POLYTECH TOURS

université de TOURS

Rapport de stage individuel

4ème année

Qualité et ponctualité dans les transports maritimes réguliers de voyageurs

Autorité de la Qualité de Service dans les Transports
Tour Séquoia, La Défense

AQST
AUTORITÉ DE LA QUALITÉ DE SERVICE DANS LES TRANSPORTS

Tuteur entreprise :
Alain Sauvant
Directeur de l'AQST

Tuteur académique :
Hervé Baptiste

Côme Geoffray
IUT
2020-2021

Sommaire :

L'Autorité de la Qualité de Service dans les Transports

Présentation Générale

D'après son site[1], l'Autorité de la Qualité de Service dans les Transports (AQST) est une entité autonome du ministère chargé des transports créée par les décrets n°2012-211 et 2012-216 datant de février 2012 qui a pour vocation de chercher des moyens d'améliorer la qualité de services dans les transports. L'AQST est placée au sein du conseil général de l'Environnement et du Développement durable pour ce qui est de sa gestion administrative. Ainsi, depuis 2012, l'AQST diffuse des statistiques concernant la ponctualité et la régularité des lignes régulières aériennes et ferroviaires nationale et internationales. D'autres documents sont aussi publiés sur le site concernant la qualité de l'information diffusée aux voyageurs en toutes situations. La qualité de service dans les transports est définie comme étant « l'aptitude du système de transports à répondre aux besoins de ses utilisateurs » d'après A. Sauvant (2020). La norme NF EN 13816 de l'AFNOR donne ainsi les huit principales parties de cette qualité de service que sont : l'offre de service, l'accessibilité, l'information, la durée, l'attention portée au client, le confort, la sécurité et l'impact environnemental[2]. L'AQST est dirigée par Alain Sauvant depuis 2016 après une proposition du vice-président du Conseil général de l'environnement et du développement durable ainsi que les avis favorables du ministre chargé des transports et de la ministre chargée de la consommation. A.Sauvant est un ingénieur général des ponts, des eaux et des forêts, qui enseigne aussi l'économie des transports à l'école des ponts ParisTech.

Les missions de l'AQST

L'AQST a notamment pour mission de publier sur son site (ww.qualitetransports.gouv.fr) des statistiques concernant la ponctualité et la régularité de différents modes de transports de voyageurs en France que sont, pour l'instant, le mode de transport aérien ainsi que les liaisons ferroviaires. Ces statistiques sont produites selon une méthodologie fixe à partir de données transmises par la Direction générale de l'Aviation civile (DGAC), les autorités organisatrices de la mobilité (AOM) et les opérateurs de transport en accord avec des conventions-cadres. Cette première mission consiste donc en la création et l'utilisation d'une grande base de données concernant la qualité de service dans les transports. La seconde mission de l'AQST est de renseigner les usagers sur leurs droits et de les accompagner dans leurs démarches. Ce site est le premier d'Europe à fournir une approche multimodale de la qualité de service dans les transports. Par ailleurs, l'AQST s'intéresse aussi aux systèmes et dispositifs de traitement des réclamations et de médiations proposés au public dans le but d'étudier leur efficacité et leur impartialité au besoin. L'AQST s'occupe donc des relations avec les différents acteurs de la qualité de service dans les transports. Enfin, l'AQST fournit régulièrement des analyses et propositions afin d'améliorer la qualité de service dans les transports grâce à des études de satisfaction ou d'autres méthodes encore. L'AQST produit aussi un rapport annuel d'activité.

[1] « Site officiel de l'AQST », s. d., http://www.qualitetransports.gouv.fr/.

[2] Comité Européen de Normalisation, « Norme NF EN 13816 » (AFNOR, septembre 2002).

La mission

La fiche de stage mentionne les informations suivantes : « l'AQST souhaiterait préciser quelques éléments de la ponctualité dans les transports de voyageurs maritimes réguliers (donc hors croisières et assimilées).
Divers sites permettent un suivi temps réel des navires (ferries et/ou passages d'eau), incluant aussi les navires de marchandises. On pourra ainsi apparier les observations avec les horaires théoriques pour constituer une maquette de la ponctualité (transmanche, Corse, outre-mer, passages d'eau, …).
Dans certains cas il peut s'agir de dessertes opérées par un transporteur pour son propre compte, et dans d'autres pour celui d'une autorité organisatrice, ou d'un groupement d'autorités organisatrices. Un regard sur les contrats et les incitations à la qualité qui y figurent pourra également être utilement effectué.
Pourrait être examinée également la desserte terrestre des ports de voyageurs. (Par exemple l'existence d'une offre, les fréquences, les amplitudes, de la connexion du port maritime avec les transports publics terrestres interurbains et urbains…)
Des comparaisons internationales pourront être tentées. »
Plus généralement donc, la mission de ce stage est de fournir une première maquette, voir des premiers chiffres concernant la ponctualité des transports maritimes réguliers de passagers. Ce domaine n'ayant pas encore -à l'heure actuelle- été étudié par l'Autorité de la Qualité de Service dans les Transports, les points d'intérêts sont divers. Tout d'abord, il s'agira de chercher un ou des moyens de récupérer des données sur les déplacements réels effectués par les navires et sur ceux prévus, annoncés par les compagnies de transports maritimes de voyageurs. Dans un second temps et dans le but de produire des taux de retard, il sera nécessaire d'apparier les données sur les horaires d'arrivée théoriques et effectives des navires. Pour en déduire des taux de retard à l'arrivée, il sera également nécessaire de déterminer un ou des seuils de retard appropriés permettant de coller au mieux à la réalité et au ressenti des voyageurs. Enfin, il s'agirait de reproduire ces méthodes avec des lignes étrangères afin d'effectuer des comparaisons qui pourraient permettre de mettre en évidence les forces et faiblesses des transports maritimes de passagers français.
Une seconde partie de cette mission serait, en suivant le texte de la fiche de stage, de s'intéresser à la desserte terrestre des ports de voyageurs. Il s'agirait donc ici d'étudier directement les divers modes d'accès aux ports de voyageurs afin de déterminer s'ils offrent une qualité de service satisfaisante. .

Le déroulé de la mission

Collecte des données théoriques

Introduction

Afin de produire des chiffres illustrant les retards dans les transports maritimes réguliers de voyageurs, il me faut nécessairement comparer les horaires d'arrivée effectives à ce qui était annoncé par les différentes compagnies opérant des liaisons dans les zones étudiées. Pour ce faire, il me faut donc tout d'abord collecter des données dites théoriques auprès des compagnies avant de collecter les données réelles. En effet, il n'est nullement possible de partir collecter des données réelles sans connaître au préalable ne serait-ce que les navires à étudier et les ports desservis. Cette première partie s'intéresse donc à la collecte des données théoriques auprès des compagnies. La question de l'automatisation de cette collecte de données est assez vite réglée. En effet, dans un premier temps, il a été décidé avec M. Barbusse de s'intéresser à des périodes d'environ un mois pour chaque zone étudiée, afin de disposer d'un échantillon de trajets suffisamment large pour produire des statistiques représentatives de la ponctualité des liaisons maritimes étudiées. Après une brève observation des différents sites des compagnies, j'estime que la création d'un programme capable d'aller chercher automatiquement les horaires théoriques sur ces sites serait une perte de temps pour de petites périodes. En effet, la variété de ces sites ainsi que les nombreuses constructions différentes imposeraient de personnaliser très précisément le programme pour chacun des sites. Qui plus est, avant d'avoir inventorié les liaisons d'une compagnie sur une période donnée, il est difficile d'estimer si leur volume justifierait ou non la création d'un programme. Aucune sélection des sites préalable n'est donc efficacement réalisable.
C'est pourquoi je décide de collecter « à la main » ces données. Seul le stockage de ces données tendra donc à être optimisé concrètement. Cette tâche étant toutefois laborieuse, elle sera répartie tout au long de ce stage, tout en respectant bien entendu les différentes échéances liées à la collecte des données de géolocalisation des navires – basée sur la technologie Automatic Identification System (AIS). En effet, il me faut connaître l'intégralité des navires à étudier avant de pouvoir appliquer les méthodes de la partie suivante.

Constitution d'une base de données

Dans l'optique décrite précédemment d'optimiser la collecte à la main, je choisis de créer une base de données, sous Microsoft Access afin de stocker les nombreux trajets théoriques annoncés par les compagnies maritimes. Ce format présente plusieurs avantages ; Tout d'abord, il permettra lors de l'utilisation de ces grandes quantités de données de gagner un temps considérable dans la sélection des entrées à utiliser au moyen de requêtes adaptées. D'autre part, la recherche du minimum de redondances pratiquée communément lors de la création d'une base de données limite intrinsèquement le temps passé à entrer les données dans la base. Finalement, ce format pourra permettre par la suite de relier les entrées AIS aux entrées Théoriques (j'utilise ici le mot « entrées » car on verra par la suite que ces dernières ne décrivent pas la même chose) au moyen d'identifiants, ce qui simplifiera grandement l'étape dite du « couplage des données ».
Je reçois alors l'autorisation de travailler sur Access. Je construis donc dans un premier temps le modèle MCD de la base de données *Figure 1*.

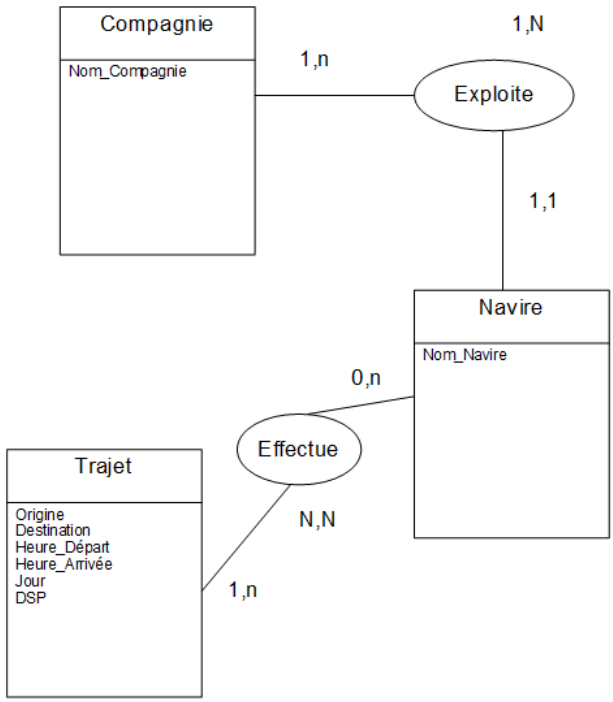


Figure 1: Premier modèle MCD de la base de données de l'offre de transport maritime

Il s'agit donc ensuite de transformer ce modèle théorique en base de données concrète dans Access. Après avoir créé l'architecture de la base de données sous Access, je m'attèle donc à la remplir avec les données des sites de compagnies de transports maritimes de voyageurs. En récoltant de nombreuses donnés, je constate parfois des incohérences ou imprécisions sur ce que proposent ces sites. Dans ces cas-là, j'essaye de rester vague pour englober l'ensemble des possibilités (tout particulièrement sur les trajets qui ne mentionnent pas explicitement quel navire opérera la liaison). Je collecte ainsi dans un premier temps les données théoriques de la zone Transmanche pour le mois de Mai. Ensuite, celles des liaisons avec la Corse pour le mois de Juin et, enfin, j'essaye de récolter des données concernant d'autres zones de transport maritime de voyageur en France (Outre-mer, passages d'eau sur le littoral de France métropolitaine, etc.) pour le mois de Juillet. A la fin de cette vaste collecte de données théoriques sur les trajets maritimes de voyageurs, ma base de données recueillera quelques 6504 trajets théoriques différents.

Des vieilles et des nouvelles tables

Une modification a été amenée afin de gagner du temps sur la collecte des données théoriques et suite à mon étude de la réalité et de la régularité des trajets. La base a tout d'abord été construite avec un champ « jour » pour chaque trajet qui correspondait au jour dans la semaine (du 3 au 10 Mai) de ce trajet. En effet, la première version ne concernait qu'une seule semaine, c'était une forme de test. L'utilisation de cette base devant par la suite être étendue à de plus grandes périodes, mon système devenait obsolète et nécessitait maintenant l'association des trajets à une date précise. Plusieurs options furent alors envisagées pour rajouter l'information sur la date de départ du trajet annoncé par la compagnie maritime.

Un champ « Date »

Une première option est étudiée. Il s'agirait simplement de transformer notre table trajet et de remplacer le champ « Jour » par un champ Date *Figure 2*.

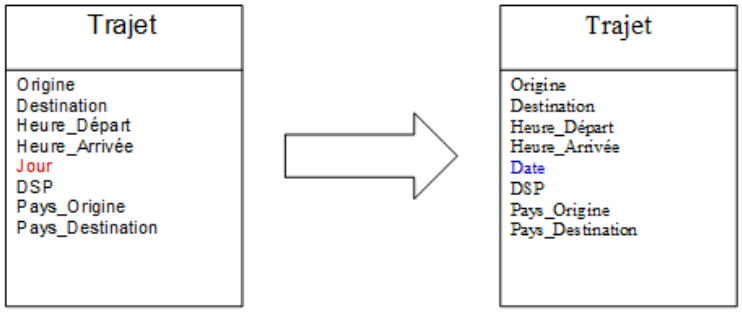


Figure 2: Première solution envisagée

Il serait alors assez facile de convertir les données précédemment enregistrées en passant par excel avec une condition IF pour obtenir les dates correspondantes. Cependant, il faudrait alors copier et cloner les données en grande quantité sur la totalité de la période étudiée en modifiant manuellement les dates pour les horaires récurrents. Cela prendrait beaucoup de temps lors de la collecte.

Un booléen

L'idée ici serait de créer un champ booléen qui, s'il est vrai impliquerait que le trajet en question serait récurrent sur l'intégralité de la période étudiée : le champ jour serait alors conservé. S'il est faux en revanche, il donnerait accès à un champ « Date » qui permettrait donc de noter la date précise de l'entrée de la table « Trajet ». On notera que je considère les récurrences des trajets semaine après semaine.

Ce modèle présente toutefois des inconvénients. Si un trajet était en fait récurrent sur une période incluse dans la période totale d'étude, il faudrait alors l'entrer plusieurs fois (comme dans le premier modèle) pour chacune des dates correspondantes (cependant, étant donné que la période d'étude sera probablement assez limitée, ce problème pourrait bien ne pas en être un). Un second problème serait celui de la divergence des formats des dates. Pour obtenir les trajets d'un jour j, les requêtes seraient alors plus complexes à construire.

Un champ occurrence

Enfin, une dernière idée de solution serait de remplacer le champ « Jour » par un champ « Date_Départ ». Un second champ serait ensuite créé : un champ occurrence contenant un entier qui correspondrait au nombre de semaines consécutives pour lesquelles ce trajet aura lieu. Concrètement c'est probablement la meilleure solution car elle apporte les avantages de la précédente tout en enlevant ses inconvénients. Je choisis donc cette dernière option mais je conserve pour l'instant le champ « Jour » par simplicité (qui plus est la donnée n'est pas vraiment redondante et fournie une information facile qui évite de longues requêtes servant à récupérer le jour correspondant à chaque date).

Grâce à cette méthode, j'enregistre au 11 Mai près de 2255 trajets stockés dans moins de 800 enregistrements différents (on voit donc bien là l'efficacité de cette méthode qui limite grandement le nombre d'entrées différentes).

Par la suite, en étudiant l'architecture de la base de données et dans l'optique de réduire les redondances, j'opère de nouvelles modifications. Je décide de créer une table « Port » qui contiendra le champ « Nom_Port » ainsi qu'une table « Pays » permettant de limiter les redondances. De plus, cela ouvrira des possibilités pour la suite afin de qualifier les ports plus précisément en utilisant les données par ports.

Les données des noms des ports et des pays anciennement entrées sont converties au moyen d'un champ calculé sur excel en ID numériques et viennent remplacer les anciens champs dans la table Trajet. Au moyen d'une requête particulière, je peux toujours récupérer une table fournissant les mêmes informations qu'auparavant. Bien que cela complexifie quelque peu certaines requêtes, la base de données s'en voit renforcée et allégée *Figure 3*.

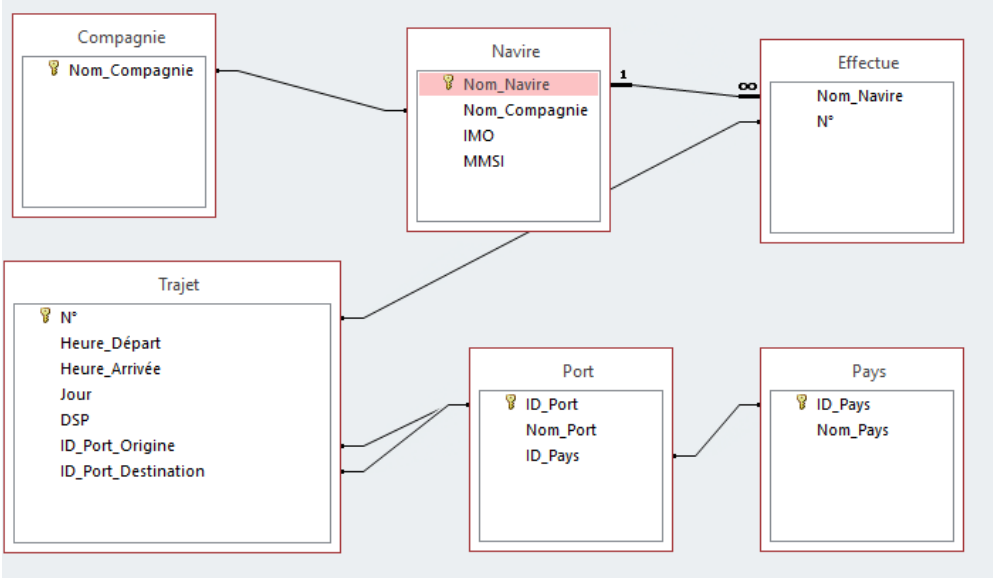


Figure 3: Structure finale de la base de données des trajets Théoriques

Difficultés dans la collecte des données

Une des conditions nécessaires à l'exploitation de données est l'existence d'un port enregistré simultanément sur le site de la compagnie et sur www.vesselfinder.com – le site source utilisé pour récupérer les données de géolocalisation des navires. Par exemple, après avoir enregistré les trajets de la compagnie Océane vers Belle-Île, je notais qu'ils reliaient aussi l'ile de Hoëdic. Cependant, ce port n'étant pas enregistré sur www.vesselfinder.com, la page du bateau effectuant ces trajets n'affichait alors que des arrêts à Quiberon. Les autres ne sont tout simplement pas détectés. Ce problème m'empêche ainsi de prendre les données vers : L'île de Hoëdic, l'île de Groix, l'île de Houat et l'île de Ré en ce qui concerne les passages d'eau de la côte Atlantique. De plus, les petits bateaux ne sont eux aussi pas nécessairement enregistrés comme par exemple, les vedettes servant de navettes entre Marseille, l'île du Frioul et le Château d'If. Par la suite de nombreux cas impliquant l'un ou l'autre de ces écueils m'empêcheront de récupérer des données. Les « petits » trajets et passages d'eau demeurent ainsi très compliqués à étudier.

