Etude des marqueurs de l'interdisciplinarité par l'analyse d'échanges mails

Rapport - Projet de programmation 2

L3 CMI Informatique
Faculté des Sciences
Université de Montpellier
2021-2022

HADDAD Gatien
COSSU Arnaud
BOURRET Maxime
SAID Adam

Encadrés par LAURENT Anne
ALEVEQUE Guillaume
LIBOUREL Thérèse
NOURRIT Deborah

Table des matières

1 Présentation

1.1 Le sujet

L'HUman at home projecT est une étude menée depuis 2018 à Montpellier qui a pour but d'en apprendre plus sur les comportements humains dans un habitat connecté. Cela a débuté par l'emménagement de deux étudiants volontaires dans un appartement rempli de capteurs de divers types. Les précieuses données récoltées sont ensuite utilisées par une soixantaine de chercheurs de branches différentes (juridique, informatique, économique, électronique, sanitaire...) afin d'étudier les aspects des interactions humaines au fil des jours. Ces expériences étant donc menées par des chercheurs provenant de différents domaines scientifiques, elles révèlent des marqueurs de l'interdisciplinarité [1].

Le sous-projet de HUT sur lequel nous avons travaillé en découle pour s'insérer dans l'analyse des marqueurs de l'interdisciplinarité au sein d'un groupe de travail. Pour ce faire, nous avons eu à notre disposition des mails confidentiels échangés par des personnes de différents milieux travaillant sur le même sujet.

Grâce à nos connaissances en informatique, nous savions que même si l'analyse de langage naturel est très compliquée, il est possible de faire ressortir des informations importantes à l'aide des métadonnées [2] comprises dans les mails. C'est avec cette idée que nous avons abordé le sujet, dans le but de faire ressortir des données intéressantes à étudier par la suite.

L'informatique étant en plein essor, le groupe de chercheurs utilisait beaucoup de moyens de communications dématérialisés dont les mails pour échanger leur point de vue, prévoir des réunions ou autre. La messagerie utilisée est `Thunderbird` et les mails ne pouvaient pas être utilisés depuis l'application, heureusement il est possible de les extraire avec toutes leurs métadonnées dans un format texte brut. Ainsi, après avoir étudié l'interdisciplinarité au travers d'une recherche bibliographique, nous avons implémenté un programme capable de rendre ces données brutes utilisables par une machine et compréhensibles par un humain. Nous avons créé un outil qui va nettoyer le fichier retourné par `Thunderbird`, recréer les fils de discussion, puis, selon les demandes de l'utilisateur, faire ressortir les informations contenues dans les mails telles que les temps de réponse, la présence de pièce jointe ou la longueur du message.

1.2 L'approche

Pour ce faire, nous avons choisi d'utiliser `Python`. Ce langage permet une manipulation facile de fichiers volumineux et est peu contraignant en étant facile d'utilisation. Chaque membre du groupe était à l'aise avec la syntaxe et a facilement réussi à l'utiliser de chez lui. Cela s'est avéré être un bon choix, car même avec de longues chaînes de caractères, le

1. Interdisciplinarité : Approche de problèmes scientifiques à partir des points de vue de spécialistes de disciplines différentes (CNRTL)
2. Métadonnées : Données servant à caractériser une autre donnée, physique ou numérique (Larousse)

typage faible de `Python` et ses fonctions implémentées dans la librairie de base nous ont permis d'arriver à nos fins. La grande compatibilité du langage nous permet d'obtenir un programme facilement exécutable (au-delà du temps d'exécution variant selon la taille des mails à traiter) sur presque n'importe quelle machine.

Afin d'être capable de manipuler les mails avec l'outil, il a fallu faire en sorte que ce dernier soit fluide et optimisé. La meilleure manière d'aborder ce problème que nous avons trouvé est tout d'abord de découper le fichier source à analyser en plusieurs petits pour alléger la suite. Ensuite, une fois les mails plus facilement manipulables, il nous a fallu les nettoyer pour supprimer les données inutiles et les formater pour faire ressortir l'essentiel. Enfin, la dernière étape avant l'analyse du contenu est la reconstruction des fils de discussion qui consiste à regrouper les fichiers mails ayant le même objet dans un dossier ayant le même nom.

