection de la base de données `optimode`. Elle stocke toutes les données des segments de vols, la liste des champs suivante n'est pas exhaustive :

- aéroport de départ ;
- aéroport d'arrivée ;
- mois d'opération ;
- compagnie aérienne ;
- revenu généré ;
- `pax` ;
- type de segment (`segment_split`).

Exemple 6.2.1 (Collection `segment`)**.** Un exemple d'extrait de document de la collection `segment` se trouve dans le listing 6.10.

```
1  /* 1 */
2  {
3          "leg_origin" : "AAA",
4          "leg_destination" : "BBB",
5          "segment_split" : "Local",
6          "trip_origin" : "AAA",
7          "record_ok" : true,
8          "segment_revenue_usd" : 8808,
9          "trip_destination" : "BBB",
10         "operating_airline" : "JJ",
11         "passengers" : 3,
```

```
        "year_month" : "2010-09"
}
```

Listing 6.10 – exemple d'extrait de document de la collection `segment`

Les segments de vols sont stockés dans la collection `segment`. Les champs `leg_origin` et `leg_destination` spécifient l'origine et la destination du vol. `trip_origin` représente d'où on vient. Quant au champ `trip_destination`, il indique la prochaine destination.

L'exemple de document 6.10 nous montre que les vols partant de l'aéroport *AAA* vers *BBB* opéré au mois de septembre de l'année 2010 par la compagnie aérienne *JJ*. Ce vol a généré un revenu de $8 808 et trois passagers l'ont pris.

Dans la collection `segment`, Un segment de vol est représenté par `year_month`, `leg_origin`, `leg_destination`, `operating_airline`, `segment_split`, `trip_origin` et `trip_destination`.

La requête du listing 6.11 renvoie les statistiques de la collection *segment*.

```
db.segment.stats()
```

Listing 6.11 – requête des statistiques de la collection `segment`

```
{
        "ns" : "optimode.segment",
        "count" : 1542918929,
        "size" : NumberLong(1264710538165),
        "storageSize" : 329463439360,
        "nindexes" : 9,

}
```

Listing 6.12 – résultat de la requête du listing 6.11

D'après le résultat illustré dans 6.12, la collection *segment* contient environ un milliard et demi de document stockés dans 1 *To*.

La base `optimode` de *Milanamos* possède dix-sept ans d'historique de données. La requête du listing 6.13 renvoie la date du plus ancien `year_month` stockée dans la base, ainsi que la date du `year_month` le plus récent (voir le résultat du listing 6.16). La méthode *sort* permet de trier les résultats, qui sont bien évidemment sous forme de document *JSON*. Elle comporte un seul paramètre pour spécifier l'ordre dans lequel on veut faire le tri :

— -1 : tri ascendant (équivalent à *ASC* en `SQL`) ;

— 1 : tri descendant (équivalent à *DESC* en `SQL`).

```
db.segment.find({},{"_id":0,"year_month":1}).sort( {"year_month": 1} ).limit(1)
```

Listing 6.13 – requête pour trouver l'ancien `year_month` dans la collection `segment`

La méthode `limit` permet de limiter le nombre de documents renvoyés dans le résultat. Ainsi, `limit(1)` renvoie le document le plus récent dans le cas du tri descendant. De la même façon, quand il s'agit d'un tri ascendant, elle renvoie le document correspondant au `year_month` le plus ancien.

```
        "year_month" : "2002-01"
```

Listing 6.14 – résultat de la requête du listing 6.13

```
db.segment.find({},{"_id":0,"year_month":1}).sort( {"year_month": -1} ).
    limit(1)
```

Listing 6.15 – requête pour trouver le `year_month` le plus récent dans la collection `segment`

```
{
        "year_month" : "2019-01"
}
```

Listing 6.16 – résultat de la requête du listing 6.15

Avec les données de la collection *segment*, on peut répondre à ce genre de questions :

— Quel est le nombre total des passagers pour une paire `OD` par année ?

— Quel est le revenu total par année ?

— Quelles sont les destinations possibles depuis un certain aéroport ?

— Quelles sont les compagnies aériennes qui opèrent un certain segment de vol ?

Si on cherche à savoir le flux total des passagers par région ou par pays, il faut faire une jointure avec la collection `airport` qui stocke les informations des aéroports.

6.2.2.2 Perspectives des segments

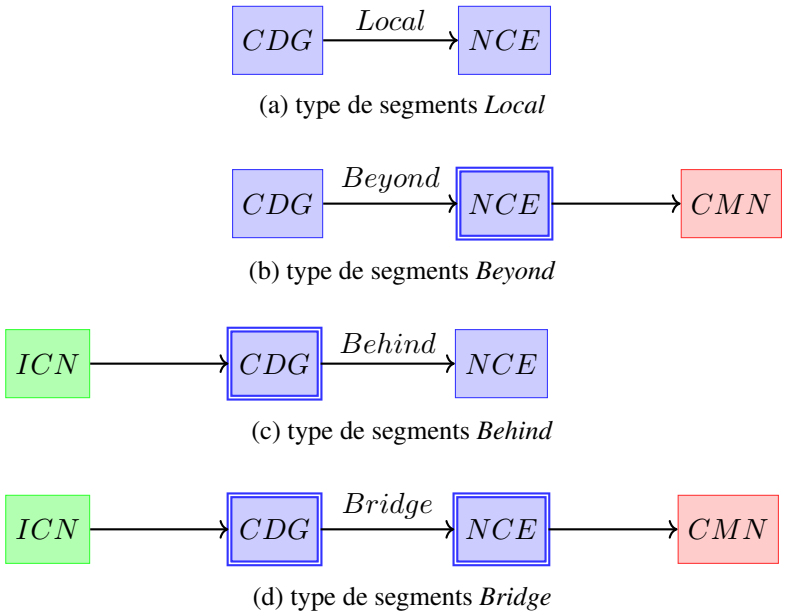
Les aéroports utilisent le terme *segment_split* pour distinguer les différents types de connexions (voir Figure 6.3 de la page 146). Les responsables des aéroports ou bien des compagnies aériennes ont souvent besoin de connaître le flux de passagers suivant une certaine perspective, plutôt que de considérer le point de vue des passagers impliqués par la demande. Ainsi, selon ces visions, nous distinguons quatre types de segments [Goedeking, 2010]. Prenons l'exemple des deux hubs *CDG* et *NCE* (voir la Figure 6.3 de la page 146) :

— Local : Il représente le segment local, c'est-à-dire, le flux de passagers qui a fait le vol sans escale de *CDG* à *NCE* (voir la figure 6.3a).

— Beyond : le segment *Beyond* concerne plutôt la perspective du hub *NCE*, et représente le flux des passagers venant du hub *CDG* pour rejoindre *NCE*, et faisant une correspondance à *NCE* pour aller à d'autres destinations (voir la figure 6.3b).

— Behind : Du point de vue du hub *CDG*, le segment *Behind* représente le flux des passagers qui arrive en amont de ce hub de n'importe quel aéroport, en faisant une correspondance à *CDG* pour pouvoir rejoindre *NCE* (voir la figure 6.3c).

— Bridge : le segment *Bridge* combine les trois perspectives. Cette vision représente les flux de passagers venant de n'importe quel aéroport, passant par le segment *CDG-NCE* pour aller vers n'importe quelle destination (voir la figure 6.3d).



(a) type de segments *Local*

(b) type de segments *Beyond*

(c) type de segments *Behind*

(d) type de segments *Bridge*

FIGURE 6.3 – Perspectives des segments. L'aéroport concerné par la vision est représenté par un double carré.

Cette perspective spécifique à un aéroport reflète le fait que les aéroports ne cherchent pas à comprendre le véritable flux *O&D*, mais à comprendre les flux *via* leurs aéroports respectifs. Typiquement, voici les questions qu'on peut poser :

— D'où viennent les passagers qui se connectent à l'aéroport qu'on observe ?
— Où vont-ils ensuite ?
— Combien d'origines/destinations ?
— Quelles sont les origines/destinations les plus importantes ?
— Combien de passagers restent sur place ?
— Combien de passagers sont en transit ?

Cette étude préalable permet aux responsables de mieux gérer la facturation et la comptabilisation des redevances aux compagnies aériennes.

Quand il s'agit d'un segment local, alors le champ `trip_origin` (respectivement `trip_destination`) est systématiquement le même que `leg_origin` (respectivement `leg_destination`). Si on représente ces informations sur l'exemple de la figure 6.3, et en prenant le cas de *Bridge* comme type de *segment*, on aura alors les informations de tableau 6.2.

champ	valeur
leg_origin	"CDG"
leg_destination	"NCE"
segment_split	"Bridge"
trip_origin	"ICN"
trip_destination	"CMN"

TABLE 6.2 – exemple de la représentation de segment *Bridge* de la figure 6.3 dans la collection segment.

6.2.3 Modélisation des marchés

La collection o&d représente les marchés. Elle contient environ 600M documents et sa taille est 586 Go. Cette collection contient des informations sur le trafic aérien vis-à-vis la vision *o&d* (voir la section 2.2.2.1 du chapitre 2), nous trouvons, ainsi, des informations sur les compagnies qui vendent les sièges et les aéroports de correspondance (voir la figure de la page 139).

6.2.4 Modélisation des horaires

La collection schedule contient toutes les données relatives aux plannings des vols. Elle contient les deux types de vols : *direct* et *sans escale*. Pour distinguer ces deux types de vols, il y a un champ spécifique qui est via. Quand via est *null*, alors il s'agit d'un vol sans escale, autrement dit c'est un vol direct. Chaque document de la collection schedule représente les informations relatives à un vol : origine, destination, numéro de vol, heure de départ, heure d'arrivée, durée, distance, terminal de départ, terminal d'arrivée, capacité de l'équipement, capacité totale, saisonnalité des vols, les dates des vols, le nombre de jours du vol, la fréquence et le *code share* pour préciser si le vol est partagé ou non. Cette collection possède environ quatre cents millions de documents et sa taille est 609 Go.

6.3 Analyse des données aériennes

La partie de l'analyse des données aériennes consiste à analyser les collections décrites auparavant. Comme les données viennent du monde réel, certaines fausses manipulations ont entraîné les problèmes suivants lors de l'extraction des données :

— Certains aéroports n'ont pas de vols commerciaux.

— Certaines origines ou destinations manquent dans la collection des aéroports.

— Les segments de vols n'ont pas d'information sur les horaires.

— Des données sont manquantes sur plusieurs champs.

— La distance récupérée du système de réservation et celle calculée par *Milanamos* ne sont pas les mêmes.

— Des données sont erronées : les coordonnées géographiques des aéroports, le nombre de passagers, *etc*.

— Des segments n'ont aucun passager.

Cette analyse nous permettra d'estimer les données manquantes pour chaque collection utilisée. En effet, la qualité des données joue un rôle très important sur le résultat attendu et dans l'expérience des utilisateurs. Ainsi, les questions auxquelles nous cherchons à répondre :

— Combien d'aéroports n'ont pas de données sur le trafic et les horaires ?

— Combien de segments n'ont pas d'informations sur les horaires et sur les aéroports ?

— Combien de vols n'ont pas de données sur le trafic ?

— Quelles sont les caractéristiques des aéroports sans coordonnées géographiques ?

Cette étape est nécessaire pour estimer la taille et la complexité de notre problème étudié. On cherche à comprendre quelles sont les données manquantes et si possible la cause de ces erreurs pour faire des hypothèses. Par la suite, cela nous permettra de bien mener les démarches suivantes :

1. modéliser le graphe ;
2. comprendre sa structure et son évolution dans le temps ;
3. choisir la méthode adéquate pour résoudre le problème étudié,
4. valider les hypothèses par les résultats des expériences.

6.3.1 Analyse de la distance

La distance est stockée dans deux collections : `segment` et `schedule`. La distance dans la collection `schedule` est une donnée brute fournie par le système de réservation. Lors de l'analyse des données en entrée, nous avons constaté que la valeur de la `distance` n'est pas identique. L'erreur absolue entre les deux métriques se calcule comme suit :

$$E(d_{se}, d_{sc}) = d_{se} - d_{sc}$$

où d_{se} (respectivement d_{sc}) est la distance de la collection `segment` (respectivemment `schedule`).

Si l'erreur absolue $E(d_{se}, d_{sc})$ dépasse un certain seuil que nous avons fixé à 200 km car 90% des segments ont une distance de 4 000 km, et que les avions roulent en moyenne à 700km/h, alors nous estimons que cette distance est erronée et par conséquent, on la récupère directement de la collection `segment`. Nous estimons qu'il y a 62% des documents de la collection `schedule` dont la distance est erronée. Nous avons donc corrigé la distance en utilisant la formule de **haversine** que *Milanamos* utilise. Ce calcul tient compte de la rotondité de la planète.

6.3.2 Analyse du trafic

Nous avons fait une seconde analyse pour voir s'il existe bien des segments avec plus de 10 passagers par mois. En effet, des segments avec en moyenne 30 passagers par mois impliquent un seul passager par jour. Nous avons donc fixé le seuil à 10 par mois pour prendre en compte le cas où il y a 0 passager. La requête du listing 6.17 est faite pour un seul mois qui est celui de janvier 2017 (`2017-01`).

La requête du listing 6.17 regroupe l'ensemble des documents correspondant au mois `2017-01` par origine et destination et calcule le nombre total de passagers pour chaque paire origine-destination. Ainsi, nous filtrons 13% des segments sur 51953 segments ayant moins de 10 passagers par mois.

```
pipe = [
      {'$match': {'record_ok': True, 'year_month': year_month}},
   {'$group': {
   '_id': { 'origins': '$leg_origin',
   'destinations': '$leg_destination'},
   'pax': {'$sum': '$passengers'}
   }}]
cursor = SegmentInitialData.aggregate(pipe, allowDiskUse=True)
```

Listing 6.17 – segments avec moins de dix passagers par mois.

6.3.3 Analyse de la collection des aéroports

Nous avons estimé le nombre de données manquantes par champs. Le tableau 6.3 résume le pourcentage pour chaque champ identifié par ce manque.

champ	pourcentage
city	2%
country	1%
region	1,8%
lonlat	8%

TABLE 6.3 – les données manquantes de la collection `airport`.

Comme constaté dans la table 6.3, le champ `lonlat` est le plus affecté par ce manque de données, c'est le champ qui stocke les informations sur les coordonnées géographiques. Suite à cela, nous avons fait une analyse plus détaillée pour identifier les caractéristiques de ces entités et ainsi comprendre cette absence. La requête du listing 6.18 est faite en `MongoDB` pour trouver la région des aéroports n'ayant pas de coordonnées géographiques. Le résultat est illustré graphiquement dans la figure 6.4.

```
db.airport.find({'code_type':'airport',lonlat':null},{"lonlat":1,"region":1})
```

Listing 6.18 – requête pour trouver la liste des régions des aéroports sans coordonnées géographiques

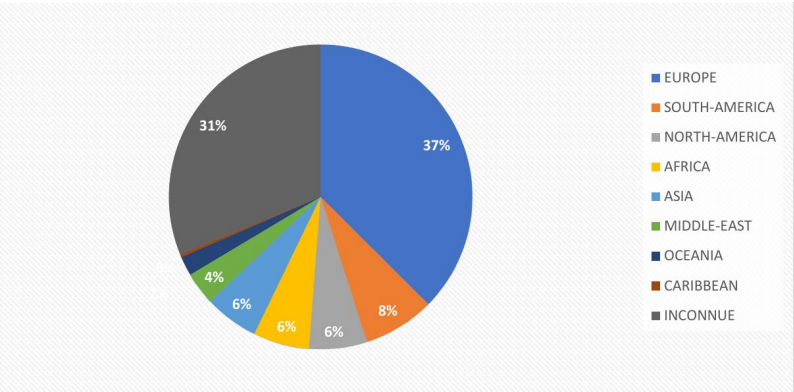


FIGURE 6.4 – Répartition des aéroports par région.

Une grande partie des aéroports fait partie de la région européenne. Ceci peut être expliqué par la présence des aéroports, nommés *offLine* (57%) dans le cas des vols charters. Un vol charter est un vol non régulier, souvent reservé par un groupe de vacances. La destination peut être vers une ville non desservie par la compagnie aérienne (*offLine*). 43% sont des gares de trains et des bus.

6.3.4 Analyse de la collection des segments de vols

6.3.4.1 Information sur le trafic

Après avoir étudié la collection des aéroports, nous avons analysé la collection `segment` vis-à-vis de la collection `airport`. C'est-à-dire, nous avons estimé le pourcentage des aéroports sans données sur le trafic.

Nous avons trouvé 6 862 aéroports parmi 11 518 qui n'ont pas de vols commerciaux (environs 60%), on les trouve ni dans `segment` ni dans `schedule`. Ce qui représente un grand volume par rapport aux aéroports existants.

6.3.4.2 Information sur les horaires de vols

Nous avons estimé combien de segments de la collection `segment` n'ayant pas de données de *schedule*. La requête du listing 6.19 agrège les données par mois et par segment. Un segment est représenté par une paire d'aéroports (origine, destination).

```
pipe = [
      {'$match': {'record_ok': True,'year_month': year_month}},
      {'$group': {
            '_id': {
                  'origin': '$leg_origin',
                  'destination': '$leg_destination'},
            'pax':{'$sum':'$passengers'}}},
      {'$match': {'pax': {'$gte': 10}}},
      {'$sort': SON([('_id.origin', 1), ('_id.destination', 1)])}]
data_seg = SegmentInitialData.aggregate(pipe, allowDiskUse=True)
```

Listing 6.19 – segments sans horaires de vols

Ainsi, sur 52 474 segments, nous avons 24% des segments ayant uniquement des informations sur le trafic, en termes de `pax` et revenu. En sélectionnant les segments ayant plus de 10 passagers par mois (voir la section 6.3.2), on passe à 9% (voir le programme dans le listing 6.21). Comme exemple, le vol entre **Malte** et **Dubaï**.

6.3.5 Analyse de la collection des horaires de vols

Comme les sources de données fournies sur le trafic et les horaires sont différentes, nous avons également identifié le nombre de segments de vols sans informations sur le trafic pour le mois août de l'année 2017 (voir la requête du listing 6.20). Ainsi sur 51 120 des segments de vols, nous avons 1% des segments n'ayant pas d'informations sur le trafic. Ce pourcentage est assez négligeable par rapport au pourcentage des segments sans informations sur les *schedule* (voir le programme dans le listing 6.21). Comme exemple, le vol entre **Singapour** et **Los Angeles**.

```
pipe = [
      {'$match': {'record_ok': True, 'active_rec': True, 'via': None,
                        'code_share': False,'year_month': '2017-08'}},
      {'$group': {
                  '_id': {'origin': '$origin','destination': '$destination'}
                        }},
      {'$sort': SON([('_id.origin', 1), ('_id.destination', 1)])}]
data_sch = ScheduleInitialData.aggregate(pipe, allowDiskUse=True)
```

Listing 6.20 – segments de vols sans informations sur le trafic

Certains segments de vols n'ont pas d'horaires notamment les vols en cabotage et les vols charters.

```
def main():

    print("connect to optimode")
    Model.init_db()

    data_seg = data_sch_seg(year_month)
    data_sch = data_seg_sch(year_month)

    scheduleSet =  set()
    segmentSet =  set()

   for sch in data_sch:
            id = sch['_id']
           od =  id['origin'] + "-" +  id['destination']
           scheduleSet.add(od)

   for seg in data_seg:
            id = seg['_id']
           od =  id['origin'] + "-" +  id['destination']
           segmentSet.add(od)

```

```
22     diff_sch_seg =scheduleSet.difference(segmentSet)
23     diff_seg_sch = segmentSet.difference(scheduleSet)
24
25
26 if __name__ == '__main__':
27      main()
```

Listing 6.21 – script pour l'analyse des collections en `Python`

6.3.6 Données sur les compagnies aériennes

Nous avons analysé également les données sur les compagnies aériennes. En particulier, les compagnies à bas coûts (LCC) (en anglais *low-cost carrier*s). Cette donnée nous sera utile pour la section de l'identification des hubs (voir la section 7.6.5).

La collection `company` stocke toutes les informations sur les compagnies aériennes. Le champ `airline_cluster` nous permet de distinguer les types de compagnies aériennes. Les *LCC* appartiennent à deux classes : `LOWCOST` et `VALUE`, la célèbre compagnie *easyJet* fait partie de cette dernière alliance.

```
1 db.company.find({"airline_cluster":{"$in":["VALUE","LOWCOST"]}, "active":
      true},{"_id":0,"iata_code":1}).count()
```

Listing 6.22 – requête pour le nombre des compagnies aériennes *LCC*

Le champ `active` dans la requête du listing 6.22 permet de sélectionner que les compagnies qui sont toujours actives. En fait, *Milanamos* ne supprime pas de données. Nous avons, ainsi, 138 *LCC* dont 29 de l'alliance `VALUE` sur 4 475 compagnies aériennes (voir la requête du listing 6.22). Nous avons aussi remarqué qu'il y a 704 compagnies aériennes sans catégorie, `null` (voir la requête du listing 6.23).

```
1 db.company.find({"airline_cluster":null, "active":true},{"_id":0,"iata_code
      ":1}).count()
```

Listing 6.23 – requête pour des compagnies aériennes sans catégorie

6.4 Comparaison avec l'aviation civile

6.4.1 Données sur le trafic

Toujours dans le cadre de l'analyse du trafic, nous avons comparé le trafic (le nombre de passagers) sur un ensemble de segments de vols publiés par la direction de l'aviation civile française *DGAC* [Bureau de l'observation du marché, 2017] avec les données fournies par *Milanamos*. Cette étude comporte trois types de relations :

— Liaison radiale : entre la région parisienne et une autre ville française.
— Liaison transversale : entre les villes françaises ou la région parisienne.
— Liaison intérieure : entre les villes du département Outre-Mer.
— Liaison internationale : entre la France et les autres pays du monde.

6.4.1.1 Statistiques de la comparaison

Nous nous sommes basés sur la collection `Segment` pour le calcul des `Pax`. La requête 6.24 renvoie le nombre de passagers pour un segment de vol pour l'année 2017. Le paramètre `codes` est la liste des codes d'aéroports figurant dans le rapport de la *DGAC*.

```
pipe = [
      {'$match': {'record_ok': true, 'year_month': {'$gte':'2017-01', '$lte':
         '2017-12'},{'leg_origin': {'$in':codes}}, {'leg_destination': {'$in':
         codes}}}},
      {'$group': {
            '_id':
                  {'origin':'$leg_origin', 'destination': '$leg_destination'},
            'pax': {'$sum': '$passengers'}}}]
cursor = SegmentInitialData.aggregate(pipe, allowDiskUse=true)
```

Listing 6.24 – requête pour le calcul du `Pax`

On calcule l'écart δ entre le nombre de passagers calculé et celui de la *DGAC*. Nous considérons, ainsi, qu'un écart au dessous de 5% est négligeable. Selon les résultats, nous avons classé qu'il y a trois catégories :

— Catégorie 1 : $0 \leq \delta \leq 5\%$.

— Catégorie 2 : $\delta > 5\%$, le cas où il y a une sous-estimation.

— Catégorie 3 : $\delta < 0\%$, le cas où il y a une surestimation.

Le tableau 6.4 nous donne le nombre de routes pour chaque catégorie. La comparaison a été faite pour les trois premiers types de liaisons, au total 181 liaisons. Nous avons, ainsi, 80% des routes telle que l'écart δ est non négligeable. 45% de ces routes ont moins de `Pax` que la *DGAC*.

n°	inter. (δ)	nbr_routes
1	$[0\,;5\%]$	35
2	$]5\%\,;100\%]$	82
3	$[-150\%\,;0\%[$	62

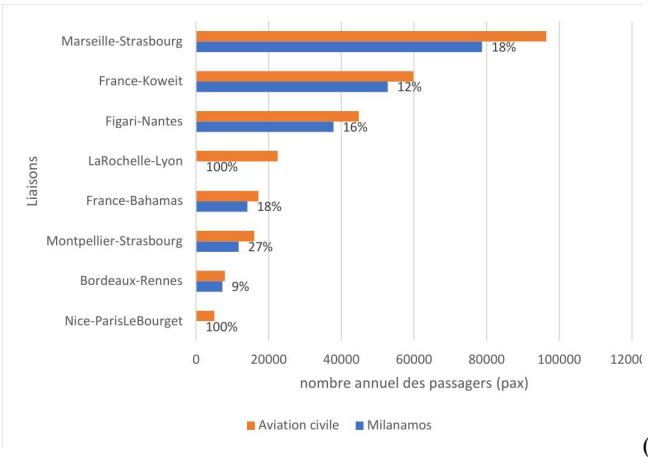
TABLE 6.4 – statistiques de la comparaison.

6.4.1.2 Exemples des routes

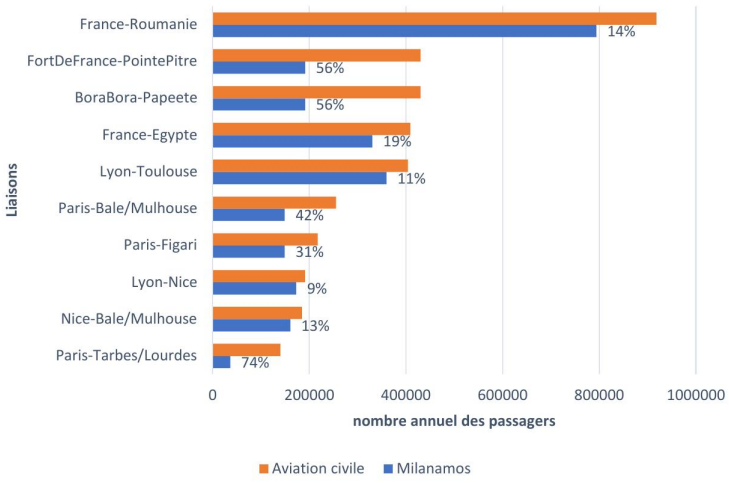
Nous avons donc classé cet ensemble de segments de vols en terme de nombre annuel de passagers :

1. liaisons ayant moins de 100 000 passagers.
2. liaisons ayant entre 100 000 et 1 000 000 de passagers.
3. liaisons ayant plus de 1 000 000 passagers.

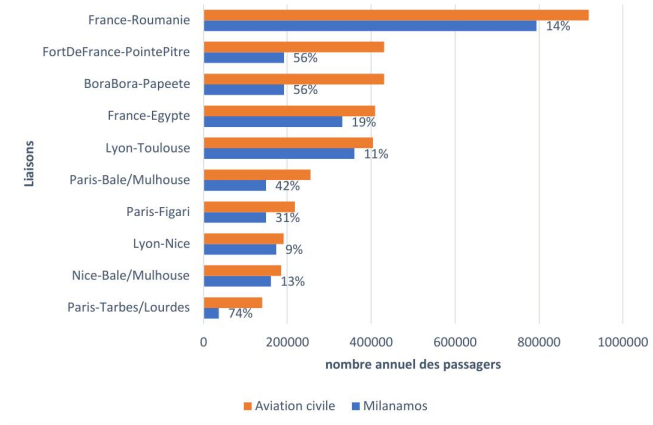
La figure 6.5 est un histogramme illustrant les écarts pour les trois catégories. On peut voir un écart assez important pour les liaisons intérieures mais également extérieures à la France. L'écart est illustré sur les figures.



(a) les données des liaisons ayant moins de 100 000 `pax`.



(b) les données des liaisons ayant entre de 100 000 et 1 000 000 `pax`.



(c) les données des liaisons ayant plus de 1 000 000 `pax`.

FIGURE 6.5 – Écart des passagers entre les données de *Milanamos* et les statistiques publiées.

Pour mieux comprendre ce manque, nous avons cherché les capacités de ces routes. La figure 6.6 présente les capacités des liaisons présentées dans la figure 6.5a. Le taux de remplissage (en anglais *Passenger Load Factor*) est illustré sur les barres bleues.

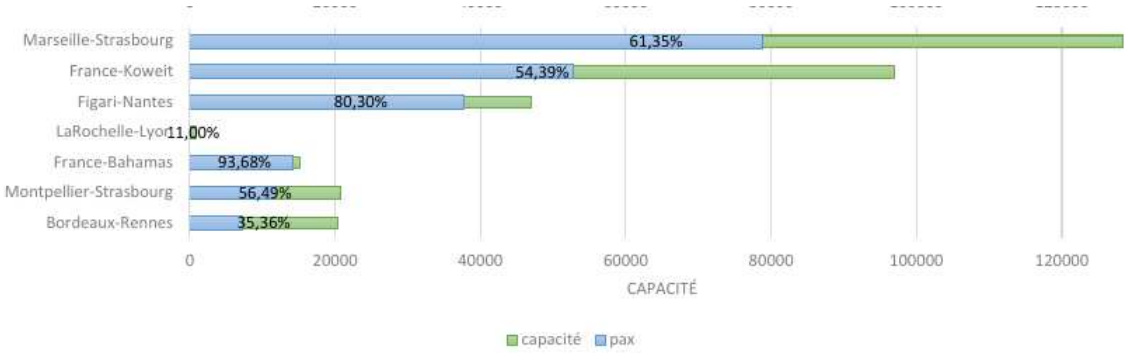


FIGURE 6.6 – Capacité des liaisons ayant moins de 100 000 `pax`.

6.4.1.3 Explications

Les statistiques publiées par l'aviation civile française [Bureau de l'observation du marché, 2017] compte le nombre de passagers effectuant des vols avec escale d'un point vers un autre point comme un segment de vols. Alors que *Milanamos* considère ce type de vol comme deux segments de vols. Après discussion avec le responsable de données, il estime que la manière de comptage et la vision du marché ne sont pas du tout identiques.

Une analyse est donc poussée dans ce sens puisque l'écart est assez important, notamment le cas du segment (Paris-Tarbes/Lourdes) où l'écart dépasse 70%. Nous avons trouvé un taux de remplissage moins élevé qui est dû à une capacité importante allouée pour ces vols mais avec un nombre de passagers moins élevé. C'est une situation assez étrange. Nous avons regardé en détail la liste des compagnies opérant ces vols et avons détecté que la compagnie aérienne *Royal Air Maroc* avait un souci. Après une vérification sur le site de la compagnie pour voir si elle desservait bien cette destination, nous avons trouvé que la destination n'était pas du tout présente parmi ses destinations. Nous avons dans ce cas une donnée erronée et manquante.

En réalité, les données aéroportuaires fournies correspondent à un estimatif des mouvements opérationnels et du trafic de l'aviation commerciale de transport de passagers : le fret, l'aviation militaire, l'aviation générale, et toutes les activités de loisir, de sport ou de travail aérien ne sont pas inclus dans les chiffres manipulés. Les données sont issues de différentes bases de données, comme suit :

— Mouvements commerciaux : l'Official Airline Guide est la source principale de données, corrigées par des informations de suivi de vol (base Flight Radar [2] ou autres) et complétées par des données des aviations civiles et plate-formes aéroportuaires. item Trafic passagers : les données estimées croisent différentes sources d'informations telles que les chiffres publiés par les aviations civiles et plate-formes aéroportuaires, la banque de données européenne Eurostat, et les données issues des billets émis par les agences de voyages, y compris en ligne.

Aujourd'hui, les données de *Milanamos* sont fournies par la compagnie *Sabre*, qui se base sur son système de réservation des billets électroniques comme la compagnie *Amadeus*. Or, pour la partie manquante comme les vols commerciaux, les billets achetés sur place dans une agence commerciale, les passagers privés, *etc.*, Sabre se base sur une estimation de ces données manquantes.

2. site web qui affiche les informations en temps réel sur les vols

6.4.2 Données sur les aéroports

Nous avons comparé le nombre d'aéroports du pays *France* avec les données publiés sur le site *Open data* [ourairports.com/, 2019]. Nous avons trouvé qu'il y a 327 aéroports dans la base `optimode` (requête du listing 6.4 avec un filtre sur le pays *France*).

En regardant les noms des aéroports, nous avons détecté qu'il y a des gares de trains parmi les aéroports. Ces gares sont stockées avec la même étiquette que les aéroports (`code_type:airport`). Cela ne permet pas de récupérer directement les données aériennes (voir la requête du listing 6.25 et 6.26).

```
1 db.airport.find( {"code_type":"airport", "country":"FR",name: /.*Rail.*/ } )
```

Listing 6.25 – requête pour trouver les documents correspondant aux données ferroviaires

```
1 db.airport.find( {"code_type":"airport", "country":"FR",name: /.*Bus.*/ } )
```

Listing 6.26 – requête pour trouver les documents correspondant aux données de bus

Finalement, il y a 27 gares de trains, 3 stations de bus et 300 aéroports. Or, les dernières statistiques de l'aviation civile qui datent de juillet 2019 indiquent qu'il y a 440 aéroports français. 310 aéroports publiés avec le code `iata` et 130 avec le code `icao` en absence du code `iata`.

6.4.3 Données sur les horaires

Autre problème, les *Schedule* fournis par la compagnie *OAG* présentent les programmes annoncés par les compagnies aériennes. Or, ce qui se passe en temps réel n'existe pas dans les données de *Milanamos*, notamment les derniers changements d'un vol tels que l'annulation des vols et les vols privés. Un travail de stage est en cours sur ce sujet.

Conclusion.

L'analyse des données de *Milanamos* nous a aidé à avoir connaissance des données manquantes et erronées qui vont par la suite créer des problèmes lors de la génération du graphe. En effet, la qualité des données joue un rôle très important sur le résultat attendu et dans l'expérience utilisateur. Nous avons, ainsi pu gérer ces soucis en amont de la génération du graphe pour sélectionner, nettoyer et corriger les données. Cette étape nous a permis de mieux estimer la taille et la complexité de notre problème. Le tableau 6.5 présente un récapitulatif des collections présentées dans ce chapitre avec les champs ayant des soucis.

Collections	#Documents	Taille	Champs
`airport`	11 999	28	`lonlat`
`segment`	1 542 918 929	1 264 710	`segment_revenue_usd`, `distance`
`schedule`	400 000 000	609 000	`distance`, `duration`
`o&d`	600 000 000	586 000	-

TABLE 6.5 – récapitulatif des collections de la base de données aérienne `optimode`.

CHAPITRE 7

Modélisation du réseau aérien

Dans ce chapitre, nous présentons le modèle proposé pour le réseau aérien de Milanamos *ainsi que les différents modèles existant dans la littérature pour modéliser les réseaux de transport, notamment, les réseaux aériens. Nous décrivons en détail la construction de notre réseau aérien à partir de la base de données* `optimode` *sur la nouvelle base de données orientée graphe* `Neo4j`*. Dans le cadre du test de performance, nous montrons que l'intégration des données sur* `Neo4j` *est plus rapide que l'extraction depuis* `MongoDB`*. Nous analysons la structure du réseau aérien et étudions les propriétés des réseaux dites "petit-monde". Nous présentons en dernier l'approche proposée pour l'identification des* hubs.

7.1 Modélisation des réseaux de transport

Dans le domaine des transports, il existe de nombreux réseaux avec différentes caractéristiques. Bien que les réseaux routiers puissent être considérés comme des réseaux statiques, les réseaux aériens dépendent intrinsèquement du temps. Il est important que les différentes approches de modélisation prennent en considération les données d'entrée, notamment, l'aspect temps. En outre, tout type de réseau peut être affecté par des modifications nécessitant des ajustements de la structure de données sous-jacente. Cette section présente les modèles les plus pertinents étudiés, ainsi que leurs applications et leurs caractéristiques. Dans la littérature, il existe deux approches pour modéliser les réseaux de transport qui varient en fonction du temps : les approches qui sont indépendantes du temps et celles qui en dépendent [Pajor, 2009].

7.1.1 Modèles indépendant du temps

L'approche indépendante du temps est la plus utilisée pour modéliser des réseaux routiers : chaque arc u est associé à une valeur constante $w(u)$ comme, le temps de trajet, la distance ou d'autres métriques qu'on cherche à minimiser. Quand il s'agit d'une information statique, ce modèle est largement suffisant pour représenter ce genre de réseau de transport, parce que la distance géographique ne dépend que du temps. Un tel modèle est plus simple et compact en raison d'absence de dépendances temporelles. C'est la raison pour laquelle, de nombreuses recherches s'articulent autour des accélérations des algorithmes du plus court chemin sur ce type de graphe. Souvent, ce sont des réseaux routiers qui donnent des meilleurs résultats [Bast et al., 2016].

Dans un réseau routier, l'absence de dépendance temporelle génère encore des requêtes utiles. D'autres modèles, comme les réseaux ferroviaires, ne le font certainement pas. Le problème de la dépendance temporelle est inhérent à ces modèles, car les trains ne fonctionnent qu'à certains moments et donc la recherche de la route la plus rapide dépend fortement de l'heure de départ dans la journée [Geisberger, 2011].

7.1.2 Modèles dépendant du temps

Le modèle dépendant du temps est une approche plus efficace pour modéliser les dépendances temporelles. Les poids des arcs dans un tel modèle ne sont plus des valeurs constantes. Dans le cas des réseaux aériens, un nœud correspond à chaque aéroport et un arc est inséré s'il existe une connexion directe entre deux aéroports. Pour chaque arc, on associe une fonction plutôt qu'une valeur constante. Cette fonction renvoie le temps d'arrivée et l'aéroport de destination (l'aéroport d'arrivée) selon l'heure de départ de l'aéroport d'origine [Delling et al., 2009a,b]. Pour tenir compte de la dépendance temporelle, nous utilisons une fonction temporelle f qui calcule le poids de chaque arc en fonction de l'heure de départ. Ainsi, le calcul des plus courts chemins dépend fortement de l'heure de départ t_s depuis la source. Ce qui peut donner des chemins de longueurs différentes mais aussi des chemins complètement différents.

En principe, la fonction utilisée $f : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ est une fonction périodique, le choix porte sur une fonction linéaire par morceaux pour une implémentation efficace [Pajor, 2009]. Lorsqu'on traverse le graphe, le temps d'arrivée peut être calculé à chaque aéroport. Ainsi, avec cette façon, on peut calculer le temps de parcours en prenant en compte le temps d'attente avant le prochain passage [Fortin et al., 2016].

Après avoir présenté la différence entre les deux approches de la modélisation d'un réseau de transport, nous présentons l'application de ces modèles dans le cas d'un réseau aérien. Nous focalisons sur les modèles indépendant du temps. Ce sont les modèles utilisés dans ce travail de thèse. Nous commençons par présenter les modèles utilisés pour les réseaux ferroviaires qui sont la base de notre approche proposée pour la modélisation du réseau aérien. En effet, les modèles utilisés dans la littérature pour les réseaux aériens utilisent le modèle dépendant du temps pour inclure l'information temporelle y compris le temps d'enregistrement et le temps de départ [Pajor, 2009, Delling et al., 2009a].

7.1.3 Modélisation des réseaux ferroviaires

Contrairement aux réseaux de transport routiers qui sont assez canoniques, il existe plusieurs approches pour modéliser les réseaux ferroviaires. Après avoir présenté les horaires, qui sont la base de tous les modèles ferroviaires, nous donnons un aperçu des modèles existants, discutant de leurs avantages et de leurs inconvénients, et abordons plus en détail le modèle indépendant du temps proposé que nous utilisons réellement dans ce travail tout en conservant la souplesse et le réalisme requis.

7.1.3.1 Table des horaires

Le réseau ferroviaire est basé sur une table des horaires des trains. Une *table des horaires des trains* est définie par un 4-uplet $(\mathcal{C}, \mathcal{S}, \mathcal{Z}, \mathcal{T})$ tel que $\mathcal{S}$ est l'ensemble des gares de trains, $\mathcal{Z}$ est l'ensemble des trains, $\mathcal{T}$ est la périodicité de la table, et $\mathcal{C}$ est l'ensemble des connexions élémentaires. Une *connexion élémentaire* $c \in \mathcal{C}$ est un 5-uplet $c = (z, s_o, s_d, t_s, t_e)$ qui représente un *train* $z \in \mathcal{Z}$ partant de la gare $s_o \in \mathcal{S}$ à l'heure $t_s < \mathcal{T}$ et arrivant à l'aéroport $s_d \in \mathcal{S}$ à l'heure $t_e < \mathcal{T}$ [Pajor, 2009]. Concrètement, une connexion élémentaire correspond à un événement dans la table des horaires. La route d'un train consiste en plusieurs connexions élémentaires de la table des horaires.

Concernant le temps de trajet d'une connexion élémentaire c. Si le temps d'arrivée t_e est plus grand que le temps de départ t_s alors le temps de trajet de c : $\delta(t_s, t_e)$, est égale simplement à la différence $t_e - t_s$. Toutefois, il est possible que le départ soit la nuit et que l'arrivée soit le lendemain, cela est dû à la périodicité de la table horaire. Dans ce cas-là, le temps de trajet est composé du temps de trajet de t_s à minuit plus le temps de trajet de minuit à t_e (voir l'équation 7.1)

$$\delta(t_s, t_e) = \begin{cases} t_e - t_s & \text{si } t_e \geq t_e, \\ T - t_s + t_e & \text{sinon.} \end{cases} \tag{7.1}$$

7.1.3.2 Modèle condensé

L'approche la plus basique pour modéliser les réseaux ferroviaires est le modèle condensé indépendant du temps. Un tel modèle représente uniquement la structure du graphe et omet les horaires. Ainsi, pour chaque gare $s \in \mathcal{S}$ dans la table des horaires des trains, il existe un seul unique nœud $x \in \mathcal{X}$ dans le graphe $G = (\mathcal{X}, \mathcal{U})$ qui représente le réseau de transport. Un arc $u = (x, y) \in \mathcal{U}$ est inséré s'il existe au moins une connexion directe entre deux gares dans la table des horaires qui va de la gare x à la gare y [Pajor, 2009, Kirchler, 2013].

Le modèle indépendant du temps génère un graphe condensé où chaque arc correspond à toutes les connexions agrégées entre une paire de nœuds. Dans ce type de modèle, une gare de train est

représentée par un unique nœud, au lieu de multiples nœuds. Cette approche génère des graphes compacts. Par conséquent, on obtient une borne inférieure sur le temps de trajet entre deux gares. En fait, le modèle condensé donne un aperçu de la structure du système de transport ce qui permet de tester rapidement la connectivité. En revanche, un tel modèle n'est pas utilisé dans le calcul exact du plus court chemin.

7.1.3.3 Modèle temporel étendu

Toujours dans le cadre d'une approche indépendante du temps et pour faire face aux limites du modèle condensé qui ne calcule pas des plus courts chemins exacts, le modèle temporel étendu (en anglais *time-expanded*) a été proposé pour inclure les horaires. Dans le graphe résultant, chaque nœud est associé à un événement de la table des horaires de telle sorte qu'un arc connecte deux événements consécutifs. Un événement représente soit un départ soit une arrivée. Par conséquent, cette approche travaille quand même sur un graphe qui inclut une partie de l'information sur les dépendances temporelles, ce qui permet d'ajouter un degré d'analyse supplémentaire [Fortin et al., 2016]. Dans la littérature, il existe deux versions : version simple et version réaliste. La version simple est une représentation directe de la table des horaires sur le graphe. Toutefois, cette version n'inclut pas des temps de transfert réalistes. En effet, il y a bien un temps de transfert minimum **transfer**(s) pour changer d'un train à un autre dans une gare s [Pajor, 2009]. Pour incorporer cette information, la version réaliste a été développée pour améliorer la version simple du modèle étendu.

Version Simple

Les nœuds dans le graphe ne représentent plus des gares mais plutôt des événements associés à la table des horaires (voir 7.1.3.1). Dans la version simple du modèle étendu, il existe deux types d'événements : événements de départ et événements d'arrivée. Pour chaque connexion élémentaire de la table des horaires $c = (z, s_o, s_d, t_s, t_e)$, il y a un train z qui part de la gare s_o à l'instant t_s et arrive à la gare s_d à l'instant t_e. Ainsi, deux nœuds sont insérés dans le graphe correspondant aux événements de départ et d'arrivée. Pour garder trace de la gare à laquelle un nœud appartient, chaque nœud est affecté à sa gare s ainsi que son horodatage t lorsque l'événement se produit.

Pour les arcs, deux types sont insérés. Un arc est inséré entre les nœuds s_o et s_d avec un poids égal au temps de trajet $\delta(t_s, t_e)$. Par ailleurs, le deuxième type d'arc est inséré entre les nœuds associés à la même station.

Pour autoriser les transferts, les nœuds représentant la même gare sont triés par ordre croissant [Pajor, 2013] (voir la figure 7.1).