D'autres problèmes encore ont été rencontrés pour certains territoires. Il s'agit de l'accès à l'information qui est bien plus difficile sur les îles. En effet, récupérer une liste exhaustive de

compagnies opérant en Guadeloupe par exemple a été compliqué et a nécessité de croiser plusieurs documents et sites entres eux[3][4][5].
Ainsi, il a été très difficile de récupérer des données dans les départements et territoires d'Outre-Mer de par l'utilisation récurrente de navires de petites tailles non identifiés par l'AIS. La couverture des trajets hors zones Transmanche ou liaisons avec la Corse n'est donc que très sporadique et ne sera probablement pas suffisamment représentative pour sortir des statistiques fiables en fin de stage.

Collecte des données de géolocalisation AIS

Introduction

Il s'agit ici de chercher un ou des moyens afin d'automatiser la collecte des données provenant de l'AIS. Après avoir envisagé d'autres méthodes, je note que de nombreux sites comme (www.vesselfinder.com, www.marinetraffic.com, www.myshiptracking.com etc…) proposent des données de l'AIS en ligne obtenues grâce à des partenariats avec des stations radio collectant elles-mêmes ces données AIS. Ces sites proposent tous de vendre leurs données historiques ou futures pour des sommes que l'AQST ne peut à priori pas débloquer aussi simplement. Toutefois, ces sites affichent aussi en temps réel les dernières informations enregistrées pour chaque navire de leur base de données –On notera alors ici que ces sites possèdent une base de données de ports et navires, identifiés par différents codes. Les navires enregistrés sont ceux possédant un système AIS (définition à la page suivante) et correspondant aux critères de la convention « Safety Of Life At Sea » évoquée plus tard. Il apparait donc très vite ici que l'enjeu de cette partie va être le développement d'un programme (automatisation de clics ou récupération de flux d'informations) permettant de prendre régulièrement les données gratuites données par ces sites afin de reconstituer sur des périodes entières les déplacements des bateaux ayant retenus mon attention.
Il faut noter ici que bien que les sites en question se réservent le droit de bannir des machines ou connexions internet pour ce genre de pratiques, il n'est nullement répréhensible d'utiliser ce genre de pratiques à des fins non commerciales de recherche comme il en est question ici…
Par la suite, je serai amené à mettre en évidence de nombreuses lignes de codes ou réponses de console Python. Afin d'éviter d'alourdir démesurément ce rapport dans sa forme, je choisis d'intégrer ces bouts de codes directement au texte du rapport au moyen de captures d'écran. Ainsi, bien que ces captures soient des images, je ne les considère pas comme étant des figures et elles ne seront donc pas traitées comme telles. Quoi qu'il en soit, les codes produits durant le stage sont consultables en annexe dans leur intégralité.

Définitions

Cette partie contient une liste des définitions utiles données aux différents termes et concepts utilisés dans l'intégralité de ce rapport de stage. Elle contient des définitions posées spécialement par moi-même dans le cadre de ce stage ainsi que des explications plus générales sur certains concepts techniques utilisés qui peuvent ne pas être évidents à première vue.

[3] Institut d'Emission d'Outre Mer, « L'économie bleue dans l'Outre-mer », janvier 2018.
[4] Adriana Raveau, Christophe Rynikiewicz, et Béatrice Degaulejac, « Economie bleue en Martinique », janvier 2016,
https://martinique.deets.gouv.fr/sites/martinique.deets.gouv.fr/IMG/pdf/martinique_economie_bleue_rapport_final_janvier2016-2.pdf.
[5] Agence De l'Environnement et de la Maitrise de l'Energie, « Analyse de la desserte inter-iles en Guadeloupe », août 2010, http://www.guadeloupe.developpement-durable.gouv.fr/IMG/pdf/Analyse_de_la_desserte_intra_archipel_en_Guadeloupe.pdf.

AIS : (Automatique Identification System). C'est un système de communication automatisé de messages entre navires par ondes radios. Cela permet aux stations de contrôles CROSS (Centres Régionaux Opérationnel de Surveillance et de Sauvetage) pour la France ainsi qu'aux autres navires d'avoir des informations sur l'identité, le « statut », la position et même la route des navires en zone de navigation. Par extension, on appellera « données AIS » les données réelles collectées via ce système. La convention « Safety Of Life At Sea » internationale adoptée en 1914 a beaucoup évoluée et impose depuis 2007 à tous les navires de jauge brute supérieure à 300T et effectuant des voyages internationaux de s'équiper d'un système AIS en 2007.

Scraping : Ou « webscraping », désigne globalement les techniques utilisées afin d'extraire des informations de sites web de manière automatisée. Cela regroupe l'ensemble des scripts et programmes informatiques permettant de prélever ces informations. Ici on s'intéresse donc à un programme de webscraping en Python 3.6.

Les codes MMSI et IMO : Le numéro Maritime Mobile Service Identity (MMSI) permet d'enregistrer les équipements d'appel sélectif numérique (ASN), c'est-à-dire les équipements apparentés à celui présent dans le système de l'AIS qui permet d'entrer en communication entre deux postes de communication. Un système de sécurité permet de comparer les MMSI uniques des bateaux et vérifier que tout est normal avant d'entrer en communication. Ainsi, chaque navire devant être équipé d'un système AIS est donc identifié par un MMSI qui correspond à son système de communication. Ce numéro peut donc changer plusieurs fois pendant la période d'exploitation d'un navire en raison de changements d'exploitant, de détérioration ou de mise à niveau des appareils électroniques du navire. Le numéro IMO (Organisation maritime internationale) est un numéro unique permettant d'identifier les bateaux. Il est associé à la coque du bateau. Il ne change donc pas une fois qu'il a été attribué à un bateau. Depuis le 1er janvier 1996, tout navire de commerce de jauge brut supérieure à 100T doit s'en voir associer un lors de sa construction (plus exactement lors de la pose de la quille). Depuis 2009, les propriétaires (particuliers ou compagnies) de navires enregistrés doivent aussi se munir d'un numéro IMO. Ces deux codes permettent ainsi d'identifier les navires de manière unique.

Parser : Un parseur est un programme qui analyse la syntaxe d'un code. Ce genre de programmes permettent de formaliser le texte et d'en faire ressortir l'arbre syntaxique formé par l'ensemble des nœuds du code.

CSV : Le csv ou Coma-Separated Values est un format de données assez simple qui permet de découper les données en plusieurs champs grâce à des virgules. Concrètement on peut utiliser n'importe quel autre caractère pour séparer ces champs. Les données peuvent donc être affichées dans excel sous forme de tableau de données.

JSON : Le json ou JavaScript Objet Notation est un langage utilisé pour échanger des données textuelles. Il est de syntaxe simple et se structure en arborescence ce qui lui permet de fournir des données structurées. Ce format est notamment utilisé pour répondre à des requêtes GET sur internet.

Requête : Une requête correspond à l'interrogation d'une base de données sur internet. Il existe plusieurs langages de requêtes comme le SQL employé en base de données (que l'on utilise sous Access notamment). Cependant, le terme est étendu sur internet et désigne aussi les messages envoyés par le navigateur internet aux différents serveurs afin que ces derniers leurs envoie le contenu souhaité.

Dictionnaire Python : Le dictionnaire est un type de données de Python. Aussi connus sous le nom de « tableaux associatifs » dans d'autres langages, ils correspondent à une table qui n'est pas indexée par des nombres, mais par n'importe quel autre type de données. On appel ces index des clés. Le dictionnaire correspond à un ensemble de paires clé-donnée.

API : Application Programming Interface ou interface de programmation d'application. Sur internet, les API sont une forme d'applications qui permettent aux développeurs d'intégrer et de créer des logiciels qui rendent possible la communication entre différents services (là en l'occurrence, l'API de vesselfinder.com sert vraisemblablement à lier le site aux programmes qu'ils ont développés en parallèles afin de recueillir et retourner les données des stations AIS partenaires).

Heure de Départ AIS : On définit l'Heure de Départ AIS comme étant l'heure de départ collectée sur le site www.vesselfinder.com. Il s'agit donc concrètement de l'heure à laquelle un navire quitte la zone d'un port enregistré.

Heure de Départ Théorique : On définit l'heure de Départ Théorique comme étant l'heure de départ (du quai) d'un navire annoncée par la compagnie sur son site. Ces dernières sont donc collectées manuellement et stockées dans ma base de données.

Heure d'Arrivée AIS : On définit l'Heure d'Arrivée AIS comme étant l'heure d'arrivée collectée sur le site www.vesselfinder.com. Il s'agit donc concrètement de l'heure à laquelle un navire entre dans la zone d'un port enregistré.

Heure d'Arrivée Théorique : On définit l'heure d'Arrivée Théorique comme étant l'heure de départ (du quai) d'un navire annoncée par la compagnie sur son site.

Temps de manœuvre moyen : D'après les hypothèses posées plus haut, le temps de manœuvre moyen correspond à la différence moyenne entre les heures de Départ AIS et Théorique. Il est aussi égal à la différence entre les heures de d'Arrivée Théorique et AIS si le navire arrive à l'heure. Il est donc défini comme étant une caractéristique des ports.

Développement d'un premier outil de webscrapping

Après une revue via internet des différentes méthodes utilisées pour le webscrapping (voir définition plus loin), je décide d'utiliser le langage de programmation Python afin de réaliser un programme automatique de collecte des données. Je décide d'utiliser la dernière version de Python dans l'environnement « Spyder » sur Anaconda. Je souhaite dans un premier temps utiliser les bibliothèques BeautifulSoup et urlib principalement afin de collecter ces données.

Après étude des différents sites proposant des données issues de l'AIS, je choisis de m'intéresser plus particulièrement au site www.vesselfinder.com pour des raisons de structure et de couverture grâce à leurs nombreuses stations radio partenaires *Figure 4*.
Dès le début, en essayant d'ouvrir la page internet et de la stocker dans une variable Python, je reçois l'erreur suivante : `HTTPError: Forbidden`
En me renseignant ensuite, j'apprends que cette erreur est due à un système de sécurité du site qui empêche de se connecter via urllib de Python. Il convient donc ici de détourner cette sécurité en faisant croire que l'on utilise un navigateur tout ce qu'il y a de plus classique :

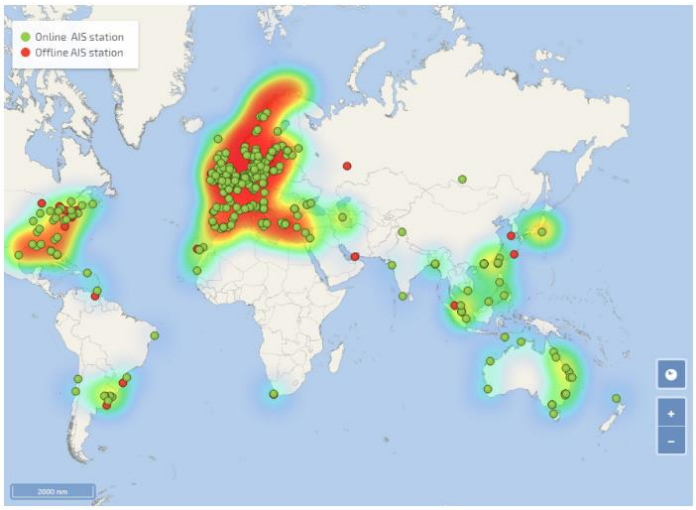


Figure 4: Couverture des Stations de réception AIS partenaires du site vesselfinder.com au 20/04/2021

```
class AppURLopener(urllib.request.FancyURLopener):
    version = "Mozilla/5.0"
#Donc on crée une instance de cet objet que l'on appelle "opener
opener =AppURLopener()
```

Après avoir réussi cette première action, je peux alors ouvrir la page et l'analyser avec le parser de la bibliothèque BeautifulSoup. Python affiche alors si on le lui demande la page internet analysée et enregistrée dans la variable « soup » :

```
#ici on note l'url de la page que l'on va étudier
urlpage = 'https://www.vesselfinder.com/fr/vessels/EUROPEAN-CAUSEWAY-IMO-9208394-MMSI-311027000'
#puis on ouvre cette page avec notre ouvreur
page = opener.open(urlpage)
#Ici on le parse avec btflSoup
soup = BeautifulSoup(page, 'html.parser')
```

Dans un premier temps, on choisit d'essayer de scraper la page suivante: « https://www.vesselfinder.com/fr/vessels/EUROPEAN-CAUSEWAY-IMO-9208394-MMSI-311027000 » qui correspond à la page donnant des informations sur le bateau « European Causeway » sur le site vesselfinder. Je rencontre alors un nouveau problème : les données des ports d'escale affichées sur le navigateur lorsque je visite la page ne sont pas détectées en cherchant leur balise html dans le document parsé par beautifulsoup (ici, la balise en question est déterminée grâce aux nombreux « outils de développement » de google chrome). Dans un premier temps, j'essaye donc de passer par d'autres sites utilisant les données AIS en espérant avoir de meilleurs succès. En visitant ces autres sites, il apparaît que ces deniers utilisent pour la grande majorité d'entre eux un script (en javascript) qui s'occupe d'afficher l'intégralité du contenu des pages. Cependant, BeautifulSoup ne permet pas d'interagir avec les pages et les scripts. J'envisage alors l'utilisation d'autres bibliothèques comme « Selenium » à ce moment.
Toutefois, en revenant sur le premier site, je décide alors d'analyser plus profondément la page web. Il en ressort finalement qu'un script de la page se charge de mettre en forme et d'afficher les données

obtenues grâce à une requête GET qui retourne les données dans un format json. Je choisis alors d'étudier directement cette requête pour récupérer les données du dernier départ du bateau.
Je convertis ces données grâce à la bibliothèque json déjà intégrée à Python en un format dictionnaire de Python :

```
soup_str = str(soup)
soup_dict = json.loads(soup_str)
```

Ensuite, je récupère les entrées du dictionnaire qui m'intéressent et je les transforme en chaînes de caractères afin de pouvoir les intégrer par la suite à un tableur :

```
#on récupère les entrées qui nous intéressent
        lieu_Départ=str(ligne['dp'])
        pays_Départ=str(ligne['c'])
        date_Arrivée = str(ligne['a'])
```

Je choisis d'écrire ces données dans un format .csv qui me permet de facilement découper les champs grâce à des virgules. Cela permettra par la suite de les utiliser dans un fichier Excel classique afin de les sélectionner, les classer et les utiliser à des fins statistiques.

```
#Partie remplissage du .csv
        with open('E:\COME\cours\stage 4a\Progammes\Donnees.csv', mode='a+', newline='') as csvfile:
            writer = csv.writer(csvfile, delimiter=';')
            writer.writerow([str(j),date_Arrivée,lieu_Départ,pays_Départ])
```

Finalement, maintenant que j'ai réussi à récupérer et entrer ces données dans un fichier CSV, il ne reste plus qu'à créer une boucle afin d'aller chercher les informations de tous les bateaux retenus. Je remarque que l'adresse de la requête utilisée contient le MMSI du bateau étudié et, après vérification, toutes les pages des autres bateaux possèdent une requête formée de la même façon. Je récupère donc grâce à une requête SQL dans ma base de données théorique l'intégralité des MMSI des bateaux que je souhaite étudier, je les intègre dans une liste sur Python et je boucle mon programme sur cette liste. Enfin, je boucle à nouveau le tout en rajoutant un temps d'attente d'une heure entre chaque prise de données (les temps de trajets étant tous supérieurs à une heure, je ne risque ainsi pas de rater un quelconque passage au port). Mon premier programme est donc prêt et récupère toute les heures les informations des bateaux étudiés. Toutefois, de par la requête qu'il exploite, ce programme ne récupère que les informations des arrivées en port. Je ne récolte pas les horaires de départ. On verra par la suite que ces informations devront elles aussi être récupérées. Il semblerait cependant que je ne possède alors pas les droits sur le site vesselfinder.com pour accéder à la requête supposée fournir les horaires de départ des navires. C'est-à-dire que je reçois une erreur 403 en essayant d'y accéder.

Développement d'un second outil de webscrapping

Contexte

A ce stade, une recherche documentaire est effectuée pour essayer de préciser le sens concret des horaires d'arrivée en port collectées grâce au programme de webscrapping (à quel moment de l'arrivée au port correspondent-t-elles ?). Je cherche tout d'abord dans le code de la marine marchande[6] si l'on ne pourrait pas trouver des informations utiles. Cependant le chapitre sur le transport de passagers ne mentionne nullement les retards tout comme celui sur les titres de navigation ou celui donnant les définitions utiles. J'en déduis que ce point n'est pas traité par le code de la marine marchande.
Je décide donc à ce moment de contacter les responsables du site www.vesselfinder.com afin de leur demander à quoi correspondent réellement ces heures d'arrivée. Alexander Tonev, responsable

[6] République Française, « Code disciplinaire et pénal de la marine marchande », s. d., https://www.legifrance.gouv.fr/codes/id/LEGITEXT000006071188/.

communication du site me répond ainsi le mardi 11 mai à mon questionnement : « The exact time at which *a vessel has docked can not be provided. The PortCalls method provides arrival or departure information but for a specific port, i.e. the movements of a vessel within the port area are not covered. Such detailed information can be obtained by analyzing the position records of a vessel, however, obtaining the data in real-time would be relatively expensive and would involve some additional processing and filtering. »*. D'après lui donc, les temps que je récupère déjà correspondent au départ ou à l'arrivée dans la zone géographique définie comme étant celle du port. Il me faudrait donc plus de données comme celles de la vitesse du navire pour déterminer une heure d'arrivée à quai précise. Il faut alors résoudre ce nouveau problème.

Un outil utilisant la bibliothèque Selenium pour Python

Comme vu au-dessus, j'ai un problème avec la qualité de mes données : les « heures d'arrivées » recueillies grâce au système AIS ne correspondent qu'à l'arrivée du bateau dans le port. On ne peut donc pas précisément donner des temps de retards grâce à ces données. Afin d'estimer un temps d'arrivée plus précis, une solution est proposée par M. Barbusse, ingénieur à l'AQST. Cette solution consiste à recueillir les temps de manœuvre au départ du port des navires. En posant les hypothèses suivantes on pourrait définir une méthode qui nous permettrait d'estimer un temps d'arrivée proche de la réalité :

_ On suppose d'abord que les temps de manœuvre au départ (l'Heure d'arrivée AIS moins l'Heure d'accostage) et à l'arrivée (l'Heure de Départ AIS moins l'Heure de départ du quai) sont égaux. On appellera alors cela « Temps de manœuvre moyen au port ».