1.3 Le cahier des charges

A l'aide de nos encadrants, nous avons évalué, lors d'une réunion sur `Zoom`, ce qu'il nous était possible de faire en fonction du temps donné, de leurs attentes et de nos capacités. Au terme de ce débat, nous en avons conclu que nous devions être capables de fournir un programme capable d'analyser un corpus de mails de taille quelconque et d'en ressortir plusieurs données, notamment la longueur des mails, la fréquence par jour, les adresses les plus présentes, le temps de réponse d'un fil de discussion... L'outil se devait d'être léger, facile d'utilisation pour des personnes étrangères à la programmation et fonctionnel sur tous types d'OS. Aussi il doit permettre l'export des données extraites vers un format texte compréhensible pour que ces dernières puissent être utilisées par la suite en dehors du programme.

2 Technologies utilisées

Pour accomplir les tâches qui nous ont été données nous avons donc, comme vu plus haut, utilisé `Python` comme langage de programmation. Pour communiquer au sein de l'équipe, nous avons utilisé `Discord`, nous restions en contact avec les encadrants par mail et faisions des réunions sur `Zoom`. Chacun d'entre nous utilisait `Visual Studio Code` sur sa machine, un éditeur simple et pratique d'utilisation, mais afin de mettre en commun nos parties, nous codions beaucoup sur `replit` également. Il s'agit d'un environnement de programmation en ligne permettant à plusieurs personnes de programmer en même temps et de pouvoir tester les résultats directement. Cela s'est avéré très utile pour travailler ensemble et aussi pour faire des sauvegardes avec `Github` nous permettant de garder des versions antérieures de notre code et récupérables facilement. `Github` nous a également permis de créer des paquets afin de proposer petit à petit des versions finalisées de notre code prêt à l'emploi. Enfin, nous avons utilisé `SonarCloud` afin de vérifier, améliorer notre code et éviter des mauvaises pratiques de développement (redondance des lignes, mauvaise syntaxe...)

3 Développement

3.1 Le programme

Notre programme final est donc un analyseur de mail qui permet, à partir d'un corpus de mails, de retrouver différentes informations et statistiques telles que les jours de la semaine où les personnes ont le plus communiqué, les adresses ayant envoyé le plus de mails ou encore le temps mis par une personne pour répondre à un précédent mail.
Le programme permet également de filtrer un corpus de mails afin de ne garder que ceux voulus, par exemple il est possible de trier par adresse, par date, par mot présent dans l'objet ou encore par contenu du mail.
Une fois le corpus sélectionné et filtré comme voulu, il est possible d'appliquer deux méthodes différentes. La première est un rapport contenant toutes les statistiques explicitées précédemment. La seconde est le temps de réponse et ne fonctionne que sur les fils de discussions.
Après chaque action, il est possible d'exporter les résultats en `CSV` pour de futures analyses ou pour les importer dans d'autres logicels comme `Excel`.