Exemple 7.1.1 (Modèle temporel étendu)**.**
La figure 7.1 illustre l'exemple du modèle simple. Les nœuds en couleur verte représentent les départs, les nœuds en couleur jaune représentent les arrivées. La version réaliste est présentée dans la figure 7.2, les nœuds en bleu correspondent aux nœuds de transfert.

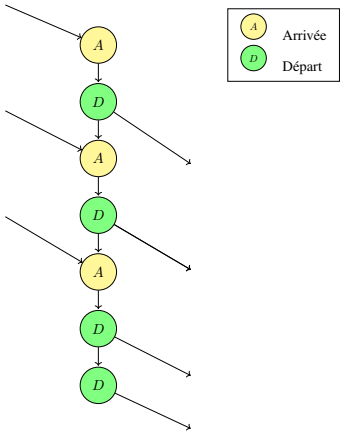


FIGURE 7.1 – Version simple du modèle temporel étendu.

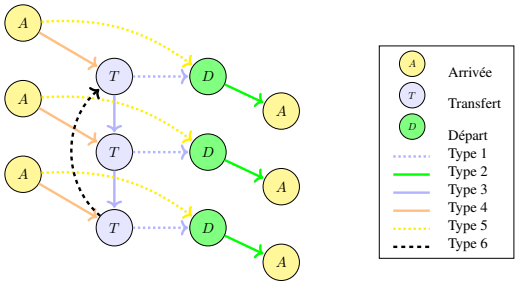


FIGURE 7.2 – Version réaliste du modèle temporel étendu

Version Réaliste

Pour faire face à des transferts réalistes, une fonction de transfert est associée à chaque gare. Cette fonction a pour but d'affecter des temps de transfert à chaque station de train. Ainsi, on distingue trois types de nœuds : nœuds de départ, nœuds d'arrivée et nœuds de transfert. Alors que les nœuds d'arrivée représentent toujours les événements d'arrivée de la table des horaires, les événements de départ sont modélisés par une paire de nœuds qui consiste en un nœud de transfert et un nœud de départ.

Pour les arcs, un nouveau type est ajouté représentant le transfert avec un temps minimum de transfert [Pajor, 2009, Kirchler, 2013]. Ainsi, il y a cinq types d'arcs dans la version réaliste (voir la figure 7.2) :

1. Arcs-Départ [T => D] : Chaque paire de nœuds de transfert et de départ est connectée par un arc de départ à poids 0.
2. Arcs-Connexion [D => A] : Chaque nœud de départ est connecté à son nœud d'arrivée correspondant dans la connexion élémentaire c.
3. Arcs-Gare [T => T] : Pour chaque gare $s \in \mathcal{S}$, ses nœuds de transfert sont triés par ordre croissent (comme dans le cas de la version simple).

4. Arcs-Transfert [A => T] : Ce type d'arc sert à modéliser les transferts. Un nœud d'arrivée est connecté au prochain nœud de transfert. Pour chaque nœud d'arrivée à t_e, on cherche pour le plus petit nœud de transfert, t_{tr}, tel que : $\delta(t_e, t_{tr}) \geq \textbf{transfer}(s)$.
5. Arcs-Train [A => D] : Ce type d'arc modélise le fait qu'un passager puisse rester dans le même train pour poursuivre son trajet.
6. Arcs-Nuit : Ce type d'arc sert à modéliser les transferts pendant la nuit.

7.1.4 Synthèse des modèles de réseaux de transport

Le tableau 7.1 présente un résumé des approches présentées dans la littérature pour la modélisation des réseaux de transport. Pour chaque modèle, nous présentons les propriétés, les avantages et les inconvénients. Le modèle dépendant du temps est présenté sous forme de deux versions comme le cas du modèle temporel étendu. Comme cette approche a été écartée de notre travail du thèse, nous avons donné seulement un résumé de ce modèle (voir la section 7.1.2).

7.1.5 Discussion

Tout d'abord, nous avons commencé par présenter les deux types de modèles : modèle indépendant du temps qui est le plus utilisé pour les réseaux routiers et le modèle dépendant du temps. Dans le cas de ce travail du thèse, nous sommes intéressés par le modèle indépendant du temps qui répond plus à notre question de recherche. Ainsi, nous avons laissé de côté le modèle dépendant du temps.

Dans le cadre d'une approche indépendante du temps qui permet de travailler sur des graphes assez compacts, il y a deux types de modèle :

— Le modèle temporel étendu qui inclut le temps de transfert en représentant les événements de la table des horaires comme des nœuds, ce modèle inclut statiquement les dépendances temporelles mais génère des graphes plus larges.

— Le modèle condensé permet de représenter uniquement la structure du réseau et tester rapidement la connectivité. Ce modèle permet ainsi d'obtenir des bornes inférieures sur le calcul du plus court chemin.

Pour revenir à notre cadre du travail qui ne fait pas le calcul d'itinéraires mais le but est de trouver une opportunité économique qui peut être basée sur les décisions suivantes :

1. augmenter la fréquence ;
2. proposer un nouveau vol ;
3. proposer un vol partagé.

Le travail de thèse ainsi consiste à analyser et modéliser le réseau aérien. Dans le cadre aérien, la plupart des modèles dans la littérature utilisent une approche dépendante du temps.

Pour répondre à notre question, nous avons proposé une nouvelle approche qui est basée sur les modèles présentés avant pour les autres réseaux de transport dans le cas d'une approche indépendante du temps.

Notre approche permet de mieux modéliser notre réseau aérien suivant nos contraintes :

1. représenter la structure du graphe ;
2. modéliser les transferts ;
3. calculer des bornes inférieurs sur les plus courts chemins.

	Indépendant du temps			**Dépendant du temps**	
	Condensé	Étendu			
		Simple	Réaliste	Simple	Réaliste
Propriétés	nœud → aéroport arc → toutes les connexions élémentaires poids → temps de trajet minimum	nœud → événement arc → connexion élémentaire poids → temps de trajet	nœud → événement,transfert arc → connexion élémentaire, transfert poids → temps de trajet, temps de transfert	nœud → aéroport arc → toutes les connexions élémentaires poids → f(temps d'attente, temps de trajet)	nœud → événement,transfert arc → connexion élémentaire poids → f(temps d'attente, temps de trajet)
Avantages	graphe compact structure de graphe test de la connectivité	prise en compte des horaires calcul du plus court chemin	prise en compte des transferts adaptation des plus courts chemins classiques	prise en compte des horaires prise en compte du temps d'attente	calcul exact des plus courts chemins minimum temps de transfert pour changer un train
Inconvénients	absence d'information temporelle	large graphe absence de transfert	heure d'arrivée inconnue	absence de minimum de temps de transfert	graphe trop grand

TABLE 7.1 – Comparaison entre les différentes approches de modélisation des réseaux de transport.

7.2 Modélisation du réseau aérien

Dans cette section, nous présentons notre approche pour modéliser le réseau aérien de la base `optimode`. Tout d'abord, nous commençons par introduire la table des horaires qui est la base du réseau aérien. Ensuite, nous passons à la description du modèle du graphe condensé aérien. Puis, nous décrivons la représentation du modèle ainsi que les métriques du graphe condensé aérien. Finalement, nous présentons la transformation que nous avons proposée pour améliorer notre modèle. C'est une combinaison entre le modèle condensé, qui garde la structure compacte du réseau et la version réaliste du modèle temporel étendu, qui inclut les temps de transfert.

7.2.1 Table des horaires

Comme le réseau ferroviaire, le réseau aérien est basé sur une table des horaires des vols. Une *table des horaires des vols* est définie par un 4-uplet $(\mathcal{C}, \mathcal{A}, \mathcal{F}, \mathcal{T})$ tel que $\mathcal{A}$ est l'ensemble des aéroports, $\mathcal{F}$ est l'ensemble des vols, $\mathcal{T}$ est la périodicité de la table, et $\mathcal{C}$ est l'ensemble des connexions élémentaires. Une *connexion élémentaire* $c \in \mathcal{C}$ est un 5-uplet $c = (f, o, d, t_s, t_e)$ qui représente un *vol* $f \in \mathcal{F}$ partant de l'aéroport $o \in \mathcal{A}$ à l'heure $t_s < \mathcal{T}$ et arrivant à l'aéroport $d \in \mathcal{A}$ à l'heure $t_e < \mathcal{T}$ [Pajor, 2009]. Concrètement, une connexion élémentaire correspond à un événement dans la table des horaires. Un *itinéraire* $(c_1, c_2, \ldots, c_{n-1}, c_n)$ est une séquence de connexions élémentaires, l'origine d'une connexion élémentaire c_i étant la même que la destination de c_{i-1} dans la séquence. Le temps parcouru entre deux connexions successives est au moins égal au temps minimum pour faire la correspondance :

$$o(c_{i+1}) = d(c_i) \wedge t_e(c_i) + MCT(d) \leq t_s(c_{i+1}) \; \forall 1 \leq i \leq n-1$$

Où MCT est le temps minimum pour faire une correspondance à l'aéroport $d(c_i)$. Ce temps dépend de l'aéroport destination de la connexion élémentaire.

7.2.2 Modèle du graphe condensé aérien

Pour modéliser un réseau aérien, qui est habituellement représenté par un graphe orienté, nous avons opté pour une approche basée sur le modèle condensé que nous avons améliorée pour répondre à notre besoin. C'est une approche indépendante du temps qui inclut le temps de transfert.

Pour chaque aéroport $a \in \mathcal{A}$ dans la table des horaires des vols, il existe un unique nœud $x \in \mathcal{X}$ dans le graphe $G = (\mathcal{X}, \mathcal{U})$ qui représente le réseau aérien. Un arc $u = (x, y) \in \mathcal{U}$ est inséré s'il existe au moins un vol direct entre deux aéroports dans la table des horaires qui va de l'aéroport x à l'aéroport y.

Nous présentons plus tard la transformation que nous avons proposée pour inclure le transfert tout en gardant la structure compacte du graphe (voir la section 7.3).

7.2.3 Représentation du modèle

Pour représenter le graphe condensé aérien *CFN* (en anglais *condensed flight network*), nous avons proposé une représentation en fonction des données de la base `optimode`. Les données sont stockées comme propriétés dans les arcs sous la forme d'un dictionnaire qui contient des paires clé-valeur. Un dictionnaire est une structure de données qui associe à chaque clé une valeur. Ainsi, un arc est construit par période. Nous avons proposé le mois de l'année (`year_month`) comme période (voir la figure 7.3).

Algorithmiquement parlant, Les données sont stockées sous la forme de clé-valeur telle que la clé est le mois de l'année, alors que sa valeur est un dictionnaire de clé-valeur pour tous les critères. Cette représentation est assez canonique et plus flexible pour différentes raisons :

— Les données sont stockées mensuellement dans la base `optimode`.

— Pour ajouter les données d'un mois spécifique, il suffit de récupérer la paire de nœuds à mettre à jour. Ensuite, ajouter les données aux propriétés existantes. La même chose est faite pour la suppression.

— Le temps de réponse est plus rapide.

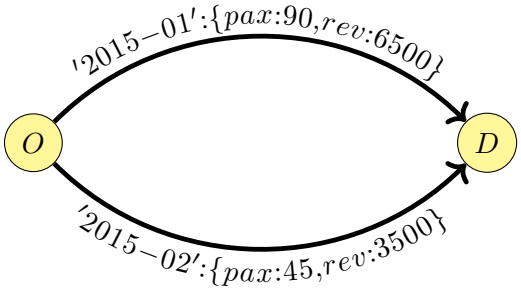


FIGURE 7.3 – Représentation du modèle du graphe condensé aérien. Dans cet exemple, nous présentons deux périodes : `2015-01`, `2015-02` et deux critères : **pax** (nombre des passagers), **rev** (revenu).

7.2.4 Représentation des métriques

Dans le graphe condensé, un arc entre deux nœuds indique qu'il existe au moins un vol entre ces deux aéroports. Chaque arc est construit en agrégeant toutes les connexions élémentaires entre toute paire d'aéroports. Soit $\mathcal{C}_{\wr\lceil} = \{c \in \mathcal{C} \mid o = o(c) \wedge d = d(c)\}$ l'ensemble des connexions élémentaires entre les deux aéroports o & d. Les étiquettes suivantes sont associées à l'arc (o, d) [Idrissi et al., 2017] :

— $F_{od} = |\mathcal{C}_{od}|$ est le nombre de connexions élémentaires entre o et d ;

— $c_{od} = \sum_{c \in \mathcal{C}_{od}} cap(c)$ est la capacité totale en terme de nombre de passagers ;

— $P_{od} = \sum_{c \in \mathcal{C}_{od}} pax(c)$ est le nombre total de passagers ;

— $R_{od} = \sum_{c \in \mathcal{C}_{od}} r(c)$ est le revenu total ;

— $\bar{R}_{od} = \min_{c \in \mathcal{C}_{od}} \frac{r(c)}{pax(c)}$ est le revenu minimum par passager ;

— $t_{od} = \min_{c \in \mathcal{C}_{od}} t(c)$ est la durée minimale du vol ;

— $T_{od} = \max_{c \in \mathcal{C}_{od}} t(c)$ est la durée maximale du vol ;

— D_{od} est la distance entre les deux aéroports ;

— A_{od} est la liste des compagnies aériennes qui opèrent les vols existant entre les deux aéroports ;

— W_{od} est la liste des jours quand le vol est opéré ;

— WF_{od} est le nombre d'opérations hebdomadaires.

7.3 Transformation du graphe condensé

Le graphe représentant le réseau aérien est stocké dans une base de données orientée graphe `Neo4j`. Dans notre contexte, une base de données orientée graphe est plus adaptée à la résolution des problèmes auxquels est confronté *Milanamos* qu'une base de données `NoSQL` (voir la section 5.5.6). On peut en citer quelques-uns :

— Problème de jointure en `MongoDB`.

— Absence de structure de graphe.

— Gestion des données manquantes.

— Problème de visualisation.

Le graphe condensé est un graphe indépendant du temps où les nœuds représentent les aéroports alors que la présence d'un arc entre une paire d'aéroports indique qu'il existe au moins un vol reliant cette paire d'aéroports. Ce modèle omet l'aspect temps pour plus de simplicité. Cela permet de mieux tester la connectivité. Pour inclure le temps de transfert, nous proposons de le représenter par un arc dans le graphe. Cette technique est souvent utilisée pour modéliser l'information sur le transfert puisqu'il est important dans le calcul du plus court chemin [Cherkassky et al., 1996].

7.3.1 Modèle du graphe condensé transformé

Dans cette section, nous allons présenter les deux techniques proposées pour modéliser un réseau aérien réel [Idrissi et al., 2018].

7.3.1.1 Ajout des terminaux

Pour modéliser les correspondances, chaque aéroport $A \in \mathcal{A}$ est représenté par trois nœuds dans le graphe transformé : un *nœud de départ* qui rassemble les vols partant de A et un *nœud d'arrivée* pour modéliser les arrivées et le nœud aéroport A. De cette façon, nous avons modélisé le fait que tous les vols partent ou bien arrivent à un aéroport suivant la table des horaires des vols présentée dans la section 7.2.1. Les arcs sont créés comme suit. Il y a trois types d'arcs au sein de chaque entité aéroport, et nous désignons par le poids d'un arc, la valeur temps :

— arc de type *board_at* est inséré à partir du nœud A au nœud de départ et son poids est mis à 0 ;

— arc de type *alight_at* reliant le nœud d'arrivée au nœud A avec un poids égal à 0 ;

— arc de type *connect_to* pour modéliser le transfert entre le nœud d'arrivée et le nœud de départ de l'aéroport A. Ce type d'arc a un poids fixé à 120. Cette valeur représente le MCT qui est le même pour tous les arcs du graphe G de même type. En moyenne, le temps minimum pour faire le transfert est 120.

La figure 7.4 présente un exemple du graphe condensé transformé. Le graphe contient 4 aéroports (*NCE* : Nice, *BKK* : Bangkok, *PEK* : Pékin, *ICN* : Séoul) qui sont représentés par un trait en double et 4 arcs de type *year_month*. Les nœuds en trait plein représentent les nœuds de départ (origine)et ceux en pointillé sont les nœuds d'arrivée (destination). Les arcs en trait double sont les arcs de transfert et les arcs en pointillé sont les arcs d'arrivée.

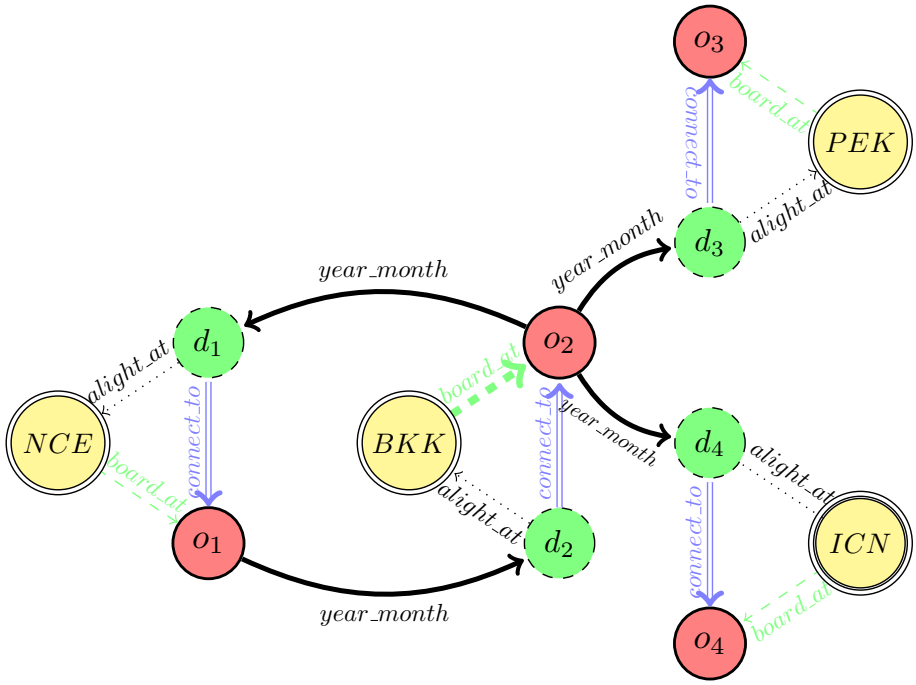


FIGURE 7.4 – Graphe condensé transformé.

7.3.1.2 Connectivité du graphe condensé

Comme nous travaillons sur un réseau de transport aérien réel, il est censé être fortement connexe. Cela veut dire tout simplement que tous les nœuds doivent être accessibles depuis la source. Or, notre graphe a été généré à partir des données manquantes (voir la section 6.3). Pour pallier cela, nous ajoutons une source fictive et des arcs pour connecter tous les nœuds. Ainsi, nous introduisons un nœud de type *connector* pour connecter chaque nœud de départ (*origin*) à l'ensemble des nœuds d'arrivée (*destination*) (voir la figure 7.5). Cela permet d'avoir des résultats ayant plus de sens lors de la mise en place des algorithmes de plus court chemin [Cherkassky et al., 1996].

7.3.2 Taille du graphe condensé transformé

Après la transformation du graphe condensé suivant le modèle décrit dans la section 7.3.1, le graphe transformé contient 13 732 nœuds et 1 148 303 arcs. Nous rappelons que le graphe condensé a été généré avec des données d'une période de 2 ans (24 mois d'historique), l'ordre et la taille du nouveau graphe sont calculés de la façon suivante :

$$\begin{cases} n' = 3 * n + 1 \\ m' = m + 5 * n \end{cases}$$

où n est le nombre de nœuds et m le nombre d'arcs du graphe G.

La figure 7.5 présente le schéma du réseau aérien stocké dans la base de données orientée graphe `Neo4j`. Nous avons choisi un seul mois (`year_month`) pour la présentation. En pratique, le nœud de type *origin* est lié au nœud *destination* par 24 relations où chaque relation représente un mois de l'année considéré (`year_month`).

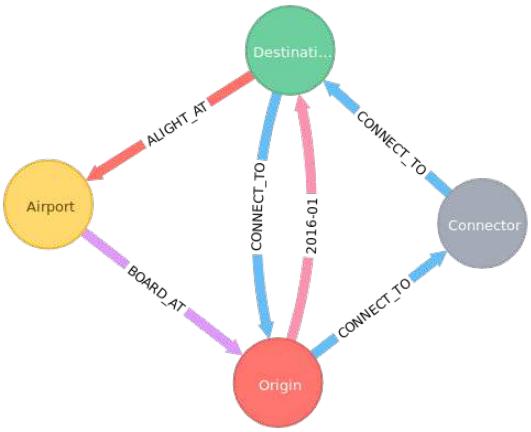


FIGURE 7.5 – Schéma du réseau aérien en `Neo4j`.

7.4 Construction du graphe condensé

Une fois que nous avons modélisé le graphe condensé et déterminé l'ensemble des métriques à inclure dans les arcs du graphe, nous procédons à la construction et au stockage du graphe. La figure 7.7 de la page 172 décrit les différentes étapes du processus. Ces étapes seront présentées plus en détail dans les prochaines sections.

Nous présentons une nouvelle version améliorée du processus décrit par [Idrissi et al., 2017]. En effet, nous avons constaté des données manquantes en analysant le graphe condensé. Nous avons, ainsi procédé à une phase de pré-traitement des données avant la génération du graphe sur la base de données orientée graphe `Neo4j`.

7.4.1 Génération du graphe condensé

Dans cette section, nous commençons par décrire le processus de la génération du graphe condensé. Ensuite, nous présentons l'algorithme implémenté ainsi que l'ensemble des requêtes utilisées sur `MongoDB` pour récupérer les informations.

7.4.1.1 Problèmes rencontrés

Comme on a expliqué dans la section 6.3, il y a des données manquantes lors de l'analyse des données et du graphe condensé (estimation fournie dans le chapitre 6). Voici une liste exhaustive des divers problèmes constatés :

1. absence d'informations pour certains aéroports : coordonnées géographiques, région ;
2. certains aéroports n'ont pas de segments de vols : absence de trafic ;
3. des segments de vols ont les aéroports n'ayant pas d'informations sur les coordonnées géographiques ;
4. l'information sur la distance manque sur certains segments de vol ;
5. une partie des segments n'ont pas de données sur les horaires de vols ;
6. une partie des segments de la collection `schedule` n'ont pas d'informations sur le trafic ;

7. une partie des segment n'ont pas d'informations sur la durée.

Suivant cette analyse, nous avons procédé à l'estimation des données manquantes concernant la distance et la durée. L'opération d'estimation nécessite l'information sur les coordonnées géographiques. Nous avons ainsi décidé de régler ces problèmes suivant ce protocole :

1. Ne garder que les aéroports dont ils disposent de coordonnées géographiques ;
2. Se baser sur la collection `Segment` pour générer la structure du graphe ;
3. Filtrer les segments de vols n'ayant pas de distance et de durée et dont les aéroports n'ont pas de coordonnées géographiques ;
4. Estimer la distance et la durée pour les segments de vols dont les aéroports ont des coordonnées géographiques ;
5. Filtrer les segments de vols n'ayant pas d'information sur le revenu (nécessaire pour nos algorithmes, voir le chapitre 8).

Nous avons décidé que le graphe se baserait sur la collection `segment` pour les raisons suivantes :

— `segment` représente les segments de vols réalisés par un avion.

— `schedule` représente plutôt la partie marketing, c'est-à-dire, le programme du vol annoncé mais qui n'est pas forcément réalisé.

— `schedule` stocke le même vol mais sous un format différent, notamment, dans le cas du partage de code ou bien dans le cas d'un *vol direct* (voir la définition 3.1.6).

La figure 7.6 illustre la représentation d'un vol direct dans les deux collections. Comme le vol *direct* est opéré par le même avion et que les segments de vol concernent les routes des avions (voir la section 6.1.2.2), alors il est représenté par une seule entrée : `SegmentNCE-BKK`. Alors que dans `schedule`, il est présent dans deux entrées : `vol123` et `vol456`. C'est la raison pour laquelle nous nous sommes basés sur la collection `segment` qui informe réellement si l'avion a volé et donc que le vol a été réalisé.

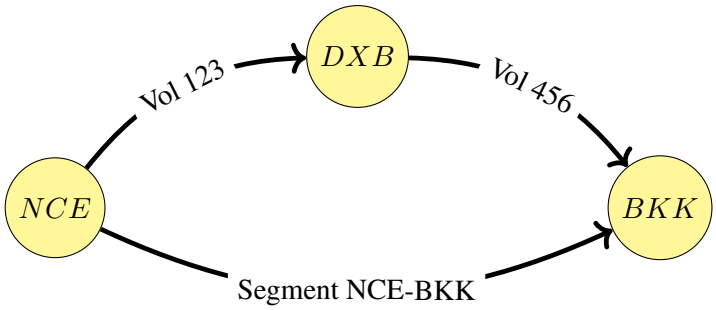


FIGURE 7.6 – Représentation d'un vol *direct* dans `segment` et `schedule`.

7.4.1.2 Description du processus

Le graphe condensé a été généré à partir de la base de données `optimode`. Le processus de la génération s'est fait en trois étapes principales :

1. récupération des données de la base de données `optimode` ;
2. traitement des données ;
3. création du graphe sur la base de données orientée graphe `Neo4j`.

Étape 1

La première étape consiste à récupérer l'ensemble des informations sur les vols stockées dans les deux collections `segment` et `schedule`. Les étiquettes du graphe condensé sont calculées à partir de ces informations. Ensuite, nous agrégeons les données suivant les trois champs : origine (O), destination (D) et M (`year_month`) (voir l'étape *Aggregation (O,D,M) de la figure 7.7*).

Étape 2

Comme la base de données `MongoDB` ne dispose pas d'opération effectuant la jointure, la deuxième étape consiste à fusionner les données agrégées. Nous utilisons alors le langage *Python* grâce au pilote *pymongo*. Après la jointure, on estime les durées qui manquent.

Étape 3

L'étape finale consiste à créer ces données sur le graphe de `Neo4j` en utilisant le pilote *py2neo*. Ce pilote dispose de l'ensemble des fonctions pour créer des nœuds, insérer des données, *etc*. Ainsi, pour générer les nœuds, nous nous sommes basés sur la collection `airport` pour récupérer toutes les informations concernant l'entité aéroport, requête du listing 7.1.

```
1 db.airport.findOne(
2 {"code_type": "airport", "lonlat": {"$ne": None}, "code": code},
3 {"lonlat": 1, "name": 1, "code": 1, "city": 1, "country": 1, "region": 1, "_id
    ": 0})
```

Listing 7.1 – requête pour récupérer les aéroports

La requête du listing 7.1 renvoie toutes les informations sur le nom, le code, la ville, le pays, la région et les coordonnées géographiques d'un aéroport (valeur 1 pour renvoyer uniquement les champs demandés). Comme on a expliqué dans la section 7.4.1.1, certains aéroports n'ont pas de coordonnées géographiques. Pour les exclure de la recherche, nous utilisons l'opérateur de `MongoDB` `$ne` qui exclue les aéroports ayant des coordonnées non nulles ou qui n'existent pas.

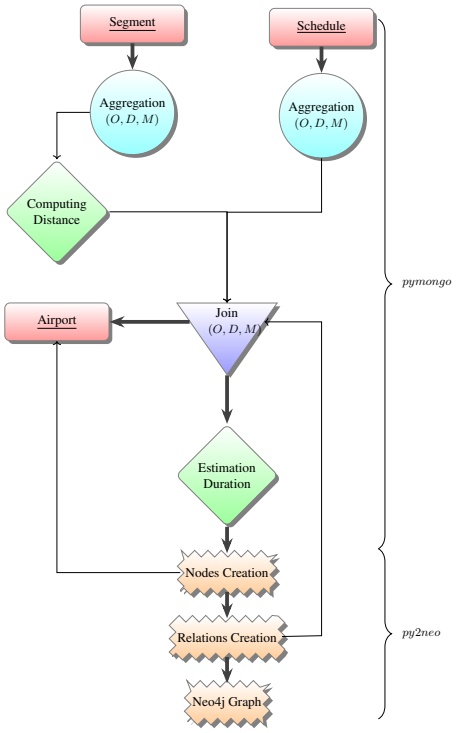


FIGURE 7.7 – Processus de génération et stockage du graphe condensé.

7.4.1.3 Algorithme de génération

Le code du listing 4 présente l'algorithme pour la génération du graphe condensé. La période T est de 24 mois (`year_month`). Pour chaque mois ym de la période T (ligne 2 de l'algorithme du listing 4), nous répétons le même processus décrit dans la section 7.4.1.2.

Algorithme 4 : Génération du graphe condensé

input : *segment* collection, *airport* collection, *schedule* collection, T period
output : A digraph $G = (\mathcal{X}, \mathcal{U}, \mathcal{W})$

```
1 G ← G.create_constraints(code);
2 foreach ym ∈ T do
3     request_seg ← segment.aggregate(ym);
4     request_sch ← schedule.aggregate(ym);
5     (data_join, airport_dict) ← segment_join(request_seg, request_sch);
6     data_join ← compute_missing_duration(data_join);
7     nodes ← create_nodes(data_join);
8     G ← create_arcs(G, nodes, data_join, ym);
9 end
```

Pour avoir les informations mensuelles sur le trafic pour chaque paire d'aéroports origine-destination, à savoir, le nombre de passagers (`pax`), le revenu minimum par passager, le revenu, la distance, la liste des compagnies aériennes, nous faisons une agrégation sur la collection `segment`. Ensuite, pour construire les arcs, nous sommes partis de la collection `segment` (ligne 3 de l'algorithme 4). Puis, pour compléter les données relatives au planning de vol, notamment, la

capacité, la fréquence, la durée et le jour de l'opération du vol, nous procédons à une agrégation sur la collection `schedule` (ligne 4 de l'algorithme 4). Les données sont agrégées suivant le mois de l'année (`year_month`). La fonction `segment_join` consiste à fusionner les résultats des deux agrégations (ligne 5 de l'algorithme 4). Concernant la distance qui manque, nous faisons un calcul lors de la jointure.

Opération de jointure

On part du principe que le graphe est construit à partir de l'ensemble de segments de la collection `segment` dont les aéroports sont connus par notre système (la collection `airport`). La fonction `segment` parcourt le résultat de l'agrégation sous forme d'un curseur et cherche la correspondance sur la paire d'aéroports origine-destination, qui est la clé de la jointure, avec le résultat de l'agrégation de `schedule`. Comme c'est une jointure à gauche, on souhaite récupérer toutes les informations de la collection gauche (`segment`) et compléter par les informations sur les horaires de vols. La recherche se fait sur les ensembles triés par un ordre lexicographique sur les codes des paires d'aéroports origine-destination. On commence par l'origine, si on trouve la correspondance, on passe alors à la destination et on refait la même procédure. Les cas de figures rencontrés sont :

— Cas 1 : l'origine n'est pas encore trouvé dans `schedule`, on ajoute le segment.

— Cas 2 : on n'a pas trouvé l'origine dans `schedule`, on ajoute le segment et on avance sur `schedule`.

Comme la distance est absente sur certains segments de vols, elle est calculée au cours de la jointure (opération *Computing Distance* dans la figure 7.7). Pour cela, nous récupérons en même temps les informations relatives aux aéroports de segment de vol, notamment les coordonnées géographiques qui serviront à estimer la durée manquante. Pour estimer celle-ci, nous utilisons un algorithme basé sur l'apprentissage automatique (ligne 6 de l'algorithme 4).

Agrégation des données en `MongoDB`

La requête du listing 7.2 présente la requête d'agrégation qui est une requête paramétrée, appelée pour chaque mois de la période considérée T. La ligne 2 consiste à chercher les documents correspondant au mois demandé. Ensuite, on fait une projection pour sélectionner les champs qui nous intéressent et calculer le revenu moyen par passager qui sert par la suite à calculer le revenu minimum par passager $\bar{R}_{od}$. En effet, pour chaque connexion élémentaire représentée par un document dans la collection `segment`, on calcule le revenu moyen par passager `average_revenue`. Ensuite, nous effectuons un `group` pour regrouper les données par paire origine-destination. Cela permettra d'agréger tous les segments existants entre une paire d'aéroports origine-destination.

Comme pour certains segments, on a le nombre de passagers à 0 : on estime que le revenu moyen est nul pour ces segments (ligne 12 de la requête du listing 7.2).

On ne garde, dans le résultat, que les segments dont le nombre de passagers est supérieur à 10 et bien évidemment ayant des revenus non nuls (ligne 26 de la requête du listing 7.2). C'est le seuil qu'on avait fixé puisqu'on avait estimé que de tels segments correspondaient plutôt à une faute de frappe (voir la section 6.3.2).

Une méthode d'agrégation contient plusieurs étapes. `MongoDB` écrit dans des fichiers temporaires sur le disque. La mémoire limite est de 100Mo. Quand cela dépasse, c'est le cas notamment

avec des grandes données comme la base `optimode`, il faut passer alors le paramètre *allowDiskUse* pour donner la permission à `MongoDB` d'allouer plus de mémoire sur le disque, à la méthode (ligne 28 de la requête du listing 7.2).

```
1  pipe = [
2        {'$match': {'record_ok': True, 'year_month': ym}},
3        {'$project': {
4              'leg_origin': '$leg_origin',
5              'leg_destination': '$leg_destination',
6              'segment_revenue_usd': '$segment_revenue_usd',
7              'passengers': '$passengers',
8              'distance': '$distance',
9              'operating_airline': '$operating_airline',
10             'average_revenue': {
11                   '$cond': [{'$eq': ['$passengers', 0]}, 0,
12                   {'$divide': ['$segment_revenue_usd', '$passengers']}]}
13       }},
14       {'$match':{'average_revenue': {'$ne': 0}}},
15       {'$group': {
16             '_id': {
17                   'origin': '$leg_origin',
18                   'destination': '$leg_destination'
19       },
20             'revenue': {'$sum': '$segment_revenue_usd'},
21             'pax': {'$sum': '$passengers'},
22             'distance': {'$first': '$distance'},
23             'airlines': {'$addToSet': '$operating_airline'},
24             'revenue_min': {'$min': '$average_revenue'}
25       }},
26       {'$match': {'pax': {'$gte': 10}, 'revenue_min':{'$ne':0}}},
27       {'$sort': SON([('_id.origin', 1), ('_id.destination', 1)])}]
28 cursor = SegmentInitialData.aggregate(pipe, allowDiskUse=True)
```

Listing 7.2 – requête pour l'agrégation de la collection `segment`

La requête du listing 7.3 présente la requête d'agrégation des données de la collection `schedule` pour compléter les données relatives au planning de vol, C'est étape correspond à la ligne 4 de l'algorithme de génération 4. Les données sont agrégées suivant le mois de l'année (`year_month`).

```
1   pipe = [
2         {'$match': {'record_ok': True, 'active_rec': True,
3                           'code_share': False, 'year_month': ym}},
4         {'$group': {
5               '_id': {
6               'origin': '$origin',
7               'destination': '$destination'
8         },
9               'frequency': {'$sum': '$departure_count'},
10              'capacity': {'$sum': '$total_capacity'},
11              'duration_min': {'$min': '$flight_duration'},
12              'duration_max': {'$max': '$flight_duration'},
13              'week_days': {'$first': '$week_days'}
14        }},
15        {'$sort': SON([('_id.origin', 1), ('_id.destination', 1)])}]
16  cursor = ScheduleInitialData.aggregate(pipe, allowDiskUse=True)
```

Listing 7.3 – requête pour l'agrégation de la collection `schedule`.

Le tableau 7.2 résume les champs récupérés par les requêtes et insérés par la suite comme des étiquettes du graphe condensé. Les couleurs des étiquettes représentent le type de traitement et la source de chaque donnée. Ainsi, on distingue 4 couleurs :

— Couleur `rouge` : donnée de la collection `segment`.

— Couleur `gris` : donnée de la collection `schedule`.

— Couleur `vert` : donnée de la collection `schedule` estimée.

— Couleur `jaune` : donnée calculée à partir d'une autre donnée.

étiquette *CFN*	champ	manque	formule
P_{od}	pax	oui	
R_{od}	revenue	non	
A_{od}	airlines	oui	
F_{od}	frequency	non	
c_{od}	capacity	non	
W_{od}	week_days	non	
t_{od}	duration_min	oui	Forêts aléatoires [Gachoki and Muraya, 2019]
T_{od}	duration_max	oui	Forêts aléatoires
D_{od}	distance	oui	formule de haversine [Robusto, 1957]
WF_{od}	week_frequency	non	cardinal de l'ensemble `week_days`
$\bar{R}_{od}$	revenue_min	oui	moyenne arithmétique

TABLE 7.2 – Correspondance des étiquettes du *CFN* avec les métriques des requêtes 7.2 et 7.3.

7.4.2 Stockage du graphe condensé

En `MongoDB`, il n'y a pas de structure de graphe. En effet, ajouter des nouvelles données mensuelles consiste à mettre à jour toutes les collections impactées par ces données. Pour ce faire, il faut passer par un script en *Python* comme nous l'avions fait pour générer le graphe condensé afin de chercher des données dans une autre collection. D'où la nécessité de trouver une structure qui regroupe les données, stocke et visualise le graphe (voir la section 5.4.5).

Nous avons proposé la base de données `Neo4j` comme une deuxième alternative pour faire face à ces limites. Nous avons déjà présenté la base de données orientée graphe d'une façon plus détaillée dans le chapitre 5. Pour mettre en place cette nouvelle base, nous avons commencé par réaliser certains tests pour voir comment elle se comportait. L'ensemble des tests sont les suivants :

1. le calcul d'une métrique pour un mois absent ;
2. l'évolution temporelle des données envoyées à la base sur un certain horizon.

Pour le premier test, nous avons testé le calcul d'une somme pour un ensemble de périodes dont une période n'existe pas. On rappelle que la période est le *year_month*. On supposera que

```
1 MATCH (n:aeroport)-[r]->()
2 WHERE type(r) IN ['2015-01', '2015-02', '2015-05']
3 RETURN SUM(r.cap) AS sum
```

Listing 7.4 – requête pour le test 1

nous avions deux périodes (`2015-01` et `2015-02`) et que nous souhaitions calculer la capacité totale (voir la requête du listing 7.4).

La requête du listing 7.4 ne génère aucun blocage. En effet, `Neo4j` commence par filtrer le graphe et ensuite faire le calcul. Il est à noter que `Nul` signifie que « la propriété n'existe pas ». Donc, `Neo4j` ne crée pas une propriété qui a une valeur nulle. On présentera le deuxième test dans la section suivante.

7.4.3 Test des performances de `Neo4j`

Le deuxième test a pour but de mesurer les performances de cette base de données. Ce test se compose de deux tests :

1. tester les performances sur l'horizon d'une année ;
2. refaire le premier sur un horizon de cinq ans.

Test 1

Le test consiste à intégrer les étiquettes calculées mois par mois dans le *CFN*. Nous avons commencé par les données `2015` puisqu'à l'époque, c'était les données les plus récentes. Le graphe généré contient 11 300 nœuds et 604 280 arcs. Ce graphe a été généré en 2 heures environ (voir le tableau 7.3).

nœuds	relations	T_n	T_{tot}
11 300	604 280	176,65	6 847,74

TABLE 7.3 – Statistiques sur la période `2015`. T_n représente le temps de création des nœuds, T_{tot} est le temps du processus total en secondes.

Le tableau 7.4 représente les informations détaillées pour chaque mois des deux opérations du processus de la mise en place du graphe condensé : l'agrégation et la création. Cette dernière opération concerne le stockage du graphe sur `Neo4j`, alors que l'opération d'agrégation concerne plutôt la base de données `MongoDB`. Les informations concernent le nombre de relations générées à partir du nombre de documents ainsi que le temps d'exécution de chaque opération (T_{ag} (respectivement T_{cr}) pour l'agrégation (respectivement la création)).

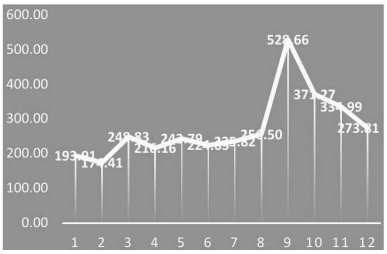
Pour mieux analyser les données du tableau 7.4, nous les avons présentées graphiquement. On peut ainsi évaluer le comportement du temps par rapport au volume des données (voir les figures 7.8).

On remarque que l'agrégation prend deux fois plus de temps que la création des arcs. Les courbes des deux étapes ont la même tendance que celle de nombre de relations. Le pic est atteint

year_month	T_{ag}	T_{cr}	relations	documents
2015-01	193,91	263,82	47183	9006082
2015-02	174,41	268,47	47421	8336266
2015-03	248,83	274,95	49052	9326464
2015-04	216,16	265,48	49379	8935626
2015-05	243,79	280,06	50767	9481010
2015-06	224,65	288,90	52845	9766482
2015-07	235,82	289,83	53594	9756868
2015-08	256,50	300,89	53610	18846342
2015-09	528,66	290,75	51809	10089961
2015-10	371,27	286,63	51409	10925629
2015-11	334,99	277,31	48438	9846764
2015-12	273,81	281,20	48773	9854326

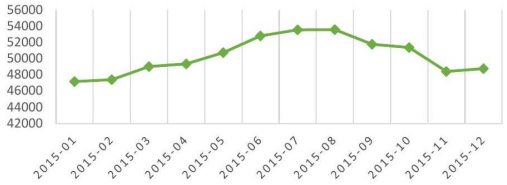
TABLE 7.4 – Statistiques de la génération du graphe condensé pour l'année 2015.

au mois 9 de l'année 2015. C'est le mois qui coûte cher pour faire l'agrégation selon le tableau 7.4.



(a) Courbe de temps d'agrégation des documents.

(b) Courbe de temps de création des arcs.



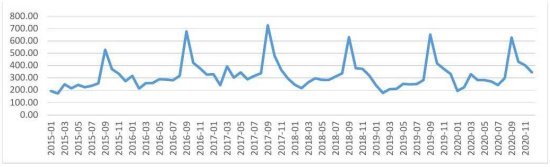
(c) Évolution mensuelle de nombre de relations en Neo4j.

FIGURE 7.8 – Résultats des calculs des performances.

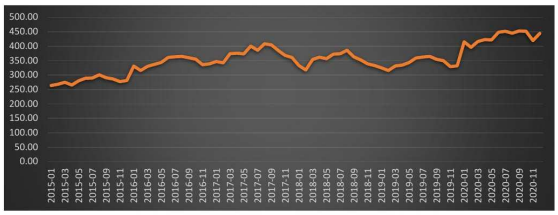
Test 2

L'horizon choisi pour ce test est de 5 ans à partir de l'année 2015. Comme le temps est sensible aux flux d'arrivée, cette mesure a une tendance linéaire : le temps augmente quand le volume de

données reçu est important. C'est tout à fait le comportement normal que nous devions constater. Le graphique de la figure 7.9b illustre l'évolution mensuelle du temps pour la création des arcs en `Neo4j`. La courbe a tendance à croître pour se stabiliser les six derniers mois de l'année 2020. Ce qui n'est pas le cas pour la courbe de l'agrégation des données en `MongoDB` (voir la Figure 7.9a) qui commence à se stabiliser vers la moitié de la période considérée (`2017-09`). Cela peut s'expliquer par le fait qu'à partir de ce stade, les opérations dédiées au stockage des données sur la base s'exécutent.



(a) Agrégation des document.



(b) Création des arcs.

FIGURE 7.9 – Résultats du deuxième test.

7.5 Analyse structurelle du réseau aérien

Dans cette section, nous analysons la structure du graphe condensé qui représente le réseau aérien. Cette analyse étudie un réseau réel et s'inscrit dans la théorie des graphes. Elle permet d'étudier le comportement du graphe à travers le temps ainsi que détecter les anomalies liées à un manque et/ou une erreur des données. Les propriétés de graphe permettent de faire cette analyse pour adapter les algorithmes à proposer afin de résoudre les problèmes sur ce graphe.

Selon ces propriétés, nous étudions quatre métriques qui nous aident à mener cette analyse : connectivité, densité, distribution des degrés et excentricité. Tout d'abord, nous commençons par étudier la connectivité de notre graphe qui représente l'élément principal de l'étude. En effet, la plupart de ces métriques se basent sur la connectivité. Nous présentons également une comparaison avec l'ancienne version du graphe.

Nous avons ainsi généré le graphe condensé sur une période de deux ans : *2016* et *2017*. Le graphe contient 4 602 nœuds et 1 127 558 relations occupant 710 Mo sur la machine.