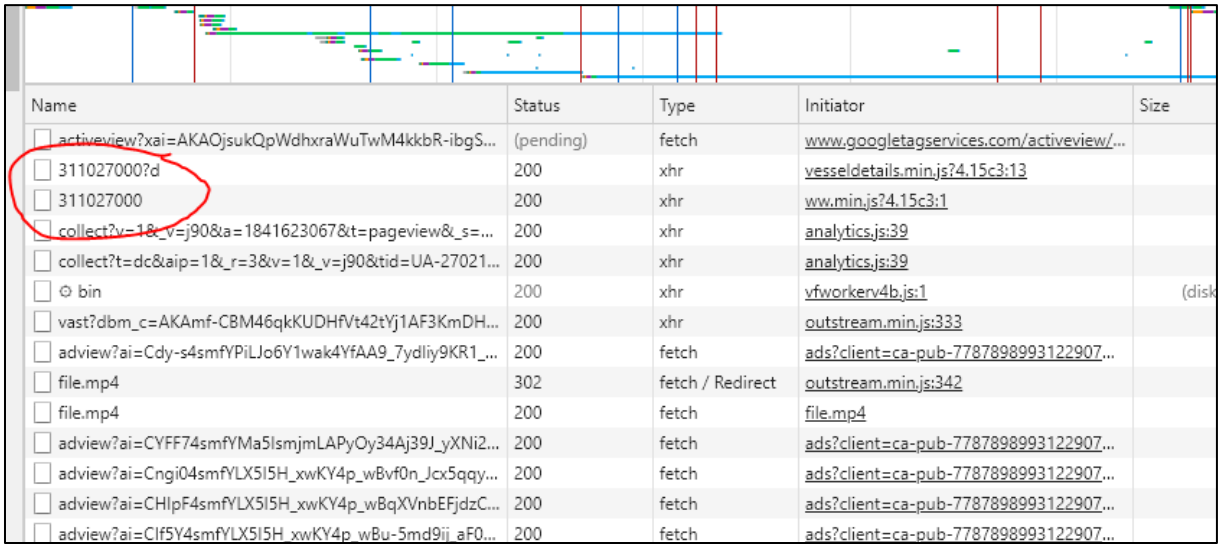
_ On suppose ensuite que les bateaux partent toujours à l'heure. Cela implique donc que l'on a la formule suivante : Temps de manœuvre au départ = Heure de Départ AIS-Heure de Départ théorique

Selon ces hypothèses, on pourrait donc calculer le Temps de manœuvre moyen en calculant simplement le Temps de départ. Par la suite, ces temps de manœuvre permettront de donner une estimation de l'heure d'accostage réelle des bateaux selon la formule suivante :

Heure d'arrivée réelle= Heure d'arrivée AIS + Temps de manœuvre moyen.

Je vais donc m'intéresser à un nouveau programme Python qui permettra de recueillir aussi les informations sur les heures de départ AIS. Cependant, la tâche n'est pas triviale : en effet, les requêtes recueillies précédemment au format json ne contenaient pas les données de départ des bateaux et aucune adresse « simple » à récupérer ne permet à première vue d'obtenir ces informations. Je vais, dans un premier temps, m'intéresser à la page internet :

« https://www.vesselfinder.com/fr/vessels/EUROPEAN-CAUSEWAY-IMO-9208394-MMSI-311027000 » qui correspond à la page du navire « EUROPEAN CAUSEWAY ». En inspectant la page, je peux récupérer une liste des requêtes exécutées par la page *Figure 3*.



Name	Status	Type	Initiator	Size
activeview?xai=AKAOjsukQpWdhxraWuTwM4kkbR-ibgS...	(pending)	fetch	www.googletagservices.com/activeview/...	
311027000?d	200	xhr	vesseldetails.min.js?4.15c3:13	
311027000	200	xhr	ww.min.js?4.15c3:1	
collect?v=1&_v=j90&a=1841623067&t=pageview&_s=...	200	xhr	analytics.js:39	
collect?t=dc&aip=1&_r=3&v=1&_v=j90&tid=UA-27021...	200	xhr	analytics.js:39	
bin	200	xhr	vfworkerv4b.js:1	(disk
vast?dbm_c=AKAmf-CBM46qkKUDHfVt42tYj1AF3KmDH...	200	xhr	outstream.min.js:333	
adview?ai=Cdy-s4smfYPiLJo6Y1wak4YfAA9_7ydliy9KR1_...	200	fetch	ads?client=ca-pub-7787898993122907...	
file.mp4	302	fetch / Redirect	outstream.min.js:342	
file.mp4	200	fetch	file.mp4	
adview?ai=CYFF74smfYMa5IsmjmLAPyOy34Aj39J_yXNi2...	200	fetch	ads?client=ca-pub-7787898993122907...	
adview?ai=Cngi04smfYLX5I5H_xwKY4p_wBvf0n_Jcx5qqy...	200	fetch	ads?client=ca-pub-7787898993122907...	
adview?ai=CHIpF4smfYLX5I5H_xwKY4p_wBqXVnbEFjdzC...	200	fetch	ads?client=ca-pub-7787898993122907...	
adview?ai=Clf5Y4smfYLX5I5H_xwKY4p_wBu-5md9ij_aF0...	200	fetch	ads?client=ca-pub-7787898993122907...	

Figure 5:Liste des requêtes effectuées à l'ouverture de la page étudiée

J'obtiens alors en particulier deux requêtes. Celle nommée « 311027000 » (numéro MMSI du navire) est celle utilisée dans mon premier script. En revanche celle de paramètre « d » (c'est une requête GET https://www.vesselfinder.com/api/pub/pcext/v4/311027000?d). Reçoit la réponse suivante :
« *[{"dp":"Belfast","c":"United Kingdom (UK)","l":"GBBEL001","s":1,"dy":"0","a":"May 12, 07:04","d":"-"},{"dp":"Larne","c":"United Kingdom (UK)","l":"GBLAR001","s":1,"dy":"17403","a":"May 12, 00:47","d":"May 12, 05:37"},{"dp":"Cairnryan","c":"United Kingdom (UK)","l":"GBCYN001","s":1,"dy":"3679","a":"May 11, 22:00","d":"May 11, 23:01"},{"dp":"Larne","c":"United Kingdom (UK)","l":"GBLAR001","s":1,"dy":"7930","a":"May 11, 16:48","d":"May 11, 19:01"},{"dp":"Cairnryan","c":"United Kingdom (UK)","l":"GBCYN001","s":1,"dy":"8042","a":"May 11, 12:46","d":"May 11, 15:00"}].* »
Concrètement il s'agit de toutes les informations dont j'ai besoin ; l'intégralité des données du tableau affiché sur la page de chaque navire de leur base de données *Figure 6*.

Figure 6: Tableau affichant les derniers passages au port d'un navire

Mais je reçois -comme il l'est brièvement mentionné au-dessus- une erreur « 403 forbidden » lorsque je tente d'accéder directement à l'url de la requête dans mon navigateur internet.
Bien qu'à première vue l'utilisation de ces données est condamnée, j'ai choisi d'approfondir cette opportunité afin de « scrapper » les données voulues. Cette requête fait appel à un code javascript qui se charge lui-même d'aller chercher les données dans ce qu'il semble être l'API du site vesselfinder.com. Le module BeautifulSoup utilisé précédemment permet seulement de récupérer des fichiers textes correspondant à la page html ou – dans mon cas – à du json. Il ne permet aucune interactivité avec le site, c'est-à-dire qu'on ne peut pas « naviguer » sur internet grâce à ce module ni lancer les requêtes automatiques du site qui lui permettent d'afficher les données (en effet, les données issues des requêtes sont ajoutées au html à posteriori après interrogation de l'API par des scripts javascript ; le html recueilli par le module BeautifulSoup ne contient donc pas les balises html contenant l'affichage des données). Il s'agit donc d'employer une autre méthode.
Après quelques recherches, je choisis d'utiliser le module Selenium de Python. Ce module permet en fait d'ouvrir un navigateur internet automatisé et de naviguer sur ce dernier. Je pense qu'une solution peut être construite en empruntant ce chemin. Après m'être brièvement familiarisé avec ce module, j'arrive dans un premier temps à ouvrir le site et, au moyen d'une méthode propre au driver chrome, à récupérer l'intégralité des requêtes émises lors de la navigation par le site (un peu comme avec les outils de développeur chrome finalement). En les épluchant une par une, je constate que la requête qui m'intéresse est effectivement capturée par selenium :

```
Caught https://www.vesselfinder.com/api/pub/pcext/v4/311027000?d
Mime: application/json
```

Il s'agit alors de récupérer l'identifiant de la requête pour sélectionner seulement cette dernière. Je remarque ici que le lancement des requêtes prend un temps non négligeable à l'ouverture, j'ajoute donc un temps d'attente avant de demander la récupération de ces dernières pour limiter les erreurs potentielles dues à un chargement trop long (problèmes de connexion par exemple). De plus, il semblerait que de manière un peu aléatoire la page ne se charge pas intégralement à l'ouverture. Une boucle while est donc créée afin de garantir que la page s'est intégralement chargée avant d'essayer

d'exploiter les données. Ces mesures seront particulièrement utiles étant donné que ma connexion n'est pas très stable…
Après avoir récupéré le précieux identifiant on s'aperçoit cependant qu'en répétant l'opération, l'identifiant change. Ce n'est donc pas une méthode fiable. On décide de simplement l'identifier grâce à son url trouvée précédemment. Ainsi, grâce à une condition j'arrive à isoler la réponse à la requête souhaitée et à recevoir ses paramètres. Pour obtenir le corps de la réponse, je tente d'abord l'instruction suivante :
content = driver.execute_cdp_cmd("Network.getResponseBody",{"requestId":request_id})

Elle me donne le résultat suivant :

```
In [12]: content
Out[12]:
{'base64Encoded': False,
 'body': '[{"dp":"Belfast","c":"United Kingdom
(UK)","l":"GBBEL001","s":1,"dy":"0","a":"May 12, 07:04","d":"-"},
{"dp":"Larne","c":"United Kingdom (UK)","l":"GBLAR001","s":
1,"dy":"17403","a":"May 12, 00:47","d":"May 12, 05:37"},
{"dp":"Cairnryan","c":"United Kingdom (UK)","l":"GBCYN001","s":
1,"dy":"3679","a":"May 11, 22:00","d":"May 11, 23:01"},
{"dp":"Larne","c":"United Kingdom (UK)","l":"GBLAR001","s":
1,"dy":"7930","a":"May 11, 16:48","d":"May 11, 19:01"},
{"dp":"Cairnryan","c":"United Kingdom (UK)","l":"GBCYN001","s":
1,"dy":"8042","a":"May 11, 12:46","d":"May 11, 15:00"}]'}
```

La méthode fonctionne. J'arrive ainsi à enregistrer dans une variable en chaîne de caractères (« str ») le corps de la requête. Je peux donc m'atteler ensuite à la tâche d'analyse/décomposition pour pouvoir l'exploiter. Grâce aux méthodes de conversion de json en dictionnaire et d'écriture dans le csv mentionnées précédemment, j'arrive à atteindre l'objectif de récupération de ces données. Ce nouveau programme aura donc été utilisé dès le lundi 17 mai et aura permis de récolter trois mois durant les données AIS des navires voulus.

Couplage des données

Une fois les données théoriques et les données réelles AIS collectées, il faut maintenant s'occuper du traitement de ces dernières. Toutefois, à ce moment précis, ces données ne sont encore pas « reliées » entre elles et ne permettent donc pas tout de suite leur exploitation. Un important travail de couplage des données entre elles est donc nécessaire à ce stade afin d'associer chaque entrée théorique (qui prend la forme d'un trajet théorique, c'est-à-dire comprenant les heures de départ au port A et d'arrivée au port B) à une entrée AIS (les entrées AIS prennent la forme d'un passage au port, c'est-à-dire que je dispose pour un port donné de l'heure d'arrivée d'un navire donné et de son heure de départ, éventuellement à une autre date). L'idée est que les données AIS, après avoir été triées et mises en forme, seront intégrées à une nouvelle table de ma base de données Access. Un nouveau champ sera alors créé qui pointera depuis les entrées AIS vers une entrée Théorique (relation 1, N). L'enjeu de cette partie est donc de créer cette colonne contenant ces clés étrangères. J'ai rapidement pu mettre en évidence qu'une manipulation de ce genre est bien trop complexe pour pouvoir être directement intégrée à Access. Un nouveau programme Python va donc être développé à cette fin.

Premier programme de couplage

Sur Python donc, je dois dans un premier temps récupérer les données du fichier csv produit par le programme de webscrapping. Grâce à l'instruction « readlines() », j'arrive à récupérer les lignes de

données. Je sélectionne pour se faire un échantillon de données de deux semaines correspondant aux deux premières semaines de données enregistrées via le second programme de webscrapping. Je les sépare ensuite en « cases » que j'introduis méthodiquement dans une seconde liste « Tableau » qui est en fait une liste contenant elle-même 16 listes (plus tard j'en ajouterai une 17ème) regroupant les données des 16 colonnes de mon fichier csv. Ainsi, je dois maintenant créer des conditions et réécrire dans le csv le champ de la clé étrangère dont j'ai besoin pour le couplage de la table de données AIS avec celle des trajets théoriques.

Ces conditions concernent: la date, le navire, les ports et les horaires. Je vais chercher des correspondances pour tous ces paramètres. Pour ce qui concerne les horaires, nous avons eu l'idée d'autoriser une certaine marge avant et après l'heure d'arrivée théorique pour prendre en compte les éventuelles arrivées en avance, ainsi que le délai entre l'arrivée dans le port (données de géolocalisation AIS) et l'arrivée au quai (donnée théorique). De plus, dans ce premier programme je n'utilise pas du tout les départs des ports. Je regarde donc seulement si le bon bateau arrive aux alentours de l'heure prévue par la compagnie au bon endroit.
Le premier problème rencontré survient lors du test des dates, en effet, je ne peux pas simplement comparer les chaînes de caractères des dates pour faire correspondre les dates d'arrivées étant donné la structure de ma base de données. Cette dernière, comme vu plus haut, utilise le champ occurrence. Je résous le problème en créant une liste d'occurrence à partir de la première ce qui est effectué au moyen de manipulations sur les formats des dates :

```
def Construction_Occurences (y,Tableau):
    for d in range(1,int(Tableau[13][y])):
        if int(Tableau[16][y][0:2])+7*(d-1)<10:
            dates_occurences.append('0'+str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:])
        else:
            dates_occurences.append(str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:]) #co
```

J'écris alors une première version des conditions de couplage assez intuitive.
Cependant ce premier code ne fonctionne pas et les commandes « print » utilisées pour donner les correspondances dans la console ne donnent aucun résultat positif. Je teste alors ces conditions une par une. De nombreuses correction d'indices et le développement de fonctions de contrôle du tout et d'affichage séquencé des actions effectuées par mon programme permet de résoudre ces premiers problèmes.
A ce stade, ce premier programme fonctionne et fournit dans la console les appareillages (sur une ligne j'affiche le numéro de la ligne de l'entrée AIS avec la clé de l'entrée théorique). Cependant, je remarque grâce à la console que le programme a tendance à appareiller une entrée AIS avec plusieurs entrées théoriques. Bien que le système des occurrences autorise qu'une entrée théorique soit associée à plusieurs entrées AIS différentes, l'inverse n'est concrètement pas possible. Un même bateau ne peut pas effectuer deux trajets différents en un seul. Il faut donc corriger ce problème avant de demander l'écriture finale du champ de la clé étrangère. Pour ce faire j'affiche dans un premier temps en plus des couplages, les heures des trajets correspondants pour voir si le problème vient de ma marge de deux heures (en effet, pour cette première version, j'utilise une marge entre l'arrivée théorique et l'arrivée AIS égale à deux heures choisie de façon arbitraire. Cela évoluera par la suite).

En effet, je constate deux écueils possibles avec ma méthode. Premièrement, ma condition ne conçoit pas comme couplés des horaires si ces derniers se situent entre deux jours (heure théorique et heure AIS autours de minuit). D'autre part, plusieurs correspondances sont possibles. Grâce au dernier affichage j'ai constaté des cas de ce type qui peuvent – je pense – être dus aux valeurs théoriques comportant un doute sur le bateau effectuant le trajet. En effet, de nombreuses compagnies ne précisent pas dans la rubrique dédiée aux horaires théoriques sur leur site le nom du bateau effectuant le trajet. Dans ces cas-là, j'ai ajouté toutes les liaisons effectuées à tous les bateaux susceptibles de les effectuer dans ma base de données. Ainsi, si un bateau A doit arriver à l'heure H et un bateau B à l'heure H+1 (pour une liaison courante principalement) les deux entrées théoriques peuvent être

associées toutes deux aux mêmes MMSI en raison du doute lors de la collecte sur le site de la compagnie. Le cas échéant, les deux entrées Théoriques seront donc couplées à la même entrée AIS.

En étudiant comment résoudre ce problème, je me rends compte aussi que la condition sur les jours pose de même un problème non négligeable étant donné qu'elle n'accepte pas les correspondances du jour précédent. Pour faire d'une pierre deux coups, je choisis donc de convertir les valeurs des heures et des jours en une seule donnée : un entier qui sera en fait un nombre de minutes initialisé au début du mois de Mai afin d'éviter les problèmes dus au mois de février. Je définis donc dans un premier temps une fonction qui prend en argument une date et une heure pour retourner un code sous la forme d'un entier :

```
def Code_Date_Heure(date,heure):
    x=(int(date[3:5])-5)*31*1440+int(date[0:2])*1440+int(heure[0:2])*60+int(heure[3:5])
    return(x)
```

Je réécris la condition des deux heures et je supprime celle sur la date. Cette solution fonctionne.
Pour ce qui est du problème des couplages « doubles » je créée une liste qui regroupera les différentes correspondances possibles pour une entrée AIS donnée. Je pose alors l'hypothèse que l'heure d'arrivée théorique la plus proche de l'heure d'arrivée AIS correspond au bon trajet. Je sélectionne cette valeur dans la liste grâce à un algorithme de tri classique. Cette solution reste donc cependant potentiellement inexacte – en imaginant un grand retard par exemple –. A cause de cette limite, on cherchera à minimiser le recours à cette condition de sélection du trajet par la suite.

Après ces corrections, le code fonctionne. J'ajoute alors de quoi écrire la colonne clé étrangère dans un fichier csv. Je teste ensuite le programme sur un échantillon d'une dizaine de correspondances dans ce csv. Je constate alors que les trajets ne correspondent pas ; il subsiste au moins une erreur dans le programme. Cependant la méthode de vérification est assez laborieuse et la collecte d'indices sur la raison de ces incohérences est complexe. J'aurais en effet dû créer par le passé un identifiant unique pour chaque trajet ce qui aurait simplifié la tâche (jusqu'alors j'utilisais simplement le numéro de ligne mais il s'avère que cela pose de gros problèmes pour résoudre les erreurs d'indices par exemple). Je décide donc de l'ajouter maintenant et de modifier le programme en conséquence pour pouvoir finalement résoudre ce dernier problème. Ainsi, grâce à ce nouveau témoin de vérification, j'arrive à corriger les quelques erreurs d'indexage qui causaient le problème. Je récupère donc finalement une colonne clé étrangère dans un fichier csv qui pourra par la suite être ajoutée à la base de données.

Etude des performances de ce programme

Je cherche à étudier dans cette partie les résultats obtenus grâce à l'échantillon pour essayer de déterminer si notre méthode nécessite d'être améliorée et –surtout- de quelle façon. Le premier test consiste simplement à donner un taux de couplage des données AIS avec les données Théoriques. Je décide d'effectuer ces tests de performances sur Python directement dans une nouvelle version du programme de couplage qui fournira directement dans la console les informations utiles. Je calcule ainsi quatre indicateurs différents permettant d'avoir une vision d'ensemble des performances de mon programme (et des programmes futurs) :

- Taux de Couplage Théorique : Il s'agit du nombre d'entrées théoriques ayant trouvé un couple AIS divisé par le nombre total d'entrées théoriques.