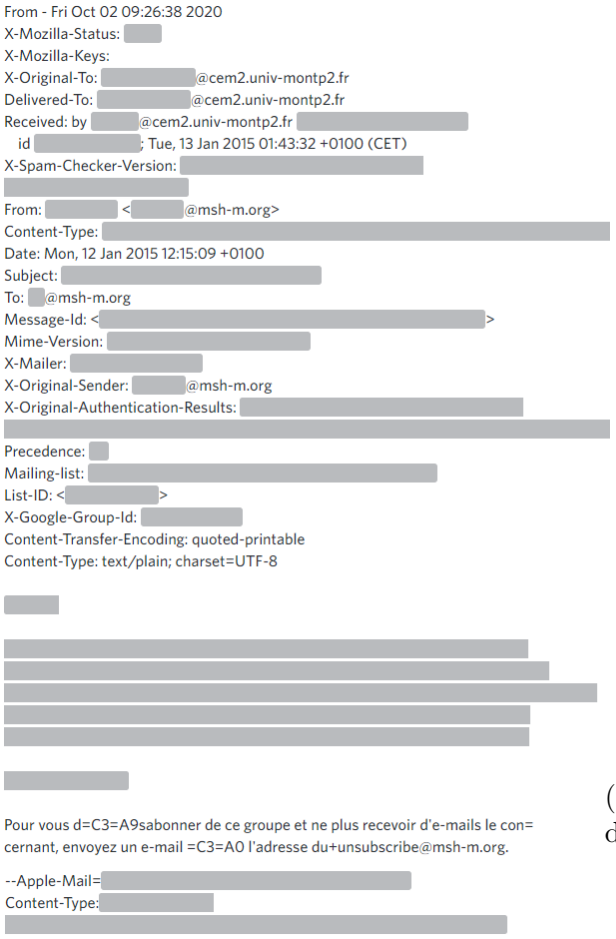
3.2 Les scripts

Notre programme est donc composé de plusieurs scripts qui permettent son fonctionnement. Dans l'ordre d'exécution nous avons d'abord le "main", qui est le programme principal. C'est lui qui va s'occuper de faire les affichages dans le terminal, de créer les dossiers nécessaires au bon fonctionnement et de lancer les fonctions contenues dans les autres scripts. Ensuite vient le "parser", ce fichier contient les fonctions qui vont découper, nettoyer et créer les fils de discussions pour permettre les analyses futures. C'est également lui qui va s'occuper de réencoder les fichiers en cas de problème. Après cette première étape de préparation des mails, nous avons le "script filter", son utilisation permet de filtrer un groupe de mails suivant différents critères afin de n'avoir que les mails voulus. Puis nous avons les deux scripts d'action : "responseTime" et "statistique". Le premier permet de retracer la chronologie d'un fil de discussion en calculant le temps de réponse entre chaque mail ayant un même objet. Le second quant à lui, va générer selon le corpus un rapport conséquent contenant les statistiques explicitées précédemment. Enfin nous avons "clean", ce petit script est utilisé tout au long de l'exécution du programme, il permet d'importer de petites fonctions utiles tout au long des différentes étapes, comme par exemple pour nettoyer l'écran du terminal, ou pour générer des chemins vers des fichiers (avec / ou \) suivant le système d'exploitation de la machine.

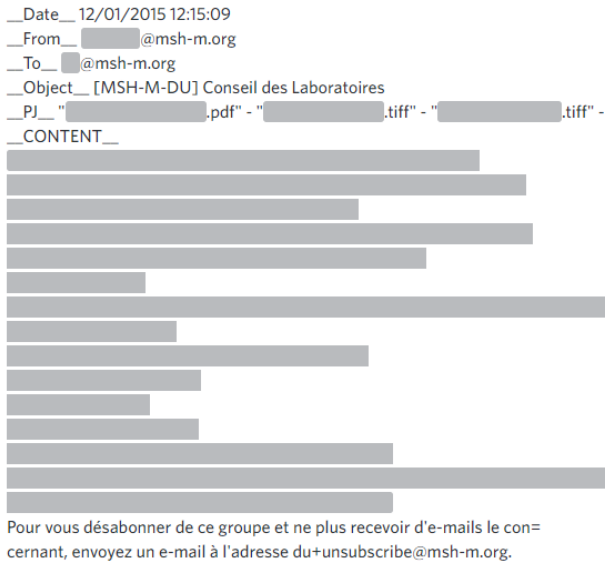
3.3 Les données

Les données en entrée sont des données brutes, généralement un gros fichier extrait grâce à un gestionnaire de mails. Les fichiers ont une taille pouvant varier selon la longueur de la discussion originale et le nombre de mails envoyés. Ils comportent en général les encodages, ainsi que toutes les informations sur les mails : la date d'envoi, l'objet, l'adresse de l'expéditeur, l'adresse du destinataire, les pièces jointes...
Mais le fichier comporte aussi nombre d'informations et de lignes inutiles telles que les serveurs par lesquels transite le mail ainsi que leur `ip`, des informations sur la sécurité du