7.5.1 Évolution des données

En 2019, *Milanamos* a intégré complètement les données fournies par la compagnie *Sabre*. Cette compagnie se base sur son système de réservation des billets électroniques comme la compagnie *Amadeus*. Nous avons regénéré ainsi une deuxième version du graphe condensé à partir de ces nouvelles données pour la même période que la première version. Le graphe de la première version contenait 4 577 nœuds et 1 125 418 relations (voir le tableau 7.5).

Taille \ Version	1	2
nœuds	4 577	4 602
arcs	1 125 418	1 127 558

TABLE 7.5 – Versions du *CFN*.

7.5.2 Vérification des données manquantes

7.5.2.1 Vérification des données du graphe condensé

Nous avons commencé par vérifier si les données manquées ont été bien traitées lors de l'étape 2 du processus de la génération du graphe condensé (voir la section 7.4.1.1). Ainsi, quatre points sont à vérifier :

— nœuds isolés ;

— coordonnées géographiques ;

— distance et durée ;

— détection des cycles.

```
1  //isolated node
2
3  MATCH(a:Airport) WHERE SIZE((a)--()) = 0 RETURN COUNT(a);
4
5  //coordinates
6
7  MATCH(a:Airport) WHERE NOT EXISTS(a.lon) OR NOT EXISTS(a.lat) RETURN COUNT(a);
8
9  //metrics
10
11 MATCH()-[r]-() WHERE NOT EXISTS(r.distance) AND NOT EXISTS(r.duration_min) AND
       NOT EXISTS(r.distance) RETURN COUNT(r);
12
13 MATCH()-[r]-() WHERE NOT EXISTS(r.distance) OR NOT EXISTS(r.duration_min) OR
       NOT EXISTS(r.distance) RETURN COUNT(r);
14
15 //cycle
16
17 MATCH (a:Airport)-[r]->(a:Airport) RETURN COUNT(r);
```

Listing 7.5 – script de vérification du *CFN* en *Cypher*.

Les requêtes renvoient des résultats nuls, ce qui montre que notre graphe respecte bien les quatre points.

Comparaison avec l'ancienne version

Dans l'ancienne version du graphe condensé, on gardait les aéroports sans données sur le trafic et les horaires, l'idée était de garder ces aéroports pour le jour où on reçoit des nouvelles données. Le graphe, ainsi, comportait des nœuds isolés. La requête du listing 7.6 renvoie le nombre de nœuds isolés, c'est-à-dire le nombre d'aéroports dont aucune information sur le mois est renseignée.

```
MATCH (a:Airport) WHERE SIZE ((a)--())=0
RETURN COUNT(a) AS totNodes
```

Listing 7.6 – requête pour le calcul des nœuds isolés.

Nous avons trouvé 6 862 nœuds isolés sur 11 439 nœuds. Cela veut dire qu'environ 70% des nœuds ont un degré nul. Ces nœuds isolés correspondent à des aéroports pour lesquels *Milanamos* n'a pas de données sur leurs vols en terme de données de trafic et des horaires de vols. Cela représente un grand volume par rapport aux aéroports existants. Cela est dû au fait que nous construisons le graphe en nous basant sur la collection `airport`.

Suivant ces analyses, nous avons procédé à une transformation consistant à supprimer les nœuds isolés (voir la requête du listing 7.7).

```
MATCH (a:Airport)
WHERE SIZE ((a)--())=0
DETACH DELETE a
```

Listing 7.7 – requête pour la suppression des nœuds.

Nous avons également détecté des cycles, provenant de la collection `schedule`. Nous avons estimé 289 cycles 1 125 707 relations, qui représentent des *vols* circulaires.

7.5.2.2 Estimation des arcs manquants

Comme il y a un manque des données, nous avons estimé le nombre de vols sans retour. Ceci est traduit par l'absence des arcs dans l'autre direction. La requête du listing 7.8 estime le nombre d'arcs manquants pour tout le graphe. Nous avons, ainsi, 68 886 parmi 1 127 558 arcs n'ayant pas d'arcs retours (7% des arcs du graphe condensé).

```
MATCH (n:Airport)-[r]-(m:Airport)
WITH n AS o, m AS d, type(r) AS relType, 1 * CASE WHEN startNode(r) = m THEN -1
     ELSE 1 END AS dir
WITH o, d, relType, SUM(dir) AS balance
WHERE balance = -1
RETURN DISTINCT COUNT(relType) AS missingArcs
```

Listing 7.8 – requête pour l'estimation des arcs manquants.

La requête du listing 7.9 présente un exemple d'estimation pour une paire d'aéroport comme *Nice* (`NCE`) et *New York* (`JFK`). Cette requête renvoie que 3 arcs de retours manquent, ce qui veut dire 3 mois de données sur les vols.

7.5.2.3 Estimation des propriétés manquantes

Trois propriétés nous intéressent dans le calcul de nos algorithmes : la distance, la durée et le revenu minimum. Après avoir estimé et calculé la distance et la durée manquantes lors de la génération du graphe condensé (voir la section 7.4.1), nous estimons le revenu minimum manquant (voir la requête du listing 7.10).

On a moins de 1% des revenus manquants dans l'ensemble des arcs, ce qui reste assez négligeable. Contrairement pour l'ancienne version, on avait 4% des revenus manquants. De plus, la

```
MATCH p=(n:Airport{code:'NCE'})-[r]-(m:Airport{code:'JFK'})
WITH SUM(SIZE((n:Airport{code:'NCE'})-[r]-(m:Airport{code:'JFK'}))) AS balance,
     type(r) AS relType
WHERE SUM <2
RETURN COUNT(relType) AS missingArcs
```

Listing 7.9 – requête pour l'estimation des arcs manquants pour une paire de nœuds.

```
MATCH (n:Airport)-[r]->(m:Airport)
WHERE NOT EXISTS(r.revenue_min)
RETURN COUNT(r) AS missRevProp
```

Listing 7.10 – requête pour l'estimation des arcs sans la propriété revenu minmum.

distance et la durée non estimées pour certains nœuds comme leurs coordonnées géographiques sont absentes (voir la section 6.3.3). Nous avions proposé une distance et une durée qui sont plus élevées pour les arcs liant ces nœuds. Ceci n'avait pas pu impacter nos algorithmes.

7.5.3 Connectivité

La *connectivité* permet d'étudier comment un réseau est connecté, ce qui joue un rôle extrêmement important pour analyser et interpréter les différents résultats. Nous avons commencé par vérifier si le graphe condensé comportait une seule composante connexe.

7.5.3.1 Calcul des composantes connexes

Nous utilisons des algorithmes de graphe implémentés dans la base de données orientée graphe `Neo4j` et appelés depuis le langage `Cypher` [NeoTechnology, 2018, Allen et al., 2019]. Le calcul se fait pour un mois donné, nous avons choisi le mois `2016-01` pour faire le calcul, le graphe *CFN* représentant les données mensuelles.

La requête du listing 7.11 présente le calcul des composantes connexes dans le graphe non orienté, l'algorithme utilisé est *Union Find*. Les paramètres `loadMillis`, `computeMillis` et `writeMillis` représentent respectivement le temps de chargement, le temps de calcul et le temps d'écriture en millisecondes.

```
CALL algo.unionFind('Airport', '2016-01', {write:true, partitionProperty:"
    partition"})
YIELD nodes, setCount, loadMillis, computeMillis, writeMillis;
```

Listing 7.11 – requête pour le calcul des composantes connexes.

Suite à ces analyses, notre graphe est censé être non connexe. Le graphe condensé est, ainsi, composé de 893 composantes connexes (voir le tableau 7.6).

nodes	setCount
4 602	893

TABLE 7.6 – Composantes connexes du graphe condensé.

```
MATCH (a:Airport)
RETURN a.partition AS compC,COUNT(*) AS taille
ORDER BY taille DESC
LIMIT 20;
```

Listing 7.12 – requête pour le nombre et la taille des composantes connexes.

La requête du listing 7.12 nous donne plus de détails sur les composantes connexes en termes du nombre et de taille. La grande composante contient 3 680 aéroports (80% d'aéroports) contre 880 composantes qui correspondent à des nœuds isolés (20% d'aéroports) : des aéroports n'ayant pas de données pour ce mois-ci (voir le tableau 7.7). Ce résultat est cohérent puisque ça correspond à la requête du listing 7.13 qui calcule le nombre des nœuds isolés pour le mois `2016-01`.

CC	taille
1	3 680
2	9
4	3
6	2
880	1
893	4 602

TABLE 7.7 – Statistiques des composantes connexes. CC est le nombre des composantes connexes.

```
MATCH (n:Airport)
WHERE SIZE ((n)-[:`2016-01`]-())=0
RETURN COUNT(n)
```

Listing 7.13 – requête pour le calcul des nœuds isolés pour le mois de janvier de 2016.

7.5.3.2 Calcul des composantes fortement connexes

On refait le même calcul en tenant compte de l'orientation du graphe. Les résultats montrent encore que le graphe n'est pas fortement connexe. On a 1 033 composantes fortement connexes dont la plus petite contient un seul nœud (aéroport) et la plus grande composante dominante est formée de 3 551 nœuds (voir le tableau 7.8). Avec les nouvelles données, les composantes fortement connexes sont passées de 962 à 1 033. Ce qui veut dire qu'on a plus d'aéroports avec plus de données.

```
CALL algo.scc('Airport','2016-01', {write:true,partitionProperty:'partition'})
YIELD loadMillis, computeMillis, writeMillis, setCount, maxSetSize, minSetSize;
```

Listing 7.14 – requête pour le calcul des composantes fortement connexes.

setCount	minSetSize	maxSetSize
1 033	1	3 551

TABLE 7.8 – Composantes fortement connexes.

On reprend la même requête du listing 7.12 pour obtenir plus de détails sur les composantes fortement connexes. Nous avons, ainsi, la composante dominante qui contient 3 551 nœuds contre 1 024 composantes contenant un seul nœud (voir le tableau 7.7). Comme on sait qu'il y a 880 nœuds isolés, on constate alors qu'on a 144 aéroports qui correspondent à des nœuds non accessibles (5% des aéroports accessibles). Parmi ces nœuds non accessibles, il y a 76 nœuds dits *sources* et 62 nœuds dits *puits*. Les requêtes du listing 7.15 et 7.16) confirment cette analyse. Cela veut dire que l'avion ne peut partir depuis ces aéroports ou bien qu'il ne peut pas les rejoindre.

SCC	taille
1	3 551
1	9
1	4
2	3
4	2
1 024	1
1 033	4 602

TABLE 7.9 – Statistiques des composantes fortement connexes. SCC est le nombre des composantes fortement connexes.

```
1 MATCH (n:Airport)
2 WHERE SIZE (()-[:`2016-01`]->(n))=0 AND SIZE((n)-[:`2016-01`]->())>0
3 RETURN COUNT(n) AS nbrSource
```

Listing 7.15 – requête pour calculer le nombre des nœuds *sources* pour le mois janvier 2016

```
1 MATCH (n:Airport)
2 WHERE SIZE (()-[:`2016-01`]->(n))>0 AND SIZE((n)-[:`2016-01`]->())=0
3 RETURN COUNT(n) AS nbrSink
```

Listing 7.16 – requête pour calculer le nombre des nœuds *puits* pour le mois de janvier 2016.

Pour avoir plus de détails sur les régions où il y a un manque de données, nous avons utilisé la requête du listing 7.17 qui renvoie le nombre d'aéroports sans données pour le mois `2016-01` par région.

La figure 7.10 illustre le graphique de la répartition des aéroports par région. Une grande partie des aéroports fait partie de la région d'Amérique nord et de l'Europe. Ceci coïncide avec les résultats obtenus pour l'analyse des coordonnées géographiques (voir la section 6.3.3). En Europe, le résultat peut être expliqué par la présence des aéroports de type *offLine* (57%). En Amérique

```
MATCH (a:Airport)
WITH a.partition AS partition,count(*) AS size_of_partition, COLLECT(a) AS
    airports
WHERE size_of_partition = 1
UNWIND airports AS a
RETURN count(a) AS nbrAirReg, a.region
ORDER BY nbrAirReg DESC;
```

Listing 7.17 – requête pour le calcul des composantes fortement connexes pour une période d'un an.

du nord, il y a des faux aéroports c'est-à-dire qui ne sont pas des aéroports en réalité C'est-à-dire qu'il n'y pas des vols commerciaux partants de ces vols.

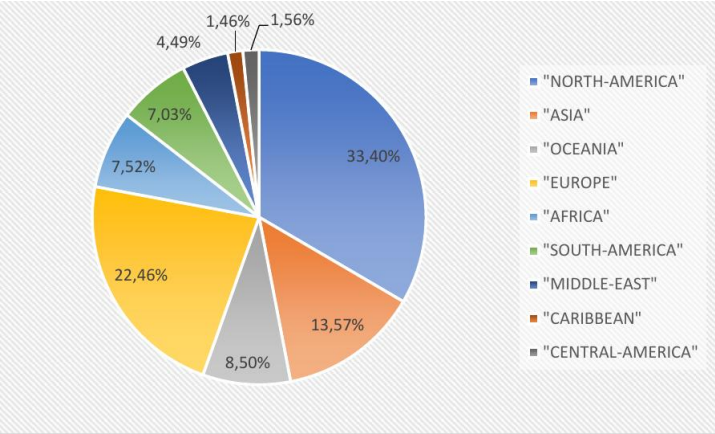


FIGURE 7.10 – Répartition des régions des nœuds isolés.

Suite à ces résultats, nous avons analysé la connexité du graphe pour une période de 12 mois. Le nombre des composantes connexes est censé diminuer (voir la requête du listing 7.18).

```
CALL algo.scc('MATCH (a:Airport) RETURN id(a) AS id ',
'MATCH (n:Airport)-[:`2016-01`|:`2016-02`|:`2016-03`|:`2016-04`|:`2016-05`
|:`2016-06`|:`2016-07`|:`2016-08`|:`2016-09`|:`2016-10`|:`2016-11`
|:`2016-12`]->(m:Airport) RETURN id(n) AS source, id(m) AS target',
{graph:'cypher',write:true, partitionProperty:"partition"});
```

Listing 7.18 – requête pour le calcul des composantes fortemment connexes pour une période d'un an.

On obtient 609 composantes fortement connexes contre 1 033 dont 50% des nœuds qui deviennent connectés (594 nœuds isolés contre 1 024) (voir les tableaux 7.10 et 7.11).

setCount	minSetSize	maxSetSize
609	1	3 963

TABLE 7.10 – Composantes fortement connexes dans le graphe condensé pour une période d'un an.

SSC	taille
1	3 963
1	9
1	4
1	6
2	5
10	2
594	1
609	4 602

TABLE 7.11 – Statistiques des composantes fortement connexes pour une période d'un an.

Conclusion. Suite à ces analyses, nous concluons que notre graphe n'est pas fortement connexe ce qui peut impacter le calcul des autres métriques typiquement la métrique de l'excentricité que nous allons voir par la suite.

7.5.4 Densité

La *densité* d'un graphe est définie comme étant le ratio du nombre d'arcs sur le nombre de nœuds. Cette métrique mesure si le graphe a beaucoup ou peu d'arcs (voir le paragraphe 4.1.1).

La figure 7.11 nous donne l'évolution du nombre de nœuds en fonction du nombre d'arcs pour chaque mois. Nous avons choisi de calculer la densité pour une période d'un an pour voir s'il y a un changement à travers les mois de l'année. La densité est donc calculée pour l'année 2016.

Le comportement de la courbe est assez étrange. En effet, les réseaux aériens deviennent plus connectés avec l'avancement du temps. Donc, on a tendance à voir une courbe droite qui croît avec le temps mais ce n'est pas le cas, cela vient du manque de données. Le pic est atteint pour les deux mois `2016-07` et `2016-08`, la saison de l'été où les vols sont plus nombreux : d'où le nombre d'arcs plus élevé par rapport au nombre de nœuds qui reste quasiment le même durant toute l'année.

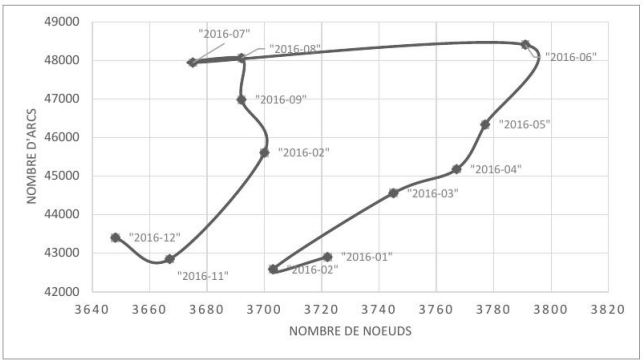


FIGURE 7.11 – Densité du graphe condensé pour l'année 2016.

Comme la start-up *Milanamos* a procédé à un remplacement de ses données par les données de *Sabre* (voir la chapitre 6), nous avons comparé la nouvelle densité calculée avec celle de l'ancien

graphe condensé pour la même année. On constate, alors, avec les nouvelles données de *Sabre* qu'on a plus de manques (voir la figure 7.12). Il fallait peut-être garder les anciennes données et les comparer avec les nouvelles sans les remplacer, et envisager une correction des données.

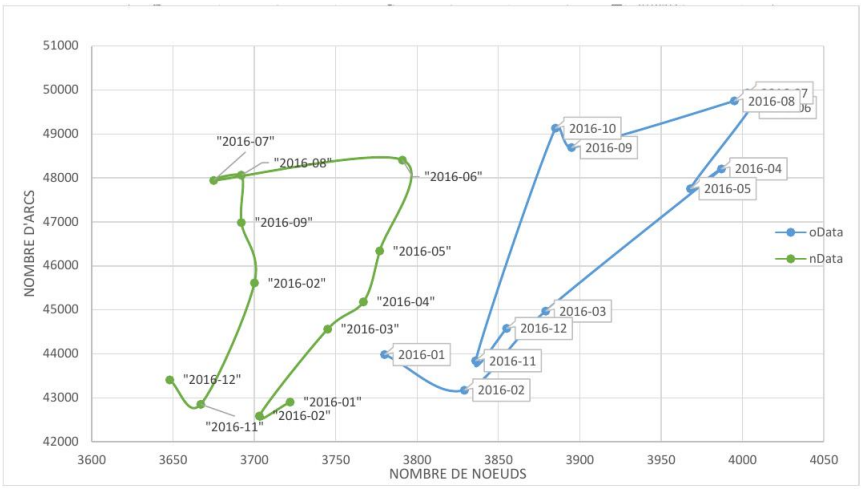


FIGURE 7.12 – Comparaison de la densité de l'année 2016 pour les données mises à jour. `oData` désigne les anciennes données, et `nData` les nouvelles données.

Toujours dans le cadre de l'analyse de la densité, nous avons analysé également l'évolution des données. Nous avons, ainsi, comparé la densité de l'année 2016 avec celle de l'année 2017. On remarque que les données évoluent bien.

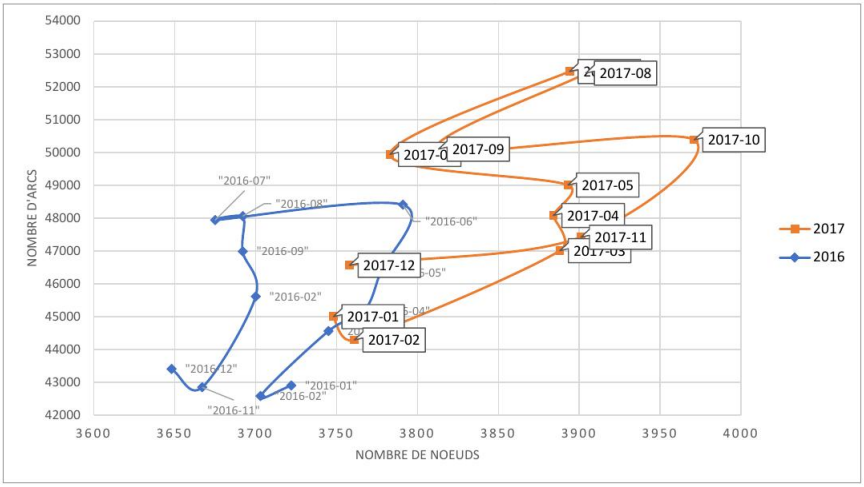


FIGURE 7.13 – Comparaison des densités de l'année 2016 et 2017.

L'autre information qu'on peut tirer de ce graphe est que le pourcentage des nœuds connectés n'atteint pas 100%. Le graphe contient 4 602 nœuds et on a au minimum 20% des nœuds non connectés. Cela est dû à un manque de données pour ces segments reliant cette fraction de nœuds. Ce résultat confirme les résultats présentés avant (880 composantes connexes de taille 1).

Pour générer les données associées au graphique de la figure 7.11, nous avons utilisé une requête `Cypher`, voir la requête du listing 7.19.

```
1 MATCH(n:Airport)-[r]-(m:Airport)
2 WHERE '2016-01' <= TYPE(r) <= '2016-12'
3 WITH COUNT(r) AS deg,n, TYPE(r) AS ym
4 WITH COUNT(n) AS nodes, ym, SUM(deg) AS arcs
5 RETURN ym, nodes, arcs/2
```

Listing 7.19 – requête pour le calcul de la densité.

La requête du listing 7.19 commence par filtrer le graphe suivant les mois demandés (ligne 2). Ensuite, on calcule le nombre d'arcs pour chaque nœud et finalement, on fait la somme de tous les arcs pour chaque nœud et chaque mois (ligne 4).

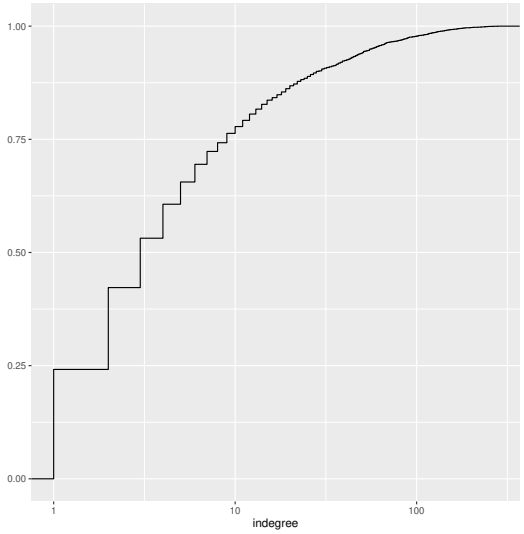
7.5.5 Distribution des degrés

Nous commençons par analyser la distribution des degrés du graphe. Le script du listing 7.20 permet de générer pour chaque `year_month` et chaque nœud, son degré entrant et sortant ainsi que le degré total. Comme les arcs dans le graphe représentent les mois de l'année, on est censé analyser la distribution des degrés pour un seul ensemble d'arcs du graphe correspondant au mois choisi et non pas l'ensemble de tous les arcs du graphe. Ainsi, nous avons choisi le premier mois du graphe qui est le mois de janvier 2016 (`2016-01`). En passant en échelle logarithmique, nous arrivons à identifier facilement les points aberrants. Au moins 20% des aéroports sont non connectés pour ce mois-là et ont donc été supprimés, ce qui confirme les résultats obtenus auparavant.

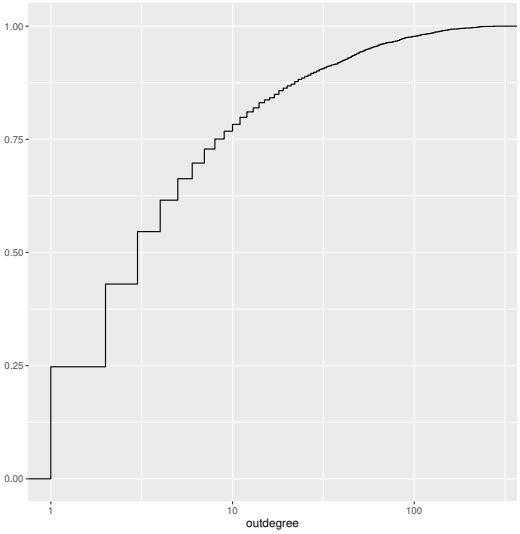
```
1 MATCH (a:Airport)-[r]-()
2 RETURN DISTINCT TYPE(r) AS yearMonth, a.code AS code, a.region AS region,
3 apoc.node.degree(a, '<'+type(r)) AS indegree, apoc.node.degree(a, type(r)+'>')
    AS outdegree
```

Listing 7.20 – requête pour la distribution des degrés

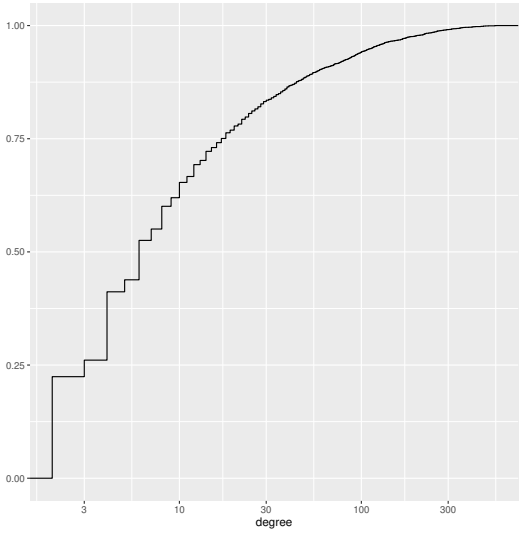
La distribution des degrés du graphe condensé suit la loi exponentielle. Il y a plusieurs nœuds avec moins de liens et peu de hubs avec un grand nombre de liens (voir la figure 7.14). Les graphiques représentent les degrés entrants (voir la figure 7.14a), sortants (voir la figure 7.14b) et total (voir la figure 7.14c) suivant la fonction de répartition ou la fonction cumulative.



(a) Distribution du degré entrant



(b) Distribution du degré sortant



(c) Distribution du degré

FIGURE 7.14 – distribution des degrés pour le graphe condensé.

La distribution des degrés sortants du graphe condensé a la même forme que celle des degrés entrants (Figure 7.14b). On a 5% des aéroports ayant plus de destinations que de provenances (voir Figure 7.14a). Cela n'a pas de sens car les vols arrivant à un aéroport sont censés repartir : pour chaque aéroport représenté par un nœud dans le graphe, on a une conservation de flux. En examinant la répartition des degrés sortants, on a 62% des 3 720 (voir la figure 7.11 de la densité du mois `2016-01`) aéroports qui ont moins de 5 destinations, ce qui représente 2 277 aéroports : cela nous paraît assez étrange.

Aéroports avec moins de cinq destinations.

Pour vérifier l'anomalie détectée dans la distribution des degrés concernant les aéroports ayant moins de 5 destinations, nous avons fait la requête du listing 7.21 en *Cypher* pour le mois ayant plus de données : c'est le mois *2017-08* qui est le plus optimiste pour notre cas.

```
MATCH (n)-[r:`2017-08`]->()
WITH COUNT(r) AS dest,n
WHERE dest<5
RETURN COUNT(n) AS totalNoe, n.region AS region
```

Listing 7.21 – requête pour trouver les aéroports ayant moins de 5 destinations

2 277 aéroports ont moins de cinq destinations ce qui confirme le chiffre trouvé lors de l'analyse du degré sortant. Cela est bizarre car plus de la moitié des aéroports sont des aéroports desservant ce nombre de destinations (62% des 3 720 aéroports ayant des données (voir la figure 7.11)). Un exemple contradictoire est l'aéroport international allemand *Paderborn Lippstadt* (code `PAD`) avec onze destinations [`https\penalty\@M://www.flightsfrom.com`, 2019] et l'aéroport international *Johannesburg-Lanseria* en Afrique du sud (code `HLA`) qui est le hub de trois compagnies aériennes [WIKIPEDIA, 2019].

Suite à cela, nous avons fait une analyse poussée pour déterminer la région de ces aéroports. La figure 7.15 montre la répartition de ces aéroports par région. On trouve que nous avons plus de données manquantes pour les aéroports de la région nord de l'Amérique. Le résultat confirme les résultats obtenus pour l'analyse des composantes connexes (voir la section 7.5.3). Lors de l'analyse de la répartition des nœuds isolés par région, nous avons obtenu un chiffre plus grand que dans le cas des aéroports ayant moins de 5 destinations. Cela a un sens comme les nœuds isolés font partie des aéroports ayant moins de 5 destinations.

La figure 7.15 illustre le graphique de la répartition de ces aéroports par région. La répartition a la même forme que celle de la figure 7.10 et la figure 6.4 (voir la section 6.3.3). Ce sont les mêmes résultats trouvés lors de l'analyse des régions d'aéroports sans coordonnées géographiques et ceux correspondant aux nœuds isolés, composante connexe avec un seul aéroport.

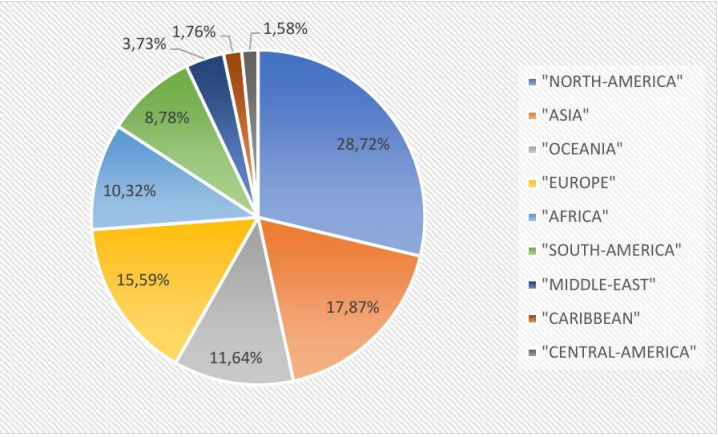


FIGURE 7.15 – Répartition des aéroports ayant moins de 5 destinations par région pour les données mises à jour.

Comparaison avec l'ancienne version

En comparant le résultat de la requête du listing 7.20 avec l'ancien résultat (ancienne version du graphe), nous constatons qu'il y a 310 aéroports qui ont été supprimés et 335 nouveaux aéroports ajoutés ($(4\,577 - 310) + 335 = 4\,602$).

Au niveau de la taille des résultats, en termes de lignes (19 071 relations), nous remarquons qu'il y a plus de données (par aéroport et par `year_month`) par rapport à la taille des nouveaux résultats, ce qui peut être expliqué par le fait que les aéroports ajoutés ont moins de données mensuelles. En effet, pour un aéroport donné, on a au minimum 1 mois de données et au maximum 24 mois de données.

Nous obtenons un chiffre de 2 149 aéroports ayant moins de cinq destinations ce qui représente 57% des 3 770 aéroports (voir la figure 7.12). L'écart a bien augmenté de 5%.

La figure 7.16 montre la répartition de ces aéroports par région. On trouve que nous avons plus de données manquantes des aéroports de la région nord de l'Amérique. En comparant avec la répartition par région pour les nouvelles données (voir la figure 7.15), on constate que ce pourcentage a bien augmenté sur les nouvelles données pour plusieurs régions notamment : l'Amérique du nord et l'Europe. La région `null` correspond aux aéroports dont on ne connaît pas leurs régions et par conséquent leurs coordonnées géographiques. Cela été filtré lors du prétraitement des nouvelles données pour générer la deuxième version du graphe. C'est pour cela, qu'il n'y a plus de région `null` sur la figure 7.15.

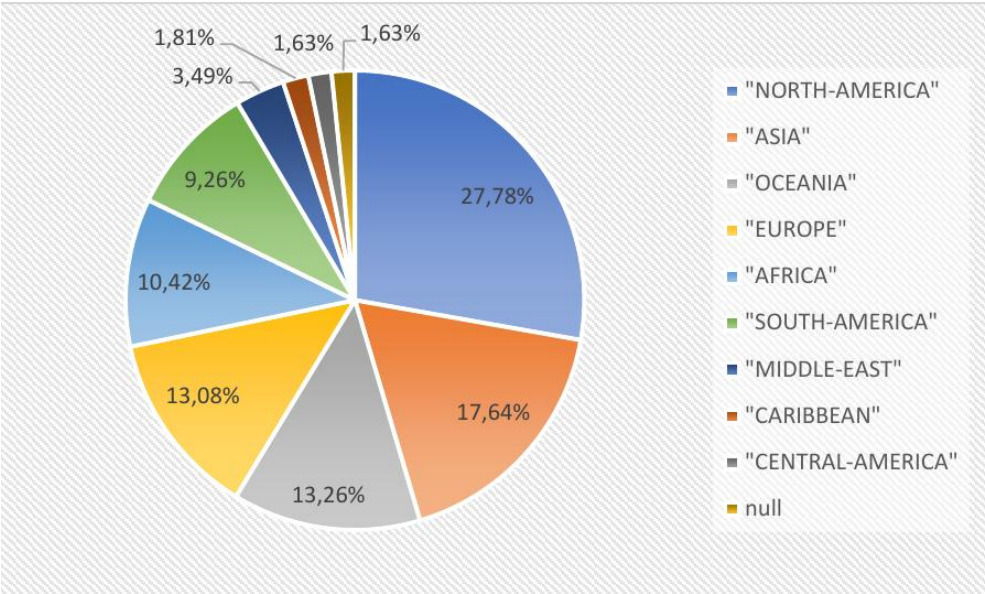


FIGURE 7.16 – répartition des aéroports ayant moins de 5 destinations par région.

7.5.6 Excentricité

Nous commençons par définir l'*excentricité* ainsi que l'ensemble des métriques liées à la métrique *excentricité* [Chakrabarti and Faloutsos, 2006].

Définition 7.5.1 (Excentricité)**.** L'*excentricité* d'un ensemble de nœuds désigne sa distance au nœud dont il est le plus éloigné. La distance entre deux nœuds est définie par la longueur d'un plus court chemin reliant ces deux nœuds en terme de sauts.

Définition 7.5.2 (Diamètre)**.** Le *diamètre* d'un graphe est l'excentricité maximale de ses nœuds. Cette mesure représente la distance la plus courte entre les nœuds les plus éloignés du graphe.

Définition 7.5.3 (Rayon)**.** Le *rayon* ou le *centre* d'un graphe est l'excentricité minimale de ses nœuds.

Définition 7.5.4 (Longueur moyenne de chemin)**.** La *longueur moyenne de chemin* (en anglais *Average path length*) d'un graphe est la distance moyenne du chemin le plus court. Ce qui équivaut, en moyenne, au nombre d'étapes à franchir pour rejoindre un nœud à partir d'un autre nœud du graphe. Le comportement de la longueur moyenne des plus courts chemins nous donne une idée sur le comportement du réseau étudié. Cette métrique représente la moyenne, pour toutes les paires de nœuds, des plus courts chemins.

Définition 7.5.5 (Diamètre moyen)**.** Le *diamètre moyen* (en anglais *Characteristic path length*) d'un graphe représente la même vision de la métrique *longueur de chemin* mais d'un point de vue médiane.

Le graphique des sauts illustré dans la figure 7.17 nous donne l'information sur l'excentricité du graphe condensé. Comme le graphe est modélisé de façon à ce que les arcs soient définis par mois de l'année (`year_month`), on est censé analyser l'excentricité pour un mois donné. On a choisi ainsi le mois de *janvier de l'année 2016* (voir Figure 7.17). Ce graphique est obtenu de la façon suivante : à partir d'un nœud x dans le graphe, on calcule un parcours en largeur (*BFS*) vers tous les autres nœuds connectés à ce nœud dans le graphe, pour ce mois étudié. On suppose que $N_s(x)$ est le nombre de paires de nœuds connectés au nœud x avec s sauts. Ensuite, on répète la même procédure pour tous les nœuds du graphe condensé et finalement, on fait la somme des résultats pour trouver le nombre de paires de nœuds pour s sauts ($N_s = \sum_x N_s(x)$). Comme le graphe contient 4 602 nœuds, on a calculé 21 173 802 paires de nœuds.

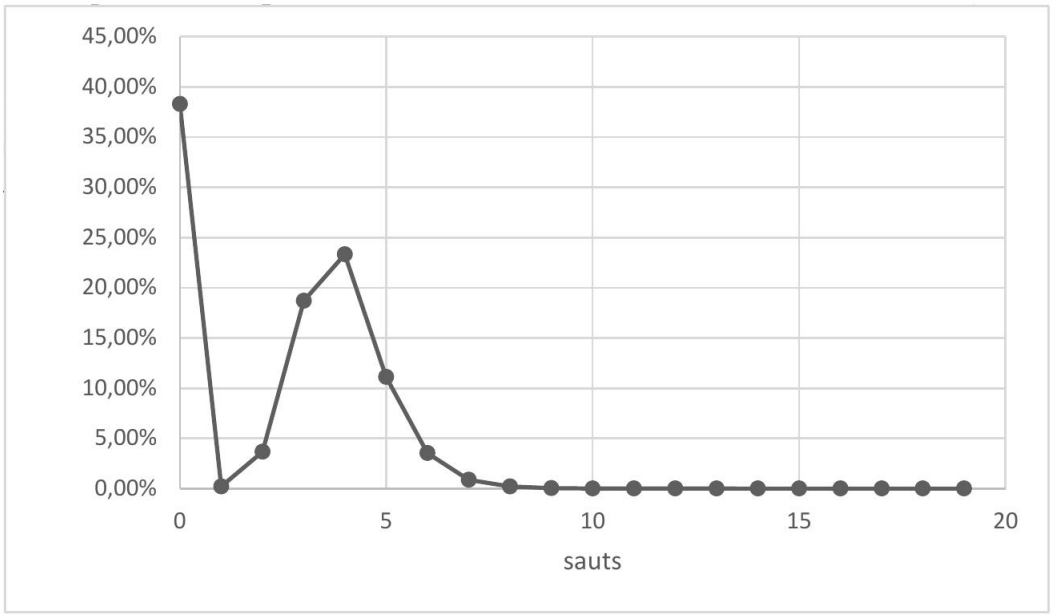


FIGURE 7.17 – Densité mensuelle du graphe condensé pour le mois de janvier 2016.

Ce graphique nous donne trois informations :

— Le *rayon* du graphe est 1. Cela veut dire que nous pouvons rejoindre un certain nombre de nœuds avec un minimum d'un seul saut. Cela représente une fraction assez négligeable par rapport aux autres sauts.

— Le *diamètre* du graphe est 14.

— Une partie importante de nœuds non joignables dans le graphe pour ce mois étudié à cause du manque de données représente 38% du nombre total de paires de nœuds.

— La *longueur moyenne de chemin* et le *diamètre* moyen sont environ 4 sauts. Cela veut dire que 25% de paires de nœuds sont connectées avec 4 sauts.

Nous avons calculé la longueur moyenne de chemin en utilisant la formule 7.2. Nous avons bien trouvé la valeur 4.

$$AV = \frac{\sum_{x \neq y} l^*(x, y)}{n(n-1)} \tag{7.2}$$

où $l^*(x, y)$ est la longueur du plus court chemin entre x et y. n est le nombre de nœuds du graphe.

En `Neo4j`, on peut faire une requête `Cypher` qui calcule le *diamètre* du graphe. Nous avons commencé par tester la requête proposée dans la littérature [Hölsch et al., 2017]. Les auteurs proposent d'analyser la performance des requêtes d'analyse d'un graphe sur la base de données orientée graphe `Neo4j` et une base de données `SQL` (voir la requête du listing 7.22). Cette requête tient compte de tous les mois du graphe (ligne 1). On rappelle qu'un arc dans le graphe condensé représente un mois de l'année. Ce qui va renvoyer un chemin avec plus d'un mois de données quand la paire de nœud n'est pas connectée pour ce mois-ci. Or, c'est irréalisable qu'un passager fasse un chemin sur plusieurs mois qui ne sont même pas consécutifs. La fonction `shortestpath()` représente dans `Neo4j` l'algorithme de *BFS* (voir le chapitre 4).

```
1 MATCH p = shortestpath((n)-[*]-(m))
2 WITH p LIMIT 400000000
3 RETURN p, LENGTH(p)
4 ORDER BY LENGTH(p) DESC
5 LIMIT 1
```

Listing 7.22 – requête pour le calcul du diamètre proposé dans l'article Hölsch et al. [2017]

Ainsi, nous avons adopté la requête du listing 7.22 à notre cas (voir la requête du listing 7.23).

```
1 MATCH p = shortestpath((n)-[:`2016-01`*]-(m))
2 WITH p LIMIT 400000000
3 RETURN p, LENGTH(p)
4 ORDER BY LENGTH(p) DESC
5 LIMIT 1
```

Listing 7.23 – requête du listing 7.22 pour le mois de janvier 2016

La requête met des heures pour calculer uniquement le diamètre. En analysant la structure de la requête, nous avons constaté qu'il y a des calculs redondants des plus courts chemins à cause de la syntaxe `(n)-[:`2016-01`*]-(m)`, cette syntaxe inclut les 4 cas suivants :

N°	Chemin considéré	Cas
1	`(n) -[]-> (m)`	$n \neq m$
2	`(n) <-[]- (m)`	
3	`(m) -[]-> (n)`	$n = m$
4	`(m) <-[]- (n)`	

TABLE 7.12 – Cas considérés dans la requête du listing 7.23.

D'après le tableau 7.12, on constate que le cas 1 correspond au cas 4 et le cas 2 n'est autre que le cas 3. Ainsi, la requête calcule les plus courts chemins pour $(4602^2) \times 2 = 42\,356\,808$ paires. Pour remédier à ce problème, il faut ajouter la direction et une clause qui calcule le plus court chemin entre une origine différente de la destination, ce qui revient à $4\,602 \times 4\,601 = 21\,173\,802$ paires (voir la requête du listing 7.24).

```
1 MATCH path = shortestpath((n:Airport)-[:`2016-01`*]->(m:Airport))
2 WHERE n <> m
3 RETURN n.code, m.code, LENGTH(path)
4 ORDER BY LENGTH(path) DESC LIMIT 1
```

Listing 7.24 – requête pour le calcul du diamètre du graphe condensé

La requête met a peu près 30 minutes pour calculer uniquement le diamètre. Pour calculer également l'excentricité pour une période quelconque, nous avons écrit un programme en *Java* (l'ensemble des librairies de `Neo4j` étant écrites en *Java*). Le script met, ainsi, 2 fois moins de temps que la requête.

Comparaison avec l'ancienne version

Nous avons procédé à une comparaison avec les anciennes données en choisissant la même période que dans l'analyse de densité. On constate les points suivants :

— Le diamètre de l'ancien graphe pour le mois janvier est 14 (courbe verte de la figure 7.18) ;

— le diamètre est faible pour certains mois, ce qui explique qu'on a plus de données notamment, les mois de haute saison comme *juillet* et *août*.

— le diamètre maximal du graphe est 14 contre 15 pour les anciennes données.

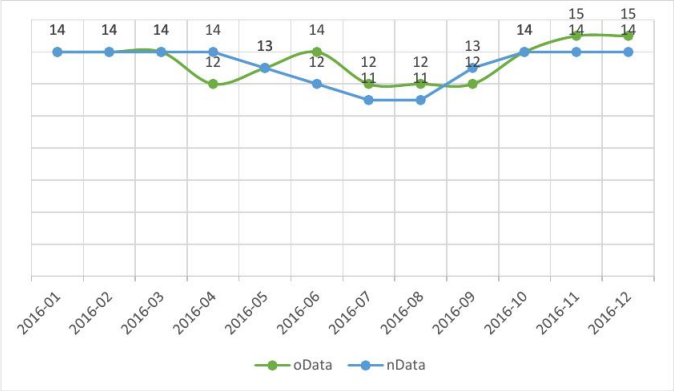


FIGURE 7.18 – Comparaison du diamètre mensuel avec les anciennes données. `oData` désigne les anciennes données, et `nData` les nouvelles données.

7.6 Propriétés des réseaux petit-monde

Un réseau dit *petit-monde* (en anglais *small-world*) est caractérisé par deux propriétés :

— Les réseaux deviennent plus denses avec le temps typiquement dans le domaine aérien, car le flux de passagers augmente avec le temps et le réseau contient donc plus de segments de vols et de routes d'avions.

— Plus le réseau est grand, plus le monde est connecté.

Pour vérifier les propriétés du réseau *petit-monde*, nous nous basons sur les métriques calculées dans la section précédente :

1. connexité ;
2. distribution des degrés ;
3. excentricité, en particulier, le diamètre ;
4. densité.

7.6.1 Connexité

Dans un réseau *petit-monde*, il y a une grande probabilité qu'il existe une composante dominante du graphe dont la taille est une fraction de n. Nous avons montré que la composante la plus dominante contient 3 551 aéroports (80% des aéroports connectés). En outre, il existe d'autres composantes qui ont des tailles très petites de l'ordre de $log(n)$.