- Taux de Couplage AIS : Il s'agit du nombre d'entrées AIS ayant trouvé un couple Théorique divisé par le nombre total d'entrées AIS.
- Taux de Doublons : Je définis un « doublon » dans mes couplages comme étant une entrée AIS associée à un trajet théorique lui-même associé à plus de trajets AIS qu'il ne possède d'occurrences. C'est-à-dire qu'au moins une des entrées identifiées comme doublon est fausse. Par extension, le doute rend tous les doublons inutilisables. Le taux de Doublons est défini comme le nombre d'entrées AIS couplées et considérées comme doublons divisé par le nombre total d'entrées AIS couplées.
- Taux Dernière Condition : Le taux de Dernière Condition sert à quantifier sur la série étudiée l'utilisation de la fonction permettant d'éviter l'association d'une même entrée AIS à plusieurs entrées théoriques en retenant le trajet théorique dont l'horaire d'arrivée est la plus proche que l'horaire d'arrivée réelle selon le système AIS. Cependant, cette condition repose sur une hypothèse forte engendre donc une grande incertitude. Le taux de Dernière Condition est défini comme le nombre de fois où cette condition est utilisée divisé par le nombre total d'entrées AIS couplées.

Ainsi, les deux premiers indicateurs permettent de caractériser l'efficacité pure du programme et les deux derniers permettent de caractériser la « qualité » du couplage retourné.

Améliorations du programme de couplage

Je me penche à présent sur une meilleure solution que la marge arbitrairement choisie pour permettre le couplage entre trajet théorique et passage au port selon le système AIS, tout en tenant compte de la différence systématique dans les horaires d'arrivée théorique (au quai) et réelle (entrée dans le port) liée au temps dans le port.
Dans l'optique de limiter l'utilisation de la dernière condition (horaires théoriques et réels les plus proches après pré-sélection), la première idée qui me vient est de modifier la marge de couplage en la rendant dépendante de la durée du trajet. Je dois donc travailler avec le temps de trajet de nos données théoriques.
Pour faire cette première variante, je définis d'un côté une fonction qui calcule un temps de trajet puis je calcule une marge qui sera associée à chaque liste d'occurrences. Cette marge dépend donc d'un paramètre arbitraire définissant la proportion du temps de trajet conservée pour faire office de marge :

```
def Temps_De_Trajet(x):
    return(Code_Date_Heure(Tableau[16][x], Tableau[10][x])-Code_Date_Heure(Tableau[13][x], Tableau[9][x]))
def Construction_Marge(y):
    tps=Temps_De_Trajet(y)
    return(tps*Marge_Relative)
```

Après une brève étude des résultats, je constate que la méthode permet de grandement limiter l'utilisation de la dernière condition par rapport au précédent programme pour des valeurs du paramètre « Marge_Relative » inférieures à 0.8. Cependant le taux de couplage s'en voit extrêmement diminué. Concrètement, cette méthode fournit de bonnes performances de couplage pour les longs trajets mais perd de son intérêt pour les plus courts qui voient alors leur taux de couplage chuter drastiquement. Cette méthode ne produit par ailleurs aucun doublon sur l'échantillon étudié.

Suite à ces résultats, il me vient très vite une seconde idée pour pallier en partie aux problèmes de la précédente méthode. Cette seconde idée a pour objectif de conserver un taux d'utilisation de la dernière condition faible tout en remontant un peu le taux de couplage pour les trajets les plus courts. Suivant l'analyse que j'ai faite de la méthode précédente, l'idée de fixer un seuil minimum à la marge relative afin de ne pas « défavoriser » les trajets courts me vient à l'esprit.
Je modifie donc simplement la fonction « Construction_Marge » définie précédemment dans un nouveau fichier et je rajoute donc un second paramètre « seuil » :

```
def Construction_Marge(y):
    tps=Temps_De_Trajet(y)
    if tps*Marge_Relative<Seuil_Marge:
        return(Seuil_Marge)
    else:
        return(tps*Marge_Relative)
```

Ainsi, si la marge calculée de la même manière que pour la version précédente est plus petite que le paramètre « Seuil_Marge », la marge prendra la valeur du seuil, qui sera fixé de manière arbitraire suite à une étude empirique de ses variations. Sinon, on conservera la même méthode que pour le programme précédent.
Mon intuition paraît alors plutôt vérifiée. J'arrive, pour des valeurs du paramètre « Seuil_Marge » inférieures à 1 heure à des taux de couplage bien plus élevés (environ 20% de plus) que précédemment. Les performances quant à l'utilisation de la dernière condition sont quant à elles conservées avec des valeurs du paramètre « Seuil_Marge » ne dépassant pas l'heure. Ce programme semble donc être pour l'instant la meilleure méthode développée jusque-là.

Développement d'un programme inspiré des idées de M. Barbusse

Introduction

Suite à une discussion avec M. Barbusse, je m'intéresse à réaliser un programme correspondant à l'idée qu'il a eue pour le comparer aux miens. Pour cela M. Barbusse m'a fourni une fiche regroupant ses quelques idées directrices. Bien évidemment, cette fiche contenait des choses qui ne correspondaient pas à la réalité du code et j'ai donc dû adapter certains passages afin d'avoir un tout cohérent. Afin que le code soit un peu plus transparent et lisible j'ai essayé de l'organiser au maximum en divisant les tâches en de nombreuses fonctions chacune aillant un but aussi simple que possible. La première tâche concrète a été de créer un nouveau tableau de données en entrée, incluant uniquement les données qui me seront utiles par la suite (j'ai construit ce tableau sur la base de la fiche résumant l'idée). Je ne m'attarderai pas dans cette partie sur les difficultés rencontrées afin de me concentrer sur le fonctionnement du code.

Les fonctions héritées des autres algorithmes

Afin de gagner du temps et ayant déjà eu l'occasion de résoudre certains problèmes similaires, j'ai pu récupérer quelques fonctions développées précédemment après les avoir adaptées dans la forme (notamment en raison de l'utilisation du nouveau tableau de données) :

- La première de ces fonctions est celle nommée « Construction_Tableau_De_Données » qui est la fonction permettant d'extraire les informations entrées dans le fichier .csv pour en faire une grande liste imbriquée de dimension 2 utilisable par la suite. La correspondance des données avec leur index dans ce tableau est notée en commentaire au début du code afin de mieux s'y retrouver.
- On retrouve aussi la fonction « Construction_Occurences » qui, grâce au tableau de données et à un numéro de ligne théorique, retourne une liste de dates au format chaîne de caractères « str » correspondant aux n occurrences de l'entrée théorique.
- La fonction « Code_Date_Heure » elle, prend pour arguments une date ainsi qu'une heure et retourne un code numérique entier utilisable facilement à des fins calculatoires (en minutes).
- La fonction « Temps_De_Trajet » reçoit en argument un numéro de ligne théorique et permet de retourner le temps de trajet théorique associé à ce numéro dans le format de notre Code_Date_Heure.
- La fonction « Construction_Marge » reçoit comme argument un numéro de ligne théorique et permettra grâce aux paramètres Marge_Relative et Seuil_Marge de retourner une marge dépendant du trajet théorique identifié par la ligne. Cette marge est en minutes et est donc compatible avec l'utilisation de notre Code_Date_Heure.

Les nouvelles fonctions constituant le programme

Plusieurs nouvelles fonctions au but plus ou moins précis ont été définies afin de faire fonctionner ce nouveau programme.

- La fonction « Trajets_Du_Bateau » reçoit en entrée un numéro de ligne Théorique et va retourner une liste contenant les numéros de ligne AIS de tous les arrêts effectués par le bateau associé au trajet théorique (il s'agit de l'identification par le code MMSI ici) ;
- La fonction « Test_Départ » prend en argument une liste de numéros de lignes AIS ainsi qu'un numéro de ligne théorique et va retourner un booléen. Ce booléen sera vrai si dans cette liste AIS, on retrouve un départ correspondant au départ du trajet théorique. C'est-à-dire que l'on

recherche un départ du même port et séparé, au maximum de « Delta » minutes du départ théorique (ce « delta » correspond au temps de manœuvre dans le port). Je considère que le bateau ne peut pas être sorti du port avant l'heure théorique de sortie étant donné que la sortie du port prend un temps non nul. Limites de cette méthode : Cette méthode est en effet limitée par ma collecte de donnée non exhaustive. Si je n'ai pas récupéré l'horaire de départ du port du bateau parce que le programme de webscrapping n'était pas actif, je ne pourrai jamais coupler le trajet même si, à l'arrivée, il est identifiable. On perd donc avec cette fonction les arrivées AIS située en début de période de collecte ;

- La fonction « Correspondance_Arrivée » reçoit les mêmes arguments que la fonction « Test_Départ », à savoir une liste de numéros de lignes AIS et un numéro de ligne théorique. Pour chacune des lignes AIS de la liste, la fonction va tester si le port d'arrivée correspond à celui de la ligne théorique et si l'heure d'arrivée correspond aussi. J'emploie pour tester l'heure la méthode de marge relative avec seuil développée dans les codes précédents. Si ces conditions sont vérifiées, le numéro de ligne AIS est ajouté à une liste qui sera retournée par la fonction (il peut donc y en avoir plusieurs) ;
- La fonction « Selection » prend une liste imbriquée de dimension 2 en entrée. La fonction va identifier les éléments identiques dans cette liste. En considérant ensuite que les valeurs correspondent à un numéro de ligne AIS et que le positionnement de ces valeurs au sein de la première liste correspond à un numéro d'entrée théorique, elle va ensuite comparer ces éléments identiques et conserver uniquement celui présentant le plus faible écart entre son heure d'arrivée théorique et son heure d'arrivée AIS. La fonction retourne alors la liste ainsi épurée et un compteur nous informant sur le nombre de fois que l'opération de suppression d'un élément a eu lieu. Pour bien comprendre, cette fonction permet en fait d'empêcher qu'une entrée AIS ne soit associée à plusieurs entrées théoriques car cela est impossible ;
- La fonction « Compter_AIS » compte tout simplement le nombre d'entrées AIS et le retourne ;
- La fonction « Calcul_Performances » reçoit en entrée une liste de numéro de lignes de la taille du nombre d'entrées théoriques. Elle contient donc de nombreux champs vides. La fonction va ensuite retourner le nombre d'entrées AIS couplées à une entrée théorique, le nombre d'entrées théoriques couplées à au moins une entrées AIS ainsi que le nombre d'entrées théoriques. Ces chiffres seront utilisés par la suite pour calculer les performances de notre programme.

La fonction de couplage

La fonction « Couplage » est le cœur du programme. Elle permet à elle seule de retourner les résultats de couplage et regroupe la quasi-totalité des autres fonctions (sauf la création du Tableau qui est appelée séparément pour simplifier la programmation et les multiples tests effectués lors de la construction des fonctions). Elle ne prend pas de réels arguments si ce n'est le Tableau, mais ce dernier est en fait stocké dans une variable globale. La fonction couplage va donc parcourir toutes les lignes théoriques et leur faire subir les mêmes choses selon l'idée de la fiche. Tout d'abord on crée une liste des entrées AIS possibles avec la fonction « Trajets_Du_Bateau » ensuite, on vérifie que cette liste retourne un résultat positif au « Test_Départ » et, le cas échéant, on peut alors la faire passer par la fonction « Correspondance_Arrivée » afin de récupérer une liste des entrées AIS couplées aux entrées théoriques. Ces petites listes sont mises bout à bout pour créer une grande liste « Liste_finale » imbriquée de dimension 2 qui contient une petite liste pour chaque entrée théorique. On recense alors la présence de doublons. Puis, on fait alors passer cette liste par la fonction « Selection » qui s'occupera de la rendre cohérente. Enfin, on récupère les performances de cette liste cohérente grâce à la fonction « Calcul_Performances ». On retourne ensuite les chiffres permettant l'étude et la comparaison du programme.

Dernières améliorations de la fonction de couplage

Toutefois, le programme présente alors des performances bien inférieures à celles enregistrées par ceux développés précédemment. Après étude du programme et après avoir isolé différentes fonctions, je constate que ces forts écarts sont dus à la condition au départ contenue dans la fonction « Test_Départ ». Etant donné les bien trop faibles résultats obtenus, il est alors décidé de retirer cette dernière fonction. Le programme fourni alors exactement les mêmes performances que mon programme développé précédemment dit « à marge relative et seuil ». L'idée du nouveau programme de couplage, suggérée par M. Barbusse puis adaptée ne consistait en fait plus, après avoir ôté cette fonction, qu'en un parcours des données différent de celui que j'avais employé. Cette version est donc sélectionnée pour être utilisée par la suite sur les données réelles (rappelons ici que je travaillais alors sur un échantillon de données).

Une dernière modification est finalement ajoutée au programme à ce stade : l'ajout du paramètre « Delta » qui renvoi au temps de manœuvre du navire au port, entre le départ/arrivée au quai et l'entrée/sortie du port, qui est une caractéristique propre à chaque port. En prenant en compte les temps de manœuvre moyens que je vais calculer plus tard, j'espère donc obtenir de meilleures performances d'appariement (couplage) entre données théoriques et données réelles selon le système de géolocalisation AIS.

Exploitation des données appariées

Maintenant que je possède des données Théoriques et AIS ainsi que le moyen de les faire correspondre, il me reste alors à traiter toutes ces données afin d'en tirer des chiffres de ponctualité des liaisons maritimes de voyageurs. Il est d'abord nécessaire d'ajouter systématiquement à l'horaire d'arrivée effective AIS (entrée dans le port de destination) le délai de manœuvre au port à l'arrivée. Initialement, nous avons utilisé une valeur globale pour l'ensemble des ports issue d'une étude Suédoise[7]. Il est préférable d'estimer ce temps de manœuvre pour chaque port puisqu'il est propre à chaque port. Une fois le temps de manœuvre moyen par port déterminé, je pourrai estimer les heures d'arrivées réelles des bateaux à partir des heures d'arrivées AIS.

Détermination de temps réguliers de manœuvre

L'idée de cette partie est de créer une méthode efficace et applicable à tous les différents ports permettant de déterminer ce qui serait un temps de manœuvre régulier dans chaque port. Pour ce faire, je pose une hypothèse. Cette hypothèse est que les bateaux partent systématiquement à l'heure du quai. Cela implique alors que la différence entre l'heure de départ théorique et celle de départ AIS correspond à notre temps de manœuvre Delta. Nous faisons également l'hypothèse que le temps de manœuvre dans le port est identique au départ et à l'arrivée (symétrie). Grâce à cette hypothèse ainsi qu'à mes données, il s'agit alors de déterminer les valeurs de Delta (port) au moyen d'une étude statistique. Les couplages sont effectués grâce à une version modifiée du programme de couplage retenu.

Afin que l'estimation soit pertinente au sens statistique, elle doit reposer sur suffisamment de trajets auxquels sont associés autant de temps de manœuvre mesurés.

[7] Linda Styhre et Hulda Winnes, « Energy Efficient Port Calls », IVL Swedish Environmental Research Institute, n° C212 (septembre 2016): 49.

Un problème est alors vite identifié : les données couplées sont irrégulièrement réparties entre les différents ports. Je ne possède donc pas suffisamment de données pour étudier de manière précises l'intégralité des ports. Afin de comprendre l'origine de ce problème, je produis un graphe qui donne la fréquentation théorique des différents ports au départ de la zone transmanche *Figure 7*.

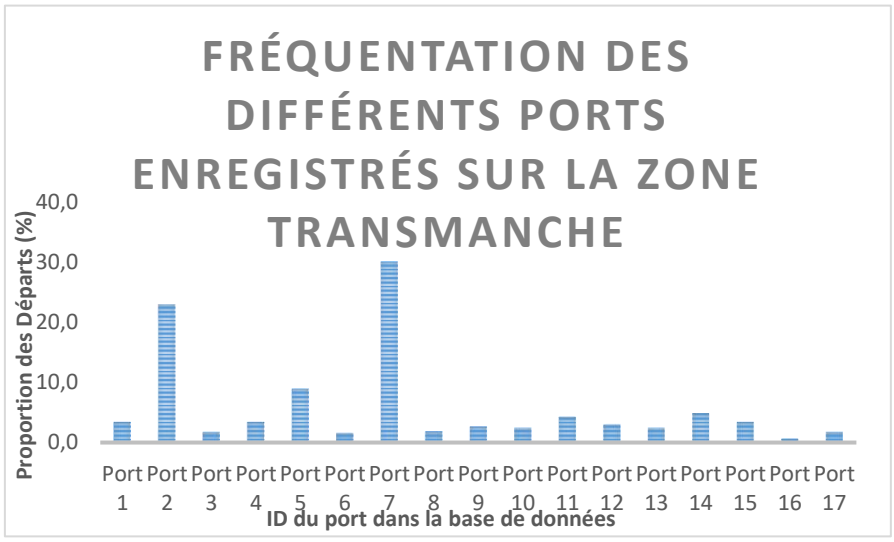


Figure 7: Fréquentation des différents ports sur la zone Transmanche

Je constate donc en comparant ce graphe aux données couplées que les chiffres sont proches de ceux obtenus avec les données couplées. Je ne pourrais donc pas étudier l'intégralité des ports de la zone.

J'étudie alors, pour les ports qui le permettent, la série des écarts entre heure de départ Théorique et heure de départ AIS – qui sont autant de temps de manœuvre mesurés - au moyen de boites à moustaches. Je supprime ainsi les valeurs aberrantes et je choisis la moyenne comme agrégateur de la série. De cette façon, j'obtiens des temps de manœuvre moyens qui seront ajoutés à la base de données dans la table « Port ». Ces temps de manœuvre moyens seront ajoutés aux heures d'arrivées réelles en port de l'AIS pour obtenir une estimation des heures d'arrivées réelles au quai.

Détermination de paramètres optimaux pour le couplage

Avant d'utiliser le programme de couplage développé précédemment, il faut choisir au mieux les paramètres qu'il utilise. Le seuil a été déterminé bien précédemment avec l'étude de mes premiers programmes. Le paramètre « Marge_relative » doit être judicieusement choisi.,Le paramètre « Delta » lpeut maintenant être fixé pour chaque port à l'aide de l'estimation du temps de manœuvre par port. Je fais donc varier ces deux principaux paramètres afin de déterminer une valeur optimale du paramètre « Marge_relative » qui puisse fournir de bons résultats quelle que soit la valeur du paramètre « Delta » du port concerné *Figure 8* (il est observé que le paramètre Delta varie globalement entre une et quarante-cinq minutes).

Delta\Marge_Relative	0,4	0,41	0,42	0,43	0,44	0,45	0,46	0,47	0,48	0,49	0,5	0,51	0,52	0,53
15	14,54	14,66	14,78	15,02	15,14	15,37	15,37	15,37	15,49	15,61	15,85	16,09	16,21	16,33
20	13,71	13,71	13,71	13,83	14,06	14,06	14,06	14,18	14,3	14,42	14,42	14,54	14,54	14,78
25	13,71	13,71	13,71	13,95	13,94	14,06	14,06	14,3	14,3	14,3	14,3	14,42	14,42	14,54
30	14,42	14,54	14,66	14,78	14,78	14,78	14,9	15,02	15,02	15,02	15,14	15,26	15,26	15,26
35	15,14	15,14	15,14	15,14	15,37	15,37	15,37	15,49	15,61	15,73	15,73	15,73	15,73	15,73
40	16,1	16,1	16,1	16,21	16,21	16,21	16,21	16,45	16,45	16,45	16,69	16,69	16,69	16,81
45	18,24	18,24	18,24	18,24	18,24	18,24	18,24	18,35	18,35	18,47	18,59	18,71	18,71	18,83
50	21,45	21,45	21,57	21,57	21,57	21,57	21,57	21,57	21,57	21,57	21,69	21,69	21,81	22,05
55	25,15	25,15	25,27	25,27	25,27	25,39	25,39	25,39	25,39	25,39	25,39	25,39	25,39	25,51
60	26,7	26,82	26,82	26,94	26,94	27,06	27,06	27,06	27,17	27,17	27,17	27,17	27,17	27,17

0%dernière condition
<1% dernière condition
<5%dernière condition
>5%dernière condition

Figure 8: Taux de couplage Théorique en fonction de Marge_relative et de Delta

Devant ces résultats, je choisis de prendre la valeur permettant le meilleur taux de couplage tout en conservant un taux nul d'utilisation de la dernière condition, soit ici de « Marge_relative »=0,53. Cette

valeur sera utilisée pour l'ensemble des régions étudiées, à défaut de données suffisantes pour déterminer une valeur optimale pour chaque zone

.