mail, son chiffrement, le contenu transcodé en `HTML` et des blocs de signatures numériques (voir figure ci-après). Afin de pouvoir exploiter les mails, il nous faut donc scinder ce fichier volumineux pour extraire les mails un par un et pouvoir réaliser les différentes opérations d'analyse, tout en triant les informations et ne garder seulement ce qui est nécessaire. Par exemple, nous allons extraire et garder l'objet du mail, ainsi que les différents champs cités plus haut qui pourront nous servir. Nous allons par la même occasion les rendre plus accessibles en les précédant de champs d'identification comme "__Objet__" ou "_PJ_" pour objet du mail et pièce jointe. Cette convention nous permettra de trouver et récupérer ces informations plus facilement par la suite.



(a) Exemple d'un mail brut avant son passage dans le programme



(b) Exemple d'un mail nettoyé après son passage dans le programme

FIGURE 1 – Comparaison d'un mail avant et après son passage dans le programme

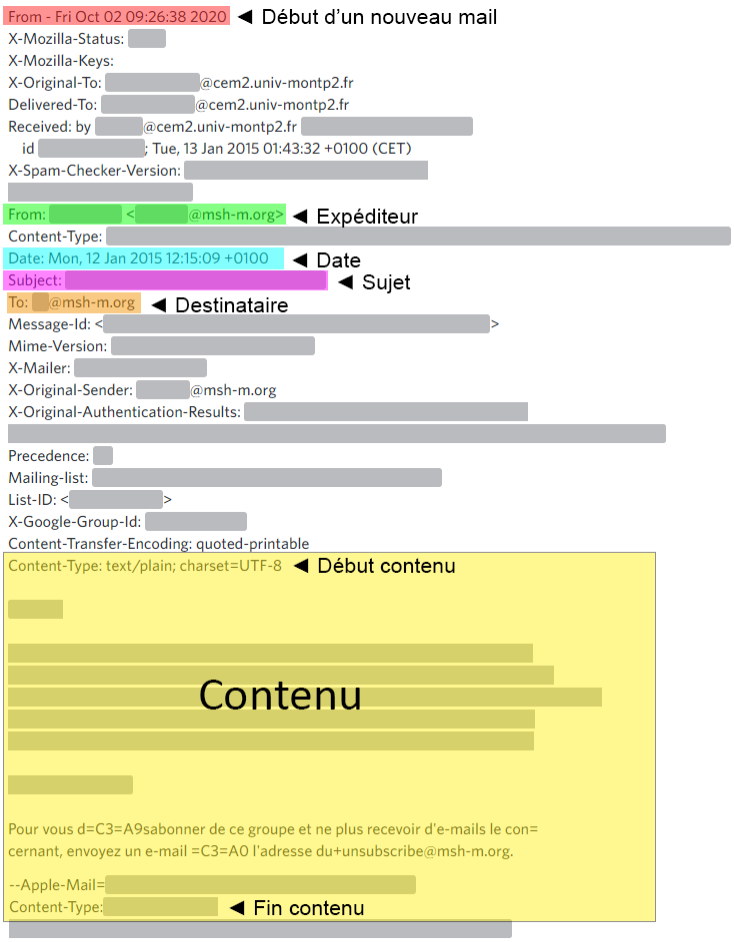


FIGURE 2 – Extrait d'un fichier brut avec les lignes clés surlignées

3.4 Problèmes rencontrés

3.4.1 Encodage

Lors de l'utilisation de la fonction découpage, qui sert à extraire tous les mails d'un même fichier, nous avons eu des problèmes d'encodage des caractères. Ce problème s'est propagé par la suite dans les mails découpés car la méthode d'encodage n'est pas récupérée dans les mails extraits par Thunderbird et les fichiers bruts n'étaient donc pas réencodés en `UTF-8`. Même si la méthode `open` de `Python` permet l'encodage, cela ne fonctionnait pas car les caractères bruts n'étaient pas bien écrits dans les fichiers de base. Il a donc fallu simplement, et à la main, réaliser un dictionnaire permettant de transformer chaque caractère en une suite de caractère en caractère `UTF-8`. Après la création de ce dictionnaire, nous avons implémenté des boucles qui se sont servies de ce dernier, par exemple dans la fonction découpage, afin d'extraire les mails et de les réécrire de façon propre et lisible.

```
=C3=A9:é
=C3=A8:è
=E9:é
=E8:è
=C3=AA:ê
=C3=AE:î
=EE:î
=C3=89:É
```