7.6.2 Densité.

L'analyse de la *densité* permet de voir si le réseau croît de manière exponentielle [Leskovec et al., 2005] : le nombre de vols croît-il rapidement par rapport aux aéroports du réseau aérien ?

Dans ce cas, la distribution des degrés connaîtra une croissance significative. Ce genre de réseau a tendance à croître exponentiellement. Or, la densité calculée pour notre graphe condensé montre un comportement assez étrange. Cela est dû au manque de données.

7.6.3 Distribution des degrés

Avec le calcul de la distribution des degrés du graphe condensé, nous avons remarqué la présence de plusieurs nœuds avec moins de liens et peu de hubs avec un grand nombre de liens. Un tel réseau est appelé, dans la théorie des réseaux complexes, *réseau invariant d'échelle* (en anglais *Scale-free*) [Chakrabarti and Faloutsos, 2006]. Un tel graphe est caractérisé par la présence des points aberrants.

7.6.4 Diamètre.

Une autre métrique liée à l'excentricité est la notion de *diamètre effectif*. Cette notion est définie par le nombre minimum de sauts à partir duquel une certaine fraction de paires de nœuds (soit 90%) est connectée [Kang et al., 2010]. Cette métrique permet de suivre l'évolution du graphe. Dans notre cas, notre réseau n'a pas de *diamètre effectif* comme il n'y a pas 90% de paires de nœuds connectées avec un certain saut du fait qu'il n'est pas connexe. Cela paraît insensé car un réseau aérien est supposé être connecté. Mais il ne faut pas oublier que les données viennent du monde réel avec un pourcentage important de données manquantes.

Ainsi, nous avons continué à analyser l'excentricité du graphe mais sur l'année complète 2016, de `2016-01` à `2016-12` (voir la Figure 7.18). Le graphique nous montre des pics sur certaines périodes. Pour certains mois comme les mois `2016-07` et `2016-08`, nous remarquons qu'ils ont un diamètre moins élevé que les autres. Cela est tout à fait normal, le diamètre par sa définition ayant tendance à diminuer quand le réseau est plus connecté. C'est le cas de la saison d'été où il y a plus de vols dans le réseau aérien. Comme un tel réseau a tendance à croître avec le temps, nous devons voir le diamètre diminuer avec le temps. Les facteurs pouvant expliquer cette évolution bizarre constatée dans la figure 7.18 [Leskovec et al., 2005] sont :

— des données manquantes ;
— le graphe n'est pas connexe, cela est traduit par la présence de plus d'une composante connexe ;
— le diamètre effectif.

On se place sur le premier facteur qui influence directement les autres facteurs. En effet, le comportement bizarre de l'évolution du diamètre à travers le temps est traduit par le manque de données ce qui explique que le diamètre effectif dans ce cas est nul quand le graphe n'est pas connexe.

7.6.5 Identification des hubs

L'algorithme *Hub labeling* est une technique d'optimisation qui nécessite de n'utiliser que des plus courts chemins déjà calculés lors d'un prétraitement (voir le chapitre 4). Pour cela, l'algo-

rithme a besoin de choisir des nœuds intermédiaires "*hubs*". Dans notre cas, les nœuds représentent les aéroports et nous choisissons les "*hubs*" suivant les critères métiers discutés avec *Milanamos* (voir la section 7.6.5.2) qui doivent respecter les contraintes suivantes :

— au moins 50% des compagnies `LCC` présentes ;

— au minimum 10 millions de passagers qui transitent ;

— présence du moyen/long courrier.

Un vol moyen courrier est un vol qui dure en moyenne 3 heures. Alors que le long courrier dure 6 heures.

7.6.5.1 Méthodologie

Description de la méthode

Nous procédons à un croisement de plusieurs sources de données pour calculer le score de chaque aéroport en nous basant sur la liste des critères présentée dans le tableau 7.13 :

Source de données

— L'encyclopédie libre Wikipédia.

— L'organisation de l'aviation civile internationale.

— Les données de *Milanamos* : base `optimode`.

— Le graphe condensé (*CFN*).

7.6.5.2 Métriques

Critère	Valeur de Préférence	Source de données
Compagnies aériennes du hub	Nombre de compagnies aériennes utilisant l'aéroport comme un hub	Liste des aéroports hubs [WIKIPEDIA, 2019]
Présence des compagnies `LCC`	La présence des compagnies `LCC` implique que cet aéroport est un hub : on estime qu'il doit y avoir au minmum 50% des *LCC*	Organisation de l'aviation civile internationale [ICAO, 2017]
Nombre de passagers (`Pax`)	Un nombre de passagers plus élevé dans un aéroport implique que celui-ci est déjà une destination populaire. Ceci est appliqué même pour le revenu généré pour un aéroport	Base `optimode`
Localisation géographique	Des informations sur le pays et la région de l'aéroport sont à prendre en compte pour mesurer la diversité géographique, dans le cas où il n'y a qu'un seul aéroport dans le pays	Graphe condensé *CFN* (voir la requête du listing 7.25)
Moyen/Long courrier	Un aéroport est considéré comme un hub s'il y a des vols moyen ou long courrier, ça veut dire des vols qui durent au minimum 3 heures	Graphe condensé *CFN* (voir la requête du listing 7.26)
Nombre de destinations	Le nombre de destinations desservies par un aéroport indique son importance, nous avons pris le minimum et le maximum comme la dispersion est trop grande	Graphe condensé *CFN* (voir la requête du listing 7.27)
Nombre de compagnies aériennes	Plus le nombre de compagnies aériennes, qui ont des vols partant de cet aéroport, est élevé, plus cet aéroport est principal	Graphe condensé *CFN* (voir la requête du listing 7.28)

TABLE 7.13 – Valeurs de préférences et sources de données des critères.

7.6.5.3 Requêtes utilisées

Nous avons utilisé deux requêtes pour calculer le nombre de passagers, `pax`, qui transitent pour chaque aéroport de notre graphe (voir le programme en annexe D).

La requête du listing 7.25 renvoie le pays et la région de chaque pays.

```
1 MATCH (n:Airport)
2 RETURN n.country AS pays, n.region AS region
```

Listing 7.25 – requête pour la localisation géographique en *Cypher*

La requête du listing 7.26 renvoie les aéroports ayant au moins un vol d'une durée de 3 heures.

```
1 MATCH (n:Airport)-[r]-()
2 WHERE r.duration_min>=180
3 RETURN DISTINCT n.code
```

Listing 7.26 – requête pour trouver les aéroports qui ont moyen et long courrier

La requête du listing 7.27 renvoie le nombre minimum et maximum de destinations pour chaque aéroport sur la période de 24 mois.

```
1 MATCH (n:Airport)-[r]->(m:Airport)
2 WITH n.code AS airp, type(r) AS relType, count(m) AS nbrDest
3 RETURN airp, Min(nbrDest) as minD, Max(nbrDest) as maxD
```

Listing 7.27 – requête pour trouver la liste des destinations en *Cypher*

La requête du listing 7.28 renvoie la liste des compagnies aériennes ayant des vols au départ de chaque aéroport pour tous les mois du *CFN* (24 mois).

```
1 MATCH(n:Airport)-[r]->(m:Airport)
2 UNWIND r.airlines AS air
3 WITH COLLECT(DISTINCT air) AS airlList, n
4 RETURN n.code, airlList
```

Listing 7.28 – requête pour trouver la liste des compagnies aériennes en *Cypher*

7.6.5.4 Résultats

En analysant le résultat, nous avons constaté qu'il y a seulement 337 aéroports sur 4 602 (soit 7%) fournis par Wikipédia. Pour des grands aéroports, notamment, l'aéroport *Charles-De-Gaulle* à *Paris*/France (CDG) et *Londres Heathrow* à *Londres*/Royaume-Uni (LHR), la *Présence des compagnies LCC* ne dépasse pas 25%, ce qui paraît peu. On savait que le manque des données sur les *LCCs* impactera le calcul de cet index (voir la section 6.3.6).

Dans la majorité des cas, il y a un grand écart entre le nombre minimum et maximum de destinations.

Conclusion.

L'analyse du graphe nous a aidé à mieux comprendre et identifier les entités manquantes sur les données. Cela n'aurait pu être le cas avec la base de données optimode. En effet, comme les données sont stockées de façon redondante et dans plusieurs endroits différents, la tâche aurait été compliquée pour arriver à cette analyse.

CHAPITRE 8

Sélection des marchés

Dans ce chapitre, nous définissons le problème FRP *dans le cadre de la sélection des marchés, qui consiste à déterminer les marchés représentant des opportunités économiques pour une compagnie aérienne. Tout d'abord, nous commençons par formuler le problème et étudier ses propriétés. Ensuite, nous exposons différentes méthodes de résolution du* FRP. *La première méthode repose sur les requêtes de* Cypher *et la deuxième méthode est basée sur des algorithmes de plus courts chemins. Dans ce contexte, nous proposons de comparer quatre algorithmes : deux algorithmes qui décomposent le* FRP *en problèmes de plus court chemin et les résolvent en parallèle avec les algorithmes de* `Dijkstra` *et* `Bellman`*; le troisième algorithme est une extension de l'algorithme* `Dijkstra` *couplé à des techniques d'accélération pour éviter les calculs inutiles ; le quatrième algorithme étend l'algorithme de* `Bellman` *pour calculer tous les plus courts chemins pour toutes les directions et tous les critères à la fois. Ces quatre algorithmes effectuent un calcul en parallèle lorsque cela est possible. Puis, nous décrivons le protocole d'expériences menées dans le cadre du test de ces algorithmes. Enfin, nous analysons les principaux résultats.*

8.1 Présentation du *Flight Radius Problem*

Dans le chapitre précédent, nous avons modélisé le réseau aérien par un graphe condensé. C'est un graphe indépendant du temps dans lequel les nœuds représentent les aéroports et un arc indique l'existence d'un vol entre deux aéroports.

Étant donné une origine et une destination pour le nouveau vol, le module *Flight Simulator* de l'application `PlanetOptim` renvoie tous les aéroports reliés par un vol vers l'origine ou issu de la destination (voir le chapitre 2). Ce sont les origines et les destinations des passagers qui seront potentiellement intéressés par ce nouveau vol. Certains chemins du sous-graphe renvoyé n'ont aucun intérêt en pratique. De plus, ce sous-graphe est souvent trop grand et dense pour être visualisé correctement dans l'application. Ensuite, l'application calcule les critères de *QSI* et estime la part de marché pour ce nouveau vol.

Exemple 8.1.1 (Exemple de motivation)**.** L'exemple de la figure 8.1 représente une capture d'écran du module *Flight Simulator*. Dans l'exemple, l'application renvoie tous les aéroports vis-à-vis de l'origine *Nice*/France (`NCE`) et la destination *Bangkok*/Thaïlande(`BKK`). Par exemple sur la figure, le trajet Nice-Bangkok-Dubaï n'est pas une alternative réaliste pour relier Nice à Dubaï puisqu'il existe un trajet reliant cette paire d'aéroports en passant par l'aéroport *Paris-Charles-De-Gaulle*/France(`CDG`) qui est plus rapide et moins cher. Et donc, le trajet Nice-Bangkok-Dubaï représente une part de marché négligeable. Ce qui revient de dire que les passagers qui voyagent entre Nice et Dubaï ne seront pas intéressés par le vol entre Nice et Bangkok. En revanche, les passagers qui voyagent entre Nice et Pékin, seront potentiellement intéressés par passer par le vol Nice - Bangkok pour aller à leurs destinations finales. C'est ainsi que le marché définit par la paire d'aéroports Nice & Pékin est un marché prometteurs centré sur le vol Nice-Bangkok. Toutefois, celui du Nice & Dubaï ne l'est pas.

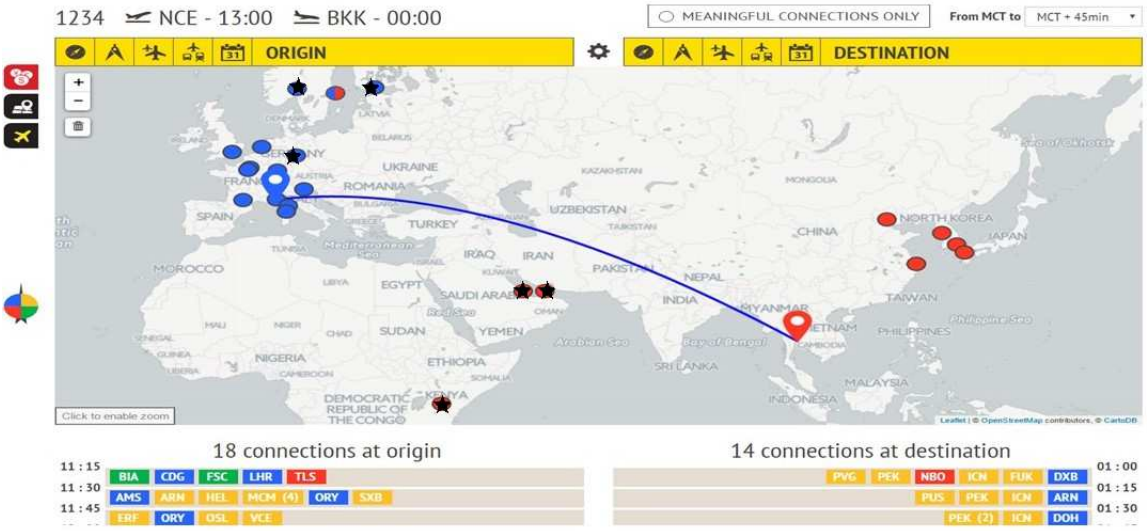


FIGURE 8.1 – Capture d'écran du module *Flight Simulator* de l'application `PlanetOptim`. Les aéroports barrés représentent des aéroports dont les routes ne correspondent pas à des itinéraires réalistes. Les couleurs des aéroports correspondent à leurs localisations suivant la boussole.

C'est dans le cadre de la sélection des marchés que nous avons défini le problème *FRP*, qui permet de déterminer un sous-graphe de parts de marchés non négligeables. Ceci permet d'améliorer la visualisation et accélérer les calculs des parts de marchés face aux alternatives existantes en termes de flux des passagers, mais pas face aux concurrences.
Les solutions de ce problème ne se restreint pas au problème de *2-hops* qui consiste à trouver les chemins à une seule escale partant d'un aéroport dans un réseau aérien (voir la section 5.1.4). Mais il est plutôt lié au problème de *développement des routes* qui consiste à déterminer les routes qui présentent des opportunités économiques pour la compagnie aérienne afin de l'inclure dans son réseau (voir le chapitre 3).

Plus formellement, le problème *FRP* est défini sur le réseau aérien condensé (*CFN*) et consiste à identifier un sous-réseau de routes vis-à-vis d'un nouveau vol suivant certains critères, comme le temps et le coût. En effet, le problème revient à déterminer un sous-graphe maximal, en termes de nœuds, dans lequel au moins un chemin valide passe par chaque nœud. Un chemin valide est un chemin qui passe par l'arc entre l'origine et la destination en limitant, par exemple, la perte de temps ou le surcoût par rapport aux meilleurs chemins.

8.1.1 Notations du *Condensed Flight Network*

Dans le graphe condensé, les nœuds représentent les aéroports et les arcs l'ensemble des vols entre une paire d'aéroports. Chaque arc est construit en agrégeant toutes les connexions élémentaires entre une paire d'aéroports. On reprend les notations présentées dans la section 7.2.4. Soit $\mathcal{C}_{od} = \{c \in \mathcal{C} \mid o = o(c) \wedge d = d(c)\}$ l'ensemble des connexions élémentaires entre les deux aéroports o & d. Le tableau 8.1 présente les étiquettes associées à l'arc (o, d) dans le graphe condensé.

Étiquette	Formule	Désignation
F_{od}	$\lvert\mathcal{C}_{od}\rvert$	nombre de connexions élémentaires entre o et d
c_{od}	$\sum_{c\in\mathcal{C}_{od}} cap(c)$	capacité totale en termes de nombre de passagers
P_{od}	$\sum_{c\in\mathcal{C}_{od}} pax(c)$	nombre total de passagers
R_{od}	$\sum_{c\in\mathcal{C}_{od}} r(c)$	revenu total
$\bar{R}_{od}$	$\min_{c\in\mathcal{C}_{od}} \frac{r(c)}{pax(c)}$	revenu minimum par passager
t_{od}	$\min_{c\in\mathcal{C}_{od}} t(c)$	durée minimale du vol
T_{od}	$\max_{c\in\mathcal{C}_{od}} t(c)$	durée maximale du vol
D_{od}	-	distance entre les deux aéroports
A_{od}	-	liste des compagnies aériennes qui opèrent les vols
W_{od}	-	liste des jours de la semaine où le vol est opéré
WF_{od}	-	nombre d'opérations hebdomadaires

TABLE 8.1 – Tableau des étiquettes d'un arc du graphe condensé.

8.1.2 Notions de regret multicritère

Par la suite, nous allons utiliser ces définitions liées à notre problème *FRP*.

Définition 8.1.1 (Regret)**.** Dans la théorie de la décision, la notion de *regret* est définie comme étant le manque à gagner en ne prenant pas la bonne décision. Par exemple pour notre problème, la décision porte sur les vols.

Définition 8.1.2 (Contrainte de regret)**.** Nous définissons la contrainte de regret $R_{od}(i,j)$ d'un chemin entre i et j passant par l'arc (o,d) dans le cas multicritère de ces façons :

$$R^{+}_{od}(i,j) = l(i,j) \leq l^{\star}(i,j) + K \tag{8.1}$$

$$R^{*}_{od}(i,j) = l(i,j) \leq l^{\star}(i,j) * K \tag{8.2}$$

Où $l^{\star}(i,j)$ est la longueur du plus court chemin de i à j tandis que $l(i,j)$ est la longueur du chemin passant par l'arc (o,d). Ce qui signifie qu'on vérifie que la longueur du chemin entre i et j qui passe par l'arc (o,d) est inférieure à celle du plus court chemin entre le couple (i,j) à une constante près K qui représente le coût supplémentaire acceptable. La contrainte de regret peut être modélisée de deux façons :

— Cas additif (voir l'équation 8.1) : la constante K est positive et indépendante du chemin considéré ;

— Cas multiplicatif (voir l'équation 8.2) : la constante K est souvent représentée par un pourcentage qui dépend du plus court chemin entre i et j et est supérieure ou égale à 1.

La contrainte de regret R est définie pour chaque critère (temps, coût ou distance). C'est une contrainte booléenne qui est vraie pour le couple (i,j) si et seulement s'il existe un chemin entre i et j passant par l'arc (o,d) et satisfait cette contrainte.

On notera $R^{k}_{od}(i,j)$ la composante de la contrainte $R_{od}(i,j)$pour le critère k.

Définition 8.1.3 (Chemin valide)**.** Un chemin *valide* entre i et j est un chemin qui satisfait une contrainte de regret pour au moins un critère. Il peut exister plus de chemins valides entre i et j.

Définition 8.1.4 (Nœud supporté)**.** On dit qu'un nœud est *supporté* s'il existe au moins un chemin valide partant ou arrivant à ce nœud dans le graphe pour un critère.

8.1.3 Notions d'optimisation multicritère

Le problème *FRP* a un aspect multicritère. Dans le cas de l'optimisation multicritère, nous cherchons à trouver les chemins qui représentent un compromis entre ces critères. Pour certains problèmes dans la littérature, on peut classer les critères par ordre d'importance. Ce type de problème consiste à trouver les chemins de *Pareto* en utilisant l'ordre lexicographique [Gandibleux et al., 2006]. Il est de complexité exponentielle en général, et ce même dans le cas de deux critères. En effet, le nombre de solutions de *Pareto* (voir la définition 8.1.7) peut être exponentiel en fonction du nombre d'étiquettes d'un nœud [Ehrgott, 2005]. Ici, nous cherchons des chemins satisfaisant la contrainte de regret pour au moins un critère : ce sont des solutions dites *supportées* (voir la figure 8.2).

Définissons formellement les notions relatives à l'optimisation multicritère [Ehrgott, 2005]. Nous commençons par définir la relation de dominance.

Définition 8.1.5 (Dominance)**.** La solution x *domine* la solution y si :

— x est au moins aussi bonne que y pour tous les critères ;

— x est strictement meilleure que y pour au moins un critère.

La solution y est dite *dominée* [1].

Définition 8.1.6 (Efficace)**.** Les solutions qui dominent les autres, mais ne se dominent pas entre elles sont appelées solutions *efficaces* ou *non dominées*.

Définition 8.1.7 (Front de Pareto)**.** Le *front de Pareto* représente l'ensemble des solutions optimales qui sont *non dominées*.

Définition 8.1.8 (Idéal)**.** Le point *idéal* correspond aux meilleures valeurs obtenues pour chaque critère, cela permettra d'obtenir une borne inférieure pour les solutions du *front de Pareto*.

Définition 8.1.9 (Nadir)**.** Le point *nadir* correspond aux pires valeurs obtenues pour chaque critère, cela permettra d'obtenir une borne supérieure pour restreindre l'espace de recherche.

Ces notions sont illustrées dans la figure 8.2. Comme le montre la figure, les domaines hachurés correspondent aux solutions *supportées* du *FRP* étant donné deux critères k_1 et k_2. En effet, nous avons introduit les solutions supportées pour deux raisons : facilité de calcul et prise en compte de plusieurs profils d'usagers. La figure nous permet de mieux situer la notion du regret par rapport aux autres notions de l'optimisation multicritère.

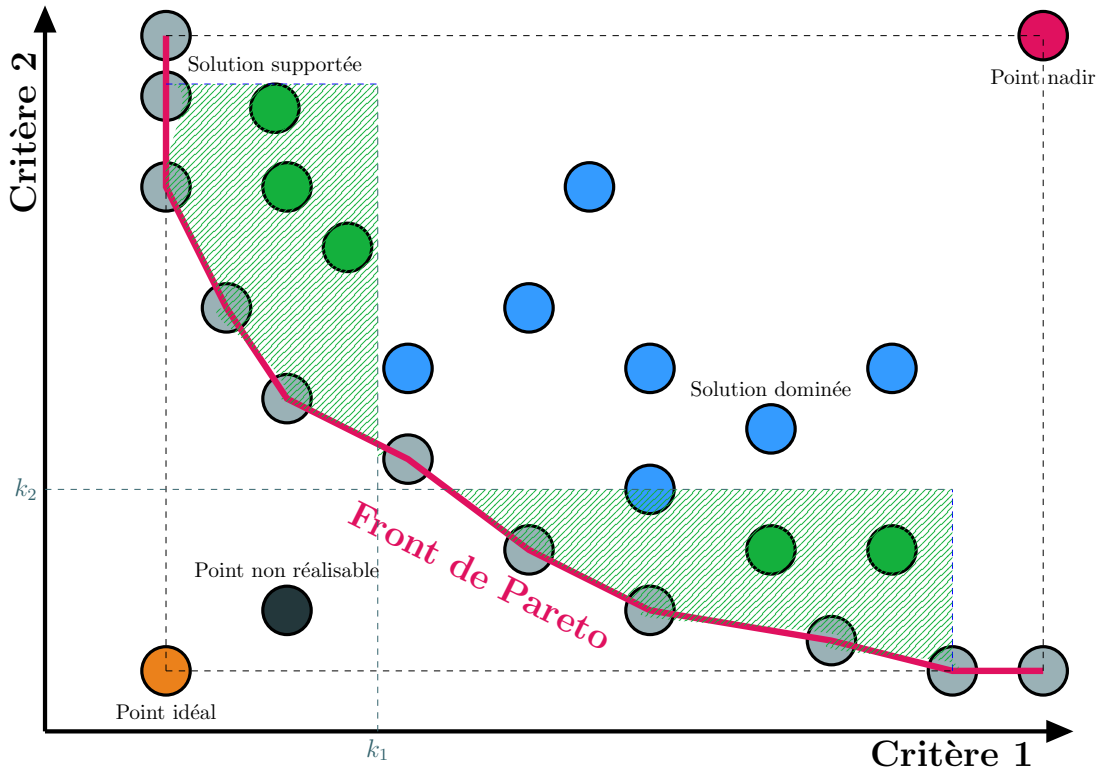


FIGURE 8.2 – Illustration des solutions supportées du *FRP*.

1. On utilise le terme *dominée* au lieu du terme *supportée* pour ne pas confondre avec les solutions du *FRP*, qui sont dites solutions *supportées*.

8.1.4 Formulation

Étant donné un graphe, la question est de décider s'il est intéressant de passer par un vol pour aller d'un aéroport i à un aéroport j. Le vol est représenté par un arc (o, d) dans le graphe condensé *CFN* tel que $o, d \in \mathcal{X}$. L'arc (o, d) peut ne pas exister dans le graphe.

Plus précisément, la question est de trouver toutes les paires d'aéroports (i, j) telles que passer par l'arc (o, d) permettra aux passagers de faire des trajets qui ne sont pas trop longs ou trop chers par rapport aux meilleurs itinéraires possibles. Nous nous intéressons alors à la récupération de tous les chemins passant par l'arc (o, d) qui pourraient être valides selon le regret défini pour chaque critère.

La contrainte de regret R est introduite pour modéliser les différentes préférences des passagers. En pratique, il y a plusieurs critères à considérer mais les trois critères principaux sont : temps, coût et distance.

De manière formelle, le problème est défini de la façon suivante [Idrissi et al., 2017] :

Entrée *Un graphe $G = (\mathcal{X}, \mathcal{U}, \mathcal{W})$, un arc (o, d) et une contrainte de regret R.*
Sortie *Un sous-graphe maximal, en termes de nœuds, $G' = (\mathcal{A}, \mathcal{V})$ tel que $\mathcal{A} \subseteq \mathcal{X}$ et chaque nœud supporte un chemin valide,* i.e. *passant par l'arc (o, d) et satisfait la contrainte de regret R.*

8.1.5 Propriétés

Le problème *FRP* a pour objectif de trouver des chemins passant par l'arc (o, d) qui ne sont pas trop longs ou trop chers que le plus court chemin. Plus précisément, voyager de $i \in \mathcal{X}$ à $j \in \mathcal{X}$ en passant par l'arc (o, d) est intéressant si et seulement si le chemin $\{i, .., o, d, .., j\}$ entre i et j satisfait la contrainte de *regret* R. La satisfaction de la contrainte dépend du plus court chemin entre i et j (voir la figure 8.3).

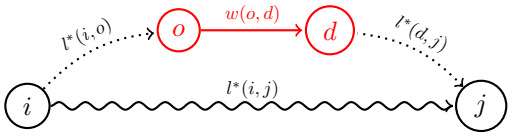


FIGURE 8.3 – Décomposition d'un chemin. L'arc en couleur rouge représente l'arc étudié (o, d). L'arc en pointillé représente le plus court chemin.

On remarque que la recherche des chemins valides peut être restreinte à la recherche des plus courts chemins valides à partir de o et arrivant à d (voir l'inégalité 8.3) :

$$l(i, j) \geq l^\star(i, o) + w(o, d) + l^\star(d, j) \tag{8.3}$$

En d'autres termes, suivre le plus court chemin de i à o, prendre le vol entre o et d et ensuite suivre le plus court chemin entre d et j est toujours un chemin valide s'il existe.

Une méthode naïve pour la vérification de la contrainte de regret R serait de calculer la matrice des plus courts chemins de tout le graphe, ce qui est plus coûteux en termes de temps. En effet, pour vérifier la contrainte de regret, il faut calculer les plus courts chemins pour toutes les paires (i, j).

Comme nous avons prouvé que la recherche des chemins valides peut être restreinte à une recherche des plus courts chemins, cela permet d'établir les trois propriétés suivantes pour le problème de *FRP*.

Propriété 8.1.1. Dans le cas additif, si K est positif ou nul dans l'inégalité 8.1 alors tout plus court chemin est un chemin valide.

Si K est nul, l'inégalité 8.1 devient alors :

$$l(i,j) \leq l^{\star}(i,j)$$

Comme le plus court chemin est unique, en termes de longueur, alors le chemin allant de i à j et passant par l'arc (o,d) est forcément un plus court chemin.

Il est à noter que dans le cas négatif, il n'existe pas de solution.

Propriété 8.1.2. Dans un graphe $G = (\mathcal{X},\mathcal{U},\mathcal{W})$, soit P un plus court chemin de i vers d en passant par o. Alors le sous-chemin de P de i vers o, noté P' est un plus court chemin de i vers o [Ahuja et al., 1993] (voir la figure 8.4).

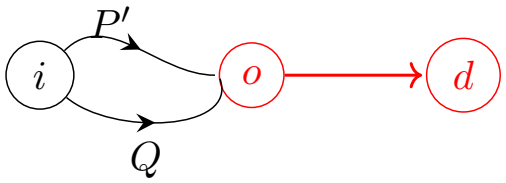


FIGURE 8.4 – Preuve de la propriété 8.1.2.

Démonstration.

$$\begin{aligned} l^{\star}(P) &\leq l(Q) + w(o,d) \\ l^{\star}(i,o) + \cancel{w(o,d)} &\leq l(i,o) + \cancel{w(o,d)} \\ l^{\star}(i,o) &\leq l(i,o) \\ l^{\star}(P') &\leq l(Q) \end{aligned} \tag{8.4}$$

Donc P' est un plus court chemin de P.

□

La propriété 8.1.2 implique la propriété suivante 8.1.3.

Propriété 8.1.3. Tout sous-chemin d'un chemin valide passant par l'arc (o,d) est un chemin valide.

En effet, on sait que tout sous-chemin d'un plus court chemin est un plus court chemin (voir la propriété 8.1.2). Cela vient du fait que le plus court chemin satisfait la propriété de l'inégalité triangulaire (voir la preuve 8.1.5).

Démonstration.

Soit $\{i, ..., o, d, ..., j\}$ un chemin valide, c'est-à-dire que c'est un chemin qui satisfait la contrainte de regret.

$$\begin{aligned} l^\star(i,j) &\leq l(i,j) + K \\ l^\star(i,o) + w(o,d) + l^\star(d,j) &\leq l^\star(i,j) + K \\ \cancel{l^\star(i,o)} + w(o,d) + l^\star(d,j) &\leq \cancel{l^\star(i,o)} + l^\star(o,j) + K \\ w(o,d) + l^\star(d,j) &\leq l^\star(o,j) + K \\ \overrightarrow{R_{od}}(j) = l(o,j) &\leq l^\star(o,j) + K \end{aligned} \tag{8.5}$$

Réciproquement, le sous-chemin de i à d est un chemin valide :

$$\begin{aligned} l^\star(i,o) + w(o,d) + l^\star(d,j) &\leq l^\star(i,j) + K \\ l^\star(i,o) + w(o,d) + \cancel{l^\star(d,j)} &\leq l^\star(i,d) + \cancel{l^\star(d,j)} + K \\ l^\star(i,o) + w(o,d) &\leq l^\star(i,d) + K \\ \overleftarrow{R_{od}}(i) = l(i,d) &\leq l^\star(i,d) + K \end{aligned} \tag{8.6}$$

□

Où $\overrightarrow{R_{od}}(j)$ est la contrainte de regret dans le cas d'une direction sortante, $\overleftarrow{R_{od}}(j)$ dans le cas d'une direction entrante et $w(o,d)$ est le poids de l'arc étudié (o,d).

Finalement, un sous-chemin d'un chemin valide est un chemin valide. Suite à cela, la recherche des chemins valides peut être restreinte à la recherche des chemins valides à partir de o (voir l'inégalité 8.5) ou bien arrivant à d (voir l'inégalité 8.6).

Le lemme 8.1.1 découle directement de la propriété 8.1.1.

Lemme 8.1.1. Soit p un plus court chemin de o. Si l'arc (o,d) appartient à p, alors tous les nœuds de p sont *supportées*.

On peut remarquer que cette propriété n'est plus vérifiée si la constante K est multiplicative. En effet, le fait que le paramètre du regret K dépende du plus court chemin entre i et j empêche la décomposition en sous-chemins.

8.1.6 Exemple

L'exemple de la figure 8.5 présente le graphe modélisant l'exemple de motivation 8.1.1 (voir la figure 8.1). Les données dans ce graphe correspondent aux données réelles de la base `optimode`.

L'exemple est fait pour un seul critère, le coût avec un regret de $K = 100\$$, les entrées et les sorties du problème *FRP* sont illustrées :

1. l'arc étudié (o,d) est représenté par la couleur rose ;
2. les nœuds en trait plein et en vert représentent le graphe G ;
3. les nœuds et les arcs pointillés en couleur verte forment les nœuds supportés et les chemins valides du sous-graphe en sortie G'.

Par exemple, le nœud correspondant à l'aéroport de `DXB` appartient au graphe G' puisqu'il existe un chemin valide partant de l'aéroport de `NCE` et arrivant à l'aéroport de `DXB` avec une correspondance à l'aéroport de `BKK`, qui satisfasse la contrainte de regret.

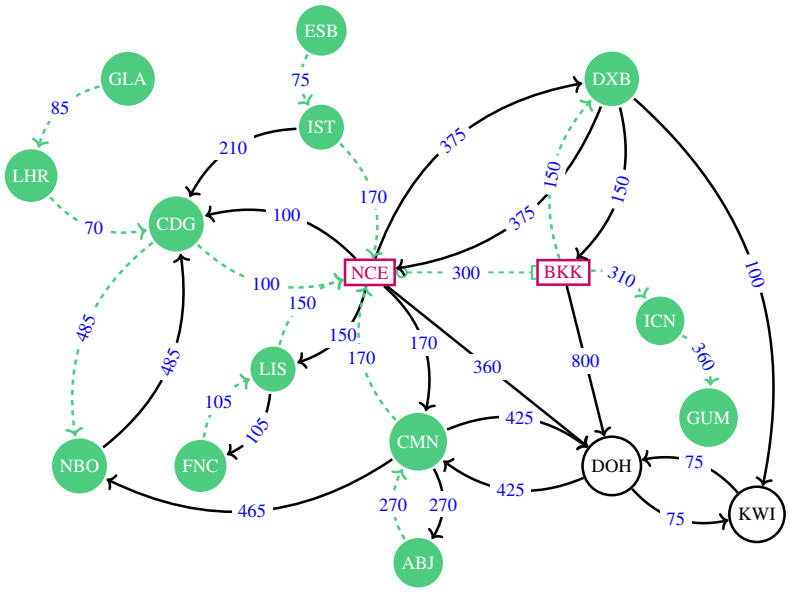


FIGURE 8.5 – Exemple du graphe *FRP* illustrant la solution obtenue.

8.2 Description des méthodes de résolution proposées

Pour trouvé l'ensemble des nœuds supportés, nous avons démontré que la recherche des chemins valides peut être limitée à la recherche des plus courts chemins valides. L'algorithme de recherche de plus court chemin depuis une source (en anglais *Single-Source Shortest Path : SSSP*) peut être ainsi utilisé pour trouver les sous-chemins valides partant de o vers tous les nœuds j. Il en est de même pour les sous-chemins valides arrivant à d depuis tous les nœuds i du graphe G.

Supposons que nous ayons calculé l'arbre des plus courts chemins depuis les nœuds o et d dans le cas de la direction sortante. Par la suite, le problème consiste à trouver l'ensemble des chemins valides tels que pour chaque nœud, il existe au moins un plus court chemin valide partant de o à j. La contrainte de regret 8.5 est définie pour chaque critère comme suit :

$$\begin{cases} w_1(o,d) + l_1^*(d,j) \leq l_1^*(o,j) + K_1, j \in V' & (8.7) \\ w_2(o,d) + l_2^*(d,j) \leq l_2^*(o,j) + K_2, j \in V' & (8.8) \\ w_3(o,d) + l_3^*(d,j) \leq l_3^*(o,j) + K_3, j \in V' & (8.9) \end{cases}$$

Cela veut dire : $\forall j \in V$ tel que 8.7 $\vee$ 8.8 $\vee$ 8.9 $\Longrightarrow j \in V'$

La résolution du problème *FRP* nécessite de faire face à plusieurs difficultés en pratique. En effet, la difficulté est de calculer les plus courts chemins dans les deux directions pour plusieurs critères depuis plusieurs sources. Par ailleurs, le graphe est stocké dans une base de données orientée graphe où les propriétés (critères) sont stockées séparément des fichiers des nœuds et arcs. Donc, le défi pour implémenter des algorithmes sur cette base est de minimiser les E/S supplémentaires. En effet, les fichiers des propriétés sont stockés séparément des fichiers des nœuds et des relations dans une base de données orientée graphe. Cependant, cet accès est statique en théorie des graphes.

Pour résoudre le *FRP*, nous avons proposé différentes méthodes de résolution. La figure 8.6 illustre l'arborescence des méthodes de résolution. La première méthode repose sur les requêtes de *Cypher*, étant donné que le graphe est stocké dans la base de données orientée graphe `Neo4j`. Ces requêtes utilisent uniquement des fonctions disponibles dans `Neo4j` (la librairie *APOC*, voir la section 5.6.4), notamment l'algorithme `Bidirectionnel` de `Dijkstra` de recherche du plus court chemin entre deux nœuds (voir la section 5.6). En effet, la variante `Bidirectionnel` était la seule variante implémentée de l'algorithme `Dijkstra` en `Neo4j` au début de ce travail de thèse (voir le chapitre 5).

Ensuite nous avons opté pour une approche algorithmique, nous avons ainsi proposé deux catégories d'algorithmes. La première catégorie décompose le *FRP* en problèmes de plus courts chemins et les résout en parallèle avec les algorithmes de `Dijkstra` ou `Bellman`. Nous avons choisi `Bellman` comme deuxième algorithme de plus court chemin car il peut être rapide si les plus courts chemins n'ont pas beaucoup d'arcs ; c'est le cas des réseaux aériens. La deuxième catégorie est basée sur l'accélération. Le premier algorithme est une extension de l'algorithme `Dijkstra` couplé à des techniques d'accélération pour éviter les calculs inutiles ; le deuxième algorithme étend l'algorithme de `Bellman` pour calculer tous les plus courts chemins pour toutes les directions et tous les critères à la fois. Tous ces algorithmes effectuent un calcul en parallèle lorsque cela est possible.

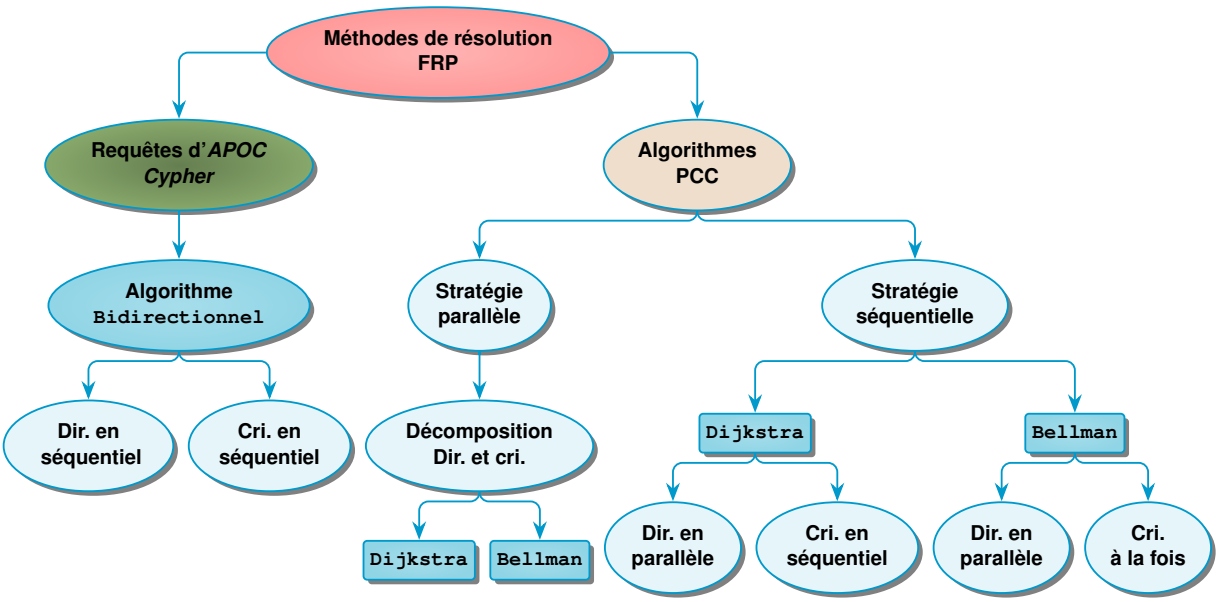


FIGURE 8.6 – Schéma des méthodes de résolution du *FRP*. **Dir.** est pour direction et **Cri.** est pour critère.

Le tableau 8.2 présente les noms des cinq méthodes proposées pour résoudre le problème de *FRP*.

Nom	Désignation
APOC	basée sur les requêtes *Cypher*
DCP_Dij	décomposition basée sur `Dijkstra`
DCP_Bel	décomposition basée sur `Bellman`
SEQ_Dij	algorithme séquentiel basé sur `Dijkstra`
SEQ_Bel	algorithme séquentiel basé sur `Bellman`

TABLE 8.2 – Tableau des noms des méthodes de résolution du *FRP*.

8.3 Méthode basée sur les requêtes *Cypher*

8.3.1 Processus de résolution

La résolution du problème *FRP* avec la méthode basée sur `APOC` se fait ainsi :

— pour chaque direction : entrante, sortante ;

— pour chaque nœud : origine, destination ;

— pour chaque critère : temps, coût et distance.

La requête 8.1 présente la requête *Cypher* simplifiée car pour un seul critère dans une seule direction. Cette requête fait appel à l'algorithme `Dijkstra` implémenté en `Neo4j` comme une procédure d'*APOC*. Bien que l'algorithme `Dijkstra` soit capable de trouver le plus court chemin à partir d'un nœud vers tous les autres nœuds, en `Neo4j` l'algorithme cherche le plus court chemin entre une paire de nœuds source et destination. Il utilise l'algorithme `bidirectionnel` de `Dijkstra`.

8.3.2 Requête d'APOC

8.3.2.1 Syntaxe

La requête du listing 8.1 résout le problème de *FRP* pour un seul critère dans une seule direction sur le graphe condensé transformé. Cette requête représente une partie du processus. Pour résoudre tout le problème, le même processus doit être répété pour chaque critère restant, et puis pour tous les critères dans l'autre direction.

En `Neo4j`, on utilise une requête paramétrée dont les paramètres sont décrits ci-dessous :

— `o_code` : spécifie le code de l'aéroport d'origine o ;

— `d_code` : spécifie le code de l'aéroport destination d ;

— `rel_type` : identifie le type de la relation à traverser ;

— `criterion` : spécifie le nom du critère (temps, coût ou distance) ;

— `regret` : détermine la valeur du regret associée au critère.

La requête 8.1 contient trois blocs majeurs. Le premier bloc (lignes 1 à 3) commence par la clause `MATCH` pour trouver le *pattern* dans le graphe qui est l'arc (o, d) en utilisant les paramètres. Le deuxième bloc (lignes 4 à 6) consiste à appeler l'algorithme `Dijkstra`. Ensuite, la procédure est appelée depuis l'origine o vers tous les autres aéroports A du graphe afin de trouver les plus courts chemins en terme de temps, puis trouver les plus courts chemins depuis la destination d.

La deuxième clause `WITH` permet d'agréger les sorties de la première procédure. En effet, les opérations en *Cypher* s'exécutent par ligne. Une ligne représente un résultat comme dans une requête `SQL`. Cela veut dire que le résultat d'une clause est l'entrée de la clause suivante. C'est réellement la taille du résultat qui détermine le nombre d'appels de la prochaine procédure. Le bloc final (lignes 7 à 9) commence par une clause `UNWIND` pour désagréger les précédentes sorties que nous avons agrégées dans la ligne 5. La dernière clause `WITH` filtre l'ensemble des chemins suivant la contrainte de regret. À la fin, la clause `RETURN` renvoie les aéroports supportés.

```
1 MATCH p=(Td:Destination)-[:CONNECT_TO]->(To:Origin{code:{o_code}})-[r]->(d:
      Destination{code:{d_code}}),(A:Destination)
2 WHERE NOT A IN [d,Td] AND type(r) = {rel_type}
3 WITH r.duration_min-{regret} AS LB,To,d,A
4 CALL apoc.algo.dijkstra(To,A,{rel}+'>|CONNECT_TO>',{criterion}) YIELD path AS
      p1,weight AS w1
5 WITH DISTINCT A,collect(w1) AS W1,LB,d
6 CALL apoc.algo.dijkstra(d,A,{rel}+'>|CONNECT_TO>',{criterion}) YIELD path AS p2
      , weight AS w2
7 UNWIND W1 AS w1
8 WITH w1-w2 AS diff,LB,A WHERE diff>=LB
9 RETURN DISTINCT A
```

Listing 8.1 – résolution avec une requête *Cypher* pour un seul critère

8.3.2.2 Explications

Recherche de l'arc étudié

Dans le *CFN*, nous avons inséré pour chaque aéroport deux terminaux correspondant aux départs (`Origin`) et aux arrivées (`Destination`) (voir la figure 7.4). L'arc étudié (o, d) correspond ainsi dans le graphe condensé au *pattern* suivant : `(To:Origin{o_code})-[r]->(d:Destination{code:{d_code}})`.