Production des résultats

Après avoir effectué les deux étapes évoquées précédemment, ai-je dispose de tous les outils en main pour produire des chiffres concernant les retards des transports maritimes réguliers de voyageurs. Tout d'abord je supprime les nombreuses entrées AIS en double obtenus lors de la collecte par webscrapping. Ces dernières sont ajoutées à une nouvelle table de la base de données. Afin d'éviter des erreurs ou « fausses manipulations », je travaille alors dans une copie de cette base de données finale. Les entrées Théoriques et AIS sont alors récupérées et intégrées à un fichier CSV mis en forme spécialement pour mon programme de couplage. Le programme de couplage peut donc produire la colonne des clés étrangères pointant vers les entrées Théoriques depuis les entrées AIS. Cette colonne est donc transformée en un nouveau champ de ma base de données. Je récupère alors via une requête les données AIS et Théoriques couplées entre elles et je les transfère sur excel afin de produire des illustrations et de calculer les temps de retards. J'obtiens les temps de retards via la formule suivante :
Temps_de_retard = Heure_Arrivée_Théorique – Heure_Arrivée_AIS - Temps_de_manoeuvre_moyen
Ainsi, un temps de retard nul ou négatif correspond à un navire à l'heure. Un temps positif signifie un retard. Au vu des premiers résultats de répartition de ces temps de retard, un premier seuil de retard de quinze minutes est sélectionné avec M. Barbusse. Toutefois, une première interrogation est levée à ce stade quant à la précision de mes résultats au vu de la répartition des temps de retards sur la région Transmanche. En effet, M. Barbusse s'attendait à une répartition proche d'une courbe d'allure log-normale. Les résultats obtenus n'ont pas tout à fait l'allure espérée et les « longs retards » semblent bien trop nombreux sur la région *Figure9*.

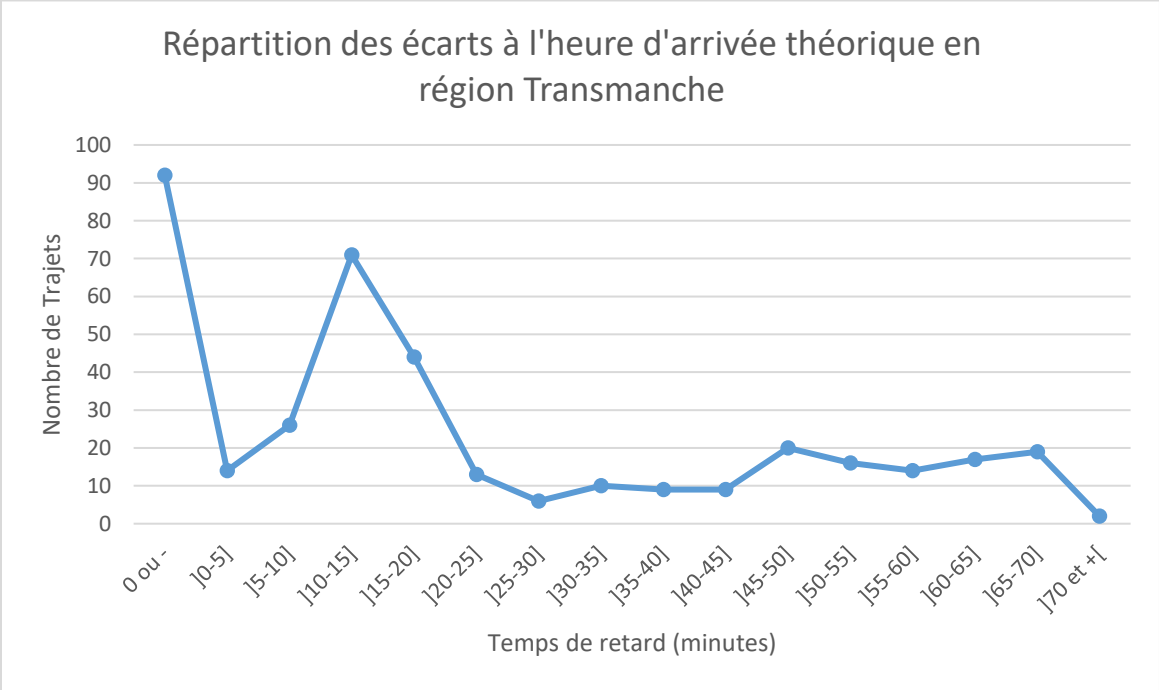


Figure 9:Répartition des temps de retard en Zone Transmanche

Suite à ça, j'essaye donc de produire divers graphes pour étudier la région. Toutefois, un second problème se pose en essayant de comparer les taux de retards des différentes compagnies opérant sur la région : 97% des données couplées concernent une seule et même compagnie. Je ne peux donc pas produire de comparaison légitime. Afin de vérifier l'origine du problème je compare le nombre d'entrées des différents types (AIS, Théorique et couplées) par compagnie *Figure 10*. Je constate ici un problème avec les quantités de données AIS recueillies ; bien trop faibles souvent par rapport au nombre de données théoriques enregistrées. De plus, je m'aperçois que la quantité de données couplées n'est pas directement proportionnelle à la quantité de données AIS. Ces irrégularités ne peuvent, à l'heure actuelle plus être résolues. Les chiffres que je m'apprête alors à produire paraissent déjà très imprécis.

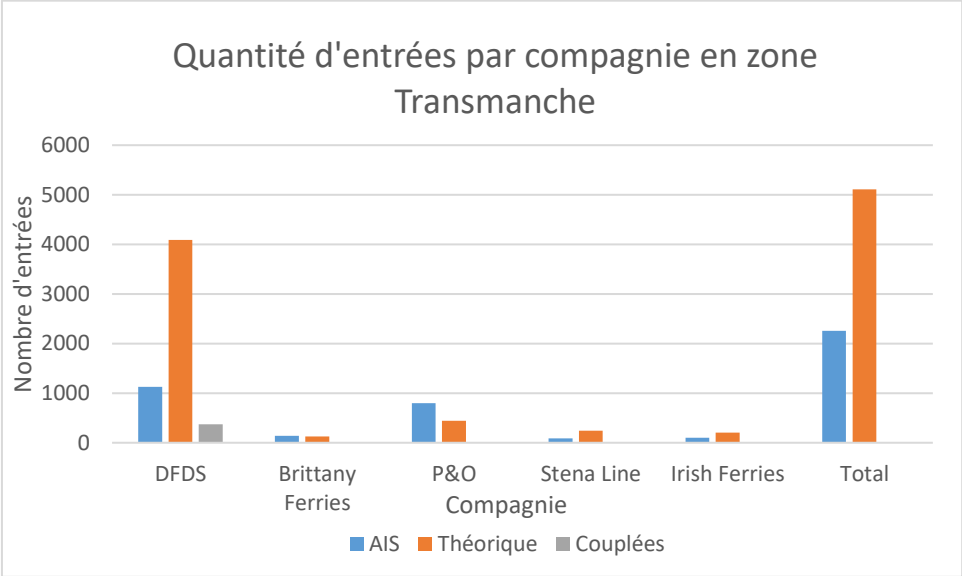


Figure 10: Quantité d'entrées par compagnie pour chaque type de données enregistré

Finalement, j'arrive à produire un comparatif des taux de retards en région transmanche selon les différentes Origine/Destination. Bien que je ne possède que deux Origine/Destination différentes, ces chiffres suffisent à poser de lourdes questions sur la validité de mes données. En effet, un taux de retard de plus de 80% est constaté pour les trajets Douvres/Calais, ce qui semble assez irréaliste à première vue *Figure 11*.

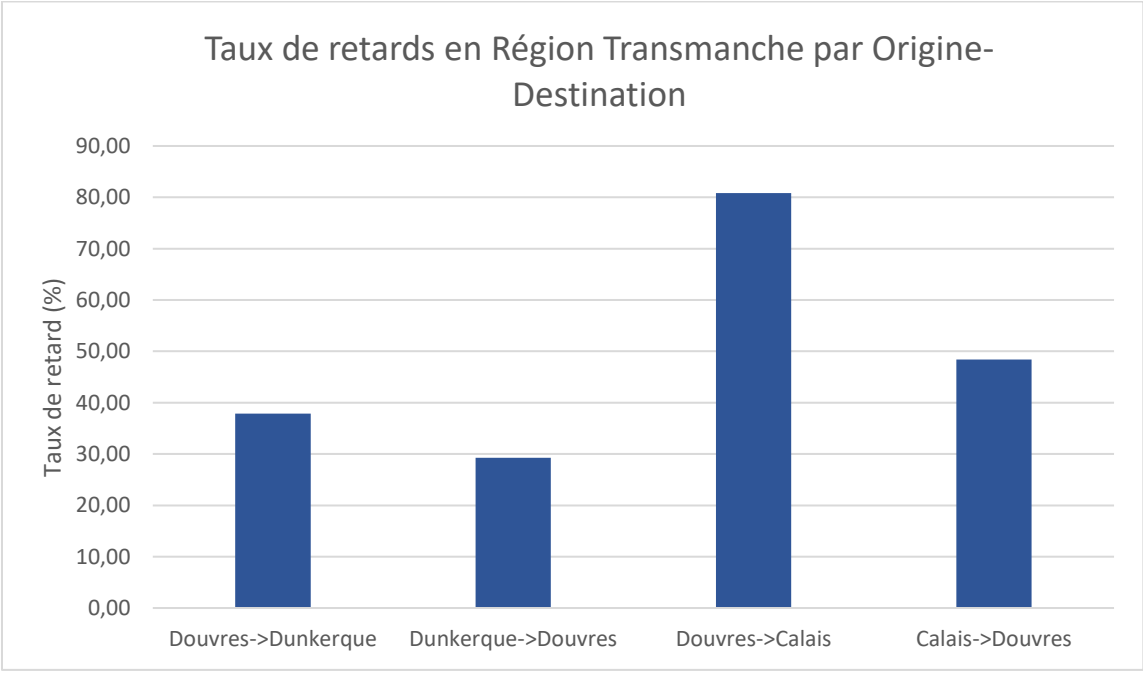


Figure 11: Comparaison des taux de retards par O/D en zone Transmanche

La production de chiffres concernant la Corse, elle, est carrément rendue impossible par le trop faible nombre de couples obtenus. Je choisis donc de ne pas sortir de chiffres sur la région.
On s'aperçoit donc globalement dans cette partie que la production de statistiques fiables est rendue presque impossible de par la quantité de données couplées récupérées, et les incertitudes relatives à la qualité des estimations ayant pu être produites avec les données à disposition. Cette partie est donc un échec et ces chiffres ne seront pas publiés en l'état, étant donné les niveaux d'incertitudes qu'ils présentent.

Retour sur expérience

Ne connaissant pas du tout ce domaine en débutant le stage, j'ai été amené à développer des méthodes en grande autonomie sur finalement peu de bases concrètes. Etant donné ma volonté de fournir des « résultats » dès le début de ce stage, je n'ai pris que peu voire pas de hauteur par rapport à mon travail au commencement. Je suis ainsi très vite parti dans des considérations techniques avec les différents programmes évoqués plus haut. Après coup, je m'aperçois donc, au vu des résultats finaux peu probants, que certains écueils auraient pu être évitables ou – du moins- envisagés à l'avance pour ceux qui ne dépendaient pas de moi directement. Je vais maintenant développer dans cette partie des solutions ou alternatives qui pourraient par la suite permettre d'obtenir de meilleurs résultats que ce que j'ai produit.

Un stage qui ouvre des possibilités

Suite à cette étude je constate que les données recueillies sont très souvent insuffisantes et ne permettent pas nécessairement de tirer des chiffres fiables. Plusieurs raisons peuvent être à l'origine de ce problème. Tout d'abord, il est important de noter que tout ceci s'étant déroulée en période de crise sanitaire, de nombreuses perturbations dans l'offre théorique de transport ont sûrement interféré avec ma collecte de données, d'autant plus que cette dernière a débuté durant le troisième confinement. Ensuite, toujours en raison de cette crise sanitaire, je n'ai pas pu accéder aux locaux de l'AQST. Cela pose problème pour la collecte régulière de données. En effet, cette dernière étant alors effectuée depuis mon ordinateur (fixe) personnel, je n'ai pas pu collecter 24h/24h et 7j/7j les informations du site étudié. De plus, mes horaires de collecte étaient nécessairement irréguliers selon ma disponibilité ainsi que les aléas de ma connexion à internet. D'autre part, les données fournies par le site www.vesselfinder.com sont récupérées par une méthode inconnue qui laisse donc de nombreuses questions en suspens, notamment quant aux temps de manœuvre que l'on a essayé de caractériser précédemment. Enfin, la collecte de données théoriques a imposé l'étude de courtes périodes d'un mois tout au plus limitant donc nécessairement la quantité de données récoltées ainsi que leur exhaustivité pour décrire la qualité de service dans les transports maritimes en général. Je vais donc, dans la suite de cette partie, tenter d'apporter des pistes de solutions qui pourraient permettre, dans le futur, de produire de meilleurs chiffres sur les transports maritimes de passagers.

Améliorer la collecte des données AIS

Tout d'abord, il semble que l'utilisation du système AIS pour la collecte de nos données soit la meilleure solution envisageable étant donné l'impossibilité de réclamer ce genre d'information auprès des armateurs français évoquée par M. Alain Sauvant au cours de ce stage. Deux options s'ouvrent alors : la première serait de conserver une méthode de webscrapping comme celle que j'ai développée et, la seconde, serait de créer une nouvelle méthode de collecte de données AIS.

Si le webscrapping devait être à nouveau utilisé, plusieurs points devraient être améliorés. Tout d'abord, l'utilisation d'un ordinateur dédié à la tâche et fonctionnant tout le temps serait optimal. Ce dernier pourrait éventuellement être contrôlé à distance et permettrait ainsi de garantir que la totalité des données du site sur la période étudiée soient récupérées.
Ensuite, vient la question de la fiabilité des données du site. En effet, j'ai pu constater notamment dans la zone Corse une certaine imprécision dans le couplage des données grâce aux indicateurs d'erreurs développés plus haut qui met en doute leur validité. On pourrait alors éventuellement étudier sur le terrain, auprès des responsables de ports par exemple. On pourrait alors ainsi mieux comprendre les écarts à la réalité enregistrés par le site afin d'établir de manière bien plus fiable des temps de manœuvre moyens.
Une autre solution, nécessitant de plus amples moyens, serait de prendre contact directement avec des fournisseurs de données AIS, comme les Centre régional opérationnel de surveillance et de sauvetage (CROSS) afin de leur demander la transmission directe de leurs données. Il s'agirait alors de créer un programme traitant ces données brutes (de positionnement des bateaux donc) permettant d'enregistrer les arrivées et départs des bateaux qui nous intéressent. En quelque sorte, il faudrait refaire le travail qui a été fait par les développeurs des sites comme vesselfinder.com, mais cette fois-ci pour un usage bien précis. Ainsi, de très grandes quantités d'informations pourraient être récupérées grâce à un serveur. Un tel système contribuerait alors à l'évaluation régulière des retards dans les transports maritimes de passagers en France.

Collecte simplifiée des horaires théoriques

La collecte des données théoriques « à la main » est très laborieuse sur de grandes périodes. Il s'agirait donc de trouver un moyen de la rendre plus simple afin de collecter rapidement de grandes quantités de données théoriques. Premièrement, on pourrait imaginer demander aux compagnies de transport maritime de voyageurs les tables de leurs horaires sous un format conventionné et permettant un traitement efficace et systématique de ces horaires sous Access ou Excel
Enfin, on pourrait autrement imaginer un système de webscrapping qui se chargerait d'aller chercher sur les sites les informations nécessaires et de les rentrer directement dans un fichier excel. Toutefois, cette méthode ne serait intéressante que pour de grands volumes de données pour chaque compagnie, étant donné que chaque site différent nécessitera un programme de webscrapping partiellement modifié. Cette dernière solution, est toutefois a priori moins efficace que la première, si tant est que cette dernière soit réalisable en pratique.

Conclusion

Pour conclure, il apparait que la mise en place d'un suivi de la qualité de service des transports maritimes de passagers est plus compliquée que prévu. Ce stage m'aura cependant permis d'avoir un premier contact avec le monde du travail et le domaine des transports tout particulièrement. Pour ce qui est des savoir-faire, il est clair que j'ai pu développer grandement mes connaissances du langage Python. En effet, le domaine très particulier du scraping m'était alors tout à fait inconnu et, bien que je possédais déjà un certain bagage en terme de code, j'ai pu ici mettre à l'épreuve mes compétences et mes capacités d'auto-apprentissage en parfaite autonomie. Plus qu'avec de nouvelles connaissances techniques, je sors de ce stage avec une toute nouvelle confiance en mes capacités d'apprentissage. En parallèle, j'ai parfois lors de ce stage commis certaines erreurs en suivant ce que me disais mon intuition, mais j'ai pu les corriger ensuite grâce à l'aide de M. Barbusse tout particulièrement.

Pour en revenir au contenu concret du stage, il apparait que la mission de développement d'une première « maquette » permettant de produire des chiffres sur le transport maritime réguliers de voyageurs a été menée jusqu'au bout, avec toutefois, malheureusement, peu de possibilités d'exploitation des résultats. En revanche, je n'ai nullement pu m'intéresser aux aspects plus « subjectifs » de la qualité de service dans les transports, concernant l'information au voyageur par exemple ou les garanties qu'ils peuvent avoir auprès des compagnies. La mission de production de quelques chiffres concernant la ponctualité des liaisons maritimes en zone Transmanche par exemple n'a, elle non plus, pas pu être amenée à son terme. Je pense donc qu'il serait à l'avenir intéressant de développer de plus amples moyens de collecte et de traitement de données de ponctualité des transports maritimes de voyageurs sur de plus longues périodes (non soumises aux aléas de la crise sanitaire si possible) afin de produire des chiffres représentatifs de la réalité. Finalement, bien que mitigée dans ses résultats, cette étude aura permis de débroussailler clairement le terrain –je l'espère – pour une étude future.

Bibliographie

Adriana Raveau, Christophe Rynikiewicz, et Béatrice Degaulejac. « Economie bleue en Martinique », janvier 2016. https://martinique.deets.gouv.fr/sites/martinique.deets.gouv.fr/IMG/pdf/martinique_economie_bleue_rapport_final_janvier2016-2.pdf.

Agence De l'Environnement et de la Maitrise de l'Energie. « Analyse de la desserte inter-iles en Guadeloupe », août 2010. http://www.guadeloupe.developpement-durable.gouv.fr/IMG/pdf/Analyse_de_la_desserte_intra_archipel_en_Guadeloupe.pdf.