FIGURE 3 – Extrait du dictionnaire créé

3.4.2 Confidentialité des mails

Pour tester notre outil, nous avons eu à disposition des mails confidentiels provenant de vraies équipes de travail. Ces derniers étaient stockés sur un ordinateur qui nous a été confié après que nous avons signé une charte de confidentialité. Ils furent très utiles pour tester nos fonctionnalités, mais cela nous forçait à transférer chaque version du programme sur l'ordinateur pour l'exécuter en local. Nous avons donc décidé de récupérer le squellette d'un fil de discussion brut et l'avons modifié pour retirer toutes les informations confidentielles tout en gardant la forme. Cela nous a permis de pouvoir tester l'exécution de notre programme en dehors de l'ordinateur de *HUT*, tout en conservant la confidentialité des données qui nous étaient fournies.

3.4.3 Analyse de gros fichiers

L'ordinateur de *HUT* stockant déjà des fichiers volumineux, il pouvait mettre longtemps à éxecuter le code que nous testions. Les fichiers mails de `Thunderbird` étaient très lourds (plusieurs centaines de Mo) et ralentissaient l'exécution car très difficiles à manipuler.

3.4.4 Comptabilité entre OS

Notre programme est fait pour ouvrir des fichiers, créer des dossiers, afficher des instructions dans la console...Nous le développions sur des machines utilisant `Linux`, or il devait être également compatible avec `Windows`. Nous avons donc rencontré des problèmes notamment lors du parcours de l'arborescence, `Linux` utilisant le séparateur "/" contrairement au "\" utilisé par `Windows`.

3.5 Statistiques

Notre paquet final se constitue de six fichiers au format `Python` qui contiennent chacun différentes fonctions utilisées tout au long de l'exécution du programme, d'un fichier texte étant le dictionnaire pour le réencodage des caractères, d'un autre fichier texte expliquant le contenu du rapport généré ainsi qu'un lanceur permettant d'exécuter le programme sous `Windows` (sous `Linux` on peut le lancer directement avec une commande dans le terminal, voir le manuel). Le programme va également créer trois dossiers au long de son exécution, "tmp" où vont être stockés les mails nettoyés, "threads" contenant les mails groupés par fils de discussion et "__MAIL_DEPOT__" qui sert à déposer ses fichiers

bruts. Mis à part ce dernier, les dossiers sont supprimés à la fermeture du programme. Voici les différents fichiers de scripts (leur utilité a été explicitée plus haut) :

- `main.py` : 166 lignes
- `parser.py` : 246 lignes
- `clean.py` : 26 lignes
- `filter.py` : 156 lignes
- `responseTime.py` : 66 lignes
- `statistique.py` : 311 lignes

Au final ce projet contient **971** lignes de code et pèse **291 ko**.

4 Algorithmes

4.1 Le "parser"

Au démarrage du programme et après avoir importé les fichiers que l'on souhaite analyser, le ‘‘`parser`’’ est le premier script qui est lancé. Il contient plusieurs fonctions qui vont chacune à leur tour découper et nettoyer les fichiers ajoutés afin de rendre des mails propres pour l'analyse. Une première fonction va ouvrir tour à tour les fichiers bruts, qui peuvent être lourds, afin de les découper en plusieurs petits. Pour ce faire, nous avons constaté que chaque début de mail peut être identifié par une ligne commençant par ‘‘`From - `‘‘. Nous allons donc ouvrir le fichier et, à chaque occurrence de ladite ligne, créer un nouveau fichier contenant le mail. Chaque mail généré va être placé dans un nouveau dossier qui portera le nom du fichier brut de base et sera identifié par un numéro (voir figure ci-après). Vous pouvez retrouver le code de cette fonction en annexe du rapport.