Pour calculer les plus courts chemins entre tous les nœuds de types `Origin` aux nœuds types `Destination`, il faut exclure le nœud correspondant au type `Destination` de l'aéroport o (ligne 2). Cela permettra d'éviter les cycles (voir la figure 8.7). Le *pattern* recherché devient ainsi (ligne 1) :

```
(Td:Destination)-[:CONNECT_TO]->(To:Origin{o_code})-[r]
->(d:Destination{code:{d_code}})
```

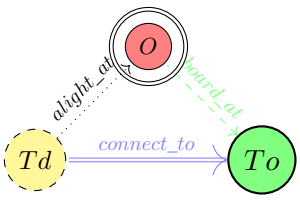


FIGURE 8.7 – Représentation de l'entité aéroport dans *CFN* transformé.

Démarches de calculs

Le critère choisi est le temps `duration_min`. D'après la contrainte de regret 8.7, on commence par calculer les membres connus de la contrainte (ligne 3), la nouvelle variable est `LB`. Ensuite on applique l'algorithme depuis l'origine `o` vers toutes les destinations `A` en tenant compte des deux types d'arcs :

1. `CONNECT_TO`, pour le transfert entre les terminaux.
2. `rel`, le `year_month` choisi.

L'algorithme renvoie une liste des chemins avec leurs longueurs qu'on stocke dans la collection `W1` : cette valeur correspond à la valeur $l^{\star}(o, j)$. On répète le même algorithme depuis la destination d vers toutes les destinations `A` (ligne 6). En effet, il faut inclure le temps de transfert à l'aéroport destination de l'arc (o, d).

Finalement la ligne 8 permet d'évaluer les chemins suivant la contrainte de regret pour le critère 1 :

$$w_1(o,d) + l_1^{\star}(d,j) \leq l_1^{\star}(o,j) + K_1$$

On peut réécrire cette contrainte de telle sorte qu'on regroupe les termes fixes d'un coté et d'un autre coté les termes variables :

$$\underbrace{l_1^{*}(o,j) - l_1^{*}(d,j)}_{\text{w1-w2=diff}} \geq \underbrace{w_1(o,d) - K_1}_{\text{LB}}$$

8.3.2.3 Plan d'exécution

Voici le plan d'exécution de la requête :

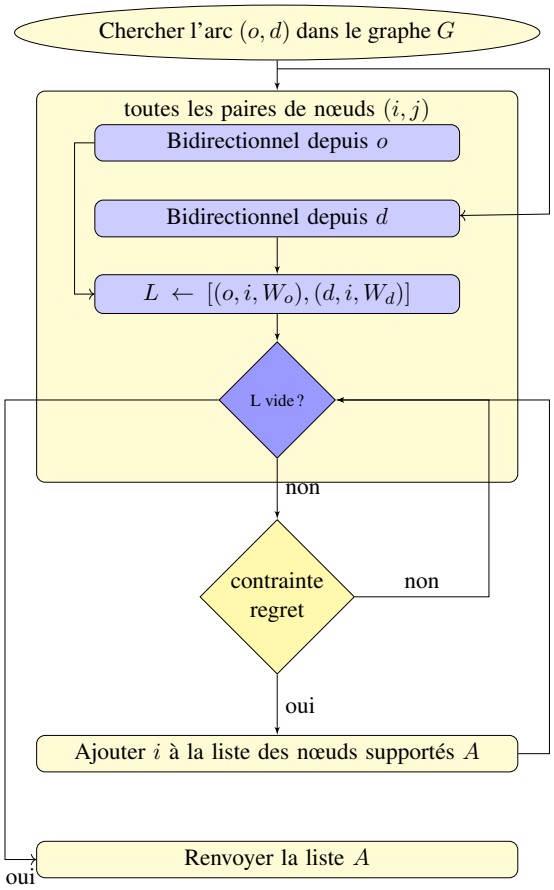


FIGURE 8.8 – Diagramme représentant la requête 8.1.

8.3.3 Limites de la méthode basée sur *APOC*

En termes de complexité, la méthode basée sur les procédures présente plusieurs limites. En termes de complexité temporelle, la requête fait appel à l'algorithme `bidirectionnel` de `Dijkstra`, qui calcule le plus court chemin entre une paire de nœuds. Ainsi, l'algorithme est répété pour chaque paire de nœuds séparément afin de trouver le plus court chemin d'un nœud à tous les autres nœuds. De nombreux calculs sont répétés. La requête d'*APOC* exécute l'algorithme bidirectionnel de o (respectivement d) vers tous les nœuds dans le graphe condensé G. Comme la complexité de cet algorithme est de l'ordre de $\mathcal{O}(m + n \log n)$, alors le temps d'exécution est de $\mathcal{O}(n \times (m + n \log n))$.

En termes de complexité spatiale, nous obtenons une liste de chemins à la place d'un arbre des plus courts chemins. Cela implique que cette complexité est bornée par n^2. En effet, la requête exécute plusieurs fois l'algorithme bidirectionnel. Ainsi, au lieu d'avoir n dans le cas d'un arbre des plus courts chemins cherché, nous avons n^2.

8.4 Méthodes basées sur les algorithmes des plus courts chemins

D'après l'analyse de la complexité et le résultat des expériences pour la méthode *APOC*, cette dernière prend trop de temps et de mémoire, car de nombreux calculs sont répétés. Nous avons ainsi proposé plusieurs algorithmes basés sur des algorithmes de plus courts chemins.

Dans cette section, nous présentons quatre algorithmes pour résoudre le *FRP*. Les deux premiers algorithmes se basent sur une décomposition du *FRP* en des problèmes de plus court chemin et procèdent en résolution en parallèle les directions et les critères en utilisant l'algorithme `Dijkstra` ou `Bellman`.

Le troisième algorithme étend l'algorithme de `Dijkstra` avec plus d'accélérations en évitant les calculs inutiles. Le quatrième algorithme étend l'algorithme de `Bellman` pour calculer tous les plus courts chemins pour tous les critères en une seule fois. Ce dernier algorithme entraîne une augmentation de la mémoire nécessaire puisque les processeurs accèdent tous en même temps à la mémoire.

Nous proposons ainsi de combiner l'exécution en séquentiel et en parallèle pour optimiser le temps d'exécution et la consommation de la mémoire. Ces algorithmes effectuent des calculs en parallèle quand c'est possible. Tous les algorithmes ont les mêmes entrées et sorties [Idrissi. et al., 2019].

---

input : le graphe $G = (\mathcal{X}, \mathcal{U})$, l'arc (o, d), l'ensemble des critères $\mathcal{W}$
output : l'ensemble des nœuds supportés S

Pour des raisons de simplicité, les algorithmes renvoient l'ensemble des nœuds supportés et s'occupent du stockage des parents des nœuds supportés. En pratique, les algorithmes renvoient également l'union des arbres supportés pour chaque direction et critère. En se basant sur les contraintes de regret, définissons :

$$R(i, dir, w) = \begin{cases} w(o,d) + l^\star(d,i) \leq l^\star(o,i) + K & \text{si } (dir = sor) \\ w(o,d) + l^\star(o,i) \leq l^\star(d,i) + K & \text{si } (dir = ent) \end{cases}$$

8.4.1 Méthodes basées sur la parallélisation

Nous proposons deux variantes de l'algorithme de `Dijkstra` et `Bellman` adaptées pour la résolution du *FRP*.

8.4.1.1 Méthode de décomposition

La méthode de décomposition basée sur les plus courts chemins résout le *FRP* en utilisant deux problèmes de plus courts chemins, un à partir de l'origine o et l'autre depuis la destination d, pour chaque direction et chaque critère en parallèle. Ainsi, les nœuds supportés sont calculés par la vérification de la contrainte de regret.

L'algorithme de plus court chemin appelé est soit l'algorithme de `Dijkstra` ou bien celui de `Bellman` (ligne 3 et 4 de l'algorithme 5). Le calcul des plus courts chemins depuis l'origine et depuis la destination se fait en séquentiel pour simplifier la recherche des nœuds supportés.

Algorithme 5 : *DCP* Decomposition

```
foreach dir ∈ {in, out} in parallel
    foreach w ∈ W in parallel
        ShortestPaths(o, w, dir);
        ShortestPaths(d, w, dir);
        foreach i ∈ V do
            if R(i, dir) then
                S ← S ∪ {i};
            end
        end
    end
end
return S
```

8.4.1.2 Exemple d'application de l'algorithme

Nous appliquons l'algorithme de décomposition sur le même exemple du graphe 8.13 pour le même critère coût avec le même regret : $K = 100\$$. Comme nous avons déjà calculé l'arbre des plus courts chemins pour la direction sortante avec l'algorithme du *PCC* : `Dijkstra`, nous présentons alors cet exemple avec cet algorithme.

La première étape consiste à calculer les arbres des plus courts chemins depuis l'origine `NCE` dans les deux directions (voir la figure 8.9). L'arbre des *PCC* pour la direction entrante est en couleur marron tandis que la couleur orange est pour la direction sortante.

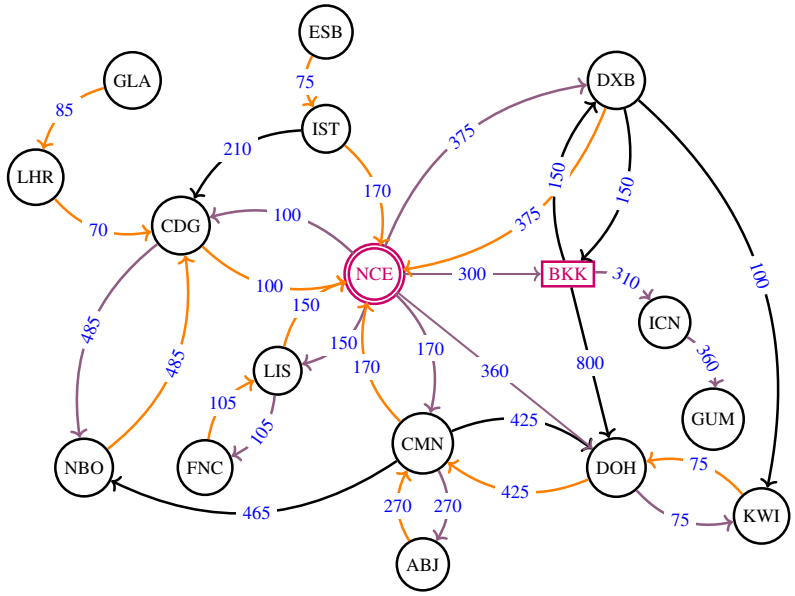


FIGURE 8.9 – Graphe avec les arbres des plus courts chemins depuis l'origine

La deuxième étape consiste à calculer les arbres des plus courts chemins depuis la destination BKK dans les deux directions (voir la figure 8.10). L'arbre des *PCC* pour la direction entrante est en couleur verte tandis que la couleur bleue est pour la direction sortante.

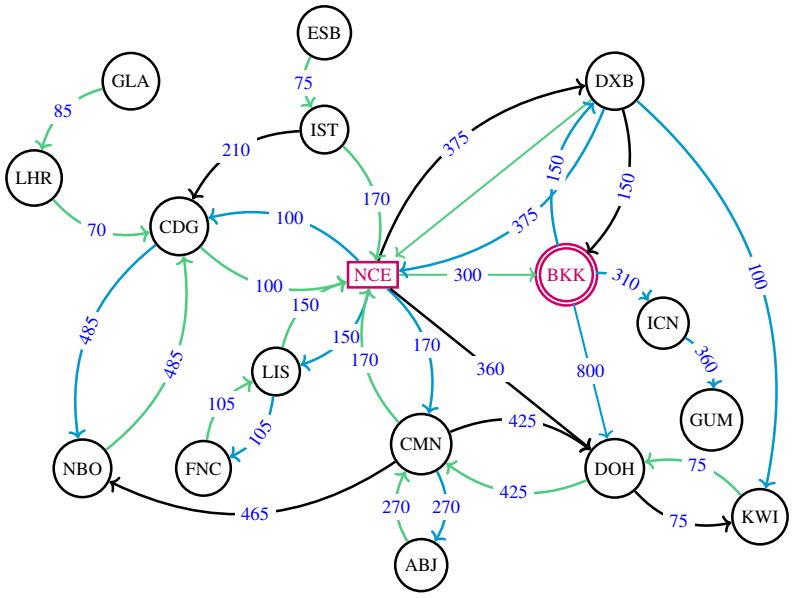


FIGURE 8.10 – Graphe avec les arbres des plus courts chemins depuis la destination.

La prochaine étape consiste à vérifier pour chaque nœud du graphe de départ pour chaque direction et chaque critère s'il satisfait la contrainte de regret 8.5 (ligne 5-7 de l'algorithme 5).

Le graphe de la figure 8.11 illustre la solution obtenue. Les nœuds en couleur verte (respectivement bleue) représentent ceux qui sont supportés pour la direction entrante (respectivement sortante). On trouve la même solution calculée par l'algorithme 8.4.2.1 (voir la figure 8.15).

Les nœuds DOH et KWI ne sont pas supportés car ce n'est pas intéressant de faire un détour par NCE-BKK pour un regret de 100$.

La majorité des nœuds sont supportés car le fait de passer par NCE-BKK reste le seul chemin pour rejoindre BKK. D'où l'idée de l'algorithme 8.4.2.1 qui est basé sur le lemme 8.1.1.

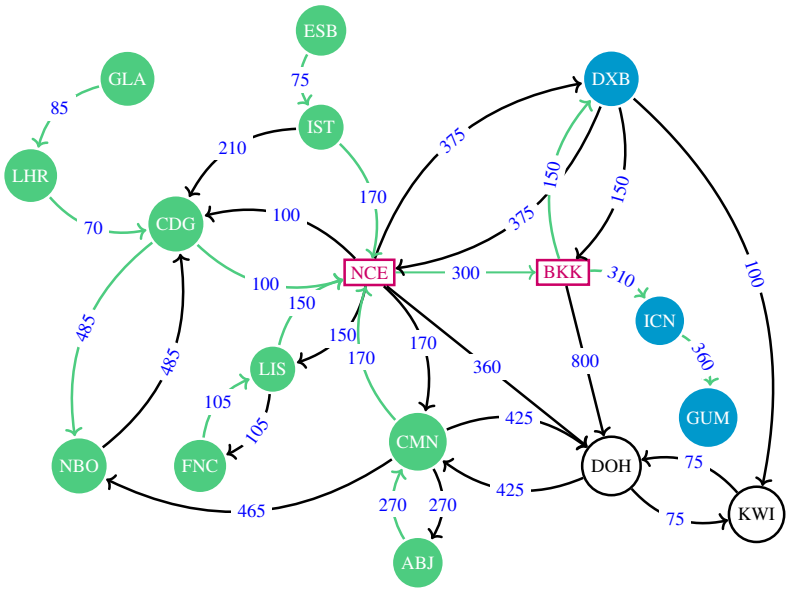


FIGURE 8.11 – Graphe solution du *FRP*

8.4.2 Méthodes séquentielles

Nous commençons par présenter l'algorithme basé sur `Dijkstra`. Ensuite, nous passons à la version générale pour la résolution du *FRP* pour plusieurs critères dans les deux directions. Enfin, nous présentons l'algorithme basé sur `Bellman`.

8.4.2.1 Méthode basée sur l'algorithme `Dijkstra`

Comme le graphe condensé est un graphe orienté pondéré par des poids positifs, nous avons proposé un algorithme qui est une extension de l'algorithme de `Dijkstra` mais en incluant plusieurs optimisations pour accélérer le calcul. Nous avons proposé une version bicritère :

1. Si l'arc (o, d) est présent dans le plus court chemin, alors c'est un chemin valide (voir le lemme 8.1.1).
2. Si le nœud visité n'est pas supporté, c'est-à-dire qu'il ne satisfait pas la contrainte de regret, alors ses successeurs ne seront pas supportés.
3. Vérification de la contrainte de regret en calculant une borne supérieure sur le deuxième critère.

Explication de l'algorithme

Tout d'abord, nous commençons par considérer un seul critère et ensuite on passe au deuxième en utilisant à chaque étape les sorties de l'étape précédente. Ainsi, l'algorithme commence par calculer l'arbre du plus court chemin depuis l'origine et vérifie si l'arc (o, d) existe dans cet arbre. Un tel chemin est considéré comme un chemin valide d'après le lemme 8.1.1. En effet, nous devons commencer par l'origine comme nous sommes centrés sur l'arc (o, d). L'algorithme s'arrête quand tous les nœuds ont été traités.

Après avoir trouvé l'arbre du plus court chemin depuis o et déterminé la présence des chemins valides grâce au lemme 8.1.1, nous passons au calcul de l'arbre du plus court chemin depuis la destination d pour pouvoir vérifier la contrainte de regret pour les nœuds restants. Un cas de figure se présente : les chemins passant par le nœud de l'origine o qu'on néglige. En effet, selon la propriété du problème *FRP*, nous sommes intéressés par les chemins qui ne passent pas par l'origine (voir l'arbre à gauche en bas (2) de la figure 8.12).

Pour vérifier la contrainte de regret, nous l'intégrons au fur et à mesure dans le calcul de borne. À chaque itération, l'algorithme scanne le nœud ayant le poids minimum et puis relâche ses successeurs. Finalement, on vérifie pour chaque nœud s'il satisfait la contrainte de regret avant de scanner ses successeurs. S'il ne la satisfait pas, on passe au prochain dans la file. En effet, on exploite le fait que `Dijkstra` cherche le plus court chemin dans un ordre croissant de distance. Avec cette manière, nous éliminons facilement les nœuds qui ne sont pas supportés. À la fin de cette étape, nous obtenons deux informations, la longueur du plus court chemin $l_1^\star(d,j)$ pour le premier critère que nous avons utilisé pour la vérification de la contrainte de regret et une borne supérieure pour le deuxième critère $\overline{l_2}(d,j)$. Cette étape est illustrée par l'arbre (3) de la figure 8.12.

Une fois que nous avons déterminé l'ensemble des nœuds supportés, si tous les nœuds du graphe ont été traités alors l'algorithme s'arrête. Sinon, nous passons au deuxième critère pour les nœuds qui ne sont pas supportés.

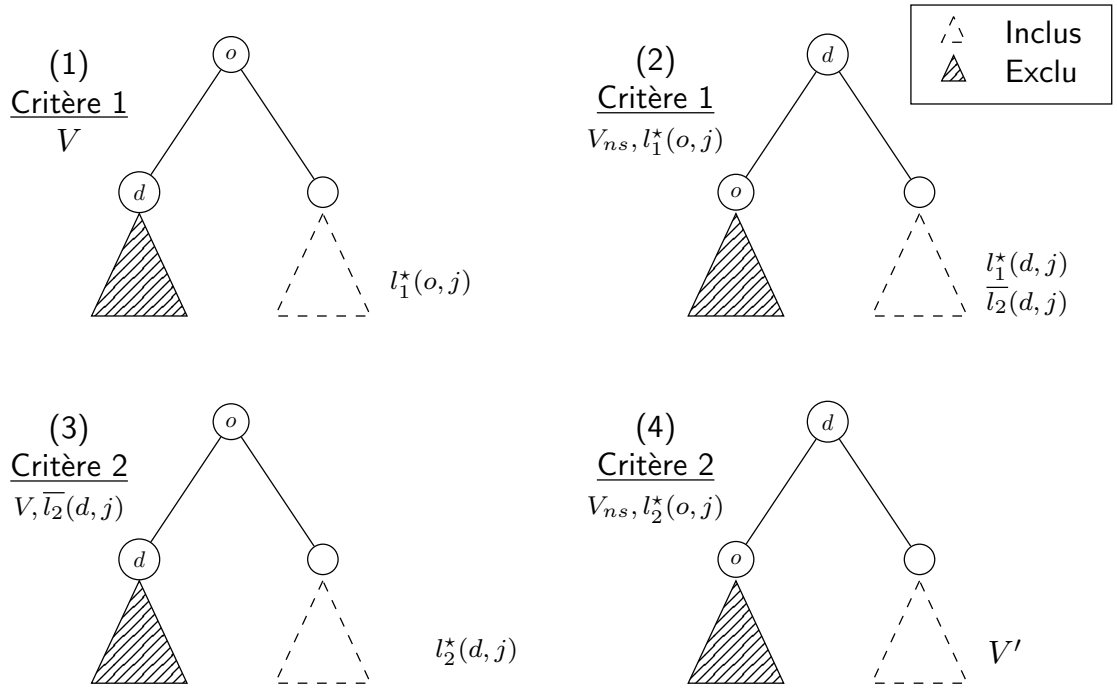


FIGURE 8.12 – Étapes de l'algorithme basé sur l'algorithme de `Dijkstra`. La partie gauche de chaque arbre représente les entrées à chaque itération, alors que la partie droite les sorties. L'étiquette "Exclu" correspond à l'ensemble des nœuds qui seront exclus dans la prochaine itération.

Nous répétons le même processus pour le deuxième critère. La troisième étape (voir l'étape 3 dans la figure 8.12) est similaire à la première étape. Une fois que l'arbre du plus court chemin

depuis l'origine pour le deuxième critère est calculé, nous utilisons la borne supérieure calculée dans l'étape précédente pour vérifier la contrainte de regret. Comme c'est une borne supérieure, alors, si la contrainte de regret est vérifiée pour cette borne, elle sera vérifiée pour la valeur du plus court chemin pour ce critère :

$$w_2(o,d) + \overline{l_2}(d,i) \leq l_2^\star(o,i) + K_2$$

Finalement, nous appliquons le même processus pour l'ensemble des nœuds qui restent non supportés pour chercher l'arbre du plus court chemin depuis la destination d.

Pseudo-code de l'algorithme

L'algorithme 7 présente le pseudo-code de l'algorithme basé sur `Dijkstra`, V_{ns} présente l'ensemble des nœuds non supportés. Dans cet algorithme, on utilise quatre fonctions :

1. `Dijkstra` calcule les plus courts chemins ;
2. `odCheck` vérifie si l'arc (o,d) appartient à l'arbre du plus court chemin ;
3. `UBCheck` vérifie la contrainte de regret pour le deuxième critère avec la borne supérieure sur les plus courts chemins ;
4. `RD` représente la version revisitée de l'algorithme `Dijkstra` qu'on applique depuis la destination.

L'algorithme peut être divisé en quatre blocs selon les étapes décrites dans la figure 8.12 :

1. étape 1 correspond aux lignes 1-4 ;
2. lignes 5-8 correspondent à l'étape 2 qui calcule le plus court chemin depuis d ;
3. étape 3 correspond aux lignes 12-13 ;
4. ligne 21 correspond à l'étape 4.

Algorithme 6 : SEQ_Dij Algorithm

procedure : **SEQ_Dij** (ensemble de nœuds $\mathcal{X}$, arc od, parameter K)
input : A digraph $G = (\mathcal{X}, \mathcal{U}, \mathcal{W})$, arc $od = (o, d)$, parameter K_1, parameter K_2
output : Subgraph $G' = (\mathcal{A}, \mathcal{V})$

$\mathcal{X}_{ns} \leftarrow \emptyset$;
$\mathcal{V} \leftarrow \emptyset$;
$T_{1o} \leftarrow$ `Dijkstra` $(o, \mathcal{X}, \mathcal{W}_1)$;
$\{\mathcal{X}_{ns}, \mathcal{V}\} \leftarrow$`odCheck` $(od, T_{1o}, \mathcal{X}_{ns}, \mathcal{V})$;
if $\mathcal{X}_{ns} = \emptyset$ **then**
  break
else
  $\{\mathcal{X}_{ns}, \overline{T}_{2d}, \mathcal{V}\} \leftarrow$ `RD` $(G, d, \mathcal{X}_{ns}, \mathcal{W}_1, \mathcal{W}_2, T_{1o}, K_1, \mathcal{V})$;
  if $\mathcal{X}_{ns} = \emptyset$ **then**
    break
  else
    $T_{2o} \leftarrow$ `Dijkstra` $(o, \mathcal{X}_{ns}, \mathcal{W}_2)$;
    $\{\mathcal{X}_{ns}, \mathcal{V}\} \leftarrow$`odCheck` $(od, T_{2o}, \mathcal{X}_{ns}, \mathcal{V})$;
    if $\mathcal{X}_{ns} = \emptyset$ **then**
      break
    else
      $\{\mathcal{X}_{ns}, \mathcal{V}\} \leftarrow$ `UBCheck` $\{\mathcal{X}_{ns}, T_{2o}, \overline{T}_{2d}, \mathcal{V}\}$;
      if $\mathcal{X}_{ns} = \emptyset$ **then**
        break
      else
        $\{\mathcal{X}_{ns}, \overline{T}_{1d}, \mathcal{V}\} \leftarrow$ `RD` $(G, d, \mathcal{X}_{ns}, \mathcal{W}_1, \mathcal{W}_2, T_{2o}, K_2, \mathcal{V})$;
      end
    end
  end
end
$\mathcal{A} \leftarrow \mathcal{X} \setminus \mathcal{X}_{ns}$;
return $G' = (\mathcal{A}, \mathcal{V})$;

Exemple d'application de l'algorithme

Dans cette section, nous allons montrer un exemple d'application de l'algorithme proposé pour la preuve de concept. Considérons l'exemple d'un graphe représentant un réseau réel de vols (voir la figure 8.13) : 17 nœuds présentant des aéroports et 33 arcs qui correspondent aux vols. Ce graphe correspond au graphe du problème *2-hops* (voir la figure 5.1 du chapitre 5).

On suppose qu'une compagnie aérienne souhaite ajouter un nouveau vol entre `NCE` et `BKK` qui est représenté par un arc rouge en pointillé dans la figure 8.13. Les étiquettes correspondent aux prix des vols. Le graphe est extrait de l'application `PlanetOptim`.

Par la suite, nous allons présenter le déroulement de l'algorithme pour un seul critère (temps) et pour une seule direction (sortante). On suppose qu'on a un regret de ($K = 100\$$) concernant ce critère. L'algorithme, ainsi, se déroulera en trois étapes :

1. calcul des plus courts chemins depuis l'origine ;

2. vérification de l'existence de l'arc (o, d) ;
3. calcul des plus courts chemins depuis la destination.

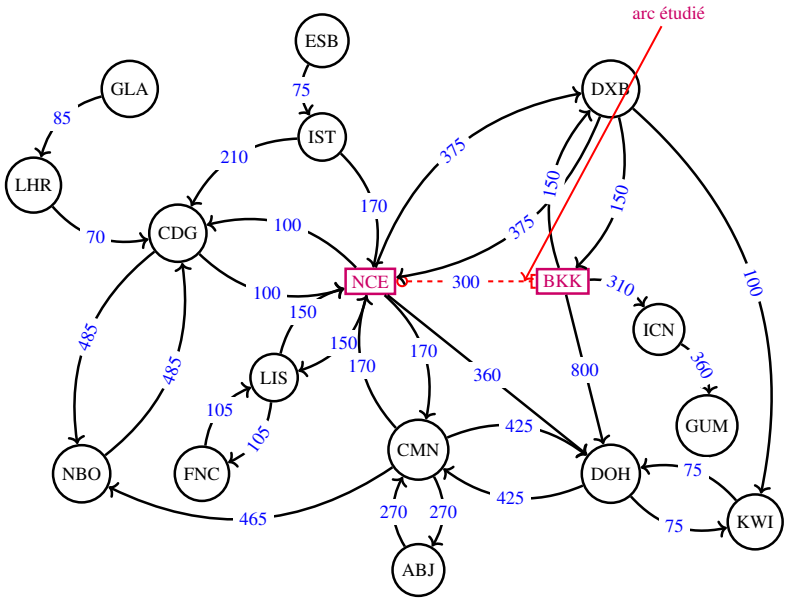


FIGURE 8.13 – Graphe de départ

Calcul des plus courts chemins depuis l'origine

La figure 8.14 illustre le graphe obtenu après avoir appliqué l'algorithme de `Dijkstra` depuis le nœud `NCE` sur le graphe présenté dans la figure 8.13. L'arbre des plus courts chemins à partir de l'origine `NCE` est représenté en couleur marron. Le nœud `NCE` doublement cerclé correspond à la racine de cet arbre.

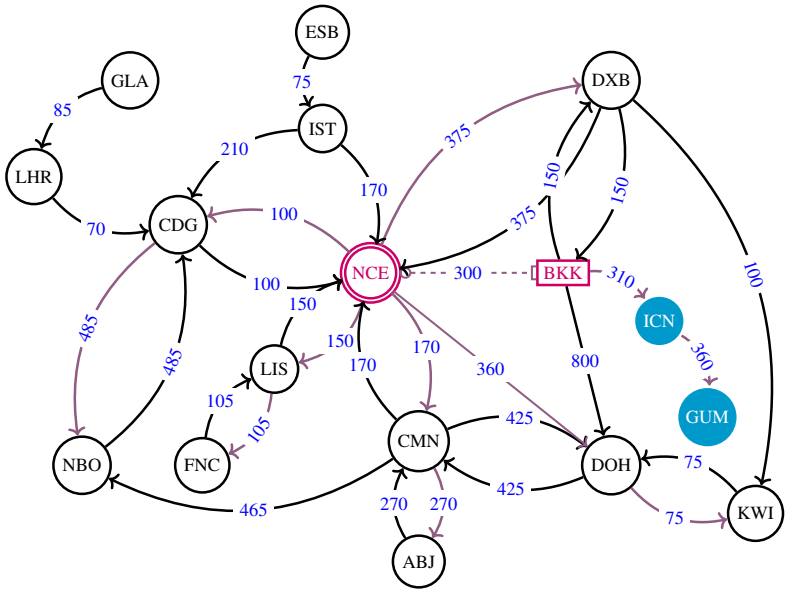


FIGURE 8.14 – Graphe avec l'arbre du plus court chemin depuis l'origine en marron.

Vérification de l'existence de l'arc (o, d)

Ensuite, on vérifie si l'arc `(NCE,BKK)` existe dans cet arbre. Dans cette étape, on obtient `{ICN,GUM}` comme l'ensemble des nœuds supportés. Ainsi, il nous reste l'ensemble des autres nœuds du graphe. Pour ce faire, nous avons besoin du coût minimum depuis la destination `BKK` vers ces nœuds.

Calcul des plus courts chemins depuis la destination

L'étape suivante consiste à calculer l'arbre des plus courts chemins depuis la destination `BKK`. La couleur bleue représente les arcs de cet arbre. Suivant la figure 8.14, on voit qu'un passager venant de `NCE`, s'envolant à `CDG` depuis `BKK` ne sera pas intéressé par un tel chemin, car il est plus cher.

À l'issue de cette étape, on trouve le nœud `DXB` qui devient supporté suivant la contrainte de regret 8.5 (voir la figure 8.15).

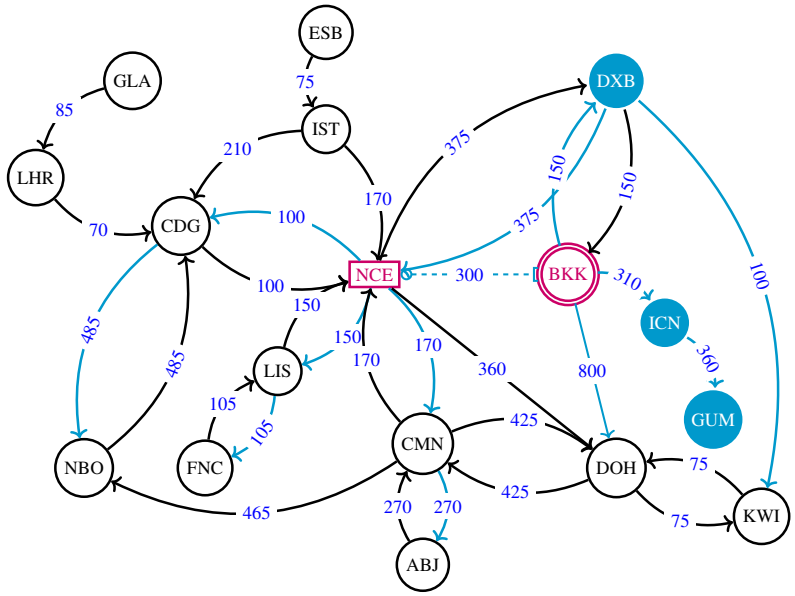


FIGURE 8.15 – Graphe avec l'arbre du plus court chemin depuis la destination en bleu.

Limites de l'algorithme basé sur `Dijkstra`

Ainsi, pour chaque critère, l'algorithme de `Dijkstra` est appelé quatre fois. Comme nous avons trois critères, alors l'algorithme sera appelé 12 fois. D'une manière générale, si on a m critères alors l'algorithme sera appelé $4 * m$ fois.

En termes de complexité, l'algorithme fait appel à l'algorithme de `Dijkstra` dans les pires cas quatre fois dans le cas du bi-critère. Les optimisations nécessitent plus de temps donc nous avons proposé de nous baser sur cet algorithme pour résoudre le problème *FRP* dans le cas général. Cette généralisation tient compte de plus de deux critères avec la possibilité de paralléliser par direction, le processus étant le même. Cela permet de ne pas garder toutes les propriétés de l'algorithme pour exclure les dépendances.

8.4.2.2 Généralisation de l'algorithme `Dijkstra`

Nous reprenons l'algorithme présenté dans la section 8.4.2.1. Cet algorithme calcule le plus court chemin à partir de o, et ensuite calcule le plus court chemin depuis d en ignorant les nœuds non supportés. Ici, les critères sont traités séquentiellement. À chaque itération, l'algorithme scanne le nœud avec le poids minimum et ensuite relâche ses voisins. Ainsi, on vérifie que si le nœud satisfait la contrainte de regret sinon on ignore le nœud. De cette façon, nous ignorons les nœuds non supportés qui ne peuvent pas nous amener à des chemins valides. L'algorithme se termine si la file devient vide ou si les plus courts chemins vers tous les nœuds non supportés ont été trouvés. La fonction `SCAN(i,w,dir)` appelle la fonction présentée dans la section 4.3 pour chaque critère et direction.

Cette deuxième méthode qui est basée sur une adaptation de l'algorithme de `Dijkstra` de recherche des plus courts chemins partant d'un nœud repose sur deux idées principales. Le problème n'est plus résolu indépendamment pour chaque critère et l'espace de recherche est réduit

Algorithme 7 : SEQ_Dij Algorithm

foreach $dir \in \{in, out\}$ **in parallel**
  foreach $w \in \mathcal{W}$ **do**
    `Dijkstra` (o, w, dir) ;
    `enqueue` (Q, d) ;
    while $Q \neq \emptyset$ *and* `isNotClosed()` **do**
      $i \leftarrow \text{argmin}_{j \in Q}(d[j])$;
      `dequeue` (Q, i) ;
      if $R(i, dir)$ **then**
        $S \leftarrow S \cup \{i\}$;
        `SCAN` (i, w, dir);
      end
    end
  end
end
return S

entre deux appels consécutifs de l'algorithme de `Dijkstra`. À chaque appel de l'algorithme de `Dijkstra`, les nœuds sont ouverts seulement lorsqu'un chemin valide les atteint.

8.4.2.3 Méthode basée sur l'algorithme `Bellman`

Nous proposons une variante de l'algorithme de `Bellman` qui calcule tous les plus courts chemins en une seule fois. Ainsi, les plus courts chemins sont calculés depuis l'origine et depuis la destination pour tous les critères. Ensuite, il y aura plus de nœuds qui seront ajoutés à la file. Ici, la fonction `SCAN` met à jour tous les chemins depuis l'origine et depuis la destination pour tous les critères.

Algorithme 8 : SEQ_Bel Algorithm

foreach $dir \in \{in, out\}$ **in parallel**
  `enqueue` (Q, o);
  `enqueue` (Q, d);
  while $(Q \neq \emptyset)$ **do**
    $i \leftarrow$ `dequeue`(Q);
    `SCAN` $(i, \mathcal{W}, dir)$;
  end
  foreach $i \in V$ **do**
    if $R(i, dir)$ **then**
      $S \leftarrow S \cup \{i\}$;
    end
  end
end
return S

8.5 Expériences

Cette section présente l'ensemble des expériences réalisées pour tester les algorithmes proposés pour résoudre le *FRP*. Ainsi, nous évaluons les deux types d'approches, basées sur une requête et sur des algorithmes.

Tout d'abord, nous commençons par décrire les données utilisées dans ces expériences. Ensuite, nous passons à la présentation du protocole des trois expériences réalisées sur ces données pour évaluer nos algorithmes. Finalement, nous concluons par les résultats.

Toutes les expériences ont été conduites sur un ordinateur avec comme système d'exploitation Ubuntu 16.04.5, 32 Go de RAM et un processeur Intel Core i7-3930K 3.20 GHz (6 cœurs). L'implémentation est basée sur Neo4j et APOC version 3.2.0. Tous les algorithmes sont implémentés en Java 8.

8.5.1 Description des données

Nous présentons les deux versions du graphe *CFN* sur lesquelles nous avons travaillé. En effet, le grand défi de ce travail était de gérer des données massives qui sont tout le temps modifiées dans le monde réel tout en adaptant nos algorithmes

8.5.1.1 Données de l'année 2015

La première version du graphe condensé a été générée sur l'année 2015 avec 12 mois (`year_month`). Ainsi, le graphe contient 11 300 nœuds et 512 294 arcs après avoir appliqué la transformation décrite dans la section 7.3. Ce graphe nous a servi pour tester la méthode basée sur les procédures d'*APOC* et l'algorithme proposé pour faire la preuve du concept.

8.5.1.2 Données de l'année 2017

Comme les données de *Milanamos* évoluent régulièrement. Nous avons alors généré une deuxième version de graphe pour avoir des données plus récentes mais surtout plus variées sur un historique plus long. Cela nous permet de bien tester la fiabilité et la sensibilité de nos algorithmes face aux changements des données. Suite à cela, nous avons généré un deuxième graphe condensé sur une période de deux ans : *2016* et *2017* (24 `year_month`). Ainsi, le graphe final contient 4 577 nœuds et 1 125 148 relations après avoir effectué le nettoyage et la correction des données aberrantes. Ensuite, nous avons appliqué la transformation pour obtenir un graphe de 13 732 nœuds et 1 148 303 arcs. Chaque mois correspond à un graphe différent.

8.5.2 Protocole des expériences

Nous commençons par évaluer les performances pour résoudre le problème *FRP* en utilisant une requête *APOC*. Ensuite, nous comparons le résultat obtenu avec celui de la méthode basée sur l'algorithme de `Dijkstra` dans le cas d'un seul critère. Finalement, nous passons au test des quatre algorithmes basés sur les plus courts chemins, qui ont été proposés en séquentiel et en parallèle. Les deux premières expériences dépendent l'une de l'autre. En effet, elles se sont réalisées sur les données du graphe *CFN1* et visent à répondre aux questions suivantes :

— Dans quelle mesure les performances de l'algorithme sont sensibles face au choix du paramètre K ?

— Les performances de l'algorithme se dégradent-elles lors de l'ajout d'un deuxième critère ?

— Comment le choix du paramètre K influence-t-il l'ordre du sous-graphe sortant ?

— Comment le degré de la paire `OD` de l'arc étudié influence-t-il le résultat ?

Alors l'expérience 3 est consacrée à l'évaluation des algorithmes basés sur les algorithmes des plus courts chemins. Par la suite, nous allons voir en détails la liste des questions.

8.5.2.1 Comparaison avec *APOC*

Cette première catégorie de tests concerne le test de la méthode basée sur *APOC* et l'algorithme proposé pour faire face aux limites de la première méthode. Les méthodes ont été testées sur des instances réelles. Nous verrons que la requête est très lente, nous avons décidé d'arrêter le processus à un seul critère pour une seule direction.

En plus du temps d'exécution, nous cherchons à évaluer deux métriques : l'ordre du sous-graphe résultant G' et le pourcentage des nœuds qui sont filtrés suivant les différentes valeurs du paramètre K.

8.5.2.2 Résolution du *FRP* dans le cas bicritère

Dans cette expérience, nous cherchons à élargir l'ensemble des tests de l'algorithme basé sur `Dijkstra`. Nous avons décidé d'inclure le deuxième critère et de varier les valeurs des regrets. Ainsi, nous avons réalisé les tests suivant les différentes valeurs du paramètre K qui représente le regret pour chaque critère. Ces valeurs ont été choisies avec des statistiques significatives. Elles sont choisies suivant la médiane, le premier quartile et le troisième quartile : Q_2, Q_1, Q_3. De cette façon, on peut mesurer la dispersion pour mieux décrire la variabilité de chaque critère. Soit K_1 le paramètre pour le critère de temps et K_2 pour le critère de coût. $\overline{T}$ représente la médiane du temps minimum de vol T décrit dans la section 7.2.2 :

$$\begin{cases} 0 \leq K_1 \leq Q_1(T) + MCT \\ Q_1(T) < K_1 \leq \overline{T} + MCT \\ \overline{T} < K_1 \leq Q_3(T) + MCT \\ K_1 > Q_3(T) + MCT \end{cases}$$

Dans le cas du critère coût, K_2 est choisi indépendamment du temps minimum de connexion MCT :

$$\begin{cases} 0 \leq K_1 \leq Q_1(R) \\ Q_1(R) < K_1 \leq \overline{R} \\ \overline{R} < K_1 \leq Q_3(R) \\ K_1 > Q_3(R) \end{cases}$$

Où $\overline{R}$ représente la médiane du revenu minimum R.

Fixer K à zéro, par exemple, signifie que le sous-graphe final G' contient tous les plus courts chemins passant par l'arc étudié (o, d). Toutefois, fixer K à une valeur très élevée donnera un sous-graphe qui contient tous les nœuds du graphe de départ G. On rappelle que la valeur du MCT est fixée à 120. En effet, cette valeur est la même pour tous les arcs du graphe condensé G de type *connect_to* entre les nœuds *origin* et *destination*.

D'une part, nous avons choisi 6 instances telles que les aéroports des arcs étudiés ont différentes caractéristiques notamment le degré. Le test a été fait pour résoudre le problème de *FRP* avec un seul critère. D'autre part, nous avons testé l'algorithme de la preuve de concept pour résoudre le *FRP* avec deux critères sur les mêmes instances. En effet, les résultats sont très satisfaisants dans le cas d'un seul critère. Finalement, nous avons généré 100 instances dont le choix de paires (`OD`) se fait aléatoirement. Pour la même paire `OD`, nous générons 10 tests qui varient en fonction des classes du regret K_1 et K_2 sur les deux critères : temps et coût.

8.5.2.3 Comparaison des algorithmes

Cette expérience est menée pour tester les algorithmes proposés en stratégie séquentielle et parallèle. Nous sommes particulièrement intéressés par la comparaison des temps d'exécution des algorithmes ayant un impact sur l'expérience utilisateur de `PlanetOptim`. Une autre métrique intéressante est le nombre de nœuds analysés par l'algorithme, qui détermine les valeurs des propriétés consultées (critères) d'une relation générant des E/S supplémentaires lors de leur premier accès. En effet, ces propriétés sont stockées séparément des fichiers des entités nœuds et relations (voir la section 5.5.10). Cependant, elles sont mises dans le cache après le premier accès Robinson et al. [2015]. Le protocole de cette expérience vise à répondre aux questions suivantes :

1. Quel algorithme est le plus rapide et fournira donc la meilleure expérience utilisateur ?
2. Est-il utile de proposer des algorithmes en stratégie séquentielle pour résoudre le problème *FRP* par rapport à ceux basés sur la méthode de décomposition en problème des plus courts chemins ?
3. La réduction du nombre de nœuds analysés provoque-t-elle une réduction du temps d'exécution ?
4. Comment la performance évolue-t-elle avec le nombre de critères ?

Nous rappelons que le graphe condensé de la deuxième version contient les données historiques de la période 2016 et 2017 (24 `year_month`), chaque mois correspond à un graphe différent. Chaque instance doit spécifier le mois, la paire `OD` de l'arc (o, d), le nombre de critères et leurs regrets.