Comité Européen de Normalisation. « Norme NF EN 13816 ». AFNOR, septembre 2002.

Institut d'Emission d'Outre Mer. « L'économie bleue dans l'Outre-mer », janvier 2018.

Linda Styhre et Hulda Winnes. « Energy Efficient Port Calls », IVL Swedish Environmental Research Institute, n° C212 (septembre 2016): 49.

République Française. « Code disciplinaire et pénal de la marine marchande », s. d. https://www.legifrance.gouv.fr/codes/id/LEGITEXT000006071188/.

« Site officiel de l'AQST », s. d. http://www.qualitetransports.gouv.fr/.

Annexes

Programme de Webscrapping utilisant BeautifulSoup

```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
#Première version du programme de webscrapping, utilisant la bibliothèque BeautifulSoup et ne permettant pas de récupérer les départs.
# import libraries
from bs4 import BeautifulSoup
import urllib.request
import urllib.error
import csv
import json
import time
import random

#Création d'un objet nous permettant d'ouvrir les liens en se faisant passer pour un navigateur internet
#On utilisera donc toujours par la suite cet "ouvreur d'url"
class AppURLopener(urllib.request.FancyURLopener):
    version = "Mozilla/5.0"

#On crée une instance de cet objet que l'on appelle "opener
opener =AppURLopener()

#Liste des MMSI utiles provenant de notre BD
MMSI=[311027000,228006800,228233600,227022800,228085000,235009590,235010500,235028825,247297100,228392900,227023100,209505000,209512000,228244700,209479000,209490000,209878000,209146000,227273000,247160400]

#Grande boucle for qui va donc analyser tour à tour toutes les requêtes des différents bateaux qui nous intéressent
for i in range (14):
    for j in MMSI:

#ici on note l'url de la page que l'on va étudier, concrètement, l'adresse est toujours la même, seul le mmsi change. J'utilise donc une concaténation pour créer l'adresse
        urlpage = 'https://www.vesselfinder.com/api/pub/pcext/v4/'+str(j)
        page = opener.open(urlpage)

#Ici je le parse avec btflSoup mais vu que c'est du json pas besoin, j'ai gardé la méthode mais j'aurais pu couper quelques lignes ou utiliser une autre méthode
        soup = BeautifulSoup(page, 'html.parser')
        soup_str = str(soup)
        soup_dict = json.loads(soup_str)

#on sépare les lignes du json
```

```
        ligne = soup_dict[0]

#on récupère les entrées qui nous intéressent
        lieu_Départ=str(ligne['dp'])
        pays_Départ=str(ligne['c'])
        date_Arrivée = str(ligne['a'])
        print(str(j),date_Arrivée,lieu_Départ,pays_Départ)



#là on l'insère dans notre excel
#Partie remplissage du .csv
        with open('E:\COME\cours\stage 4a\Progammes\Donnees.csv', mode='a+', newline='') as csvfile:
#Emplacement du fichier csv pour l'écriture
            writer = csv.writer(csvfile, delimiter=';')
            writer.writerow([str(j),date_Arrivée,lieu_Départ,pays_Départ])
#Ensuite, on ajoute un temps d'attente pour la boucle
        time.sleep(random.random()*10)
    time.sleep(3600)
```

Programme de Webscrapping utilisant Selenium

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 14 14:31:17 2021

@author: COME
"""

#Programme de Webscrapping utilisant la bibliothèque Selenium. Ici les données rentrées sont celles de la zone Corse

#Importations
from time import sleep
from selenium import webdriver
from selenium.webdriver import DesiredCapabilities
import json
import base64
import csv
import time

# Paramètrage de notre navigateur Chrome
capabilities = DesiredCapabilities.CHROME
capabilities["goog:loggingPrefs"] = {"performance": "ALL"}  # capabilities pour notre driver chrome

#On va chercher l'exécutable de notre driver chrome spécialement installé pour Selenium
driver = webdriver.Chrome(
   desired_capabilities=capabilities,
executable_path="E:\COME\cours\stage_4a\Logiciels\driverchromewebscraping\chromedriver.exe"
)



#Liste des MMSI utiles provenant de notre BD. Facilement récupérable par simple requête
MMSI=[226269000,226280000,227182000,227202000,228009700,228081000,228358800,228393700,247013400,247036100,247079000,247089500,247178700,247183200,247228600,247356500,247392000,247552000,247580000,227184000,247015400,247132400,247664000,247458000]

#J'ai laissé ici aussi les listes de MMSI et d'adresses du Transmanche, en changeant le nom des listes le programme doit être fonctionnel pour les deux.
#TranManche:
#MMSI=[311027000,228006800,228233600,227022800,228085000,235009590,235010500,235028825,247297100,228392900,227023100,209505000,209512000,228244700,209479000,209490000,209878000,209146000,227273000,247160400]
#TransManche:


#Contrairement à la requête employée par l'autre progrmme de webscrapping, les adresses des bateaux étudiés diffèrent trop. Je suis donc obligé de les entrer dans une liste une par une
#ATTENTION! IL FAUT ICI QUE LES ADRESSES SOIENT DANS LE MÊME ORDRE QUE LES MMSI DE LA LISTE MMSI!!!
```

```
Adresses_TransManche=["https://www.vesselfinder.com/vessels/EUROPEAN-CAUSEWAY-IMO-9208394-MMSI-311027000",
"https://www.vesselfinder.com/vessels/CALAIS-SEAWAYS-IMO-8908466-MMSI-228006800",
"https://www.vesselfinder.com/vessels/COTE-DALBATRE-IMO-9320128-MMSI-228233600",
"https://www.vesselfinder.com/vessels/COTE-DES-DUNES-IMO-9232527-MMSI-227022800",
"https://www.vesselfinder.com/vessels/COTE-DES-FLANDRES-IMO-9305843-MMSI-228085000",
"https://www.vesselfinder.com/vessels/DELFT-SEAWAYS-IMO-9293088-MMSI-235009590",
"https://www.vesselfinder.com/vessels/DOVER-SEAWAYS-IMO-9318345-MMSI-235010500",
"https://www.vesselfinder.com/vessels/DUNKERQUE-SEAWAYS-IMO-9293076-MMSI-235028825",
"https://www.vesselfinder.com/vessels/EPSILON-IMO-9539054-MMSI-247297100",
"https://www.vesselfinder.com/vessels/GALICIA-IMO-9856189-MMSI-228392900",
"https://www.vesselfinder.com/vessels/MONT-SAINT-MICHEL-IMO-9238337-MMSI-227023100",
"https://www.vesselfinder.com/vessels/PRIDE-OF-CANTERBURY-IMO-9007295-MMSI-209505000",
"https://www.vesselfinder.com/vessels/PRIDE-OF-KENT-IMO-9015266-MMSI-209512000",
"https://www.vesselfinder.com/vessels/SEVEN-SISTERS-IMO-9320130-MMSI-228244700",
"https://www.vesselfinder.com/vessels/SPIRIT-OF-BRITAIN-IMO-9524231-MMSI-209479000",
"https://www.vesselfinder.com/vessels/SPIRIT-OF-FRANCE-IMO-9533816-MMSI-209490000",
"https://www.vesselfinder.com/vessels/STENA-ESTRID-IMO-9807293-MMSI-209878000",
"https://www.vesselfinder.com/vessels/W.B.-YEATS-IMO-9809679-MMSI-209146000",
"https://www.vesselfinder.com/vessels/NORMANDIE-IMO-9006253-MMSI-227273000",
"https://www.vesselfinder.com/vessels/STENA-HORIZON-IMO-9332559-MMSI-247160400"]
#Corse:
Adresses=['https://www.vesselfinder.com/vessels/A-NEPITA-IMO-9211511-MMSI-226269000',
'https://www.vesselfinder.com/vessels/JEAN-NICOLI-IMO-9161948-MMSI-226280000',
'https://www.vesselfinder.com/vessels/MONTE-DORO-IMO-8911516-MMSI-227182000',
'https://www.vesselfinder.com/vessels/KALLISTE-IMO-9050618-MMSI-227202000',
'https://www.vesselfinder.com/vessels/PIANA-IMO-9526332-MMSI-228009700',
'https://www.vesselfinder.com/vessels/PASCAL-PAOLI-IMO-9247510-MMSI-228081000',
'https://www.vesselfinder.com/vessels/VIZZAVONA-IMO-9138006-MMSI-228358800',
'https://www.vesselfinder.com/vessels/PELAGOS-IMO-9136034-MMSI-228393700',
'https://www.vesselfinder.com/vessels/MEGA-EXPRESS-IMO-9203174-MMSI-247013400',
'https://www.vesselfinder.com/vessels/MEGA-EXPRESS-TWO-IMO-9203186-MMSI-247036100',
'https://www.vesselfinder.com/vessels/SARDINIA-REGINA-IMO-7205910-MMSI-247079000',
'https://www.vesselfinder.com/vessels/MEGA-EXPRESS-THREE-IMO-9208083-MMSI-247089500',
'https://www.vesselfinder.com/vessels/MEGA-EXPRESS-FOUR-IMO-9086590-MMSI-247178700',
'https://www.vesselfinder.com/vessels/MEGA-EXPRESS-FIVE-IMO-9035101-MMSI-247183200',
'https://www.vesselfinder.com/vessels/MEGA-SMERALDA-IMO-8306486-MMSI-247228600',
'https://www.vesselfinder.com/vessels/MEGA-ANDREA-IMO-8306498-MMSI-247356500',
'https://www.vesselfinder.com/vessels/SARDINIA-VERA-IMO-7360617-MMSI-247392000',
'https://www.vesselfinder.com/vessels/CORSICA-MARINA-SECON-IMO-7349039-MMSI-247552000',
'https://www.vesselfinder.com/vessels/CORSICA-VICTORIA-IMO-7305253-MMSI-247580000',
'https://www.vesselfinder.com/vessels/PAGLIA-ORBA-IMO-9050826-MMSI-227184000',
'https://www.vesselfinder.com/vessels/MOBY-WONDER-IMO-9214367-MMSI-247015400',
'https://www.vesselfinder.com/vessels/MOBY-AKI-IMO-9299393-MMSI-247132400',
'https://www.vesselfinder.com/vessels/MOBY-VINCENT-IMO-7360605-MMSI-247664000',
'https://www.vesselfinder.com/vessels/GIRAGLIA-IMO-8013235-MMSI-247458000']


# On définit ici un filtre à appliquer plus tard pour la sélection des requêtes: celles recevant une réponse
def log_filter(log):
    return (
```

```
  # Il s'agit d'une réponse
    log["method"] == "Network.responseReceived"
  )


#Création d'une grande boucle qui permettra au programme de se répèter toutes les heures
for i in range (24):

  for j in range(len(MMSI)):  #seconde boucle qui correspond donc à toutes les pages visitées; j prend la valeur de nos codes MMSI tour à tour

    #Cette boucle while contenant une for va permettre le bon chargement de la page ainsi que le stockage de la réponse dans la variable "content"
    a = 0
    erreur= 0
    while a==0 and erreur<3:    #ici on fait une boucle qui vérifie que les données se sont bien chargées, sinon on recharge la page jusqu'à les avoir, au maximum 3 fois
      driver.get(str(Adresses[j]))    # On va à l'adresse qui nous intéresse.
      sleep(8)  # On attend que la page ait le temps de se charger.

      #On extrait toutes les requêtes ici
      logs_raw = driver.get_log("performance")
      logs = [json.loads(lr["message"])["message"] for lr in logs_raw]

      for log in filter(log_filter, logs):     #On récupères ici l'identification des requêtes dans des listes, afin de pouvoir sélectionner la notre par la suite
        resp_url = log["params"]["response"]["url"]
        request_id = log["params"]["requestId"]
        mime = log["params"]["response"]["mimeType"]
        rep = log["params"]["response"]
        if resp_url == "https://www.vesselfinder.com/api/pub/pcext/v4/"+str(MMSI[j])+"?d":  #Quand la bonne requête est reconnue, on la récupère

          content = driver.execute_cdp_cmd("Network.getResponseBody",{"requestId":request_id})
          print(f"Adresse {resp_url}")
          a=1 #On anonce dans "a" que la requête est bien trouvée, j'aurais pu utiliser un booléen.
      if a==0:
        erreur+=1
        print('erreur, requête non trouvée')
        #si il n'a pas trouvé notre fameuse requête, il s'agit ici de rafraîchir la page et d'attendre avant de retenter une capture des requêtes


    #On s'attaque ici au traitement et à l'écriture de nos données dans un fichier excel, on utilise ici les mêmes méthodes à peu de choses près que pour le précédent script
    #On récupère d'abord la deuxièm ligne qui est en fait la première ligne complète et qui correspond à l'intégralité de la dernière escale (départ+arrivée)
    if content["body"]!='[]' and erreur!=3:
      contenu=json.loads(content["body"]) #récupération d'une partie du contenu du dictionnaire que l'on convertit en un autre dictionnaire finalement
```

```
        Port = contenu[1]["dp"]     #On récupère dans les prochaines lignes les entrées voulues de
notre dictionnaire "contenu" que l'on stocke dans des variables plus appropriées
        Pays = contenu[1]["c"]
        Arrivée = contenu[1]["a"]
        Départ = contenu[1]["d"]
        print(str(MMSI[j]),Arrivée,Départ,Port,Pays)

        #écriture dans un fichier csv selon la même méthode que le premier script
        with open('E:\COME\cours\stage_4a\Progammes\Donnees_Corse.csv', mode='a+', newline='')
as csvfile:  #Emplacement du fichier csv pour l'écriture
            writer = csv.writer(csvfile, delimiter=';')
            writer.writerow([str(MMSI[j]),Arrivée,Départ,Port,Pays])
        #driver.quit() on ne trouve pas ici de solution pour fermer la fenêtre entre les prises de
données horaires. Le navigateur restera donc ouvert
      else:
        print('Päs de données pour ce navire')
  print("Fin session")    #Facilite la lisibilité de la console, cela correspond à la fin de la capture
horaire des données
  time.sleep(3600)    #temps d'attente entre chaque prise de données: ici une heure
```

Programme de couplage dit « à marge fixe »

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 23 13:47:11 2021

@author: COME
"""
#Importations

import csv
import datetime

#Ici on note les numéros d'index de nos différentes colonnes excel dans la grande liste imbriquée "Tableau" déclarée par la suite
#N°AIS 0
#MMSI_AIS 0 1
#Date_Arrivé_AIS 1 2
#Heure_Arrivée_AIS 2 3
#Date_Départ_AIS 3 4
#Heure_Départ_AIS 4 5
#ID_Port_AIS 5 6
#ID_Pays_AIS 6 7
#N_Théorique 7 8
#Heure_Départ_Théorique 8 9
#Heure_Arrivée_Théorique 9 10
#Jour 10 11
#ID_Port_Destination 11 12
#Date_Départ_Théorique 12 13
#Occurence 13 14
#MMSI_Théorique 14 15
#ID_Trajet_AIS 15 0
#Date_Arrivée_Théorique 16

#Initialisation du Tableau des données
Tableau=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
lignes=[]
Lignes_Sans_La_Première=[]
Delta=30
Marge=50  #On définit ici la marge avec l'horaire théorique dans laquelle on souhaite valider le trajet
Liste_finale=[] #initialisation de la liste qui sera ensuite écrite dans un csv

#INTRODUCTION ICI DES VARIABLES STATISTIQUES
Compteur_Couplage=0 #ici on compte les couples pour le taux de couplage
nbr_entrées_AIS=0
doublons=0
Nbr_Dernière_Condition=0

#Dans cette fonction, on crée une liste de dates comprenant les x dates théoriques dûes aux occurences, on trouve dedans des conversions int/str à répétition
def Construction_Occurences (y,Tableau):
```

```
   for d in range(1,int(Tableau[14][y])):    #Ligne bonne testée et approuvée par le comité
      if int(Tableau[16][y][0:2])+7*(d-1)<10:     #ligne validée par le conseil de guerre
         dates_occurences.append('0'+str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:])
#corrigée par mes soins
      else:
         dates_occurences.append(str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:]) #corrigée
aux petits oignons



#On définit une fonction qui construit le Tableau contenant toutes nos données excel grâce à deux
boucles ainsi que la commande readlines.
def Construction_Tableau_De_Données():
   with open('E:\COME\cours\stage_4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Données_Brutes.csv', newline='') as
fichier:  #Chemin des données brutes à lire
      global lignes
      global Lignes_Sans_La_Première
      lignes = fichier.readlines()
      Lignes_Sans_La_Première=lignes[1:]
      for row in Lignes_Sans_La_Première:
         b=row.split(';')    #on sépare la ligne en une liste contenant toutes les cases de cette ligne
         for a in range (16):
            Tableau[a].append(b[a])
    #là pour la dernière colonne on enlève les caractères "\r\n" indiquant le changement de ligne
         Tableau[16].append(b[16][0:10])



#Ici fonction qui écrit ce tableau dans un autre excel pour voir s'il a bien été construit, simple
fonction de vérification
def Verification_Du_Tableau_De_Données():
   #écriture dans un fichier csv selon la même méthode que le premier script
   with open('E:\COME\cours\stage_4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Données_Brutes.csv', mode='a+',
newline='') as csvtest:   #Chemin des données brutes à lire
         writer = csv.writer(csvtest, delimiter=';')
         for k in range(len(Tableau[16])):

writer.writerow([Tableau[0],Tableau[1],Tableau[2],Tableau[3],Tableau[4],Tableau[5],Tableau[6],Tabl
eau[7],Tableau[8],Tableau[9],Tableau[10],Tableau[11],Tableau[12],Tableau[13],Tableau[14],Tableau[
15],Tableau[16]])

#On définit ici une fonction transformant nos dates et heures en codes numériques basés sur cette
année
def Code_Date_Heure(date,heure):
   x=(int(date[3:5])-5)*31*1440+int(date[0:2])*1440+int(heure[0:2])*60+int(heure[3:5])
   return(x)

#ON DEFINIT ICI UNE FONCTION AFFICHANT NOS STATS
def Affichage_Stats(L):
   print ("Taux de couplage AIS : "+str(Compteur_Couplage/nbr_entrées_AIS))
```

```
   print("Taux de couplage Théorique : "+str(Calcul_Performances(L)[0]/Calcul_Performances(L)[1]))
   print("nbr de doublons="+str(doublons)+" proportion de doublons :
"+str(doublons/Compteur_Couplage))
   print ("Taux utilisation dernière condition : "+str(Nbr_Dernière_Condition/Compteur_Couplage))


#Fonction de Calcul des performance importé du couplage "Inversé" afin de standardisé les
programme et réalisé une comparaison optimale
def Calcul_Performances(Liste_Clé_Etrangère):
   Couplage_Théorique=0

   #On calcule ensuite le nombre de lignes théoriques couplées
   Intermé=[]
   for i in Liste_Clé_Etrangère:
      while i in Intermé:
         Intermé.remove(i)
      Intermé.append(i)
   Couplage_Théorique=len(Intermé)

   #On calcule le nombre d'entrées théoriques réelles (étant donné qu'un même trajet peut avoir
plusieurs lignes en raison de l'incertitude sur le bateau)
   Intermédiaire=[]
   for j in Tableau[8]:
      while j in Intermédiaire:
         Intermédiaire.remove(j)
      Intermédiaire.append(j)
   Nombre_Entrées_Théoriques=len(Intermédiaire)
   return(Couplage_Théorique,Nombre_Entrées_Théoriques)


#Recherche de couplage, on essaye donc de créer un nouveau champ "clé étrangère" que l'on pourra
ajouter à notre base de données dans la table des données AIS

Construction_Tableau_De_Données()   #on construit les données
for x in range(len(Lignes_Sans_La_Première)-1):
   #On reset ici la liste des correspondances possibles
   Possibilités=0
   # ON RESET POUR NOS STATS LE TEST DERNIERE
   Test_Dernière=0

   #ON COMPTE LES ENTREES AIS POUR NOS STATS
   if Tableau[1][x]!="":
      #ON COMPTE CES ENTREES POUR NOS STATS
      nbr_entrées_AIS+=1
      for y in range(len(Lignes_Sans_La_Première)-1):
         if Tableau[1][x]==Tableau[15][y]:

            if Tableau[6][x]==Tableau[12][y]:   #On construit à chaque fois la liste des des occurences
puis on la reset
               dates_occurences=[]
               Construction_Occurences (y,Tableau)
```

```
   #ensuite on test toutes ces dates grâce à une enième boucle for (oui c'est un peu complexe à
comprendre à la lecture du code)
                for date in dates_occurences:
                   if (Code_Date_Heure(Tableau[2][x], Tableau[3][x])+Delta)>(Code_Date_Heure(date,
Tableau[10][y])-15) and (Code_Date_Heure(Tableau[2][x],
Tableau[3][x])+Delta)<(Code_Date_Heure(date, Tableau[10][y])+Marge):
                      if Possibilités==0:
                         Possibilités=int(Tableau[8][y])
                      elif abs(Code_Date_Heure(Tableau[2][x], Tableau[3][x])-Code_Date_Heure(date,
Tableau[10][y])) < abs(Code_Date_Heure(Tableau[2][x], Tableau[3][x])-
Code_Date_Heure(Tableau[16][Possibilités], Tableau[10][Possibilités])) :
                         Possibilités=int(Tableau[8][y])
                         Test_Dernière=1

                      #print("AIS n°"+Tableau[0][x]+'H AIS:'+Tableau[3][x]+'se couple
avec'+Tableau[8][y]+'H théo:'+Tableau[10][y])

   Liste_finale.append(Possibilités)
   Nbr_Dernière_Condition+=Test_Dernière
   if Possibilités!=0:
      Compteur_Couplage+=1

#CREATION D'UNE BOUCLE PERMETTANT DE COMPTER LES "DOUBLONS"

for x in range(len(Liste_finale)-1):
   if Liste_finale[x]!=0 and Liste_finale.count(Liste_finale[x])>int(Tableau[14][x]):
      doublons+=1


#Affichage des stats
Affichage_Stats(Liste_finale)




#LA PARTIE ECRITURE EST DESACTIVEE DANS CETTE VERSION DEDIEE A ETUDIER LA PRECISION ET
L'EFFICACITE DU PROGRAMME
#ON CONSIDERES DONC QUE LA "Liste_finale" EST NOTRE PRODUIT FINI: NOTRE COLONNE CLE
ETRANGERE
#IL SUFFIT D'ENLEVER LES "#" DEVANT LES LIGNES SUIVANTES POUR LA REACTIVER
#écriture dans un nouveau fichier csv (on copiera ensuite la colonne mais je préfère ne pas
endommager le premier au cas où)
#for j in Liste_finale:
   #with open('E:\COME\cours\stage 4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Clés_Etrangères.csv', mode='a+',
newline='') as csvfile: #ICI CHEMIN DU FICHIER CSV POUR L'ECRITURE DE LA COLONNE DE CLES
ETRANGERES
      #writer = csv.writer(csvfile, delimiter=';')
      #writer.writerow([str(j)])
```