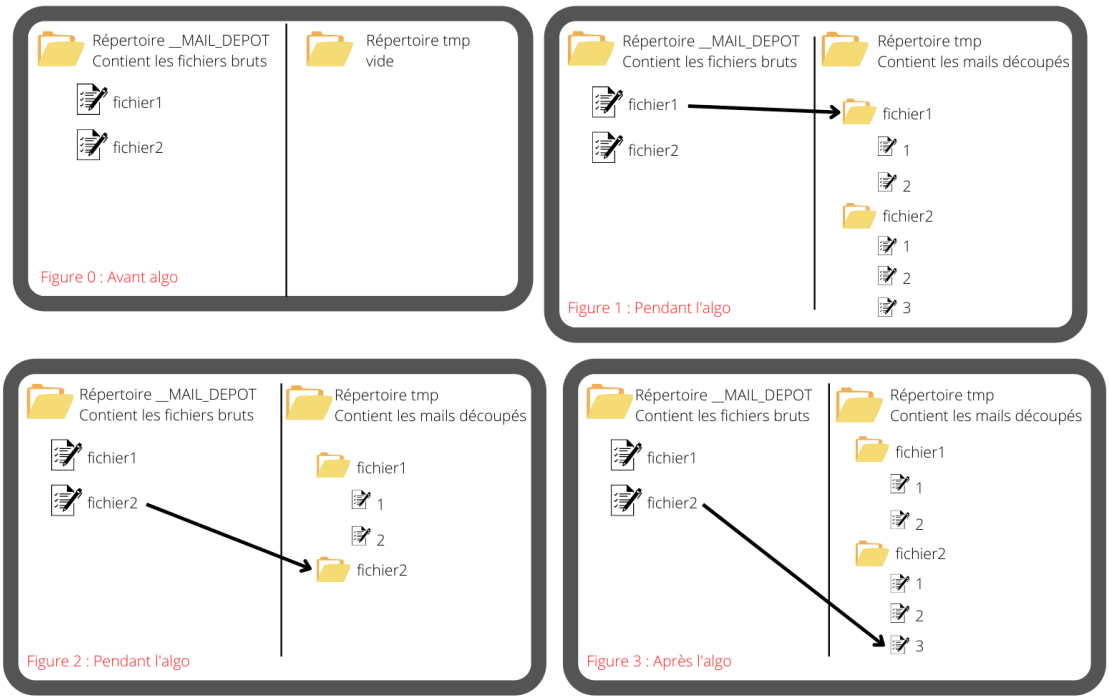


FIGURE 4 – Trace d'exécution de la fonction de découpage

1. L'algorithme ouvre le premier fichier et crée un dossier dans `"tmp"` portant le même nom (fichier1)
2. Il va avancer ligne par ligne dans ce fichier en cherchant celles commençant par `"From - "`
3. A chaque fois qu'il en trouve une, l'algo va créer un nouveau fichier dans le nouveau dossier avec pour nom un nouveau numéro
4. Tant qu'on ne tombe pas sur une nouvelle occurrence de ladite ligne, on va écrire les lignes dans le nouveau fichier

   On obtient l'arborescence `tmp` de la figure 1 dans l'image ci-dessus

5. L'algorithme ouvre le second fichier et va faire la même opération
6. Il crée un nouveau dossier du même nom dans `"tmp"`

   On obtient l'arborescence `tmp` de la figure 2.

7. On analyse ligne par ligne et, à chaque fois que l'on passe sur la ligne spéciale, on ouvre un nouveau fichier et on ferme le précédent

   On obtient l'arborescence `tmp` de la figure 3.