Le nombre de critères est un (temps), deux (temps, distance) et trois (temps, distance, coût). Pour chaque critère, deux valeurs sont considérées pour leurs regrets K : 0 et la médiane (à travers toutes les relations du graphe y compris les relations portant les données mensuelles (type `year_month`). La valeur 0 signifie que seuls les plus courts chemins suivant l'arc (o, d) sont valides, tandis que les autres chemins sont valides par le biais de la médiane. Par exemple, la durée médiane d'un vol est environ deux heures. Ainsi, un chemin entre i et j qui passe par l'arc (o, d) est valide s'il ne dépasse pas la durée du plus court chemin entre i et j de quatre heures (durée médiane plus le temps minimum de connexion MCT).

Pour chaque mois-année, chaque nombre de critères et chaque combinaison de valeurs de critères, quelques paires d'origine et destination (o, d) sont tirées au hasard. À la fin, plus de dix mille instances ont été testées.

8.5.3 Résultats des expériences

Cette section présente les résultats des expériences décrites auparavant. La section 8.5.3.1 présente une comparaison entre la méthode basée sur la procédure *APOC* et l'algorithme proposé

pour faire la preuve de concept. La section 8.5.3.2 présente une comparaison entre les quatre algorithmes proposés dans le cadre du passage à l'échelle.

8.5.3.1 Comparaison avec APOC

La méthode basée sur une procédure *APOC* a été testée sur le graphe de l'année 2015 pour un seul critère : temps. Nous présentons les résultats en terme de nombre et pourcentage : l'ordre du sous-graphe G' et le pourcentage des nœuds filtrés.

Le tableau 8.3 présente le résultat du test des deux méthodes pour résoudre le problème *FRP* dans le cas d'un seul critère.

— $\#noeuds$: la taille du graphe sortant G' ;

— Dur : temps de vol de l'arc étudié ;

— K_1 : regret fixé pour chaque test ;

— P : pourcentage des nœuds filtrés ;

— $tpsExc1$: temps d'exécution de la méthode basée sur la procédure *APOC*, exprimé en minutes.

— $tpsExc2$: temps d'exécution de la deuxième méthode basée sur l'algorithme de `Dijkstra`, exprimé en millisecondes.

`Inst.`	`Arc`	`Dur (min)`	K_1(min)	$\#noeuds$	P	$tpsExc1$	$tpsExc2$
1	`NCE → DXB`	360	0	287	2.5 %	53	**2321**
2	`JFK → NCE`	490	198	156	1.3 %	51	1127
3	`CDG → SCL`	870	245	56	0.49 %	55	696
4	`LHR → ATL`	565	330	771	6.82 %	51	786
5	`FRA → PEK`	550	198	416	3.68 %	53	736
6	`AMS → IST`	195	0	65	0.57 %	**57**	693

TABLE 8.3 – Comparaison avec la méthode basée sur une procédure *APOC*.

Le temps d'exécution de la méthode basée sur une requête est très important. Comme `Neo4j` implémente la version bidirectionnelle de l'algorithme `Dijkstra`, l'algorithme est répété pour chaque paire de nœuds individuellement afin de trouver le plus court chemin à partir de l'origine o vers tous les autres nœuds du graphe G. Ainsi, plus de calculs sont répétés. Suivant l'analyse faite dans la section 8.3.3, la requête renvoie une liste des chemins impliquant que la requête prenne énormément de temps. Dans le pire des cas, la requête s'exécute en 57 minutes et le graphe résultat est de l'ordre de 65 nœuds. Alors, l'algorithme prend seulement 2321 ms quand il s'agit de renvoyer un graphe de 287 nœuds. Par conséquent, l'algorithme proposé est plus performant que la requête *Cypher* basée sur une procédure *APOC*.

Nous avons montré que la méthode basée sur une approche algorithmique est plus puissante que la méthode basée sur une requête pour résoudre le *FRP* avec un seul critère. Nous avons ensuite effectué un passage à l'échelle de l'algorithme pour résoudre deux critères : temps et coût. Le tableau 8.4 présente les résultats obtenus.

Nous remarquons que le pourcentage des nœuds supportés augmentent quand on ajoute un deuxième critère. Même avec un regret fixé à zéro, le pourcentage est au moins le double que dans

Instance	Dur (min)	K_1(min)	K_2(usd)	$\#\_nodes$	P	tpsExec (ms)
1	360	0	0	1200	10.62	3604
2	490	198	17.0	590	5.22	1657
3	870	245	42.98	479	4.23	1597
4	565	330	96.14	1424	12.60	1906
5	550	198	17.0	933	8.25	1742
6	195	0	0	477	4.22	1554

TABLE 8.4 – Résultats de la résolution du *FRP* dans le cas bi-critère.

le cas d'un seul critère. Cela est tout à fait normal car la recherche continue pour les nœuds qui ne sont pas supportés pour le premier critère. Donc, on a de fortes chances de trouver des nœuds qui sont supportés par le deuxième critère. On rappelle qu'il suffit qu'il existe un chemin valide pour au moins un seul critère. Donc, la vérification se fait pour les deux critères quand on a encore des nœuds non supportés.

Pour les instances 1 et 6, le sous-graphe contient tous les plus courts chemins valides passant par les arcs suivants : `(NCE,DXB)` et `(AMS,IST)` pour les deux critères temps et coût. Dans les instances 2,3 et 4, le regret est choisi respectivement selon les quartiles Q_1, Q_2 et Q_3. Pour résoudre le problème avec deux critères, l'algorithme a besoin de plus de la moitié du temps d'exécution dans le cas du premier critère. L'instance 2 nous donne le plus petit sous-graphe en terme de taille. En effet, seulement 5% de nœuds sont supportés. Ce sont les nœuds dont on accepte de faire la moitié de trajet de vol en plus (`JFK,NCE`).

	Classe coût			
Classe temps	0	1	2	3
0	1573.6	1590.4	1657.2	**2354.8**
1	1783.2	1677.0	1759.4	2246.0
2	1877.8	2007.3	1928.1	2248.0
3	**1367.0**	2271.5	1484.1	1491.5

TABLE 8.5 – Temps moyen pour résoudre le problème *FRP*.

Le tableau 8.5 nous donne le temps moyen d'exécution en fonction des deux classes du regret associées au temps et coût : K_1 et K_2. Nous constatons que le temps moyen d'exécution augmente légèrement quand les valeurs de K_1 et K_2 augmentent. L'algorithme nous donne une meilleure borne dans le cas où le paramètre K_1 est fixé à une valeur supérieure ou égale au troisième quartile qui représente 75% des temps de vols tandis que K_2, le paramètre associé au regret sur le critère de coût, est fixé à zéro. Dans le pire des cas, l'algorithme prend deux fois plus de temps d'exécution que dans le meilleur cas. Il est atteint lorsque nous échangeons les deux valeurs du regret. Ce qui revient à dire, que le temps d'exécution de l'algorithme est influencé plutôt par le critère coût, comme l'augmentation de ce dernier entraîne l'augmentation du temps de calcul de l'algorithme. En effet, ce critère tient compte du paramètre de MCT.

En conclusion, nous avons montré que la spécialisation d'un algorithme apporte un gain de performance. Nous avons vu que l'algorithme derrière la requête *APOC* est plus lent que celui que

nous avons codé proprement en Java. Donc, à l'heure actuelle, on doit coder en Java pour que cela réponde à nos questions, notamment, la contrainte du temps de réponse de l'application. On peut ensuite l'intégrer sous forme de requêtes. Ce qui permettra à l'utilisateur de facilement le tester.

8.5.3.2 Comparaison des algorithmes

Tout d'abord, nous décrivons les temps d'exécution des algorithmes à comparer, ensuite la relation entre les temps d'exécution et le nombre de nœuds analysés est étudiée. Enfin, nous analysons le rapport entre les algorithmes en stratégie parallèle et les algorithmes basés sur la décomposition en plus court chemin en fonction du nombre de critères.

Le tableau 8.6 donne la moyenne et le temps d'exécution maximal, en millisecondes, des algorithmes basés sur la décomposition en plus court chemin et des algorithmes basés sur la stratégie séquentielle utilisant `Bellman` ou `Dijkstra`. Les variantes de `Dijkstra` sont environ deux fois plus rapides et présentent un écart-type inférieur à celui de son homologue `Bellman`. L'algorithme basé sur `Dijkstra` est légèrement plus rapide que la décomposition alors que ce n'est pas le cas pour `Bellman`.

	DCP Decomposition		*SEQ* Algorithm	
	Dijkstra	Bellman	Dijkstra	Bellman
avg	266	500	231	604
std	60	198	90	282
max	418	1328	644	1612

TABLE 8.6 – Distribution des temps d'exécution en millisecondes.

La figure 8.16 analyse la relation entre le temps d'exécution et le nombre de nœuds analysés pour chaque algorithme. Chaque point représente une instance : son abscisse correspond au temps d'exécution en millisecondes, et son ordonnée correspond au nombre de nœuds analysés. La couleur d'un point indique quel algorithme a résolu l'instance. L'algorithme en stratégie séquentielle basé sur `Bellman` scanne moins de nœuds que ceux basés sur l'algorithme `Bellman` basé sur la décomposition en plus court chemin (les points rouges sont au-dessous des points bleus), mais sans réduction du temps d'exécution (les points bleus se trouvent à gauche des points rouges). Cela signifie que le temps supplémentaire passé à lire toutes les propriétés n'est pas compensé par la diminution du nombre de nœuds analysés. Pour `Dijkstra`, le nombre de nœuds analysés pour la décomposition dépend uniquement du nombre de critères, alors que ce n'est pas le cas pour l'algorithme en stratégie séquentielle. Dans ce cas-là, un nombre inférieur de nœuds analysés implique une réduction de l'exécution, car le nombre de propriétés en lecture est également réduit (les points verts sont situés à gauche et au-dessous des points violets). Comme prévu, les algorithmes basés sur `Dijkstra` analysent moins de nœuds que ceux basés sur `Bellman`.

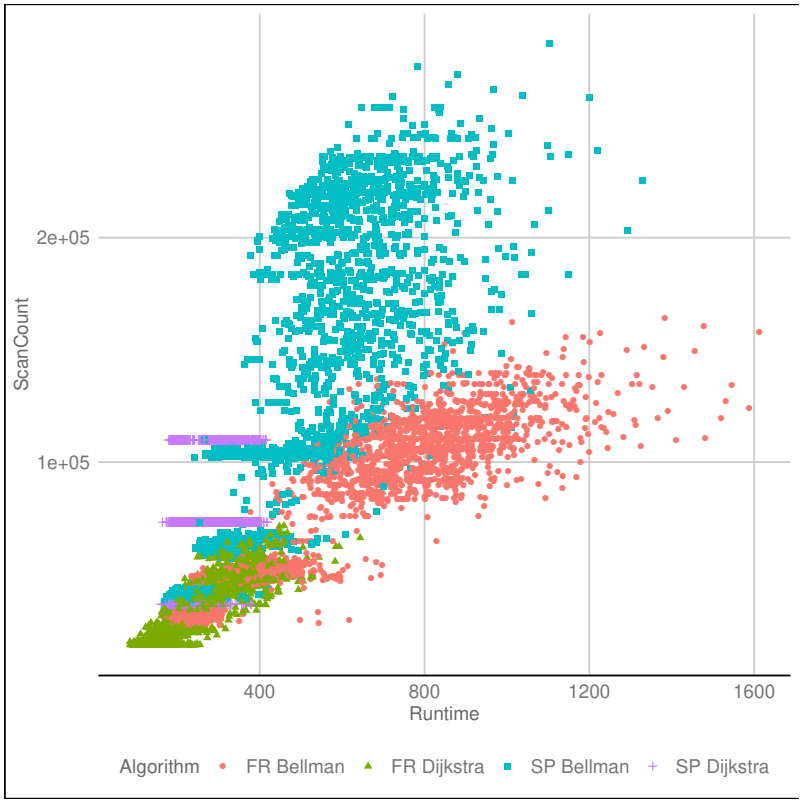


FIGURE 8.16 – Analyse des temps d'exécution et le nombre de scans.

Enfin, le tableau 8.7 donne la moyenne géométrique des améliorations apportées par les algorithmes basés sur la stratégie séquentielle par rapport à ceux basés sur la décomposition en *PCC*, en termes de temps d'exécution et de nombre de nœuds analysés. L'amélioration est le rapport entre le temps d'exécution de l'algorithme en stratégie séquentielle et la décomposition de *PCC* pour `Bellman` ou `Dijkstra`. L'amélioration est inférieure à 1 si l'algorithme en stratégie séquentielle est meilleur que la décomposition en *PCC* et supérieure à 1 sinon. Les deux algorithmes en stratégie séquentielle basés sur `Dijkstra` ou `Bellman` réduisent le nombre de nœuds analysés et la réduction augmente avec le nombre de critères. Les temps d'exécution ne sont pas réduits pour `Bellman`, mais le sont également pour `Dijkstra` lorsque le nombre de critères est égal à un ou deux. Lorsqu'il y a trois critères, la réduction du nombre de nœuds analysés ne compense pas la parallélisation supplémentaire menée dans le cas de la décomposition.

	Dijkstra		Bellman	
#Criteria	Runtime	#Scans	Runtime	#Scans
1	0.69	0.57	1.04	0.81
2	0.79	0.41	1.15	0.57
3	1.09	0.43	1.37	0.49

TABLE 8.7 – Amélioration des algorithmes *SEQ* par rapport aux décompositions de *DCP*.

Conclusion

Le *FRP* est un problème polynomial issu d'une application dans le domaine du transport aérien qui peut être résolu par l'algorithme proposé. Même si les deux méthodes renvoient une solution optimale après un nombre fini d'algorithmes de plus court chemin, la requête est plus lente que l'algorithme de plusieurs ordres de grandeur. Les performances de l'algorithme respectent déjà les contraintes sur le temps de réponse de l'application.

L'implémentation et l'évaluation de ces algorithmes nous ont permis de mieux s'interfacer avec la base de données orientée graphe. Ainsi, nous avons testé ces algorithmes sur les données de *Milanamos*. Les résultats montrent que le *FRP* peut être résolu en quelques secondes. Cela correspond parfaitement au temps de réponse de l'application. De plus, cela améliore bien le trafic puisqu'on se retrouve avec 90% des aéroports dont les routes ne sont pas intéressantes dans certains cas. On peut espérer que l'accélération du calcul des indices QSI sera conséquente et que les résultats obtenus sur le trafic seront cohérents. En effet, les itinéraires qui sont renvoyés correspondent parfaitement à des itinéraires réalistes dans notre historique.

CHAPITRE 9

Modèles d'estimation des parts de marchés

Nous présentons dans un premier temps le modèle des parts de marchés utilisé par Milanamos*, basé sur le modèle* QSI*. Ensuite, nous introduisons le nouveau modèle de choix discret basé sur la fonction* logit *pour faire face aux limites du modèle actuel. Le nouveau modèle estime les parts de marchés pour chaque marché sélectionné. Nous décrivons dans la suite le jeu de données qui a été exploité et la méthodologie développée en détails. Nous présentons enfin les études de cas réalisées pour évaluer notre approche. Dans ce contexte, nous avons proposé de valider le modèle d'estimation des parts de marchés et ensuite la méthode de la sélection des marchés sur les données historiques de* Milanamos*.*

Nous commençons par présenter le modèle de calcul de part de marché implémenté sur `PlanetOptim`, qui est basé sur le modèle *QSI* (*Quality Of Service Index*) (voir le chapitre 3). Ensuite, nous introduisons le modèle proposé basé sur le modèle *logit*. Ce modèle se base sur le graphe condensé et ne prend pas en compte la table des horaires. Il permet d'estimer les parts de marchés qui ont été identifiés par le *FRP*. Nous finissons par exposer des études de cas.

9.1 Modèle des parts de marchés utilisé par *Milanamos*

L'estimation de part de marché sur une route donnée se calcule en faisant une moyenne pondérée du *RMS* (*Referenced Market Share*) et du *QSI*. Une route est définie par une origine o et une destination d. La route fait référence au trajet d'un avion. Nous parlons d'itinéraire quand il s'agit du passager.

$$MS = QSI \times p + RMS \times (1 - p)$$

Le poids p est choisi manuellement par l'utilisateur. Par défaut, $p = 0,5$.

9.1.1 Méthode basée sur *QSI*

Le *QSI* vise à estimer une part de marché théorique en tenant compte d'éléments à la fois objectifs et subjectifs. Actuellement sur `PlanetOptim`, le *QSI* se calcule comme une moyenne pondérée de 7 critères :

$$QSI = \frac{\sum_{i=1}^{7} w_i \times c_i}{\sum_{i=1}^{7} w_i}$$

Par défaut $w_i = 1, \forall i \in C$ avec C est l'ensemble des critères.

Les sept critères sont :

1. Fréquence directe (en anglais *Direct Frequency*) : ratio de la fréquence hebdomadaire du vol étudié par rapport aux fréquences hebdomadaires des vols directs existants.
2. Fréquence indirecte (en anglais *Indirect Frequency*) : ratio de la fréquence hebdomadaire du nouveau vol par rapport à la fréquence hebdomadaire des vols avec correspondance.
3. Capacité de l'avion (en anglais *Aircraft Capacity*) : la taille de l'appareil est corrélée au confort du passager.
4. Circuit (en anglais *Circuitry*) : mesure le détour par apport au vol direct.
5. Présence de la compagnie (en anglais *Airline Presence*) : mesure la présence commerciale d'une compagnie aérienne.
6. Heure du jour (en anglais *Time of the Day*) : mesure la pertinence qualitative de l'heure de départ proposée par rapport à la préférence horaire des passagers.
7. Marque de la compagnie (en anglais *Brand Perception*) : évalue l'image de marque de la compagnie par rapport à ses concurrents sur l'itinéraire.

Les requêtes pour le calcul des critères se trouvent en annexe .

Fréquence directe

$$D_f = \frac{f}{f + E_{df}}$$

où f représente la fréquence hebdomadaire planifiée du nouveau vol.
E_{df} le nombre des fréquences existantes entre o et d en vol direct sur la semaine demandée. E_{df} se calcule en faisant une requête sur la collection `schedule`. Celle-ci est appelée dans les requêtes listées ci-dessous sous le nom :`schedule_initial_data`.

Fréquence indirecte

$$I_{df} = \frac{f}{f + (0,3 \times E_{vf})}$$

où E_{vf} est le nombre des fréquences existantes entre o et d en correspondance sur la semaine demandée du `year_month`. On applique un poids de 0,3, les passagers préfèrent en général un vol direct qu'un vol avec correspondance.

Les étapes suivantes présentent comment calculer E_{vf} :

1. Trouver tous les vols qui partent de o. On note l'ensemble contenant ces vols $V(o)$ (voir la requête dans le listing E.2 dans l'annexe E.1.2).
2. Trouver tous les vols qui partent des aéroports destinations des vols de l'ensemble $V(o)$ vers d. L'ensemble renvoyé est noté par $V(d)$.
3. Vérifier la viabilité de chaque connexion connectant o à d en passant par chaque aéroport de l'ensemble : $C = \{o(i) = d(j) | i \in V(o), j \in V(d)\}$, tel que $o(i)$ (respectivement $d(j)$)représente l'origine du vol i (respectivement la destination du vol j) (voir la section 7.2.1 pour les notations).

Fonction 9 : Calcul de E_{vf}

```
// étape 1
foreach i ∈ V(o) do
    // étape 2
    foreach j ∈ V(d) do
        // étape 3
        if o(i) == d(j) and MCT ≤ t_e(i) − t_s(j) ≤ MCT + MWT then
            E_vf ← E_vf + 1;
        end
    end
end
```

avec MCT le temps minimum pour faire la correspondance et MWT le temps maximum d'attente. Par défaut : $MCT = 60$ min et $MWT = 120$ min.
$t_s(i)$ et $t_e(j)$) sont respectivement le temps de départ et d'arrivée.

Capacité de l'avion

Nous regardons les capacités offertes par les routes similaires en termes de revenu et de volume des passagers. L'horizon de l'historique est de 12 mois avant le départ du vol étudié.

La première étape consiste à chercher les informations de toutes les routes résultant de l'étape 2 du simulateur (voir la section 2.3.2.2 du chapitre 2), cette étape consiste à trouver les routes dont les aéroports sont similaires à notre origine et destination. Ensuite, on calcule la capacité et la fréquence mensuelle pour chaque route. Puis, on calcule la capacité totale par rapport à la fréquence totale sur la période de 12 mois. Et enfin, on sélectionne les routes dont la capacité moyenne excède la capacité planifiée (cap). L'ensemble R_h présente ces routes sélectionnées.

Function 10 : Calcul

```
foreach r ∈ R do
    R̄_r = (Σ_{i=1}^{i=12} cap_r(i)) / (Σ_{i=1}^{i=12} freq_r(i));
    if R̄_r ≥ cap then
        r ∈ R_h;
    end
end
```

Par défaut, $cap = 200$. Ainsi, le critère A_c est calculé comme suit :

$$A_c = 1 - \frac{|R_h|}{|R|}$$

Pour calculer le critère A_c, nous utilisons une collection pré-calculée `SegmentRoutesAirline` et les résultats de l'étape 2 *Clusters* du simulateur, cette étape qui consiste à trouver toutes les routes similaires. La collection `SegmentRoutesAirline` contient toutes les données mensuelles et annuelles des segments par compagnie aérienne : nombre de passagers (`Pax`), revenu total, fréquence, capacité, et nombre de segments opérés par la compagnie aérienne. C'est une collection qui a été créée en se basant sur les deux collections `schedule` et `segment` (`segment_initial_data` dans les requêtes). Cette collection contient les données mensuelles de chaque segment par compagnie aérienne en terme de `Pax` et revenu.

Présence de la compagnie aérienne

On calcule le nombre de destinations à partir de l'origine o par rapport au nombre de destinations desservies par toutes les compagnies aériennes depuis o (voir la requête dans listing E.4) :

$$A_p = \frac{|D|}{|D_{all}|}$$

où D est l'ensemble de destinations desservies par la compagnie depuis l'origine et D_{all} est l'ensemble des destinations desservies par toutes les compagnies aériennes.

Circuit

C'est le rapport de la distance totale de tous les aéroports de correspondance avec le nombre de ces aéroports. Nous commençons par calculer la distance totale pour chaque aéroport en correspondance de l'ensemble C. Ensuite, nous calculons la distance moyenne $\bar{d_v}$ de cet aéroport par rapport à la distance du vol direct.

Function 11 : Calcul de la distance totale

1 $dd = dist(o, d)$;
; `// distance du vol direct`
2 $d_t = 0$;
3 **foreach** $v \in C$ **do**
4 $id = dist(o, v) + dist(v, d)$;
; `// distance par correspondance`
5 $\bar{d_v} = 1 - \frac{dd}{id}$;
6 $d_t = d_t + \bar{d_v}$
7 **end**

$\bar{d_v}$ est défini dans $[0, 1[$; proche de 0 s'il s'agit d'un petit détour. Finalement, le circuit est calculé en faisant la moyenne. Donc, plus C_t est petit, mieux c'est.

$$C_t = \frac{d_t}{|C|}$$

Les critères heure du jour et marque de la compagnie sont fixés par l'utilisateur (des critères subjectifs). Leurs valeurs sont comprises entre 0 et 120. En fait, ces deux critères dépendent directement du marché que la compagnie aérienne envisage d'opérer. Par défaut, leurs poids est $0,5$.

9.1.2 Calcul du *Referenced Market Share*

La table `OAndDRoutesAirline`, pré-calculée à partir de la table `o&d`. Elle contient toutes les informations concernant le revenu et le `Pax` mensuel et annuel pour chaque paire (origine-destination) par compagnie aérienne. Nous regardons s'il y a au moins trois routes opérées par la compagnie dans l'étape 2 (Clusters). Si ce n'est pas le cas, alors nous regardons les compagnies de la même classe (lowcost, régional, *etc.*). Sinon, nous les choisissons sur l'ensemble de toutes les compagnies. La période considérée est 12 mois avant la date du vol proposée. Cette étape permet de quantifier le nombre de passagers que l'on peut capturer de ces routes historiquement similaires à la route étudiée.

Function 12 : Calcul du RMS

1 **foreach** $r \in R$ **do**
2 $pax(p, r) = \sum_{ym=1}^{ym=12} pax(r, ym)$;
3 $MS(r) = \frac{pax(p,r)}{pax}$;
4 **end**
5 $T_{WMS} = \sum_{r \in R} MS(r) \times pax(p, r)$;
6 $RMS = \frac{T_{WMS}}{\sum_{r \in R} pax(p,r)}$;

La ligne 2 du pseudo-code calcule le nombre de passagers sur la période qu'on regarde. Ensuite, nous calculons la part de marché de chaque route. Cela permet d'estimer la part de marché capturée de chaque route similaire. Après, nous obtenons le nombre de passagers capturés (T_{WMS}) et finalement, la part de marché pour la route étudiée est obtenue (RMS).

9.1.3 Limites du modèle actuel

1. Le modèle est restreint au seul marché o & d. De ce fait, les autres marchés influencés par le vol étudié sont ignorés.
2. La concurrence se limite aux vols à une seule correspondance.
3. Par définition, le critère circuit établit le lien entre la distance et le temps. Or, le modèle actuel prend uniquement en considération la distance. De plus, la solution du *FRP* élimine déjà les itinéraires non réalistes par rapport aux trois critères : temps, distance et coût, ce qui permet d'écarter les parts de marchés négligeables.
4. L'étape 6 du simulateur qui consiste à afficher les origines/destinations des passagers est indépendante de l'étape 7 qui calcule la part du marché (voir le chapitre 2).
5. Le critère A_p exprime la dominance d'une compagnie aérienne. Il est plutôt lié au nombre de passagers qui voyagent avec cette compagnie aérienne (a) et non pas le nombre de destinations desservies. Il exprime le nombre moyen de passagers transportés par la compagnie par rapport à toutes les compagnies [Grosche and Rothlauf, 2007] :

$$A_p(r) = \frac{pax(a)}{pax_t}$$

   où pax_t est le nombre de passagers total transportés par toutes les compagnies aériennes depuis o.
6. Le *QSI* permet d'estimer la probabilité qu'un passager sélectionne un itinéraire particulier. Pour cela, des scores (QSI) sont calculés pour chaque itinéraire possible [Jacobs et al., 2012]. Le modèle actuel ne compare pas tous les itinéraires possibles entre o et d.

9.2 Nouveau modèle basé sur *logit*

Pour faire face aux limites du modèle actuel de *Milanamos*, nous avons proposé un nouveau modèle basé sur le modèle multinomial. Dans cette section, nous commençons par présenter les trois étapes principales de ce modèle. Ensuite, nous détaillons l'étape qui reprend le modèle multinomial cité dans le travail [Coldren et al., 2003], le modèle multinomial est très utilisé en pratique et ses propriétés ont été décrites dans le chapitre 3. Ainsi, l'étude du comportement du passager sera basée sur ce modèle en utilisant les données du graphe condensé.

Nous expliquons la façon dont nous avons procédé pour l'adapter à notre contexte ainsi que les données que nous avons sélectionnés pour répondre à notre besoin. Finalement, nous introduisons rapidement quelques résultats du premier jeu de données.

9.2.1 Processus d'estimation des parts de marchés

Le processus consiste en trois étapes (voir la figure 9.2) :

1. ***Prétraitement des hubs*** : calcul des plus courts chemins depuis/vers les hubs.
2. ***Résolution du* FRP** : la solution du *FRP* donne les marchés potentiels pour un vol donné.
3. ***Estimation des parts de marchés*** : estime des parts de marchés pour chaque marché en se basant sur le modèle *logit*. Cette estimation prend en considération les vols directs concurrents et les itinéraires passant par les hubs.

L'étape de prétraitement est faite une seule fois, les résultats sont stockés sur la base de données orientée `Neo4j`. C'est l'étape la plus coûteuse. Tandis que l'étape de la résolution du *FRP*, elle est faite pour chaque vol étudié et l'étape du estime des parts de marchés pour chaque marché prometteur centré sur le vol étudié.

Étape 1 : Prétraitement des hubs

L'étape du prétraitement des hubs nécessite la liste des hubs identifiée dans la section 7.6.5 et se base sur la méthode *Hub labeling* présentée dans le chapitre 4 (voir l'étape **PRE** sur la figure 9.2).

1. Pour chaque hub, calculer les plus courts chemins depuis/vers les hubs.
2. Ajouter un arc quand un aéroport est atteignable depuis un hub et un hub est atteignable depuis un hub. Ces arcs sont étiquetés par les informations ci dessous.

Dans le graphe condensé $G = (\mathcal{X}, \mathcal{U}, \mathcal{W})$, un arc entre deux nœuds représente tous les vols entre ces deux aéroports. Pour identifier les hubs de la liste (H), nous avons introduit une fonction indicatrice définie pour un nœud de cette façon :

$$\mathbb{1}_{H(x)} = \begin{cases} 1, & \text{si } x \in H \\ 0, & \text{si } x \notin H \end{cases} \tag{9.1}$$

— $f_{u'} = \min_{u \in \mathcal{U}(\mathcal{P})} Fu$ est la fréquence minimale ;
— $c_{u'} = \min_{u \in \mathcal{U}(\mathcal{P})} c(u)$ est la capacité minimale ;
— $t_{u'} = \sum_{u \in \mathcal{U}(\mathcal{P})} t(u)$ est la durée minimale totale ;
— $R_{u'} = \sum_{u \in \mathcal{U}(\mathcal{P})} R(u)$ est le revenu total ;
— $D_{u'} = \sum_{u \in \mathcal{U}(\mathcal{P})} D(u)$ est la distance totale ;
— $e_{u'}$ est le nombre des nœuds intermédiaires du chemin, ce qui représente les escales ;
— $h_{u'} = \sum_{x \in \mathcal{X}(\mathcal{P})} \mathbb{1}_{H(x)}$ est le nombre des hubs.

Avec $\mathcal{X}(\mathcal{P})$ et $\mathcal{U}(\mathcal{P})$ l'ensemble des nœuds et des arcs du chemin P depuis/vers le hub. u' est l'arc ajouté entre le hub et le nœud.

Nous enregistrons ces informations sur la base de données, qui nous seront utiles plus tard pour l'estimation des parts de marchés.

Étape 2 : résolution du *Flight Radius Problem*

Résoudre le *FRP* pour calculer l'ensemble des marchés potentiels (voir l'étape **FRP** sur la figure 9.2). Les démarches ont été présentées en détails dans le chapitre 8. On rappelle que chaque paire O&D entre un aéroport en amont et un autre en aval du vol étudié est un marché potentiel,

ce qui représente le marché en correspondance. Par ailleurs, il faudra prendre en compte dans l'estimation des parts de marchés, le marché local, ce qui veut dire le marché concerné par l'ajout du vol avec l'ensemble d'itinéraires desservant ce marché. Par exemple, le marche local dans le cas du vol NCE – BKK est *NCE & BKK*. Dans la figure 9.1), ce marché est desservi par 2 itinéraires qui passent par CDG et FRA (aéroport de Francfort).

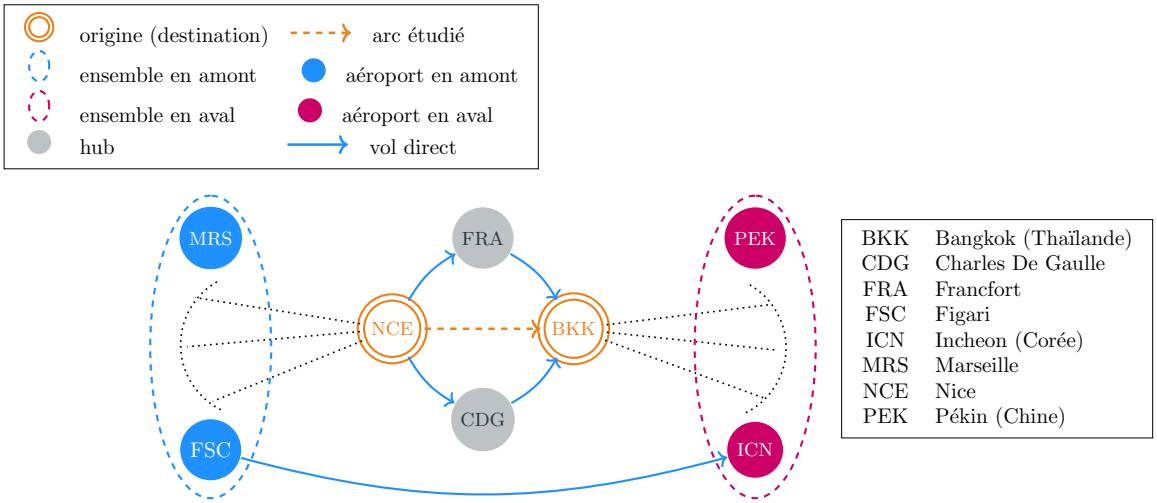


FIGURE 9.1 – Graphe de l'exemple de motivation du vol NCE – BKK.

Étape 3 : estimation des parts de marchés avec le modèle *logit*

Pour chaque marché OD, on a trois types d'itinéraires à considérer (voir l'étape **MS** sur la figure 9.2) :

1. Chemins valides (passant par l'arc (o,d)), résultats du *FRP*.
2. Chemin direct (vol direct s'il existe) reliant l'origine à la destination du marché.
3. Chemin de o à d passant par des hubs (chemins contractés dans l'étape 1).

Pour estimer le nombre de passagers, il suffit de multiplier la part du marché estimée par la taille du marché, cette dernière information est à calculer à partir de la table O&D.

9.2.2 Processus de modélisation de choix de passager

Cette section concerne l'étape 3 du calcul des parts de marchés. Plus précisément, elle se concentre sur la façon de modéliser le comportement du choix du passager dans cette étude.

Les choix d'itinéraires des passagers aériens sont initialement représentés à l'aide de modèles multinomiales (*logit*). Dans les travaux de Coldren et Koppelman ([Coldren et al., 2003, Coldren, 2005, Coldren and Koppelman, 2005, Garrow, 2016, Barnhart and Smith, 2012, Abdelghany and Abdelghany, 2018b]), l'analyse est basée sur un sous-ensemble des données utilisées. Plus précisément, un seul mois de départs de vol (janvier 2000) représentant toutes les paires d'aéroports définies pour deux entités régionales aux États-Unis est représenté. Les entités régionales sont définies par fuseau horaire. Dans cette analyse, l'entité régionale «Est-Ouest» contient des paires d'aéroport au départ du fuseau horaire de l'Est et arrivant dans le fuseau horaire du Pacifique,

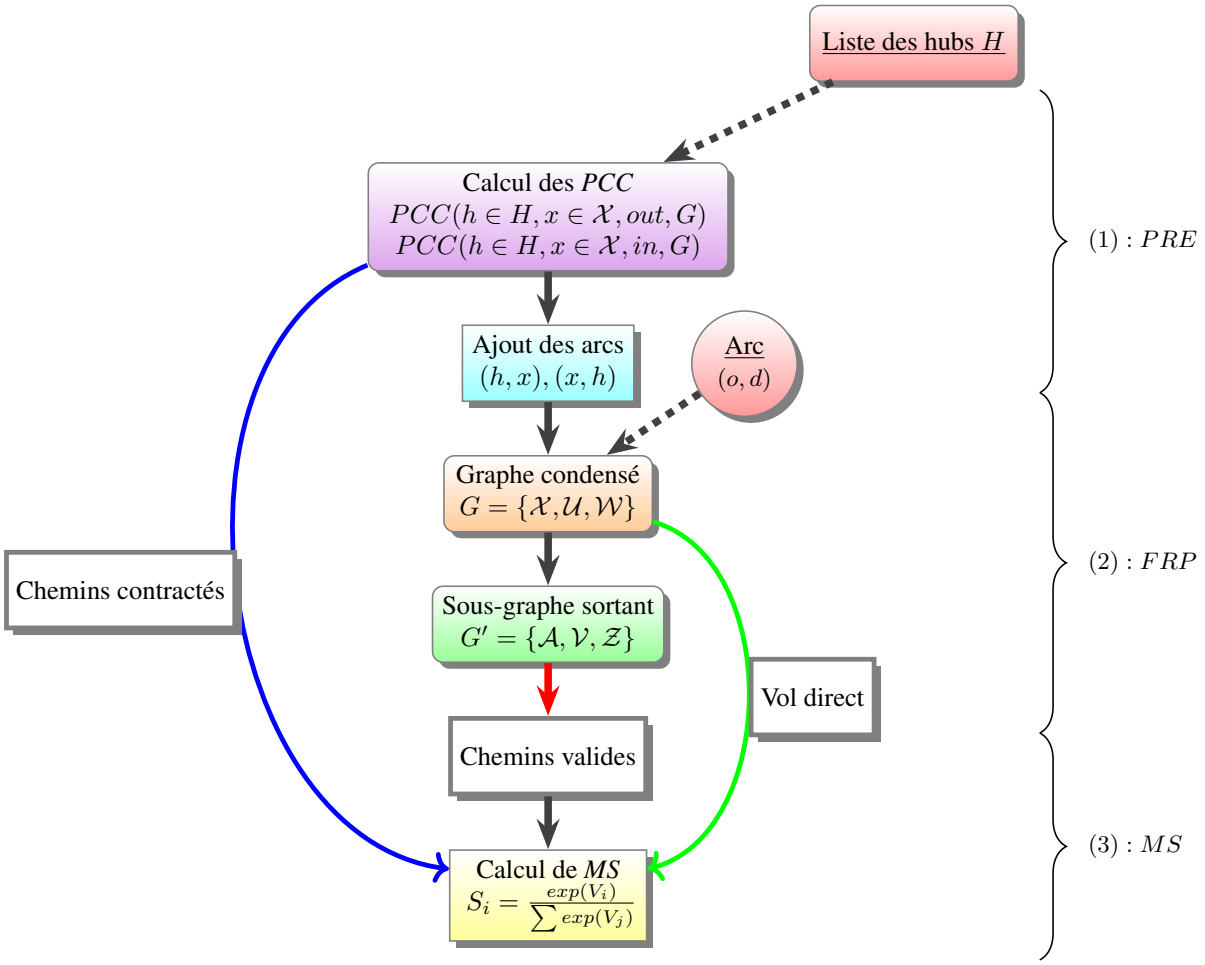


FIGURE 9.2 – Étapes du calcul des parts de marchés. .

tandis que l'entité régionale «Ouest-Est» contient des paires d'aéroports partant du fuseau horaire du Pacifique et arrivant sur la Fuseau horaire de l'Est.

Dans notre modèle, on ne tient pas en compte du fuseau horaire car on a relâché la contrainte temporelle. On travaille sur le graphe condensé avec uniquement les temps de transfert (voir le chapitre 7).

Le chapitre 3 présentait les différents modèles de choix discret utilisés en pratique dans le transport aérien. Nous étudions dans notre étude le choix d'itinéraires : on s'intéresse donc plus particulièrement aux calculs des parts d'itinéraires. Parmi les modèles de choix : nous avons opté pour le modèle *multinomial logit* qui a connu un grand succès [Coldren et al., 2003, Coldren, 2005, Coldren and Koppelman, 2005, Garrow, 2016, Barnhart and Smith, 2012, Abdelghany and Abdelghany, 2018b].

La partie qui suit présente un rappel de la fonction d'utilité du passagers, ses paramètres et ses variables. Nous nous sommes basés sur l'étude fournie par [Coldren et al., 2003]. Nous avons adapté les critères de Coldren en ignorant les critères temporels pour les calculer dans le graphe condensé. Nous allons aussi garder des critères du *QSI*.

9.2.2.1 Ensemble de choix

Les modèles de choix discret permettent de modéliser le choix d'un passager à qui il est offert un nombre fini d'itinéraires respectant certaines propriétés. Nous souhaitons évaluer la probabilité pour chacune des alternatives de l'ensemble de choix que le passager choisisse celle-ci.

On utilise dans la suite les indices i ou j pour référer aux alternatives et on omet l'indice n qui se référait au passager comme dans le chapitre 3.

9.2.2.2 Fonction d'utilité du passager

En supposant que la fonction d'utilité est composée d'une composante déterministe v_i et d'une aléatoire ϵ_i, on peut écrire, comme expliqué dans la section 3.3.2.2 :

$$U_i = V_i + \epsilon_i = \sum_{k=1}^{k=n} \beta_k X_{ki} + \epsilon_i \tag{9.2}$$

Le vecteur de variables X représente uniquement les caractéristiques des itinéraires puisque celles des passagers ne sont pas connues. Le vecteur β correspond aux poids relatifs des attributs dans la décision du passager.

Le tableau 9.1 présente les différents attributs de chaque itinéraire, qui sont utilisés en tant que variables de la fonction d'utilité : niveau de service, deuxième meilleure correspondance, différence de temps avec la deuxième meilleure correspondance, meilleure correspondance, rapport de la distance, fréquence, rapport du prix et capacité. Il existe plusieurs façons de modéliser l'impact de chacun de ces attributs dans la décision du passager, avec des variables binaires ou réelles, *etc*.

Variable	**Nom**	**Type**
Niveau de Service	ns	binaire
Deuxième meilleure correspondance	sbc	binaire
Deuxième meilleure (différence de temps)	$sbctd$	$\mathbb{R}$
Meilleure correspondance	$bctd$	$\mathbb{R}$
Rapport de la distance	d	$\mathbb{R}$
Fréquence	f	$\mathbb{R}$
Rapport du prix	p	$\mathbb{R}$
Capacité	c	$\mathbb{R}$

TABLE 9.1 – Variables de la fonction d'utilité

9.2.2.3 Variables binaires

Certains attributs sont modélisés par des variables binaires, c'est le cas ici du niveau de service et de la deuxième meilleure correspondance. L'utilisation des variables binaires nécessite l'introduction d'une variable référence. La situation de référence choisie ici correspond au meilleur service dans un marché et celui avec l'itinéraire le plus rapide pour les itinéraires avec escales.

Nous avons commencé par sélectionner les variables qui correspondent aux données que nous disposons. [Coldren, 2005] a structuré les variables suivant la présence des interactions entre elles en plusieurs catégories.

Avec l'ensemble des données dont nous disposons, nos variables sont classées suivant quatre catégories (voir le tableau 9.1) :

— Niveau de Service

— Qualité de la connexion

— Attributs du transporteur

— Taille et capacité d'avion

Le tableau 9.2 présente les coefficients estimés à partir de [Coldren et al., 2003], certains sont modifiés pour pallier les critères manquants. Le signe et la valeur du coefficient seront à comparer avec la meilleure situation. Par exemple, la variable de la référence correspond au type du meilleur service disponible sur le marché. S'il existe un vol direct sans escale reliant l'origine et la destination, alors le coefficient pour ce type d'itinéraire est 0.0000. Ainsi, s'il est négatif, cela signifie que l'utilité du passager diminue si l'itinéraire a plus d'une escale par rapport à une situation où il n'en aurait aucune. Nous expliquons plus tard, en détails, le signe et la valeur de chaque coefficient. Cela nous permettra de mieux comprendre l'impact de la variation du coefficient sur l'utilité de l'itinéraire.

Exemple 9.2.1 (Niveau de service)**.**
Prenons l'exemple du *Nice*/France (NCE) à *Charles-De Gaulle*/France (CDG) qui est considéré comme un marché sans escale car il existe au moins un service sans escale reliant cette paire de villes. Une paire de villes comme NCE à *Bangkok*/Thaïlande (BKK) est considérée comme un marché avec une escale, car il s'agit du meilleur service fourni pour cette paire de villes.

Variable	**Coefficients**
Niveau de Service (ns)	
Itinéraire direct dans un marché direct	0.0000
Itinéraire avec une seule correspondance dans le marché direct	-2.7087
Itinéraire avec deux correspondances dans le marché direct	-7.5112
Itinéraire avec une seule correspondance dans le marché avec une seule correspondance	-0.0000
Itinéraire avec deux correspondances dans le marché avec une seule correspondance	-2.5431
Qualité de la correspondance	
Deuxième meilleure correspondance (sbc)	-0.5322
Différence de temps de deuxième meilleure correspondance($sbctd$)	-0.0178
Meilleure correspondance($bctd$)	-0.0093
Rapport de la distance(d)	-0.0210
Attributs du transporteur	
Fréquence(f)	0.0078
Rapport du prix(p)	-0.0045
Taille d'avion et capacité	
Grandes lignes (c)	0.0047

TABLE 9.2 – coefficients estimés

9.2.2.4 Modélisation du nombre d'escales

Le nombre d'escales est une variable qui influence l'utilité de l'itinéraire. Cet impact concerne les itinéraires uniquement avec correspondance et modélisé par trois variables qui sont regroupées dans la catégorie "Qualité de la correspondance" :

1. sbc est une variable binaire pour comparer l'itinéraire avec celui du meilleur itinéraire avec correspondance ;
2. $sbctd$ est une variable réelle pour comparer le temps de transfert par rapport au meilleur itinéraire avec correspondance ;
3. $bctd$ est une variable réelle pour comparer la durée de l'itinéraire avec le plus rapide.