Programme de couplage dit « à marge relative »

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 23 13:47:11 2021

@author: COME
"""
#Importations

import csv
import datetime

#Ici on note les numéros d'index de nos différentes colonnes excel dans la grande liste imbriquée "Tableau" déclarée par la suite
#N°AIS 0
#MMSI_AIS  1
#Date_Arrivé_AIS  2
#Heure_Arrivée_AIS  3
#Date_Départ_AIS  4
#Heure_Départ_AIS  5
#ID_Port_AIS  6
#ID_Pays_AIS  7
#N_Théorique  8
#Heure_Départ_Théorique  9
#Heure_Arrivée_Théorique  10
#Jour  11
#ID_Port_Destination  12
#Date_Départ_Théorique  13
#Occurence  14
#MMSI_Théorique  15
#Date_Arrivée_Théorique 16

#Initialisation du Tableau des données
Tableau=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
lignes=[]
Lignes_Sans_La_Première=[]
Marge_Relative=0.2   #On définit ici la marge avec l'horaire théorique
Liste_finale=[] #initialisation de la liste qui sera ensuite écrite dans un csv
Delta=30

#INTRODUCTION ICI DES VARIABLES STATISTIQUES
Compteur_Couplage=0 #ici on compte les couples pour le taux de couplage
nbr_entrées_AIS=0   #on compte le nombre de trajets AIS
doublons=0
Nbr_Dernière_Condition=0 #On compte le nombre de trajets couplés ayant nécessité d'utiliser la dernière condition

#Dans cette fonction, on crée une liste de dates comprenant les x dates théoriques dûes aux occurences, on trouve dedans des conversions int/str à répétition
def Construction_Occurences (y,Tableau):
```

```
   a=[]
   for d in range(1,int(Tableau[14][y])):    #Ligne bonne testée et approuvée par le comité
      if int(Tableau[16][y][0:2])+7*(d-1)<10:     #ligne validée par le conseil de guerre
         a.append('0'+str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:]) #corrigée par mes soins
      else:
         a.append(str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:]) #corrigée aux petits
oignons
   return(a)


#On définit une fonction qui construit le Tableau contenant toutes nos données excel grâce à deux
boucles ainsi que la commande readlines.
def Construction_Tableau_De_Données():
   with open('E:\COME\cours\stage_4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Données_Brutes.csv', newline='') as
fichier:  #Chemin des données brutes à lire
      global lignes
      global Lignes_Sans_La_Première
      lignes = fichier.readlines()
      Lignes_Sans_La_Première=lignes[1:]
      for row in Lignes_Sans_La_Première:
         b=row.split(';')    #on sépare la ligne en une liste contenant toutes les cases de cette ligne
         for a in range (16):
            Tableau[a].append(b[a])
    #là pour la dernière colonne on enlève les caractères "\r\n" indiquant le changement de ligne
         Tableau[16].append(b[16][0:10])



#Ici fonction qui écrit ce tableau dans un autre excel pour voir s'il a bien été construit
def Verification_Du_Tableau_De_Données():
   #écriture dans un fichier csv selon la même méthode que le premier script
   with open('E:\COME\cours\stage_4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Données_Brutes.csv', mode='a+',
newline='') as csvtest:   #Chemin des données brutes à lire
         writer = csv.writer(csvtest, delimiter=';')
         for k in range(len(Tableau[16])):

writer.writerow([Tableau[0],Tableau[1],Tableau[2],Tableau[3],Tableau[4],Tableau[5],Tableau[6],Tabl
eau[7],Tableau[8],Tableau[9],Tableau[10],Tableau[11],Tableau[12],Tableau[13],Tableau[14],Tableau[
15],Tableau[16]])

#On définit ici une fonction transformant nos dates et heures en codes numériques basés sur cette
année
def Code_Date_Heure(date,heure):
   x=(int(date[3:5])-5)*31*1440+int(date[0:2])*1440+int(heure[0:2])*60+int(heure[3:5])
   return(x)

#fonction qui calcul le temps de trajet THEORIQUE à partir du numéro de la ligne conscernée.
def Temps_De_Trajet(x):
   return(Code_Date_Heure(Tableau[16][x], Tableau[10][x])-Code_Date_Heure(Tableau[13][x],
Tableau[9][x]))
```

```
#fonction qui retourne un entier (en minutes) correspondant à la marge relative
def Construction_Marge(y):
   tps=Temps_De_Trajet(y)
   return(tps*Marge_Relative)

#ON DEFINIT ICI UNE FONCTION AFFICHANT NOS STATS
def Affichage_Stats(L):
   print ("Taux de couplage AIS : "+str(Compteur_Couplage/nbr_entrées_AIS))
   print("Taux de couplage Théorique : "+str(Calcul_Performances(L)[0]/Calcul_Performances(L)[1]))
   print("nbr de doublons="+str(Calcul_Doublons(Liste_finale,Tableau))+" proportion de doublons :
"+str(Calcul_Doublons(Liste_finale,Tableau)/Compteur_Couplage))
   print ("Taux dernière condition : "+str(Nbr_Dernière_Condition/Compteur_Couplage))

#ON ECRIT ICI UNE FONCTION QUI VA TESTER LA PRESENCE DE DOUBLONS DANS LA LISTE FINALE
(celle qui sera écrite dans un csv)
def Calcul_Doublons(Liste,Tableau):
   doublons=0
   for x in range(len(Liste_finale)-1):
      if Liste_finale[x]!=0 and Liste_finale.count(Liste_finale[x])>int(Tableau[14][x]):
         doublons+=1
   return(doublons)


#Calcul des performance importé de Couplage_Barbusse
def Calcul_Performances(Liste_Clé_Etrangère):
   Couplage_Théorique=0
   #On calcule ensuite le nombre de lignes théoriques couplées
   Intermé=[]
   for i in Liste_Clé_Etrangère:
      while i in Intermé:
         Intermé.remove(i)
      Intermé.append(i)
   Couplage_Théorique=len(Intermé)
   #On calcule le nombre d'entrées théoriques réelles (étant donné qu'un même trajet peut avoir
plusieurs lignes en raison de l'incertitude sur le bateau)
   Intermédiaire=[]
   for j in Tableau[8]:
      while j in Intermédiaire:
         Intermédiaire.remove(j)
      Intermédiaire.append(j)
   Nombre_Entrées_Théoriques=len(Intermédiaire)
   return(Couplage_Théorique,Nombre_Entrées_Théoriques)



#Coeur du programme:le couplage en lui même
Construction_Tableau_De_Données()   #on construit le Tableau de données
#On parcourt avec x les données AIS
for x in range(len(Lignes_Sans_La_Première)-1):
   #Resets de variables
   Possibilités=0  #nombre d'entrées théoriques correspondantes
```

```
   Test_Dernière=0 #stats dernière condition
   #On vérifie que la ligne AIS n'est pas vide (ça fait sens en réalité)
   if Tableau[1][x]!="":
      nbr_entrées_AIS+=1  #On compte les entrées AIS
      #On parcourt avec y les trajets théoriques
      for y in range(len(Lignes_Sans_La_Première)-1):
         #On compare les MMSi
         if Tableau[1][x]==Tableau[15][y]:
            #On compare les ports AIS avec les ports d'arrivées théoriques
            if Tableau[6][x]==Tableau[12][y]:
               dates_occurences=Construction_Occurences (y,Tableau) #On construit à chaque fois la
liste des des occurences
               Marge=Construction_Marge(y) #On calcule la marge qui sera nécessairement la même
pour toutes les occurences (mdr le temps que j'ai mis à m'en rendre compte)
               #On parcourt les occurences récemment construites
               for date in dates_occurences:
                  #On test ici la compatibilité des heures grâce à la marge et notre fonction
Code_Date_Heure plutôt pratique
                  if (Code_Date_Heure(Tableau[2][x], Tableau[3][x])+Delta)>(Code_Date_Heure(date,
Tableau[10][y])-15) and (Code_Date_Heure(Tableau[2][x],
Tableau[3][x])+Delta)<(Code_Date_Heure(date, Tableau[10][y])+Marge):
                     #Si la ligne n'a pas encore eu de correspondance, on note celle là
                     if Possibilités==0:
                        Possibilités=int(Tableau[8][y])
                     #Sinon, on la compare à la correspondance actuellement validée
                     elif abs((Code_Date_Heure(Tableau[2][x], Tableau[3][x])+Delta)-
Code_Date_Heure(date, Tableau[10][y])) < abs((Code_Date_Heure(Tableau[2][x],
Tableau[3][x])+Delta)-Code_Date_Heure(Tableau[16][Possibilités], Tableau[10][Possibilités])) :
                        Possibilités=int(Tableau[8][y])
                        Test_Dernière=1

                     #print("AIS n°"+Tableau[0][x]+'H AIS:'+Tableau[3][x]+'se couple
avec'+Tableau[8][y]+'H théo:'+Tableau[10][y])

   Liste_finale.append(Possibilités)   #On construit finalement notre Liste_Finale avec les valeurs
successives des correspondances
   Nbr_Dernière_Condition+=Test_Dernière
   if Possibilités!=0:
      Compteur_Couplage+=1




#Affichage des stats
Affichage_Stats(Liste_finale)




#LA PARTIE ECRITURE EST DESACTIVEE DANS CETTE VERSION DEDIEE A ETUDIER LA PRECISION ET
L'EFFICACITE DU PROGRAMME
```

```
#ON CONSIDERES DONC QUE LA "Liste_finale" EST NOTRE PRODUIT FINI: NOTRE COLONNE CLE ETRANGERE
#IL SUFFIT D'ENLEVER LES "#" DEVANT LES LIGNES SUIVANTES POUR LA REACTIVER
#écriture dans un nouveau fichier csv (on copiera ensuite la colonne mais je préfère ne pas endommager le premier au cas où)
#for j in Liste_finale:
   #with open('E:\COME\cours\stage 4a\Test appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Clés_Etrangères.csv', mode='a+', newline='') as csvfile: #ICI CHEMIN DU FICHIER CSV POUR L'ECRITURE DE LA COLONNE DE CLES ETRANGERES
      #writer = csv.writer(csvfile, delimiter=';')
      #writer.writerow([str(j)])
```

Programme de couplage dit « à marge relative et seuil »

```
# -*- coding: utf-8 -*-
"""
Created on Sun May 23 13:47:11 2021

@author: COME
"""
#Importations

import csv
import datetime

#Ici on note les numéros d'index de nos différentes colonnes excel dans la grande liste imbriquée "Tableau" déclarée par la suite
#N°AIS 0
#MMSI_AIS  1
#Date_Arrivé_AIS  2
#Heure_Arrivée_AIS  3
#Date_Départ_AIS  4
#Heure_Départ_AIS  5
#ID_Port_AIS  6
#ID_Pays_AIS  7
#N_Théorique  8
#Heure_Départ_Théorique  9
#Heure_Arrivée_Théorique  10
#Jour  11
#ID_Port_Destination  12
#Date_Départ_Théorique  13
#Occurence  14
#MMSI_Théorique  15
#Date_Arrivée_Théorique 16

#Initialisation du Tableau des données
Tableau=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
lignes=[]
Lignes_Sans_La_Première=[]
Marge_Relative=0.4   #On définit ici la marge avec l'horaire théorique
Liste_finale=[] #initialisation de la liste qui sera ensuite écrite dans un csv
Seuil_Marge=60
Delta=30

#INTRODUCTION ICI DES VARIABLES STATISTIQUES
Compteur_Couplage=0 #ici on compte les couples pour le taux de couplage
nbr_entrées_AIS=0   #on compte le nombre de trajets AIS
doublons=0
Nbr_Dernière_Condition=0 #On compte le nombre de trajets couplés ayant nécessité d'utiliser la dernière condition

#Dans cette fonction, on crée une liste de dates comprenant les x dates théoriques dûes aux occurences, on trouve dedans des conversions int/str à répétition
```

```
def Construction_Occurences (y,Tableau):
   a=[]
   for d in range(1,int(Tableau[14][y])):    #Ligne bonne testée et approuvée par le comité
      if int(Tableau[16][y][0:2])+7*(d-1)<10:     #ligne validée par le conseil de guerre
         a.append('0'+str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:]) #corrigée par mes soins
      else:
         a.append(str(int(Tableau[16][y][0:2])+7*(d-1))+Tableau[16][y][2:]) #corrigée aux petits
oignons
   return(a)


#On définit une fonction qui construit le Tableau contenant toutes nos données excel grâce à deux
boucles ainsi que la commande readlines.
def Construction_Tableau_De_Données():
   with open('E:\COME\cours\stage_4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Données_Brutes.csv', newline='') as
fichier:  #Chemin des données brutes à lire
      global lignes
      global Lignes_Sans_La_Première
      lignes = fichier.readlines()
      Lignes_Sans_La_Première=lignes[1:]
      for row in Lignes_Sans_La_Première:
         b=row.split(';')    #on sépare la ligne en une liste contenant toutes les cases de cette ligne
         for a in range (16):
            Tableau[a].append(b[a])
    #là pour la dernière colonne on enlève les caractères "\r\n" indiquant le changement de ligne
         Tableau[16].append(b[16][0:10])



#Ici fonction qui écrit ce tableau dans un autre excel pour voir s'il a bien été construit
def Verification_Du_Tableau_De_Données():
   #écriture dans un fichier csv selon la même méthode que le premier script
   with open('E:\COME\cours\stage_4a\Test
appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Données_Brutes.csv', mode='a+',
newline='') as csvtest:   #Chemin des données brutes à lire
         writer = csv.writer(csvtest, delimiter=';')
         for k in range(len(Tableau[16])):

writer.writerow([Tableau[0],Tableau[1],Tableau[2],Tableau[3],Tableau[4],Tableau[5],Tableau[6],Tabl
eau[7],Tableau[8],Tableau[9],Tableau[10],Tableau[11],Tableau[12],Tableau[13],Tableau[14],Tableau[
15],Tableau[16]])

#On définit ici une fonction transformant nos dates et heures en codes numériques basés sur cette
année
def Code_Date_Heure(date,heure):
   x=(int(date[3:5])-5)*31*1440+int(date[0:2])*1440+int(heure[0:2])*60+int(heure[3:5])
   return(x)

#fonction qui calcul le temps de trajet THEORIQUE à partir du numéro de la ligne conscernée.
def Temps_De_Trajet(x):
```

```
    return(Code_Date_Heure(Tableau[16][x], Tableau[10][x])-Code_Date_Heure(Tableau[13][x],
Tableau[9][x]))

#fonction qui retourne un entier (en minutes) correspondant à la marge relative
def Construction_Marge(y):
    tps=Temps_De_Trajet(y)
    if tps*Marge_Relative<Seuil_Marge:
        return(Seuil_Marge)
    else:
        return(tps*Marge_Relative)

#ON DEFINIT ICI UNE FONCTION AFFICHANT NOS STATS
def Affichage_Stats(L):
    print ("Taux de couplage AIS : "+str(Compteur_Couplage/nbr_entrées_AIS))
    print("Taux de couplage Théorique : "+str(Calcul_Performances(L)[0]/Calcul_Performances(L)[1]))
    print("Nbr doublons : "+str(Calcul_Doublons(Liste_finale,Tableau))+" taux de doublons :
"+str(Calcul_Doublons(Liste_finale,Tableau)/Compteur_Couplage))
    print ("Taux dernière condition : "+str(Nbr_Dernière_Condition/Compteur_Couplage))

#ON ECRIT ICI UNE FONCTION QUI VA TESTER LA PRESENCE DE DOUBLONS DANS LA LISTE FINALE
(celle qui sera écrite dans un csv)
def Calcul_Doublons(Liste,Tableau):
    doublons=0
    for x in range(len(Liste_finale)-1):
        if Liste_finale[x]!=0 and Liste_finale.count(Liste_finale[x])>int(Tableau[14][x]):
            doublons+=1
    return(doublons)