La deuxième meilleure correspondance et la différence de temps de la deuxième meilleure correspondance sont deux variables liées qui ne sont utilisées que pour les itinéraires avec correspondance. La première variable est une variable binaire qui indique si l'itinéraire avec une correspondance a une meilleure correspondance à l'aéroport de transfert par rapport au temps au sol. Cette variable fictive est égale à 1, si l'itinéraire de correspondance n'est pas le meilleur en termes de correspondance, et zéro sinon. Dans le cas où la variable est égale à 1 pour tout itinéraire, la deuxième variable, différence de temps de la meilleure correspondance, mesure la différence de temps entre la durée de transfert de cet itinéraire et la durée de transfert du meilleur itinéraire avec correspondance. La variable meilleure correspondance mesure la différence de temps écoulé entre un itinéraire impliquant une correspondance et l'itinéraire le plus rapide impliquant une correspondance pour chaque paire de villes, indépendamment des aéroports de transfert.

Comme nous ne disposons pas d'informations sur le temps au sol, nous avons alors modélisé cette variable avec le temps de transfert, MCT, qui est fixé à 120 min.

Le nombre d'escales est un indicateur de la qualité de la connexion qui est en plus représentée par la variable rapport de la distance. Cette variable permet de comparer la distance de l'itinéraire par rapport à l'itinéraire le plus court sur le marché.

9.2.2.5 Modélisation de la fréquence

La fréquence est incluse dans la troisième catégorie, cette dernière correspond à la modélisation de l'impact de la fréquence dans l'utilité du passager. Coldren [2005] a modélisé la fréquence par le point de vente qui reflète le fait que les itinéraires proposés par les compagnies aériennes qui ont plus de présence à l'origine et/ou à la destination du voyage sont plus susceptibles d'être sélectionnés par les voyageurs. Par exemple, les voyageurs des départs de `CDG` sont plus susceptibles de choisir les itinéraires proposés par *Air France*, car cette compagnie aérienne est plus présente dans cet aéroport. Cette variable mesure la présence de la compagnie aérienne dans la ville à l'origine et à la destination de l'itinéraire.

Pour les itinéraires avec plusieurs segments de vols, nous avons regardé le premier segment de vol.

9.2.2.6 Modélisation du prix

En plus de la fréquence, nous avons également gardé le prix puisque le prix augmente avec l'amélioration du niveau de service. Dans l'étude de Coldren [2005], le rapport tarifaire mesure le tarif moyen de la compagnie aérienne par rapport au tarif moyen de l'industrie pour la paire de villes. Il convient de noter que cette variable de rapport tarifaire n'est pas mesurée au niveau de

l'itinéraire. En d'autres termes, il ne représente pas le prix du billet de chaque itinéraire. Il mesure plutôt, en général, si la compagnie aérienne est une compagnie aérienne à bas prix qui propose ou non des itinéraires bon marché dans la paire de villes.

Dans notre cas, on regarde les itinéraires par rapport au flux, donc le prix correspond plutôt au revenu minimum par passager.

9.2.2.7 Modélisation de la capacité

La capacité correspond au nombre de sièges sur le plus petit avion pour l'itinéraire. Dans notre cas, comme les données sont agrégées, nous l'avons modélisée par le taux de remplissage (en anglais *load factor*) qui correspond à la capacité minimale fournie sur l'itinéraire par rapport à la fréquence.

9.2.2.8 Interprétation des coefficients

Comme le montre le tableau 9.2, les signes et les valeurs des paramètres des différentes variables incluses dans la fonction d'utilité ont une signification. Par exemple, pour la variable de niveau de service, les coefficients sont négatifs et les valeurs de ces paramètres diminuent à mesure que le nombre de correspondances augmente. Ainsi, pour les paires de villes sans correspondance (marchés), l'itinéraire sans escale est considéré comme la référence et la valeur de son paramètre correspondant est égale à 0.

Les valeurs des coefficients qui correspondent à l'itinéraire à correspondance unique et l'itinéraire avec deux correspondances sont respectivement $-2,7078$ et $-7,5112$.

Lorsque l'itinéraire n'est pas le meilleur, en termes de correspondance (c'est-à-dire la deuxième meilleure correspondance), le score de l'itinéraire est réduit de $-0,5322$ (la valeur du coefficient de la variable fictive de la deuxième meilleure correspondance est $-0,5322$). Pour chaque minute de différence entre le deuxième itinéraire de correspondance et le meilleur itinéraire de correspondance, le score de l'itinéraire est réduit de $0,0178$. De plus, pour chaque minute de différence de temps écoulé entre un itinéraire impliquant une correspondance et l'itinéraire le plus rapide impliquant une correspondance pour chaque paire de villes indépendamment des villes de transfert, le score de l'itinéraire est réduit de $0,0093$. De plus, le score de l'itinéraire diminue à mesure que sa distance augmente.

Le score de l'itinéraire diminue de $0,0210$ pour chaque unité d'augmentation du rapport de distance. En outre, l'attraction de ces itinéraires augmente avec la fréquence. Finalement, les compagnies aériennes coûteuses sont moins susceptibles d'être sélectionnées, car le score de son itinéraire diminue de $0,0045$ pour chaque augmentation d'unité du rapport tarifaire.

De plus, pour le même type d'avion, les passagers préfèrent les avions ayant plus de sièges aux plus petits.

9.3 Études de cas

Cette section présente l'ensemble des études de cas réalisées pour tester l'approche proposée pour sélectionner les marchés et évaluer les parts de marchés. Ainsi, nous considérons globalement deux études de cas suivant le modèle à évaluer :

1. Validation de l'estimation des parts de marchés sur le réseau de la France.

2. Validation du modèle de la sélection des marchés sur le réseau mondial.

Ces études de cas permettront de valider les modèles en comparant leurs résultats aux données historiques et d'évaluer le passage à l'échelle en fonction de la taille du réseau.

9.4 Validation de l'estimation des parts de marchés

Le protocole expérimental pour la validation du modèle d'estimation des parts de marchés est le suivant :

1. Sélectionner un marché pour lequel des données historiques existent.
2. Estimer les parts de marchés d'un ensemble d'itinéraires desservant ce marché avec notre modèle.
3. Comparer les parts de marchés estimées à celles calculées à partir des données historiques.

Nous réalisons donc trois études de cas sur le réseau aérien français en fonction du niveau de service existant (direct ou indirect).

Le réseau aérien français a trois hubs, plate-formes de correspondance dans l'aérien, identifiés grâce aux données métiers : `CDG`, `ORY` et `LYS`. Une part significative des vols d'une compagnie transite par ces hubs afin de réduire les coûts de gestion des opérations de ses vols.

9.4.1 Service direct : marché Figari (FSC) & Lille (LIL)

On commence par le niveau de service direct, c'est-à-dire que le marché est desservi par un vol direct. Dans ce cas-là, notre modèle considère uniquement les itinéraires à une seule correspondance, passant par des hubs. Donc, on va considérer quatre itinéraires :

— Vol direct.

— Trois itinéraires avec une seul correspondance.

Le tableau 9.3 présente pour chaque itinéraire considéré dans l'étude son type (direct ou avec une seule correspondance), sa part calculée depuis l'historique de *Milanamos* ainsi que sa part estimée par notre modèle.

Dans l'historique, il existe des itinéraires ne passant pas par des hubs. À titre d'exemple, un itinéraire passe par l'aéroport `MRS`, qui n'est pas un hub pour la compagnie aérienne *Air France*. Pour ce marché, les résultats montrent que la majorité des passagers voyagent en pratique entre Figari et Lille par un vol direct (93,5%), ce qui est validé par notre modèle (90,5%).

En revanche, notre modèle estime qu'il y a un itinéraire qui passe par `LYS`, qui est déjà un hub, pouvant capturer une partie de ces passagers. Certains passagers font une correspondance par `MRS` (aéroport de Marseille) ou bien `NCE` (aéroport de Nice). Par conséquent, une opportunité économique serait de faire une correspondance par `LYS`, ce qui permettrait à *Air France* de réduire ses coûts sur ce marché.

Itinéraire	Type[1]	partE[2]	partH[3]
FSC -> LIL	direct	90,5%	93,5%
FSC -> CDG -> LIL	1 cor.	0,5%	0,0%
FSC -> LYS -> LIL	1 cor.	8,3%	0,0%
FSC -> ORY -> LIL	1 cor.	0,7%	0,1%
FSC -> MRS -> LIL	1 cor.	0,0%	5,3%
FSC -> NCE -> LIL	1 cor.	0,0%	1,1%

[1] *Type d'itinéraire ;* [2] *Part estimée par le modèle.*
[3] *Part calculée sur l'historique.*

TABLE 9.3 – Résultats de l'estimation des parts de marchés FSC & LIL. (1 cor. représente une seule correspondance.)

9.4.2 Service direct : marché Toulouse (TLS) & Metz (ETZ)

Le tableau 9.4 donne un seul itinéraire qui passe par un aéroport qui n'est pas un hub. C'est celui passant par l'aéroport de Nice (NCE) avec une part faible des passages (1,5%). Cependant, le modèle et l'historique donnent quasiment le même ensemble d'itinéraires avec un écart faible entre les deux parts.

Marché Toulouse (TLS) & Metz (ETZ)

Itinéraire	Type[1]	partE[2]	partH[3]
TLS -> METZ	direct	86,2%	78,4 %
TLS -> CDG -> METZ	1 cor.	0,1 %	0,0 %
TLS -> LYS -> METZ	1 cor.	13,6 %	20,1 %
TLS -> ORY -> METZ	1 cor.	0,1 %	0,0 %
TLS -> NCE -> METZ	1 cor.	0,0%	1,5%

[1] *Type d'itinéraire ;* [2] *Part estimée par le modèle.*
[3] *Part calculée sur l'historique.*

TABLE 9.4 – résultats de l'estimation des parts de marchés TLS & ETZ.

9.4.3 Service indirect : marché Poitiers (PIS) & Nice (NCE)

On considère le cas d'un marché desservi avec un itinéraire avec une seule correspondance. On a choisi le marché Poitiers (PIS) & Nice (NCE) desservi par un vol avec une correspondance par (LYS) dans l'historique. Dans ce cas-là, on regarde les itinéraires avec deux correspondances, ceux passant par des hubs selon notre modèle. Ce qui revient à étudier 10 itinéraires :

— Itinéraire avec une seule correspondance.

— Trois itinéraires avec une seule correspondance.

— Six itinéraires avec deux correspondances.

Les résultats du tableau 9.5 illustrent uniquement trois itinéraires passant par LYS parmi les 10 itinéraires possibles. En effet, pour aller de l'aéroport PIS à n'importe quelle ville française, il y a une seule possibilité en pratique, c'est celle de passer par l'aéroport LYS. Ce qui revient à dire, que notre modèle trouve parfaitement le même résultat en termes de l'ensemble d'itinéraires et également en termes de part de marchés capturées.

Marché Poitiers (PIS) & Nice (NCE)

Itinéraire	**Type**[1]	**partE**[2]	**partH**[3]
PIS -> LYS -> CDG -> NCE	2 cor.	0,0%	0,0%
PIS -> LYS -> NCE	1 cor.	100%	100%
PIS -> LYS -> ORY -> NCE	2 cor.	0,0%	0,0%

[1] *Type d'itinéraire ;* [2] *Part estimée par le modèle.*
[3] *Part calculée sur l'historique.*

TABLE 9.5 – résultats de l'estimation des parts de marchés PIS & NCE.

9.4.4 Conclusion des 3 études de cas

D'après les résultats de ces trois études de cas, on peut conclure que notre modèle permet bien de fournir une bonne estimation des itinéraires desservant un marché et de leurs parts de marchés respectives dans le réseau français.

En outre, le temps de l'estimation est plus rapide que les approches scientifiques dans l'état de l'art, qui prennent plusieurs mois pour faire ce travail [Coldren et al., 2003]. Notre modèle ne dépasse pas quelques secondes.

Ces études de cas valident la restriction des itinéraires concurrents à ceux passant par des hubs. Nous testons maintenant à toute l'approche intégrée y compris la méthode de la sélection des marchés centrée autour d'un vol.

9.5 Validation du modèle de la sélection des marchés

On propose toujours de se comparer avec l'historique pour pouvoir évaluer le passé, ce qui nous permettra par la suite de se projeter dans le futur. Nous commençons par décrire le protocole expérimental. Ensuite, nous passons à la réalisation des études de cas.

On rappelle que notre méthode de sélection des marchés consiste à centrer sur un vol et ensuite identifier la liste des marchés prometteurs vis-vis ce vol :

1. Choisir un vol (existant pour se comparer à l'historique).
2. Déterminer les marchés centrés sur ce vol.
3. Sélectionner un ou plusieurs de ces marchés.
4. Estimer les parts de marchés.
5. Comparer les parts estimées à celles calculées à partir des données historiques.

On commence par réaliser ce protocole sur les données du réseau français avant de passer au réseau mondial.

9.5.1 Analyse du passage à l'échelle

Pour valider le modèle sur le réseau mondial, comme toujours l'étape de pré-traitement est la plus coûteuse (étape 1) mais, avec le *COVID-19*, le travail est devenu plus compliqué car il a été poursuivi sur ma machine personnelle qui a une limite de 11 Go.

Nous avons ainsi analysé la requête pour le pré-traitement. La requête consiste en trois phases, calcul des plus courts chemins depuis/vers les hubs, calcul des informations sur les caractéristiques des chemins contractés et création des arcs avec ces informations pré-calculées. La première phase implique des calculs des plus courts chemins. Quant à la troisième phase de la création des arcs, qui est une opération d'écriture, elle est plus longue car elle consiste à localiser les nœuds (les aéroports) et ensuite écrire les propriétés dans des fichiers qui sont stockés séparément des nœuds/relations (voir la section 5.5.10) ce qui est même la plus longue. Cette opération dépend également du type de disque, dans notre cas c'est un disque SSD qui est plus rapide que HDD. En outre, nous avons ajouté une clause pour éviter les arcs en doublon.

Nous sommes donc arrivés aux estimations suivantes dans le cas d'une seule direction (depuis les hubs vers les aéroports) avec 10 Go de RAM pour `Neo4j`.

1. 377 hubs sur le réseau mondial. Il faudra ainsi calculer les plus courts chemins de ces hubs vers 4 602 aéroports. Puis, créer des arcs pour chaque hub. Ce qui donne 1 734 954 arcs crées (voir la requête E.11 en annexe E.2).
2. 5 minutes pour un hub, donc, 32 heures pour tous les hubs du monde entier.
3. écriture des 6 propriétés.

Nous avons ainsi opté pour une stratégie de traitement par lot. Il s'agit d'exécuter les transactions séquentiellement. `Neo4j` fournit la procédure qui convient :`apoc.periodic.iterate` (voir la requête E.10 en annexe E.2). Il s'agit de la procédure la plus rapide et la plus sûre pour ce genre de traitement [Neo4j and Cypher, 2020].

9.5.2 Vol `LYS-PIS`

À partir de l'historique, nous avons choisi le vol direct `LYS-PIS`. Supposons maintenant que ce vol n'existe pas et voyons ce que notre système nous donnera.

La figure 9.3 présente le graphe du cas centré sur le vol `LYS-PIS`, qui est illustré par un arc en orange pointillé. Une des paires d'aéroports sélectionnés (NCE) & (LRH) est représentée par les deux nœuds en couleur verte. Les arcs en bleu correspondent aux chemins de la concurrence, qui passent par les hubs suivant notre modèle. Finalement, les arcs en rose pointillé représentent les itinéraires de l'historique.

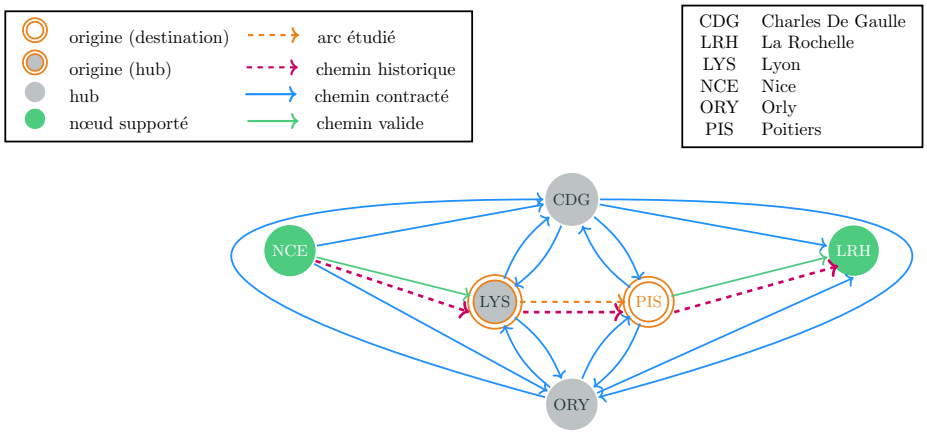


FIGURE 9.3 – graphe du cas centré sur le vol LYS–PIS.

La première étape du processus consiste à résoudre le problème de la sélection des marchés sur le réseau français pour le vol LYS – PIS. Pour raisons de simplicité, on a choisit un seul critère qui est la durée avec un regret de 3h, ce qui signifie que c'est le coût supplémentaire acceptable par rapport aux meilleurs chemins, pour un temps de transfert de 2h.

On constate que 65 aéroports en amont et 2 aéroports en aval sont sélectionnés. En termes de marchés, 130 de marchés sélectionnés parmi 4 225, ce qui veut dire qu'on arrive à éliminer déjà 97% des marchés peu prometteurs pour ce vol. En d'autres termes, ce sont les marchés dont la part des marchés est nulles.

Parmi les 130 paires d'aéroports sélectionnées, on va se focaliser sur (NCE) & (LRH). Ce qui implique 4 marchés qui offrent la possibilité de passer par le vol LYS – PIS pour aller de Nice (NCE) à la Rochelle (LRH).

Le tableau 9.6 représente les résultats obtenus de l'application du modèle de l'estimation des parts de marchés sur les 4 marchés définis précédemment.

On remarque qu'il y a un seul itinéraire pour chaque marché dans l'historique. On rappelle qu'un itinéraire est une possibilité pour aller de l'aéroport origine de départ à l'aéroport de la destination finale. Par ailleurs, on trouve le même ensemble d'itinéraires, ce qui veut dire qu'on retrouve les mêmes trajets des passagers en pratique avec la même part des marchés. On rappelle qu'on est sur un niveau stratégique, l'idée est de détecter des opportunités économiques et d'aller les revérifier sur des niveaux plus bas qui sont tactiques et opérationnelles. Dans cette étude de cas, on constate qu'une opportunité économique pour *Air France* serait d'ouvrir un vol direct entre Lyon (LYS) et la Rochelle (LRH), ce qui lui permettra de capturer les passagers venant en amont de Lyon mais également ceux passant par Poitiers (PIs) ; cet itinéraire est opéré actuellement par la compagnie aérienne française *Chalair aviation*.

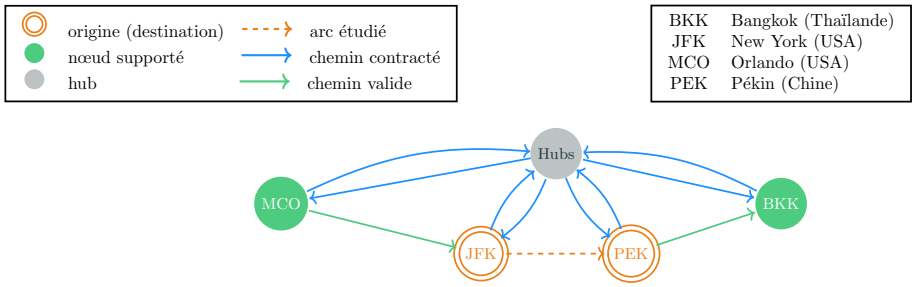


FIGURE 9.4 – graphe du cas centré sur le vol JFK–PEK.

Marché	Itinéraire	Type[1]	partE[2]	partH[3]
LYS & PIS	LYS -> PIS	direct	100%	100%
NCE & LRH	NCE -> LYS -> PIS -> LRH	2 cor.	100%	100%
LYS & LRH	LYS -> PIS -> LRH	1 cor.	100%	100%
NCE & PIS	NCE -> LYS -> PIS	1 cor.	100%	100%

[1] *Type d'itinéraire ;* [2] *Part estimée par le modèle ;* [3] *Part calculée sur l'historique.*

TABLE 9.6 – résultats du cas centré sur le vol LYS–PIS.

Ici, notre approche intégrée permet d'avoir des résultats très fiable avec un temps de calcul rapide.

9.5.3 Vol JFK–PEK

Dans cette étude de cas, on va suivre la même démarche que dans le cas précédent mais sur le réseau mondial centré sur le vol JFK – PEK (New York - Pékin) (voir la figure 9.4).

On rappelle que le réseau mondiale comporte 4 602 aéroports, 1 127 558 arcs (24 mois). La sélection des marchés permet ainsi de sélectionner 50 440 marchés, ce qui revient à éliminer 99% marchés peu prometteurs pour ce vol selon les mêmes paramètres (critère durée, avec un regret de 3h et temps de transfert de 2h). Suivant les résultats, on va se focaliser sur le marché Orlando (MCO) & Bangkok (BKK), pour lequel il existe 43 itinéraires dans l'historique, ce qui veut dire que les passagers qui voyagent entre Orlando et Bangkok ont 43 possibilités pour le faire.

Les résultats du tableau 9.7 représente le classement des 5 premiers itinéraires en termes de part de marchés. On retrouve le même classement que dans l'historique, notamment, l'itinéraire qui passe par le vol étudié JFK–PEK en deuxième position, qui capture une bonne partie du trafic. Ces premiers résultats confirment les conclusions et les hypothèses présentées avant. Par ailleurs, on remarque que l'écart entre la part des marchés estimée et celle calculée est faible. En fait, elle ne dépasse par 4%. D'après les experts métiers, cette marge est acceptable pour valider le modèle.

Les résultats des deux études de cas nous permet de valider notre méthode de la sélection des marchés. En fait, notre méthode permet bien de retrouver des marchés qui existent dans la réalité, qui sont prometteurs centré sur le vol étudié. Par ailleurs, ce cas valide à nouveau l'estimation des passagers, qui fournit des estimations de bonne qualité.

Marché Orlando (MCO) & Bangkok (BKK)

Itinéraire	**Type**[1]	**partE**[2]	**partH**[3]
MCO -> DXB -> BKK	1 cor.	24%	20%
MCO -> JFK -> PEK -> BKK	2 cor.	21%	17%
MCO -> JFK -> DXB -> BKK	2 cor.	3%	7%
MCO -> EWR -> PEK -> BKK	2 cor.	3%	5%
MCO -> IAD -> NRT -> BKK	2 cor.	2%	5%

[1] *Type d'itinéraire ;* [2] *Part estimée par le modèle.*
[3] *Part calculée sur l'historique.*

TABLE 9.7 – résultats du cas centré sur le vol LYS–PIS.

La sélection des marchés qui se fait par une approche algorithmique fournit des résultats très satisfaisants sur le niveau stratégique. En outre, le modèle de l'estimation des parts de marchés est très fiable sur les études des cas réalisées, en termes de temps de calcul, qui est très rapide à l'ordre de quelques secondes par rapport aux approches classiques et en termes d'erreur, où l'écart ne dépasse par 4%. Ces résultats préliminaires nous permettront ainsi de valider notre approche intégrée et de nous projeter dans le futur pour proposer des vols qui n'existent pas en réalité.

CHAPITRE 10

Conclusion et Perspectives

À ce jour, il n'est pas possible de proposer une démarche globale traitant les problèmes de planification en transport aérien en raison de la complexité du problème étudié. Il n'est certes pas simple d'optimiser la sélection des marchés aériens dans le domaine déjà très complexe de l'industrie aérienne.

L'objectif de cette thèse a été d'étudier les méthodes et de fournir un outil d'aide à la décision afin que les clients de la start-up *Milanamos* puissent prendre de meilleures décisions :

— modifier la fréquence ;

— proposer un nouveau vol ;

— proposer un vol partagé.

Le résultat obtenu devrait permettre à *Milanamos* d'améliorer l'efficacité globale et la qualité du calcul de part de marché, en prenant en compte des marchés *O&D* impactés ainsi que l'ensemble d'itinéraires reliant la paire *O&D*. Nos travaux sont orientés vers les niveaux anticipés de décision. *Milanamos* utilise les données historiques des 17 dernières années. L'un des défis de ce travail de thèse est la gestion d'un grand volume de données hétérogènes et incomplètes. Dans une première partie de ce manuscrit, nous présentons le contexte industriel de la thèse, l'outil développé par *Milanamos* et le modèle actuel pour le calcul des parts de marchés. La plupart des approches proposées dans la littérature traitant ce problème se focalisent sur un seul marché où le vol sera ouvert. Elles se contentent ainsi d'énumérer tous les itinéraires reliant la paire *O&D*. En pratique, la combinatoire sur l'énumération des itinéraires interdit le temps réel. Toutefois, ils ne prennent pas en considération les autres marchés impactés par l'ouverture de ce vol. Par exemple, un voyageur voyageant de *Bordeaux*/France (`BOD`) à *Bangkok*/Thaïlande (`BKK`) *via Charles De Gaulle*/France(`CDG`), sera potentiellement intéressé par l'ouverture du vol direct *Nice*/France (`NCE`) - `BKK`.

Nous consacrons ensuite la deuxième partie de ce travail aux travaux existant dans la littérature. Nous sommes intéressés par l'étude de trois axes de recherche :

1. Aide à la décision dans le transport aérien
2. Bases de données
3. Théorie des graphes

Le premier axe vise à étudier les différents problèmes d'optimisation qui découlent du processus de la planification dans le transport aérien ainsi que les approches proposées pour les résoudre.

Ensuite, le deuxième axe nous aidera à choisir la base de données adéquate, à savoir, passer à un autre système de gestion de base de données pour représenter plus naturellement notre graphe et décrire plus efficacement les algorithmes. Cela nous permet de stocker notre graphe modélisant le réseau aérien et faire face aux limites de la base actuelle de *Milanamos*. La théorie des graphes nous permet ainsi de formaliser le problème, déterminer sa complexité, et éventuellement choisir l'algorithme adéquat sur notre graphe pour résoudre la problématique de la recherche.

La troisième partie de ce manuscrit présente les contributions scientifiques de ce travail de thèse qu'on peut résumer en quatre points. Le premier point est lié à l'analyse des données aériennes de *Milanamos*. Comme les données viennent du monde réel, des données massives, incomplètes et hétérogènes, cette étape de pré-traitement des données est primordiale et est le point d'entrée pour mieux préparer nos données. En effet, la qualité des données joue un rôle très important sur le résultat attendu. Nous avons ainsi pu mieux comprendre les anomalies, faire des hypothèses tout en conservant un maximum de réalisme, estimer la taille et la complexité du problème.

Le deuxième point consiste à proposer une approche pour modéliser notre réseau aérien suivant nos hypothèses. Notre travail a pour but de trouver des opportunités économiques en tenant compte des préférences des passagers. Nous avons ainsi omis les horaires des vols et opté pour l'approche indépendante du temps. Nous avons proposé un modèle hybride entre le modèle condensé et le modèle temporel étendu. Le modèle condensé nous permettra de contrôler la taille du graphe, représenter sa structure et donc de tester rapidement la connexité. En outre, nous intégrerons quelques éléments du modèle temporel étendu, cela nous permettra d'inclure les temps de transfert dans le graphe pour avoir des trajets réalistes. Avec cette approche, nous sommes arrivés à calculer des bornes inférieures sur les plus courts chemins, qui sont largement suffisantes dans notre cadre d'étude.

En posant plusieurs hypothèses et en utilisant diverses approches pour simplifier le problème tout en conservant un maximum de réalisme, le réseau aérien est modélisé par un graphe indépendant du temps tel que les nœuds représentent les aéroports tandis qu'un arc indique l'existence d'un vol entre deux aéroports pour un *year_month*. Ce réseau sera utilisé par la suite pour estimer les parts de marchés. Toutefois, il a un intérêt immédiat pour répondre à de nombreuses questions auxquelles étaient très long voire difficile ou impossible d'y répondre avec la base actuelle `optimode`.

Le troisième point répond au problème de la sélection des marchés, qui expose la formulation mathématique proposée pour modéliser notre problème *Flight Radius*. Il s'agit de trouver un sous-graphe maximal, en termes de nœuds tels que les chemins sont valides. Un chemin valide est un chemin qui limite le surcoût sur le temps, la distance ou bien le prix. La formulation mathématique que nous avons proposée tient compte des préférences des usagers du réseau de transport aérien.

Pour résoudre le *Flight Radius Problem*, nous avons commencé par tester les méthodes proposées par la base de données orientée graphe `Neo4j`. Les résultats n'étaient pas satisfaisants en termes de temps. Nous avons ainsi proposé une approche algorithmique basée sur l'algorithme classique de `Dijkstra`. Les résultats préliminaires ont démontré que l'algorithme répond en quelques secondes dans le pire des cas. Par ailleurs, la solution renvoie également les vols avec plus d'une correspondance contrairement à l'affichage actuel sur *Milanamos*. De nombreux aéroports sont filtrés, ce qui permet d'accélérer le calcul des parts de marché.

Cependant, nous avons étudié plusieurs pistes d'amélioration : une optimisation fine de l'algorithme de `Dijkstra` et notamment de la queue de priorité, mais aussi l'évaluation des performances d'autres algorithmes de plus courts chemins comme `Bellman`. C'est dans ce cadre que

nous avons proposé deux approches différentes : une approche séquentielle basée sur la décomposition en plus court chemin et une approche basée sur le parallélisme.

Pour évaluer leur performance, nous avons choisi deux métriques : temps d'exécution et nombre de nœuds scannés par l'algorithme. En effet, le vrai défi sur les bases de données orientées graphe est de gérer les propriétés qui sont stockées dans des fichiers séparés, ce qui est coûteux puisque gourmand en accès mémoire. Les propriétés correspondent aux étiquettes en théorie des graphes. Les résultats obtenus démontrent que l'algorithme basé sur `Dijkstra` est le meilleur en termes de performance.

Le *Flight Radius Problem* est un problème polynomial lié au problème de recherche des plus courts chemins, issu d'une application dans le domaine du transport aérien qui peut être résolu par l'algorithme proposé. Même si les deux méthodes renvoient une solution après un nombre fini d'appels à des algorithmes de plus court chemin, la requête est plus lente que l'algorithme de plusieurs ordres de grandeur. Les performances de l'algorithme respectent les contraintes sur le temps de réponse de l'application.

Le dernier point de ce manuscrit présente les études de cas que nous avons proposées pour accélérer le calcul des parts de marché et faire face aux limites du modèle actuel de *Milanamos*. En effet, nous avons proposé un nouveau modèle basé sur le modèle multinomial couplé à des techniques d'accélération de calcul des plus courts chemins, notamment la technique du *Hub Labeling*, pour réduire la combinatoire sur l'énumération des itinéraires.

En comparant les résultats avec l'historique de *Milanamos*, nous avons validé le modèle. Nous espérons ainsi l'étendre pour tout marché potentiel de l'ensemble des marchés de la solution du *Flight Radius Problem*.

Pour conclure, si nous avions à résumer notre démarche de travail, nous proposerions un plan à neuf points :

1. analyser les données aériennes,
2. modéliser le réseau aérien,
3. construire le graphe,
4. transformer le graphe,
5. stoker le graphe,
6. analyser le réseau aérien,
7. identifier les hubs,
8. résoudre le problème de la sélection des marchés,
9. modéliser l'estimation des parts de marchés.

Perspectives

Les travaux développés dans cette thèse représentent la toute première étude sur l'analyse et la sélection des nouveaux marchés au sein de la start-up *Milanamos*. De ce fait, un travail d'analyse et de formalisation du problème ont été réalisés afin de disposer d'une vue claire de la problématique. Ceci constitue en soi un apport important et réutilisable pour les prochaines études sur ce sujet. C'est également le premier travail dans la littérature qui traite le modèle des parts de marché d'un point de vue plus combinatoire. Dans ce contexte, les résultats de cette thèse ouvrent de nombreuses perspectives.

À l'échelle de l'aérien, le nouveau modèle proposé pour le calcul des parts de marché se base sur le modèle multinomial (*logit*). Les coefficients ont été estimés pour le réseau des États-Unis. En analysant les coefficients, on remarque que les vols avec plus d'une correspondance sont plus pénalisés. Il est vrai que le réseau des États-Unis est plus grand que celui de la France. Ceci nous permettra d'ouvrir une piste pour estimer les coefficients en se basant sur les données de la France.

Par ailleurs, le modèle multinomial utilisé dépasse le modèle *QSI* d'après les résultats dans la littérature. Toutefois, le modèle multinomial devient difficile quand il s'agit de manipuler plusieurs critères. Dans ce cas-là, il se peut que l'utilisation des méthodes de l'apprentissage automatique puisse faciliter cette tâche pour les deux modèles. On pourrait ainsi améliorer le modèle du *QSI* développé par *Milanamos* suivant deux axes :

1. utiliser les méthodes de l'apprentissage automatique pour fixer au préalable les coefficients ;
2. reprendre la méthodologie du calcul de part de marché qui a été proposée avec le *logit* mais en remplaçant ce dernier par le modèle *QSI*.

Dans une direction similaire, nous pourrions intégrer les horaires. Il est essentiel de rappeler que notre modèle relâche les contraintes temporelles, le fait de passer d'une fréquence mensuelle à une fréquence hebdomadaire nous permettra d'apporter une aide à la décision sur le plan tactique en proposant au client une liste des choix des heures de départ du vol proposé.

À l'échelle des bases de données, plusieurs algorithmes de calcul de chemins ont été développés récemment par `Neo4j` [NeoTechnology, 2018]. Il serait intéressant d'évaluer ces algorithmes, avec l'espoir à terme, d'améliorer les temps d'exécution. Dans ce sens, l'appel à des algorithmes des plus courts chemins sur des bases de données typiquement `Neo4j`, engendre un coût qui devient vite important à cause du stockage des propriétés. Pour pallier cela, nous avons utilisé des techniques de parallélisme. L'algorithme qui a donné des bons résultats est l'algorithme basé sur `Dijkstra` dans une approche séquentielle. On rappelle que l'algorithme traite les directions en parallèle et les critères en séquentiel. Il semble ainsi raisonnable de paralléliser l'algorithme également pour les critères puisqu'ils sont traités indépendamment.

Nous avons aussi fait des pré-traitements afin de réduire la combinatoire sur l'énumération des itinéraires entre chaque paire d'aéroports de marché potentiel. La méthode basée sur l'algorithme *Hub Labeling* fonctionne parfaitement quand la liste des hubs est fournie sur un petit jeu de données. Toutefois, elle ne passe pas à grande échelle. En outre, un profilage du code lors de l'implémentation pourrait apporter des améliorations certaines en réduisant les temps de calcul de certaines parties du code qui sont sollicitées fréquemment et qui représentent la plus grande charge de travail à effectuer par l'ordinateur. Mais ce travail relève plutôt du génie logiciel.
L'idée principale reste toutefois valide : il serait envisageable de changer les techniques utilisées. En effet, le fait de se concentrer uniquement sur les itinéraires passant par des hubs nous permettra de capturer tout le flux des passagers. La démarche utilisée pour réduire le graphe et ne garder que les marchés potentiels où les parts de marchés sont significatives demeure une bonne idée.

Un dernier point sur la collecte des données qui représente la clé de tout processus d'aide à la décision. D'après nos analyses, nous avons remarqué que les données extraites de la base de *Milanamos* comportent des erreurs. L'analyse du graphe nous a permis de détecter ces erreurs et de les remonter aux responsables pour les corriger. Nous avons ainsi passé du temps à refaire la démarche mais en faisant un pré-traitement avant la génération du graphe. Travailler avec des jeux de données plus complets conduirait à des résultats plus précis, notamment, le manque sur les données des revenus, distance et durée.

Le croisement des cinq disciplines abordées dans cette thèse, l'aérien, les formules économiques, les bases de données, la théorie des graphe et l'aide à la décision, permet de mieux modéliser les problèmes réels, notamment sur le plan stratégique et opérationnel. On peut prendre l'exemple du virus *COVID-19*. En regardant la carte mondiale des cas, on remarque qu'il existe une forte corrélation entre les principaux aéroports internationaux et la venue du virus dans ces pays, car en effet, les passagers aériens en provenance de pays infectés propageaient le virus dans le pays destination : les passagers étaient donc le vecteur de propagation du virus. L'analyse du réseau aérien permettra ainsi de mieux suivre la propagation du virus.

Bibliographie

A. Abdelghany and K. Abdelghany. *Airline Network Planning and Scheduling*. John Wiley & Sons, 2018a.

A. Abdelghany and K. Abdelghany. *Airline Network Planning and Scheduling*. John Wiley & Sons, 2018b.

I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In *International Symposium on Experimental Algorithms*, pages 230–241. Springer, 2011.

I. Abraham, D. Delling, A. V. Goldberg, and R. F. Werneck. Hierarchical hub labelings for shortest paths. In *European Symposium on Algorithms*, pages 24–35. Springer, 2012.

N. Adler. Hub-spoke network choice under competition with an application to western europe. *Transportation science*, 39(1) :58–72, 2005.

S. Agrawal and A. Patel. A study on graph storage database of nosql. *International Journal on Soft Computing, Artificial Intelligence and Applications*, 5 :33–39, 02 2016. doi : 10.5121/ijscai.2016.5104.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows : theory, algorithms, and applications*. Prentice hall, 1993.

D. Allen, A. Hodler, M. Hunger, M. Knobloch, W. Lyon, M. Needham, and H. Voigt. Understanding trolls with efficient analytics of large graphs in neo4j. *BTW 2019*, 2019.

R. Angles. A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177. IEEE, 2012.

R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1) :1, 2008.

C. Barnhart and A. Cohn. Airline schedule planning : Accomplishments and opportunities. *Manufacturing & service operations management*, 6(1) :3–22, 2004.

C. Barnhart and B. Smith. *Quantitative problem solving methods in the airline industry*. Springer, 2012.

C. Barnhart, P. Belobaba, and A. R. Odoni. Applications of operations research in the air transport industry. *Transportation science*, 37(4) :368–391, 2003a.

C. Barnhart, P. Belobaba, and A. R. Odoni. Applications of operations research in the air transport industry. *Transportation science*, 37(4) :368–391, 2003b.

H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

S. Batra and C. Tyagi. Comparative analysis of relational and graph databases. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(2) :509–512, 2012.

N. Béchet. État de l'art sur les systèmes de recommandation. 2012.

P. Belobaba, A. Odoni, and C. Barnhart. *The global airline industry*. John Wiley & Sons, 2015.

I. Ben-Gan, A. Machanic, D. Sarka, and K. Farlee. *T-SQL Querying*. Microsoft Press, 2015.

C. Berge. Graphes et hypergraphes. 1973.

K. Bhamra. A comparative analysis of mongodb and cassandra. Master's thesis, The University of Bergen, 2017.

J. A. Bondy, U. S. R. Murty, et al. *Graph theory with applications*, volume 290. Citeseer, 1976.

M. Braimniotis. *Transformation from ORM Conceptual Models to Neo4j Graph Database*. PhD thesis, 2017.

R. Bruchez. *Les bases de données NoSQL et le BigData : Comprendre et mettre en oeuvre*. Editions Eyrolles, 2015.

N. Budhiraja, M. Pal, and A. Pal. Airline network design and fleet assignment : using logit-based dynamic demand-supply interaction. Technical report, working Paper, Indian Institute of Management Calcutta, 2006.

M. Buerli. The current state of graph databases. 2012.

Bureau de l'observation du marché. Bulletin statistique du trafic aérien commercial, 2017.

J. G. Busquets, E. Alonso, and A. D. Evans. Air itinerary shares estimation using multinomial logit models. *Transportation Planning and Technology*, 41(1) :3–16, 2018.

D. Chakrabarti and C. Faloutsos. Graph mining : Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1) :2, 2006.

D. G. Chandra. Base analysis of nosql database. *Future Generation Computer Systems*, 52 : 13–21, 2015.

B. V. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest paths algorithms : Theory and experimental evaluation. *Mathematical programming*, 73(2) :129–174, 1996.

K. Chodorow. *MongoDB : the definitive guide : powerful and scalable data storage*. " O'Reilly Media, Inc.", 2013.

M. Ciglan, A. Averbuch, and L. Hluchy. Benchmarking traversal operations over graph databases. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 186–189. IEEE, 2012.

E. F. Codd. Data models in database management. *ACM Sigmod Record*, 11(2) :112–114, 1981.

E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5) :1338–1355, 2003.

G. M. Coldren. *Modeling the Competitive Dynamic Among Air-travel Itineraries with Generalize Extreme Value Models*. PhD thesis, Northwestern University, 2005.

G. M. Coldren and F. S. Koppelman. Modeling the competition among air-travel itinerary shares : Gev model development. *Transportation Research Part A : Policy and Practice*, 39(4) :345–365, 2005.

G. M. Coldren, F. S. Koppelman, K. Kasturirangan, and A. Mukherjee. Modeling aggregate air-travel itinerary shares : logit model development at a major us airline. *Journal of Air Transport Management*, 9(6) :361–369, 2003.

D. Comer. Ubiquitous b-tree. *ACM Computing Surveys (CSUR)*, 11(2) :121–137, 1979.

A. Czerepicki. Application of graph databases for transport purposes. *Bulletin of the Polish Academy of Sciences Technical Sciences*, 64(3) :457–466, 2016.

C. Dasadia and A. Nayak. *MongoDB Cookbook*. Packt Publishing Ltd, 2016.

D. Delling, T. Pajor, D. Wagner, and C. Zaroliagis. Efficient Route Planning in Flight Networks. *ATMOS*, 12, 2009a. ISSN 2190-6807. doi : http://dx.doi.org/10.4230/OASIcs.ATMOS.2009.2145. URL http://drops.dagstuhl.de/opus/volltexte/2009/2145.

D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. In *Algorithmics of large and complex networks*, pages 117–139. Springer, 2009b.

D. Delling, A. V. Goldberg, and R. F. Werneck. Hub label compression. In *International Symposium on Experimental Algorithms*, pages 18–29. Springer, 2013.

D. Delling, A. V. Goldberg, R. Savchenko, and R. F. Werneck. Hub labels : Theory and practice. In *International Symposium on Experimental Algorithms*, pages 259–270. Springer, 2014.

A. Dey, A. Fekete, and U. Röhm. Scalable transactions across heterogeneous nosql key-value data stores. *Proceedings of the VLDB Endowment*, 6(12) :1434–1439, 2013.

D. Di Wang, D. Klabjan, and S. Shebalov. Attractiveness-based airline network models with embedded spill and recapture. *Journal of Airline and Airport Management*, 7(1) :1–25, 2014.

G. Dobson and P. J. Lederer. Airline scheduling and routing in a hub-and-spoke system. *Transportation Science*, 27(3) :281–297, 1993.

M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

R. Elmasri and S. Navathe. *Fundamentals of database systems*. Addison-Wesley Publishing Company, 2010.

A. E. Eltoukhy, F. T. Chan, and S. H. Chung. Airline schedule planning : a review and future directions. *Industrial Management & Data Systems*, 117(6) :1201–1243, 2017.

L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

D. Fernandes and J. Bernardino. Graph databases comparison : Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. 2018.

P. Fortin, C. Morency, and M. Trépanier. Innovative gtfs data application for transit network analysis using a graph-oriented method. *Journal of Public Transportation*, 19(4) :2, 2016.

L. R. Foulds. *Graph theory applications*. Springer Science & Business Media, 2012.

M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3) :596–615, 1987.

L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

P. Gachoki and M. Muraya. Comparison of models used to predict flight delays at jomo kenyatta international airport. *Asian Journal of Probability and Statistics*, pages 1–8, 2019.

X. Gandibleux, F. Beugnies, and S. Randriamasy. Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function. *4OR : A Quarterly Journal of Operations Research*, 4(1) :47–59, 2006.

G. Gardarin. *Bases de données*. Editions Eyrolles, 2003.

L. A. Garrow. *Discrete choice modelling and air travel demand : theory and applications*. Routledge, 2016.

R. Geisberger. *Advanced route planning in transportation networks*. PhD thesis, Universitat Karlsruhe (TH) - Institut für Theoretische Informatik (ITI), 2011.