#Calcul des performance importé de Couplage_Barbusse
def Calcul_Performances(Liste_Clé_Etrangère):
    Couplage_Théorique=0
    #On calcule ensuite le nombre de lignes théoriques couplées
    Intermé=[]
    for i in Liste_Clé_Etrangère:
        while i in Intermé:
            Intermé.remove(i)
        Intermé.append(i)
    Couplage_Théorique=len(Intermé)
    #On calcule le nombre d'entrées théoriques réelles (étant donné qu'un même trajet peut avoir
plusieurs lignes en raison de l'incertitude sur le bateau)
    Intermédiaire=[]
    for j in Tableau[8]:
        while j in Intermédiaire:
            Intermédiaire.remove(j)
        Intermédiaire.append(j)
    Nombre_Entrées_Théoriques=len(Intermédiaire)
    return(Couplage_Théorique,Nombre_Entrées_Théoriques)
```

```
#Coeur du programme:le couplage en lui même
Construction_Tableau_De_Données()   #on construit le Tableau de données
#On parcourt avec x les données AIS
for x in range(len(Lignes_Sans_La_Première)-1):
   #Resets de variables
   Possibilités=0  #nombre d'entrées théoriques correspondantes
   Test_Dernière=0 #stats dernière condition
   #On vérifie que la ligne AIS n'est pas vide (ça fait sens en réalité)
   if Tableau[1][x]!="":
      nbr_entrées_AIS+=1  #On compte les entrées AIS
      #On parcourt avec y les trajets théoriques
      for y in range(len(Lignes_Sans_La_Première)-1):
         #On compare les MMSi
         if Tableau[1][x]==Tableau[15][y]:
            #On compare les ports AIS avec les ports d'arrivées théoriques
            if Tableau[6][x]==Tableau[12][y]:
               dates_occurences=Construction_Occurences (y,Tableau) #On construit à chaque fois la
liste des des occurences
               Marge=Construction_Marge(y) #On calcule la marge qui sera nécessairement la même
pour toutes les occurences (mdr le temps que j'ai mis à m'en rendre compte)
               #On parcourt les occurences récemment construites
               for date in dates_occurences:
                  #On test ici la compatibilité des heures grâce à la marge et notre fonction
Code_Date_Heure plutôt pratique
                  if (Code_Date_Heure(Tableau[2][x], Tableau[3][x])+Delta)>(Code_Date_Heure(date,
Tableau[10][y])-15) and (Code_Date_Heure(Tableau[2][x],
Tableau[3][x])+Delta)<(Code_Date_Heure(date, Tableau[10][y])+Marge):
                     #Si la ligne n'a pas encore eu de correspondance, on note celle là
                     if Possibilités==0:
                        Possibilités=int(Tableau[8][y])
                     #Sinon, on la compare à la correspondance actuellement validée
                     elif abs((Code_Date_Heure(Tableau[2][x], Tableau[3][x])+Delta)-
Code_Date_Heure(date, Tableau[10][y])) < abs((Code_Date_Heure(Tableau[2][x],
Tableau[3][x])+Delta)-Code_Date_Heure(Tableau[16][Possibilités], Tableau[10][Possibilités])) :
                        Possibilités=int(Tableau[8][y])
                        Test_Dernière=1

                     #print("AIS n°"+Tableau[0][x]+'H AIS:'+Tableau[3][x]+'se couple
avec'+Tableau[8][y]+'H théo:'+Tableau[10][y])

   Liste_finale.append(Possibilités)   #On construit finalement notre Liste_Finale avec les valeurs
successives des correspondances
   Nbr_Dernière_Condition+=Test_Dernière
   if Possibilités!=0:
      Compteur_Couplage+=1



#Affichage des stats
Affichage_Stats(Liste_finale)
```

```
#LA PARTIE ECRITURE EST DESACTIVEE DANS CETTE VERSION DEDIEE A ETUDIER LA PRECISION ET L'EFFICACITE DU PROGRAMME
#ON CONSIDERES DONC QUE LA "Liste_finale" EST NOTRE PRODUIT FINI: NOTRE COLONNE CLE ETRANGERE
#IL SUFFIT D'ENLEVER LES "#" DEVANT LES LIGNES SUIVANTES POUR LA REACTIVER
#écriture dans un nouveau fichier csv (on copiera ensuite la colonne mais je préfère ne pas endommager le premier au cas où)
#for j in Liste_finale:
   #with open('E:\COME\cours\stage 4a\Test appareillage\Données_A_Ne_Pas_Modifier_Pour_Python\Clés_Etrangères.csv', mode='a+', newline='') as csvfile: #ICI CHEMIN DU FICHIER CSV POUR L'ECRITURE DE LA COLONNE DE CLES ETRANGERES
      #writer = csv.writer(csvfile, delimiter=';')
      #writer.writerow([str(j)])
```

Programme de couplage sélectionné (version utilisée pour le couplage sur le Transmanche

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun  4 13:40:37 2021

@author: COME
"""

#Version finale du couplage sur les arrivées utilisé pour le Transmanche ici.

import csv
import datetime


#On va essayer de faire presque entièrement ce code avec des définitions de fonction, ça sera peut-être plus clair à comprendre

#Ici on note les numéros d'index de nos différentes colonnes excel dans la grande liste imbriquée "Tableau" déclarée par la suite
#N°AIS 0
#MMSI_AIS  1
#Date_Arrivé_AIS  2
#Heure_Arrivée_AIS  3
#Date_Départ_AIS  4
#Heure_Départ_AIS  5
#ID_Port_AIS  6
#N_Théorique  7
#Heure_Départ_Théorique  8
#Heure_Arrivée_Théorique  9
#ID_Port_Origine 10
#ID_Port_Destination  11
#Date_Départ_Théorique  12
#Date_Arrivée_Théorique 13
#MMSI_Théorique  14
#Occurence  15

#Initialisation du Tableau des données
Tableau=[[]for i in range (16)]
lignes=[]
Lignes_Sans_La_Première=[]
#Delta empiriques calculés par ailleurs
#IL s'agit ici de rentrer dans la liste Delta les valeurs des temps moyens de manoeuvre. Leur rang correspond à l'ID du port dans la BD
#Attention, Python compte la première valeur d'une liste comme étant de rang 0 MAIS J'AI COMPENSE CELA PAR LA SUITE
#IL N'EST DONC PAS NECESSAIRE DE TOUT DECALER
Delta=[0,1,0,0,19,0,42,0,0,0,0,22,38,0,0,0,0]   #ce sont ici les valeurs du tranmanche
Marge_Relative=0.47
```

```
Seuil_Marge=60

#INTRODUCTION ICI DES VARIABLES STATISTIQUES
Compteur_Couplage=0 #ici on compte les couples pour le taux de couplage
nbr_entrées_AIS=0   #on compte le nombre de trajets AIS
doublons=0
Nbr_Dernière_Condition=0 #On compte le nombre de trajets couplés ayant nécessité d'utiliser la
dernière condition


#Dans cette fonction on crée une liste des dates d'occurences
def Construction_Occurences (y,Tableau):
   a=[]
   for d in range(1,int(Tableau[15][y])):    #Ligne bonne testée et approuvée par le comité
      if int(Tableau[13][y][0:2])+7*(d-1)<10:     #ligne validée par le conseil de guerre
         a.append('0'+str(int(Tableau[13][y][0:2])+7*(d-1))+Tableau[13][y][2:]) #corrigée par mes soins
      else:
         a.append(str(int(Tableau[13][y][0:2])+7*(d-1))+Tableau[13][y][2:]) #corrigée aux petits
oignons
   return(a)


#On définit une fonction qui construit le Tableau contenant toutes nos données excel grâce à deux
boucles ainsi que la commande readlines.
def Construction_Tableau_De_Données():
   with
open('E:\COME\cours\stage_4a\Couplages\TransManche\Données\Données_Pour_Python.csv',
newline='') as fichier:   #Chemin de notre feuille excel contenant les données en entrée
      global lignes
      global Lignes_Sans_La_Première
      lignes = fichier.readlines()
      Lignes_Sans_La_Première=lignes[1:]
      for row in Lignes_Sans_La_Première:
         b=row.split(';')    #on sépare la ligne en une liste contenant toutes les cases de cette ligne
         for a in range (15):
            Tableau[a].append(b[a])
    #là pour la dernière colonne on enlève les caractères "\r\n" indiquant le changement de ligne
         Tableau[15].append(b[15][0:1])




#On définit ici une fonction transformant nos dates et heures en codes numériques basés sur cette
année
def Code_Date_Heure(date,heure):
   x=(int(date[3:5])-5)*31*1440+int(date[0:2])*1440+int(heure[0:2])*60+int(heure[3:5])
   return(x)



Construction_Tableau_De_Données()
```

```
#fonction qui calcul le temps de trajet THEORIQUE à partir du numéro de la ligne conscernée.
def Temps_De_Trajet(x):
   return(abs(Code_Date_Heure(Tableau[13][x], Tableau[9][x])-Code_Date_Heure(Tableau[12][x],
Tableau[8][x])))

#fonction qui retourne un entier (en minutes) correspondant à la marge relative
def Construction_Marge(y):
   #Cette fonction test si le temps de trajet est plus petit que le seuil divisé par la marge. Dans ce cas
on retourne le Seuil comme valeur de Marge.
   #Dans le cas contraire, on retourne une marge égale au temps de trajet multiplié par la marge
relative
   tps=Temps_De_Trajet(y)
   if tps*Marge_Relative<Seuil_Marge:
      return(Seuil_Marge)
   else:
      return(tps*Marge_Relative)

#On reprend l'idée mais pour que ça marche
#1: On crée une liste de tous les trajets ais ayant le même mmsi
def Arrêts_Du_Bateau(y):
   Liste=[]
   #On parcourt ensuite les AIS avec x
   for x in range(len(Lignes_Sans_La_Première)):
      if Tableau[14][y]==Tableau[1][x]:
            Liste.append(x)
   return(Liste)

#2: On crée une fonction testant l'existence d'un départ correspondant à notre trajet (l'emmerde
c'est si on ne l'a pas pour x raisons on perd encore des données
#Test booléen pour voir si on trouve bien un départ quand on veut. Ce test dépend du paramètre
DELTA
def Test_Départ(Liste_de_Trajets_Possibles,y):
   test=False
   Occurences=Construction_Occurences(y, Tableau)
   #On parcours les Trajets possibles
   for a in Liste_de_Trajets_Possibles:
      #On test que c'est le bon port
      if Tableau[10][y]==Tableau[6][a]:
         #On cherche un départ correspondant
         for date in Occurences:
            if   (Code_Date_Heure(date,Tableau[8][y])-3)<(Code_Date_Heure(Tableau[4][a],
Tableau[5][a])-Delta) and (Code_Date_Heure(Tableau[4][a], Tableau[5][a])-
Delta)<(Code_Date_Heure(date,Tableau[8][y])+30):
               test=True
   return(test)

#3: On définit le test ici à l'arrivée (port arrivée + heure avc marge relative avec un seuil)
def Correspondance_Arrivée(Liste_Trajets_Possibles,y):
   Liste=[]
   Occurences=Construction_Occurences(y, Tableau)
```

```
    Marge=Construction_Marge(y)
    #On parcours les trajets du bateau correspondant
    for k in Liste_Trajets_Possibles:
        #On test les ports d'arrivée
        if Tableau[6][k]==Tableau[11][y]:
            #On test les horaires (on récupères le code du programme marge+seuil)

            for date in Occurences:
                if (Code_Date_Heure(Tableau[2][k], Tableau[3][k])+Delta[int(Tableau[6][k])-
1])>(Code_Date_Heure(date, Tableau[9][y])-15) and (Code_Date_Heure(Tableau[2][k],
Tableau[3][k])+Delta[int(Tableau[6][k])-1])<(Code_Date_Heure(date, Tableau[9][y])+Marge):
                    #On couple alors, enfin on ajoute à une liste en espérant qu'ils ne soient pas trop
nombreux mdr
                    Liste.append(k)

    return(Liste)

#Fonction qui est l'équivalent de la dernière condition utilisée précédemment adaptée à ce nouveau
programme
def Selection(L):
    Compteur=0
    #On parcourt une première fois la liste avec a
    for a in range(len(L)):
        #Vu que a est une liste on la parcourt avec d
        for d in range(len(L[a])):
            #On parcourt une deuxième fois la liste pour comparer du coup à tous les autres résultats avec
b
            for  b in range(len(L)):
                #Vu que b est une liste de taille variable on la parcourt avec c
                for c in range(len(L[b])):
                    #On ajoute le and afin que la boucle ne permette pas que chacun trouve son double en
lui-même
                    #De même on évite les problèmes liés aux calculs sur les str en vérifiant que ce n'est pas
une ligne déjà discriminée
                    if (L[b][c]==L[a][d] and (b!=a or c!=d) and L[b][c]!='suppr'):
                        if abs(Code_Date_Heure(Tableau[2][L[b][c]], Tableau[3][L[b][c]])-
Code_Date_Heure(Tableau[13][b], Tableau[9][b]))<abs(Code_Date_Heure(Tableau[2][L[a][d]],
Tableau[3][L[a][d]])-Code_Date_Heure(Tableau[13][a], Tableau[9][a])):    #Tableau[][L[a][d]]
                            L[a][d]='suppr'#Si l'on supprimait l'élèment alors que l'on était encore dans le boucle,
la taille de la liste changerait et causerait des erreurs d'index par la suite.
                        else:
                            L[b][c]='suppr'#On note donc simplement les élèments sélectionnés afin de les
supprimer par la suite dans une nouvelle boucle
    #Ensuite, il nous reste à delete tous les 'suppr' de nos listes
    for k in range(len(L)):
        while 'suppr' in L[k]:
            Compteur+=1
            del L[k][L[k].index('suppr')]
    return(L,Compteur)

def Calcul_Performances(Liste_Clé_Etrangère):
    Couplage_AIS=0
```

```
    Couplage_Théorique=0
    #On calcule d'abord le nombre de lignes AIS couplée
    for k in Liste_Clé_Etrangère:
        Couplage_AIS+=len(k)
    #On calcule ensuite le nombre de lignes théoriques couplées
    for i in Liste_Clé_Etrangère:
        if len(i)>0:
            Couplage_Théorique+=1
    #On calcule le nombre d'entrées théoriques réelles (étant donné qu'un même trajet peut avoir plusieurs lignes en raison de l'incertitude sur le bateau)
    Intermédiaire=[]
    for j in Tableau[7]:
        while j in Intermédiaire:
            Intermédiaire.remove(j)
        Intermédiaire.append(j)
    Nombre_Entrées_Théoriques=len(Intermédiaire)
    return(Couplage_AIS,Couplage_Théorique,Nombre_Entrées_Théoriques)



#abs(Code_Date_Heure(Tableau[2][x], Tableau[3][x])-Code_Date_Heure(date, Tableau[10][y])) < Code_Date_Heure(Tableau[2][x], Tableau[3][x])-Code_Date_Heure(Tableau[16][Possibilités], Tableau[10][Possibilités])


#On compte dans cette fonction les aberrations dans une liste en attendant de les supprimer (cette fonction n'a plus d'intérêt à l'heure actuelle)


#Fonction simple qui nous donne le nombre d'entrées AIS²
def Compter_AIS():
    Nbr=0
    for i in range(len(Lignes_Sans_La_Première)-1):
        if Tableau[0][i]!='':
            Nbr+=1
    return(Nbr)

def Compter_Théorique():
    Nbr=0
    for i in range(len(Lignes_Sans_La_Première)-1):
        if Tableau[15][i]!='\r':
            Nbr+=1
    return(Nbr)

#Fonction "finale" qui réutilise globalement toutes celles étudiées plus haut
def Couplage():
    Liste_finale=[]
    Doublons=0
    for y in range(Compter_Théorique()):
        Possibilités=[]
        L=Arrêts_Du_Bateau(y)  #on sélectionne sur les mmsi ici
```

```
        #if Test_Départ(L, y):   #On vérifie que les conditions sur le départ sont respectées
        Possibilités=Correspondance_Arrivée(L, y)   #On crée une petite liste des entrées AIS correspondants à notre trajet théorique (correspondance sur l'arrivée toujours)
        Liste_finale.append(Possibilités)
        #On compte le nombre de doublons au sens précédemment utilisé
        if len(Possibilités)>int(Tableau[15][y]):
            Doublons+=1
    Resultats=Selection(Liste_finale)
    #On compte ici le nombre d'entrées AIS associées à plusieurs trajets théoriques (ce qui est nécessairement une aberration)
    #print(Resultats[0])
    Stats=Calcul_Performances(Resultats[0])
    print('taux de couplage AIS : '+str(Stats[0]/Compter_AIS()))
    print(str(Stats[0]))
    print('taux de couplage Théorique : '+str(Stats[1]/Stats[2]))
    print(str(Stats[1]))
    print("taux de doublons : "+str(Doublons/Stats[0]))
    print('taux dernière condition : '+str(Resultats[1]/Stats[0]))
    #print(Resultats[0])
    Ecriture(Resultats[0])

def Ecriture(Liste_Source):
    Liste=[[]for i in range (Compter_AIS())]  #On crée une liste vide que l'on va ensuite remplir assez naturellement
    #Il faut d'abord créer une liste adaptée à l'écriture
    for j in range(len(Liste_Source)):  #On parcourt notre liste
        if Liste_Source[j]!=[]:
            for k in Liste_Source[j]:
                Liste[k].append(Tableau[7][j])  #On insère au bon endroit de notre liste vide le numéro unique théorique correspondant
    print(Liste)
    c=0
    for j in Liste:
        if j!=[]:
            c+=1
    print (str(c))
    for k in Liste: #écriture proprement dite
        if k==[]:
            with open('E:\COME\cours\stage_4a\Couplages\TransManche\Données\Clé_Etrangère.csv', mode='a+', newline='') as csvfile:  #Chemin de la feuille csv vide où créer la colonne de clé étrangère.
                writer = csv.writer(csvfile, delimiter=';')
                writer.writerow([])
        else:
            with open('E:\COME\cours\stage_4a\Couplages\TransManche\Données\Clé_Etrangère.csv', mode='a+', newline='') as csvfile:  #Chemin de la feuille csv vide où créer la colonne de clé étrangère.
                writer = csv.writer(csvfile, delimiter=';')
                writer.writerow([int(k[0])])
```

POLYTECH TOURS
35 ALLÉE FERDINAND DE LESSEPS
37200 TOURS

Côme Geoffray
2020-2021

Qualité et ponctualité dans les transports maritimes réguliers de voyageurs

Résumé : Dans l'optique d'étendre les compétences de l'AQST au domaine des transports maritimes réguliers de voyageurs, ce stage donne une première maquette des méthodes pouvant être mises en œuvre pour étudier la ponctualité de ces transports. Au moyen de programmes de webscrapping, des données issues de l'AIS sont récupérées et comparées aux offres annoncées par des compagnies de transports maritimes de voyageurs.

Mots Clés : Transports maritimes réguliers de voyageurs, AIS, qualité de service dans les transports, ponctualité.

Autorité de la Qualité de Service dans les Transports :
Tour Séquoia, La Défense

Tuteur entreprise :
Alain Sauvant
Directeur de l'AQST

Tuteur académique :
Hervé Baptiste