R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies : Faster and simpler hierarchical routing in road networks. In *International Workshop on Experimental and Efficient Algorithms*, pages 319–333. Springer, 2008.

P. Goedeking. *Networks in aviation : strategies and structures*. Springer Science & Business Media, 2010.

T. Grosche. *Computational intelligence in integrated airline scheduling*, volume 173. Springer-Verlag Berlin Heidelberg, 2009a.

T. Grosche. Airline scheduling process. In *Computational Intelligence in Integrated Airline Scheduling*, pages 7–46. Springer, 2009b.

T. Grosche and F. Rothlauf. Air travel itinerary market share estimation. 04 2007.

J. A. Gubner. *Probability and random processes for electrical and computer engineers*. Cambridge University Press, 2006.

O. A. Guide. *DOT ANALYSER USER GUIDE*, JULY 2018. URL http://www.oag.com.

N. Halpern and A. Graham. Airport route development : A survey of current practice. *Tourism Management*, 46 :213–221, 2015.

J. Hölsch, T. Schmidt, and M. Grossniklaus. On the performance of analytical and pattern matching graph queries in neo4j and a relational database. In *EDBT/ICDT 2017 Joint Conference : 6th International Workshop on Querying Graph Structured Data (GraphQ)*, 2017.

F. Holzschuher and R. Peinl. Performance of graph query languages : comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 195–204. ACM, 2013.

F. Holzschuher and R. Peinl. Performance optimization for querying social network data. In *EDBT/ICDT Workshops*, pages 232–239, 2014.

ICAO. List of low-cost-carriers(lccs. https://www.icao.int/sustainability/Documents/LCC-List.pdf, 2017.

A. K. Idrissi, A. Malapert, and R. Jolin. The route network development problem based on qsi models. pages 3–11, 2017.

A. K. Idrissi, A. Malapert, and R. Jolin. Solving the flight radius problem. pages 304–311, 2018. doi : 10.5220/0006654003040311.

A. K. Idrissi., A. Malapert., and R. Jolin. Flight radius algorithms. pages 370–377, 2019. doi : 10.5220/0007388503700377.

T. L. Jacobs, L. A. Garrow, M. Lohatepanont, F. S. Koppelman, G. M. Coldren, and H. Purnomo. Airline planning and schedule development. In *Quantitative Problem Solving Methods in the Airline Industry*, pages 35–99. Springer, 2012.

P. S. Jadhav and R. Oberoi. Comparative analysis of different graph databases. *International Journal of Engineering Research & Technology (IJERT)*, 3(9) :820–824, 2015.

P. Jaillet, G. Song, and G. Yu. Airline network design and hub location problems. *Location science*, 4(3) :195–212, 1996.

U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec. Radius plots for mining tera-byte scale graphs : Algorithms, patterns, and observations. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 548–558. SIAM, 2010.

D. Kim and C. Barnhart. Transportation service network design : Models and algorithms. In *Computer-Aided Transit Scheduling*, pages 259–283. Springer, 1999.

D. Kirchler. *Efficient routing on multi-modal transportation networks*. PhD thesis, Ecole Polytechnique X, 2013.

F. S. Koppelman, G. M. Coldren, and R. A. Parker. Schedule delay impacts on air-travel itinerary demand. *Transportation Research Part B : Methodological*, 42(3) :263–273, 2008.

D. M. Kroenke and D. J. Auer. *Database processing*, volume 6. Prentice Hall, 2013.

P. Lacomme, C. Prins, and M. Sevaux. *Algorithmes de graphes*, volume 28. Eyrolles Paris, 2003.

M. Lal. *Neo4J Graph Data Modeling*. Packt Publishing, 2015. ISBN 1784393444, 9781784393441.

P. Larsson. Analyzing and adapting graph algorithms for large persistent graphs, 2008.

P. J. Lederer and R. S. Nambimadom. Airline network design. *Operations Research*, 46(6) : 785–804, 1998.

J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time : densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

M. Lohatepanont and C. Barnhart. Airline schedule planning : Integrated models and algorithms for schedule design and fleet assignment. *Transportation Science*, 38(1) :19–32, 2004.

A. E. Lotfy, A. I. Saleh, H. A. El-Ghareeb, and H. A. Ali. A middle layer solution to support acid properties for nosql databases. *Journal of King Saud University-Computer and Information Sciences*, 28(1) :133–145, 2016.

J. J. Louviere, D. A. Hensher, and J. D. Swait. *Stated choice methods : analysis and applications*. Cambridge university press, 2000.

I. Maduako, E. Cavalheri, and M. Wachowicz. Exploring the use of time-varying graphs for modelling transit networks. *arXiv preprint arXiv :1803.07610*, 2018.

T. L. Magnanti and R. T. Wong. Network design and transportation planning : Models and algorithms. *Transportation science*, 18(1) :1–55, 1984.

A. Martínez Porras, R. Mora Rodríguez, D. Alvarado González, G. López Herrera, and S. Quirós Barrantes. A comparison between a relational database and a graph database in the context of a personalized cancer treatment application. 2016.

D. McFadden et al. Conditional logit analysis of qualitative choice behavior. 1974.

Milanamos. *User Manual of PlanetOptim*, 2016. URL http://po.milanamos.com/usermanual.

M. Miler, D. Medak, and D. Odobašić. The shortest path algorithm performance comparison in graph and relational database on a transportation network. *Promet-Traffic&Transportation*, 26 (1) :75–82, 2014.

J. J. Miller. Graph database applications and concepts with neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, volume 2324, page 36, 2013.

MongoDB. Mongodb manual. https://docs.mongodb.com, 2016.

MongoDB. Mongodb manual, 2017.

S. Morishima and H. Matsutani. Performance evaluations of graph database using cuda and openmp compatible libraries. *ACM SIGARCH Computer Architecture News*, 42(4) :75–80, 2014.

S. A. T. Mpinda, L. C. Ferreira, M. X. Ribeiro, and M. T. P. Santos. Evaluation of graph databases performance through indexing techniques. *International Journal of Artificial Intelligence & Applications (IJAIA) Vol*, 6 :87–98, 2015.

A. Nayak, A. Poriya, and D. Poojary. Type of nosql databases and its comparison with relational databases. *International Journal of Applied Information Systems*, 5(4) :16–19, 2013.

N. T. Neo4j and Cypher. Neo4j apoc library, 2020.

NeoTechnology. Community detection algorithms, 2018.

J. Nesetril, E. Milková, and H. Nesetrilová. Otakar boruvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1-3) :3–36, 2001. doi : 10.1016/S0012-365X(00)00224-7. URL https://doi.org/10.1016/S0012-365X(00)00224-7.

Oracle. Mysql reference manual, 2019.

T. Pajor. *Multi-modal route planning*. PhD thesis, Universitat Karlsruhe (TH) - Institut für Theoretische Informatik (ITI), 2009.

T. Pajor. *Algorithm Engineering for Realistic Journey Planning in Transportation Networks*. PhD thesis, 2013.

C. Partl, S. Gratzl, M. Streit, A. M. Wassermann, H. Pfister, D. Schmalstieg, and A. Lex. Pathfinder : Visual analysis of paths in graphs. In *Computer Graphics Forum*, volume 35, pages 71–80. Wiley Online Library, 2016.

J. Pokornỳ. Graph databases : their power and limitations. In *IFIP International Conference on Computer Information Systems and Industrial Management*, pages 58–69. Springer, 2015.

R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6) :1389–1401, 1957.

R. Rai and P. Chettri. Nosql hands on. In *Advances in Computers*, volume 109, pages 157–277. Elsevier, 2018.

S. Raj. *Neo4j high performance*. Packt Publishing Ltd, 2015.

T. Reiners, J. Pahl, M. Maroszek, and C. Rettig. Integrated aircraft scheduling problem : An auto-adapting algorithm to find robust aircraft assignments for large flight plans. In *2012 45th Hawaii International Conference on System Sciences*, pages 1267–1276. IEEE, 2012.

I. Robinson, J. Webber, and E. Eifrem. *Graph databases : new opportunities for connected data*. " O'Reilly Media, Inc.", 2015.

C. C. Robusto. The cosine-haversine formula. *The American Mathematical Monthly*, 64(1) : 38–40, 1957.

M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6) :35–41, 2010.

C. Schön. Market-oriented airline service design. In *Operations Research Proceedings 2006*, pages 361–366. Springer, 2007.

Z. Sha, K. Moolchandani, A. Maheshwari, J. Thekinen, J. H. Panchal, and D. A. DeLaurentis. Modeling airline decisions on route planning using discrete choice models. In *15th AIAA Aviation Technology, Integration, and Operations Conference*, page 2438, 2015.

G. ShefaliPatil and A. Bhatia. Graph databases-an overview. 2014.

D. Shimpi and S. Chaudhari. An overview of graph databases. In *Int. Conf. on Recent Trends in Information Technology and Computer Science*, pages 16–22, 2012.

A. Silberschatz, H. F. Korth, S. Sudarshan, et al. *Database system concepts*, volume 4. McGraw-Hill New York, 1997.

M. Stonebraker and R. Cattell. Ten rules for scalable performance in simple operation datastores. *Communications of the ACM*, 10, 2011.

R. Sylvain, R. Nicolas, and M. Nicolas. *Neo4j : des données et des graphes*. "Éditions D-BookeR", 2016.

D. Taniar, C. H. Leung, W. Rahayu, and S. Goel. *High-performance parallel database processing and grid databases*, volume 67. John Wiley & Sons, 2008.

R. Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2) : 146–160, 1972.

N. Technology. Neo4j. https://www.neo4j.com, 2017.

https://www.flightsfrom.com, 2019.

ourairports.com/. Open data, 2019.

G. Vaish. *Getting started with NoSQL*. Packt Publishing Ltd, 2013.

A. Vukotic, N. Watt, T. Abedrabbo, D. Fox, and J. Partner. *Neo4j in action*, volume 22. Manning Shelter Island, 2015.

T. F. E. WIKIPEDIA. List of hub airports. https://en.wikipedia.org/wiki/List_of_hub_airports, 2019.

Y. Xia, I. G. Tanase, L. Nai, W. Tan, Y. Liu, J. Crawford, and C.-Y. Lin. Graph analytics and storage. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 942–951. IEEE, 2014.

J. Y. Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11) : 712–716, 1971.

G. Yu and J. Yang. Optimization applications in the airline industry. In *Handbook of combinatorial optimization*, pages 1381–1472. Springer, 1998.

D. Zhang, C. Yu, and H. Y. K. Lau. An integrated flight scheduling and fleet assignment method based on a discrete choice model. *Computers & Industrial Engineering*, 98 :195–210, 2016.

Liste des figures

État de l'art

Contributions

Liste des tableaux

État de l'art

Contributions

Liste des définitions

Liste des exemples

Liste des codes

Liste des Algorithmes

Annexes

A Description des données

Voici les fichiers des données sur les aéroports et les vols utilisés dans ce chapitre.

Code	Name
NCE	Nice Côte d'Azur Airport
CDG	Chales De Gaulle Airport
DXB	Dubaï International Airport
ICN	Incheon International Airport
DOH	Hamad International Airport
CMN	Mohammed V International Airport
ABJ	Félix Houphouët Boigny International Airport
KWI	Koweït International Airport
NBO	Jomo Kenyatta International Airport
BKK	Suvarnabhumi Airport
LIS	Lisbon International Airport
ESB	Ankara Esenboğa Airport
LHR	Heathrow Airport
GLA	Glasgow Airport
FNC	Madeira Airport
IST	Istanbul Atatürk Airport

FIGURE A.1 – Données aéroports.

Id	Origin	Destination	Duration	Cost
1	NCE	BKK	660	2100
2	NCE	DXB	375	1000
3	NCE	DOH	360	900
4	NCE	CMN	170	400
5	DXB	KWI	100	150
6	DXB	BKK	380	900
7	IST	NCE	170	350
8	CDG	NCE	100	150
9	LIS	NCE	150	300
10	IST	DXB	245	500
11	BKK	ICN	310	850
12	CMN	ABJ	270	400
13	CDG	IST	210	300
14	LHR	CDG	70	80
15	CDG	NBO	485	
16	IST	ESB	75	130
17	LHR	GLA	85	120
18	LIS	FNC	105	300
20	CMN	DOH	425	
21	CMN	NBO	465	780
22	DOH	KWI	75	160

FIGURE A.2 – Données vols.

B Programme 2-hops

Nous fournissons ici le programme proposé pour la méthode 1 de la requête 2-hops en MongoDB présenté dans la section 5.4.6.2 du chapitre 5.

```
1  from pymongo import MongoClient
2
3  def destinations(origin):
4      pipe = [
5          {'$match': {'Origin': origin}},
6          {'$group': {
7              '_id': None,
8              'mesDest': {'$addToSet': '$Destination'}}}]
9      cursor = db.flights.aggregate(pipe)
10     return cursor
11
12 def destinationsDeMesDest(orgListDest,listDest):
13     pipe = [
14         {'$match': {'Origin': {'$in':listDest},
15                     'Destination':{'$nin':orgListDest}}},
16         {'$group': {
```

```
17              '_id': None,
18              'leurDest': {'$addToSet': '$Destination'}}}]
19      cursor = db.flights.aggregate(pipe)
20      return cursor
21
22  def main():
23      print("connect to flightdatabase")
24      # Create a new connection to MongoDB instance
25      connection = MongoClient()
26      # Create and get a data base (db) from this connection
27      db = connection.testFlightDB
28
29      origin = 'NCE'
30      listDest = destinations(origin)
31      for doc in listDest:
32          mesDest = doc['mesDest']
33      orgListDest =  list(mesDest) + [origin]
34      desMesDest = destinationsDeMesDest(orgListDest, mesDest)
35      for doc in desMesDest:
36          leurDest = doc['leurDest']
37
38      print("liste des destinations: ", leurDest)
39
40  if __name__ == '__main__':
41          main()
```

Listing B.1 – programme pour récupérer la liste des destinations `Python`

C Résultats des requêtes en `Neo4j`

C.1 Création

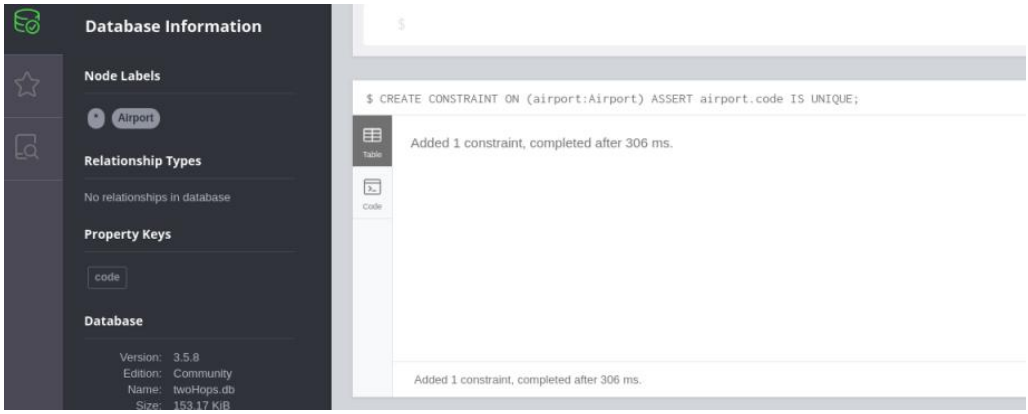


FIGURE C.1 – Résultat de la requête du listing 5.32.

C.2 Affichage

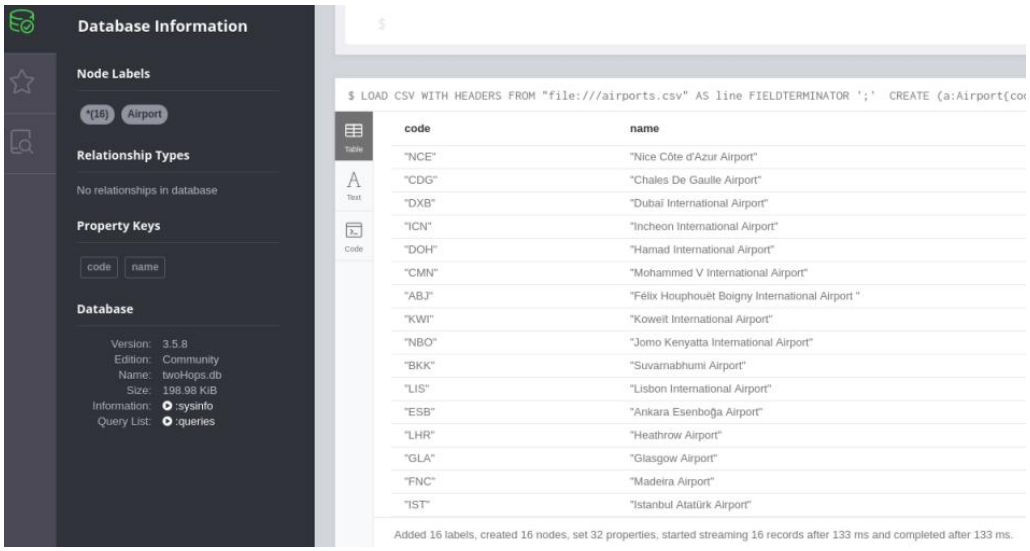


FIGURE C.2 – Résultat de la requête du listing 5.34.

C.3 Liste des destinations

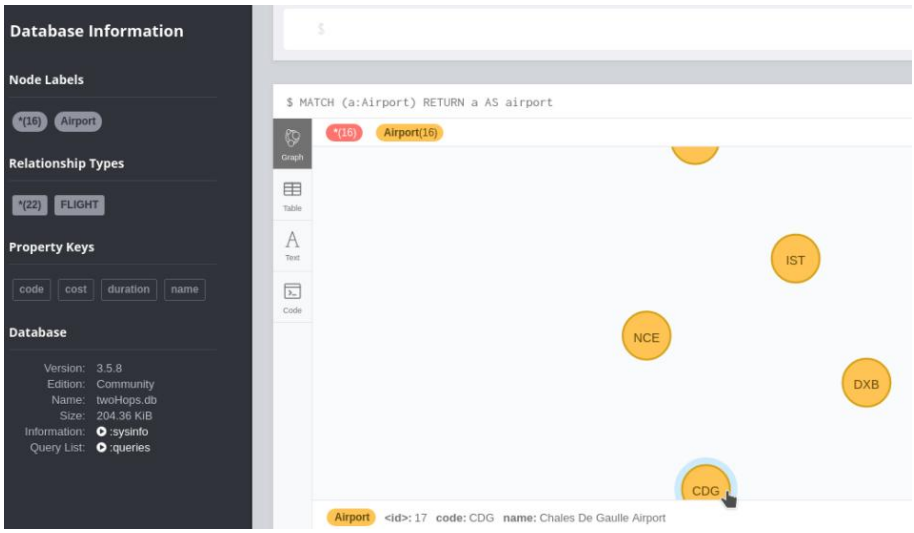


FIGURE C.3 – Résultat de la requête du listing 5.36.

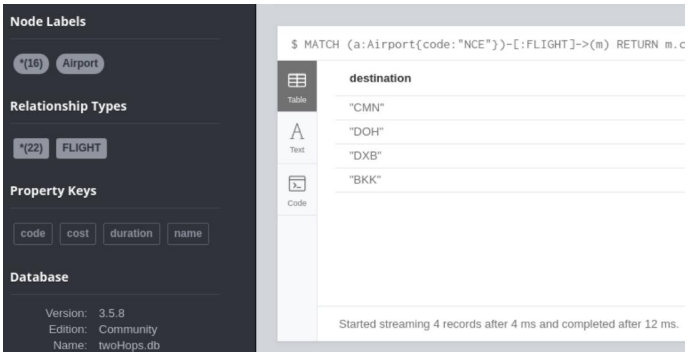


FIGURE C.4 – Résultat de la requête du listing 5.37.

C.4 Destinations en moins de 2 heures

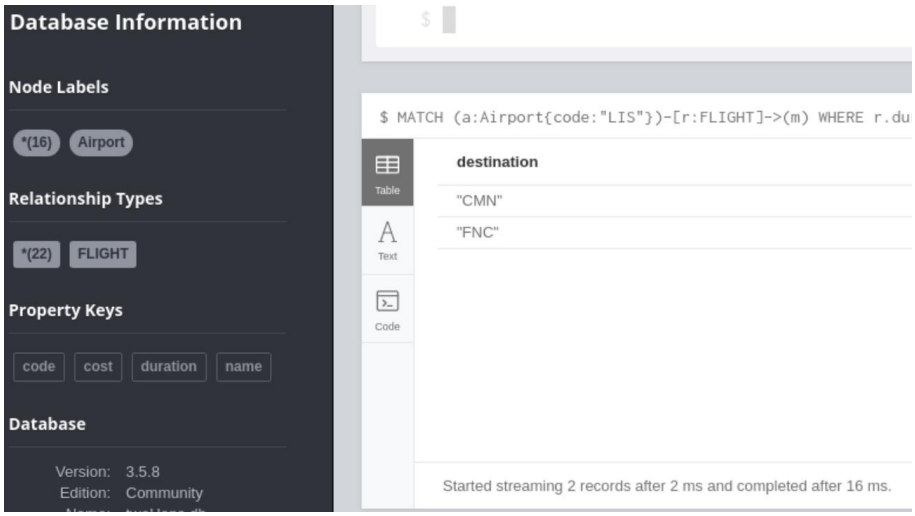


FIGURE C.5 – Résultat de la requête du listing 5.39.

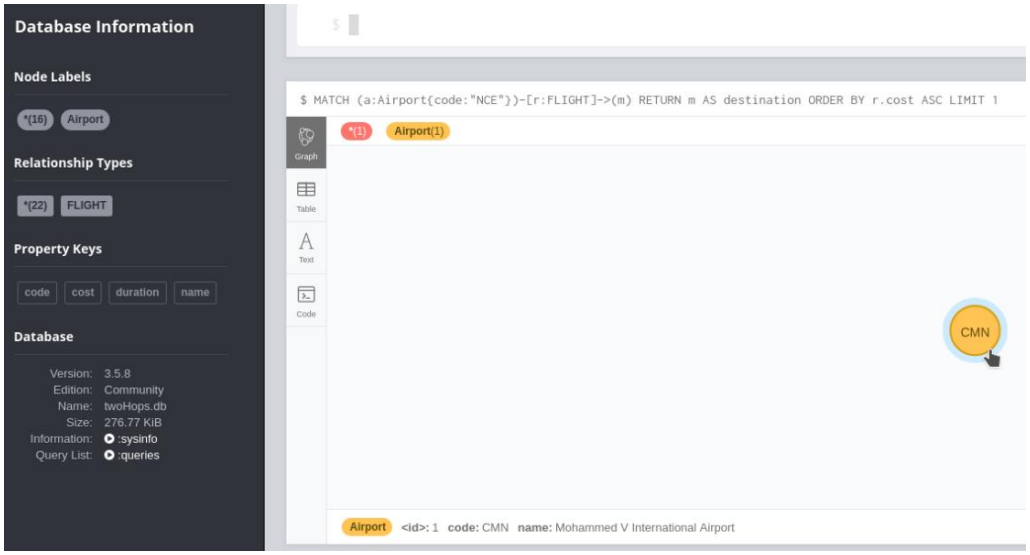


FIGURE C.6 – Résultat de la requête du listing 5.40.

C.5 Résultat de la requête 2-hops

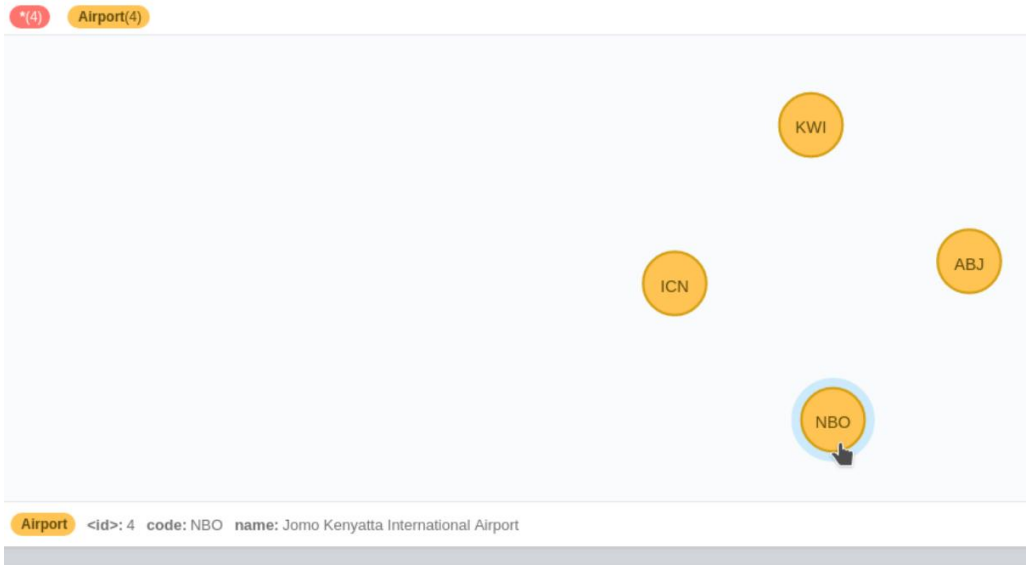


FIGURE C.7 – Résultat de la requête du listing 5.42.

D Identification des hubs

Le programme D.1 présente une partie du programme fait pour l'identification des hubs. La première fonction `get_lookup_lcc` concerne le calcul de la présence des *LCC* en termes de flux de passagers. On passe en paramètre la liste des compagnies aériennes *LCC* de notre graphe *CFN*. La deuxième méthode est pour le calcul des passagers qui transitent par chaque aéroport.

```
def get_lookup_lcc(lccList, hub_airports):
    pax_lcc = {}
    pipe = [
        {'$match': {'record_ok': True,
        'year_month': {'$gte': '2017-01', '$lte': '2017-12'}}},
        {'$group': {
            '_id': {
                'origin': '$leg_origin',
                'airl': '$operating_airline'},
            'pax': {'$sum': '$passengers'}}}]
    cursor0 = SegmentInitialData.aggregate(pipe, allowDiskUse=True)
    print("finish aggregation")

    for d in cursor0:
         id = d['_id']
        code =  id['origin']
        airl =  id['airl']
        pax = d['pax']
        if code in hub_airports.keys():
            if code in pax_lcc.keys():
                pax_data = pax_lcc.get(code)
                pax_data.update({airl:pax})
            else:
                pax_lcc[code]= {airl:pax}

    pipe = [
        {'$match': {'record_ok': True,
        'year_month': {'$gte': '2017-01', '$lte': '2017-12'}}},
        {'$group': {
            '_id': {
                'origin': '$leg_destination',
                'airl': '$operating_airline'},
            'pax': {'$sum': '$passengers'}}}]
    cursor0 = SegmentInitialData.aggregate(pipe, allowDiskUse=True)
    print("finish aggregation")

    for d in cursor0:
         id = d['_id']
        code =  id['origin']
        airl =  id['airl']
        pax = d['pax']
        if code in hub_airports.keys():
            if code in pax_lcc.keys():
                pax_data = pax_lcc.get(code)
                if airl in pax_data.keys():
                    pax_airl = pax_data[airl]
```

```
                    paxOD = pax_airl + pax
                    pax_data.update({airl:paxOD})
            else:
                pax_lcc[code] = {airl: pax}

    for code in hub_airports.keys():
        print("code ", code)
        paxT = 0
        paxL = 0
        if code in pax_lcc:
            airList = pax_lcc[code]
            print("airList ", airList)
            paxT =  sum(airList.values())
            print("paxT ", paxT)
            for air in airList.keys():
                if air in lccList:
                    print("air ", air)
                    paxL = paxL + airList[air]
                    print("paxL ", paxL)
            presLcc =  round(paxL / paxT, 3)
            hub_data = hub_airports.get(code)
            hub_data.update( dict(presLCC= float(presLcc)))
    return hub_airports


def get_pax_airport(hub_airports):
    pipe = [
        {'$match': {'record_ok': True,
        'year_month': {'$gte': '2017-01', '$lte': '2017-12'}}},
        {'$group': {
            '_id': {
                'origin': '$leg_origin',
                'destination': '$leg_destination'},
            'pax': {'$sum': '$passengers'}}}]
    cursor = SegmentInitialData.aggregate(pipe, allowDiskUse=True)
    print("finish aggregation")

    for d in cursor:
         id = d['_id']
        org =  id['origin']
        des =  id['destination']
        pax = d['pax']
        if org in hub_airports:
            hub_data = hub_airports.get(org)
            if 'pax' in hub_data:
                p = hub_data['pax']
```

```
                np = pax + p
                hub_data.update( dict(pax= int(np)))
            else:
                hub_data.update( dict(pax= int(pax)))
        if des in hub_airports:
            hub_data = hub_airports.get(des)
            if 'pax' in hub_data:
                p = hub_data['pax']
                np = pax + p
                hub_data.update( dict(pax= int(np)))
            else:
                hub_data.update( dict(pax= int(pax)))
    return hub_airports
```

Listing D.1 – partie du programme pour l'identification des hubs en `Python`

E Calcul des parts de marchés

E.1 Modèle utilisé par *Milanamos*

E.1.1 Fréquence directe

```
db.schedule_initial_data.aggregate([
{
         '$match': {
                'record_ok': true,
                'active_rec': true,
                'flights_dates': {
                    '$gte':  ISODate("2019-03-11T00:00:00.000Z"),
                    '$lte':  ISODate("2019-03-17T00:00:00.000Z")},
                'year_month': {'$gte':'2019-03','$lte': '2019-03'},
                'origin': $o,
                'destination': $d,
                'code_share': false,
                'schedule_type': 'AIR'
            }
},
        {'$unwind': '$flights_dates'},
        {'$match': {'flights_dates': {
            '$gte': ISODate("2019-03-11T00:00:00.000Z"),
            '$lte': ISODate("2019-03-17T00:00:00.000Z")}}},
        {
            '$group': {
                '_id': {
                    'origin': '$origin',
                    'operating_airline': '$operating_airline',
```

```
                    'hour': '$arrival_time.hour',
                    'mn': '$arrival_time.mn',
                    'flight_date': '$flights_dates'
                }
            }
        },
        {
            '$group': {
                '_id': 1,
                'count': {'$sum': 1}
}}],{allowDiskUse:true})
```

Listing E.1 – requête pour la première étape du calcul de la fréquence directe.

E.1.2 Fréquence indirecte

```
db.schedule_initial_data.aggregate([
{
         '$match': {
                'record_ok': true,
                'active_rec': true,
                'flights_dates': {
                '$gte':  ISODate("2019-03-11T00:00:00.000Z"),
                '$lte':  ISODate("2019-03-17T00:00:00.000Z")},
                'year_month': {'$gte':"2019-03", '$lte': "2019-03"},
                'origin': $o,
                'code_share': false,
                'schedule_type': 'AIR'}},
        {'$unwind': '$flights_dates'},
        {'$match': {'flights_dates': {
            '$gte': ISODate("2019-03-11T00:00:00.000Z"),
            '$lte': ISODate("2019-03-17T00:00:00.000Z")}}},
        {
            '$group': {
                '_id': {
                    'connection': '$destination',
                    'base_airport': '$o',
                    'operating_airline': '$operating_airline',
                    'hour': '$arrival_time.hour',
                    'mn': '$arrival_time.mn'
                },
                     'flights_dates': {
                                '$addToSet': '$flights_dates'},}}
],{allowDiskUse:true})
```

Listing E.2 – requête pour la première étape du calcul de la fréquence indirecte.

Le dernier bloc *group* de la requête ci-dessous permet de regrouper le résultat pour chaque paire $(origine, d)$.

```
db.schedule_initial_data.aggregate([
{
          '$match': {
                 'record_ok': true,
                 'active_rec': true,
                 'flights_dates': {
                 '$gte':  ISODate("2019-03-11T00:00:00.000Z"),
                 '$lte':  ISODate("2019-03-17T00:00:00.000Z")},
                 'year_month': {'$gte':"2019-03", '$lte': "2019-03"},
                 'destination': $d,
                 'code_share': false,
                 'schedule_type': 'AIR'}},
         {'$unwind': '$flights_dates'},
         {'$match': {'flights_dates': {
         '$gte': ISODate("2019-03-11T00:00:00.000Z"),
         '$lte': ISODate("2019-03-17T00:00:00.000Z")}}},
         {
             '$group': {
                 '_id': {
                     'connection': '$origin',
                     'base_airport': '$d',
                     'operating_airline': '$operating_airline',
                     'hour': '$arrival_time.hour',
                     'mn': '$arrival_time.mn'
                 },
                      'flights_dates': {
                                 '$addToSet': '$flights_dates'},}}
],{allowDiskUse:true})
```

Listing E.3 – requête pour la deuxiéme étape du calcul de la fréquence indirecte.

E.1.3 Présence de la compagnie aérienne

```
db.schedule_initial_data.aggregate([
         {
             '$match': {
                 'record_ok': true,
                 'active_rec': true,
                 'flights_dates': {
                     '$gte':  ISODate("2019-03-11T00:00:00.000Z"),
                     '$lte':  ISODate("2019-03-17T00:00:00.000Z")},
                 'year_month': {'$gte':"2019-03", '$lte': "2019-03"},
                 'origin': $o,
```

```
                'code_share': false,
                'schedule_type': 'AIR'}},
        {'$unwind': '$flights_dates'},
       {'$match': {'flights_dates': {
            '$gte': ISODate("2019-03-11T00:00:00.000Z"),
            '$lte': ISODate("2019-03-17T00:00:00.000Z")}}},
        {
            '$group': {
                '_id': {
                    'operating_airline': '$operating_airline',
                },
                'destinations': {'$addToSet': '$destination'}
            }}])
```

Listing E.4 – requête pour trouver la liste des destinations desservies par chaque compagnie aérienne.

E.2 Modèle proposé

E.2.1 Étape du prétraitement des hubs

Le code du listing E.5 présente la requête de la contraction du graphe pour la direction sortante.

```
MATCH (h:Hub:Airport),(a:Arrival)
WHERE h.code <> a.code
CALL apoc.algo.dijkstra(h,a,'2017-10>|BOARD_AT|CONNECT_TO>', 'duration_min')
    YIELD path AS pH, weight AS dur
WITH path, duration, SIZE([node IN NODES(path)[2..-1] WHERE node:Hub]) / 2 AS
    wei,
apoc.coll.sum([rel IN RELATIONSHIPS(path) | COALESCE(rel.distance, 0)]) AS
    distance,
apoc.coll.sum([rel IN RELATIONSHIPS(path) | COALESCE(rel.revenue_min, 0)]) AS
    revenue,
apoc.coll.min([rel IN RELATIONSHIPS(path) WHERE EXISTS(rel.frequency) | rel.
    frequency]) AS frequency,
apoc.coll.min([rel IN RELATIONSHIPS(path) WHERE EXISTS(rel.capacity) | rel.
    capacity]) AS capacity
RETURN {path:path, duration:duration, wei:wei, distance:distance, revenue:
    revenue, frequency:frequency, capacity:capacity} AS path
```

Listing E.5 – requête de la contraction du graphe en *Cypher* pour la direction sortante

E.2.2 Requête pour le calcul des parts de marché

Nous présentons le code en *Cypher* pour le calcul des parts de marché sous forme de quatre requêtes pour faciliter la lecture.

La requête du listing E.6 commence par récupérer le sous-graphe associé au marché demandé. Ensuite, elle calcule le critère niveau de service.

La requête du listing E.7 consiste à calculer la fréquence et la capacité.

```
MATCH p=(o:Airport{code:o_code})-[:`2017-10`|CONTRACT*1..3]->(d:Airport{code:{
    d_code}})
WITH o,d, COLLECT(p) AS pp
OPTIONAL MATCH q=SHORTESTPATH((oT:Departure{code:o.code})-[:`2017-10`|
    CONNECT_TO*1..5]->(dT:Arrival{code:d.code}))
WITH pp+q AS paths,oT,dT
UNWIND paths AS p
WITH p, LENGTH(p) AS l,oT,dT
WITH DISTINCT l,l-1 AS nbrE,oT,dT,p
WITH DISTINCT l,nbrE,p,CASE WHEN EXISTS((oT)-[:`2017-10`]-(dT))
THEN CASE WHEN nbrE=0 THEN 0 ELSE CASE WHEN nbrE=1 THEN -2.7087 ELSE -7.5112
    END END
ELSE CASE WHEN EXISTS((oT)-[:`2017-10`|CONNECT_TO*3]-(dT))
THEN CASE WHEN nbrE >1 THEN -2.5431 ELSE 0 END END END AS ls
```

Listing E.6 – requête pour le calcul du niveau de service.

```
WITH ls, nbrE,p, l
WITH p, RELATIONSHIPS(p) AS rels,ls,l
WITH p, HEAD(rels) AS frel,ls,l
WITH p, frel.frequency AS freq,ls,l
WITH {path: p, levelService: ls,length:l} AS paths,freq
WITH COLLECT(paths) AS ps, freq
WITH {nbrP:SIZE(ps),freq:freq, path:ps} AS freqs
WITH DISTINCT (freqs['nbrP']*freqs['freq']) AS waF,freqs
WITH SUM(waF) AS sf, COLLECT(freqs) AS freqALL
UNWIND freqALL AS freqs
UNWIND freqs['path'] AS p
WITH ((freqs['freq']/toFloat(sf))*100)*0.0078 AS pos, p
UNWIND p AS paths
WITH apoc.map.setKey(paths,'pos',pos) AS paths
WITH paths, RELATIONSHIPS(paths['path']) AS rr
UNWIND rr AS r
WITH paths, COLLECT(r.capacity) AS caps, COLLECT(r.frequency) AS freqs
WITH paths, caps, apoc.coll.min(caps) AS minCap, freqs
WITH paths, caps, minCap, apoc.coll.indexOf(caps,minCap) AS ind, freqs
WITH paths, (minCap/freqs[ind]) AS lf
WITH paths, lf*0.0047 AS seats
WITH apoc.map.setKey(paths,'seats',seats) AS paths
```

Listing E.7 – requête pour le calcul de la fréquence et la capacité.
La requête du listing E.8 consiste à calculer les critères : distance, durée et revenu.

```
WITH paths, REDUCE(sum=0, x IN RELATIONSHIPS(paths['path'])| sum + COALESCE(x.
    distance, 0)) AS distance,
REDUCE(sum=0, x IN RELATIONSHIPS(paths['path'])| sum + COALESCE(x.duration_min,
     0)) AS duration,
REDUCE(sum=0, x IN RELATIONSHIPS(paths['path'])| sum + COALESCE(x.revenue_min,
    0)) AS revenue
WITH apoc.map.setKey(paths,'distance',distance) AS paths, revenue,duration
WITH apoc.map.setKey(paths,'duration',duration) AS paths, revenue
WITH revenue*(-0.0045) AS fare, paths
WITH apoc.map.setKey(paths,'fare',fare) AS paths
WITH COLLECT(paths) AS pp, MIN(paths['distance']) AS minDis
UNWIND pp AS paths
```

```
10 WITH paths,(toFloat(paths['distance'])/minDis)*100 AS DR
11 WITH (DR*(-0.0210)) AS DRC, paths
12 WITH apoc.map.setKey(paths,'DistanceRatio',DRC) AS paths
13 WITH paths, CASE WHEN paths['length'] > 1 THEN paths['duration'] ELSE 36000
       END AS durC
14 WITH MIN(durC) AS minDur, COLLECT(paths) AS pp
15 UNWIND pp AS paths
```

Listing E.8 – requête pour le calcul des critères de la distance et le revenu et la durée.

La requête du listing E.9 consiste à calculer les critères de la catégorie : qualité de la correspondance.

```
1  WITH minDur, paths,REDUCE(sum=0, x IN RELATIONSHIPS(paths['path'])| sum +
       COALESCE(x.nbrEscale, 0)) AS nbrEC
2  WITH apoc.map.setKey(paths,'nbrEC',nbrEC) AS paths, minDur
3  WITH minDur,paths, CASE WHEN paths['duration']=minDur THEN (paths['length']-1+
       paths['nbrEC'])*120 ELSE 0 END AS durConA
4  WITH MAX(durConA) AS bsMCT,minDur, COLLECT(paths) AS pp
5  UNWIND pp AS paths
6  WITH paths,bsMCT,minDur, CASE WHEN paths['length'] > 1 THEN CASE WHEN paths['
       duration'] > minDur THEN -0.5322 ELSE 0 END ELSE 0 END AS sbc
7  WITH apoc.map.setKey(paths,'sbc',sbc) AS paths, minDur, bsMCT
8  WITH bsMCT,paths,CASE WHEN paths['length']>1 THEN (paths['duration']-minDur)
       *(-0.0178) ELSE 0 END AS bctd
9  WITH apoc.map.setKey(paths,'bctd',bctd) AS paths, bsMCT
10 WITH paths, CASE WHEN paths['sbc'] <>0 THEN (((paths['length']-1+paths['nbrEC
       '])*120)-bsMCT)*(-0.0093) ELSE 0 END AS sbctd
11 WITH apoc.map.setKey(paths,'sbctd',sbctd) AS paths
12 WITH paths['levelService'] + paths['DistanceRatio'] + paths['sbc'] + paths['
       sbctd'] + paths['bctd'] + paths ['fare'] + paths['pos'] + paths['seats']
       AS utility,paths
13 WITH EXP(utility) AS exp, paths
14 WITH apoc.map.setKey(paths,'utility',exp) AS paths
15 WITH SUM(paths['utility']) AS sum, COLLECT(paths) AS pp
16 UNWIND pp AS paths
17 WITH paths['path'] AS p, paths,sum
18 WITH NODES(p) AS np, paths,sum
19 UNWIND np AS nn
20 WITH COLLECT(nn.code) AS nodesP, paths,sum
21 RETURN nodesP,(paths['utility']/sum)*100
```

Listing E.9 – requête pour le calcul des critères de la qualité de la correspondance

La requête du listing E.10 représente le pré-traitement par lot :

```
1  CALL apoc.periodic.iterate(
2  "MATCH (h:Hub:Departure),(a:Arrival) WHERE h.code<>a.code Call apoc.algo.
       dijkstra(h,a,'2017-10>|CONNECT_TO>', 'duration_min') YIELD path, weight AS
       durW WITH h,a,path, durW, SIZE(NODES(path)[2..-1]) / 2 AS nbrE,SIZE([node
       IN NODES(path)[2..-1] WHERE node:Hub])/2 AS wei, apoc.coll.sum([rel IN
       RELATIONSHIPS(path) | COALESCE(rel.distance, 0)]) AS dis,apoc.coll.sum([rel
        IN RELATIONSHIPS(path) | COALESCE(rel.revenue_min, 0)]) AS rev,apoc.coll.
       min([rel IN RELATIONSHIPS(path) WHERE EXISTS(rel.frequency) | rel.frequency
       ]) AS freq,apoc.coll.min([rel IN RELATIONSHIPS(path) WHERE EXISTS(rel.
       capacity) | rel.capacity]) AS cap RETURN h,a,wei,nbrE,dis,rev,freq,cap,(
       durW-(120*nbrE)) AS dur,[n IN NODES(path) | n.code] AS chem", "match (aa:
```

```
    Airport)<-[:'ALIGHT_AT']-(a) WHERE NOT EXISTS((h)-[:CONTRACT]->(aa)) CALL
    apoc.merge.relationship(h,'CONTRACT',{},{nbrHub:wei,nbrEscale:nbrE,
    distance:dis,duration_min:dur,revenue_min:rev,frequency:freq,capacity:cap,
    chemin:chem},aa,{}) YIELD rel AS rc RETURN rc",{batchSize:100, parallel:
    false,iteratelist:true});
```

Listing E.10 – requête de prétraitement par lot

La requête du listing E.11 consiste à estimer le nombre d'arcs crées lors du prétraitement pour une seule direction :

```
1 MATCH (h:Hub),(a:Arrival)
2 WHERE h.code<>a.code
3 CALL apoc.algo.dijkstra(h,a,'2017-10>|BOARD_AT|CONNECT_TO>', 'duration_min')
      YIELD path AS pH, weight AS dur
4 WITH pH,nodes(pH)[2..] AS nn
5 UNWIND nn AS n
6 WITH pH,nn, COUNT(n) AS nbrT
7 WITH (nbrT-1)/2 AS nbrST, nn,pH
8 WITH nbrST+1 AS nbrC,nn,pH
9 RETURN COUNT(nbrC)
```

Listing E.11 – requête pour l'estimation des arcs crées