Maintenant que nous avons des petits fichiers séparés, nous pouvons les nettoyer un à un afin de ne garder que les parties intéressantes, c'est-à-dire les métadonnées et le contenu du mail. Avant nettoyage, ces fichiers contiennent un nombre très important de données utiles lors d'une discussion sur internet mais totalement inutiles pour notre analyse. Parmi elles, nous retrouvons celles que nous avons jugées utiles à savoir les destinataires, l'expéditeur, l'objet, la date, les pièces jointes et bien sûr le contenu. Pour chacune de ces données, il a fallu déterminer au préalable la syntaxe et les mot-clés qui permettent d'identifier ces lignes parmi les autres (voir structure des mails). Une fois ce travail de recherche effectué, nous avons mis en place une seconde fonction qui va, pour chaque nouveau mail, analyser ligne par ligne les mot-clés présents qui peuvent se traduire par le début ou la fin d'une ligne. Par exemple, pour identifier la ligne contenant l'objet, on va regarder si la ligne en question commence par `‘‘Subject:’’`. Chaque donnée sera stockée dans une variable et, à la fin de la lecture, on videra le fichier pour ensuite écrire chaque variable dedans (voir mail nettoyé). De plus, pour chaque ligne, nous allons regarder si on ne trouve pas de caractère mal encodé. Si c'est le cas, il sera remplacé par la valeur correspondante dans le dictionnaire. Vous pouvez retrouver le code de la fonction de nettoyage dans l'annexe.

4.2 Le "threader"

L'une de nos problématiques était de pouvoir reconstruire les fils de discussion afin de retracer la chronologie des mails.
Pour créer les fils de messages, nous avons implémenté une fonction `"threader"` qui va

regarder les mails dans `"tmp"` qui sont déjà découpés et nettoyés. Cette fonction est appelée à la demande de l'utilisateur, quand il souhaite utiliser les fils de discussion.
Pour chaque mail du dossier, la fonction va effectuer deux tests logiques, le processus est résumé dans le schéma ci-dessous.

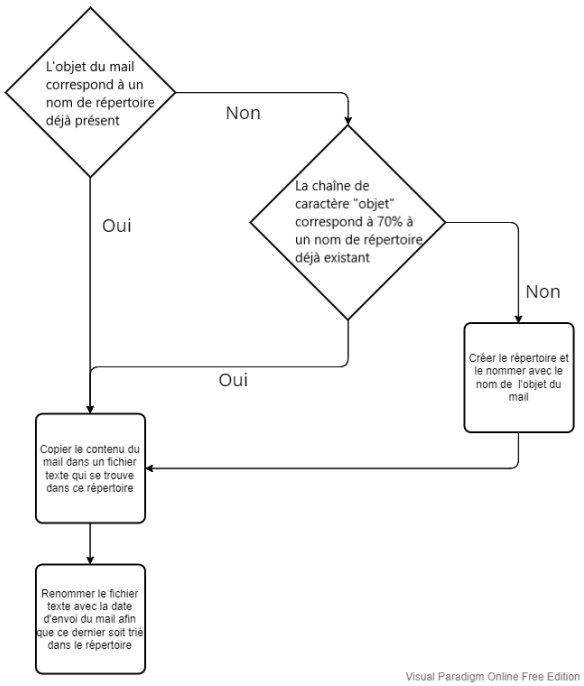


FIGURE 5 – Schéma de modélisation de la fonction de création des fils de discussion

Un premier test sera de vérifier si nous avons déjà un fil de messages avec le même nom d'objet, dans ce cas le mail actuel en cours d'analyse appartient à la discussion. Il suffit juste de créer un fichier texte dans le répertoire correspondant au nom d'objet créé précédemment afin d'y copier le contenu du mail, puis de le nommer avec la date d'envoi. La notation avec la date d'envoi permet aux systèmes qui utiliseront l'explorateur de fichier de trier directement les messages du plus vieux au plus récent, ou inversement.

Si l'objet du mail est différent de tous les noms de répertoires déjà créés, alors on effectue un deuxième test logique. Ce deuxième test va vérifier si notre objet est contenu dans un autre objet, donc dans un nom de répertoire déjà créé. On va utiliser la fonction `SequenceMatcher` qui permet de comparer les noms des objets. Cette dernière prendra en paramètre un `double` qui correspond au pourcentage de ressemblance entre les chaînes de caractères. Dans notre code, nous avons mis à 75%, soit 0.75 ce qui permet d'établir le test suivant : "Si l'objet correspond au minimum à 75% à un nom de répertoire déjà créé, alors ce mail fait partie de cette file de messages". Il faut ensuite réaliser les deux mêmes étapes à savoir copier le mail et le nommer avec la date d'envoi.

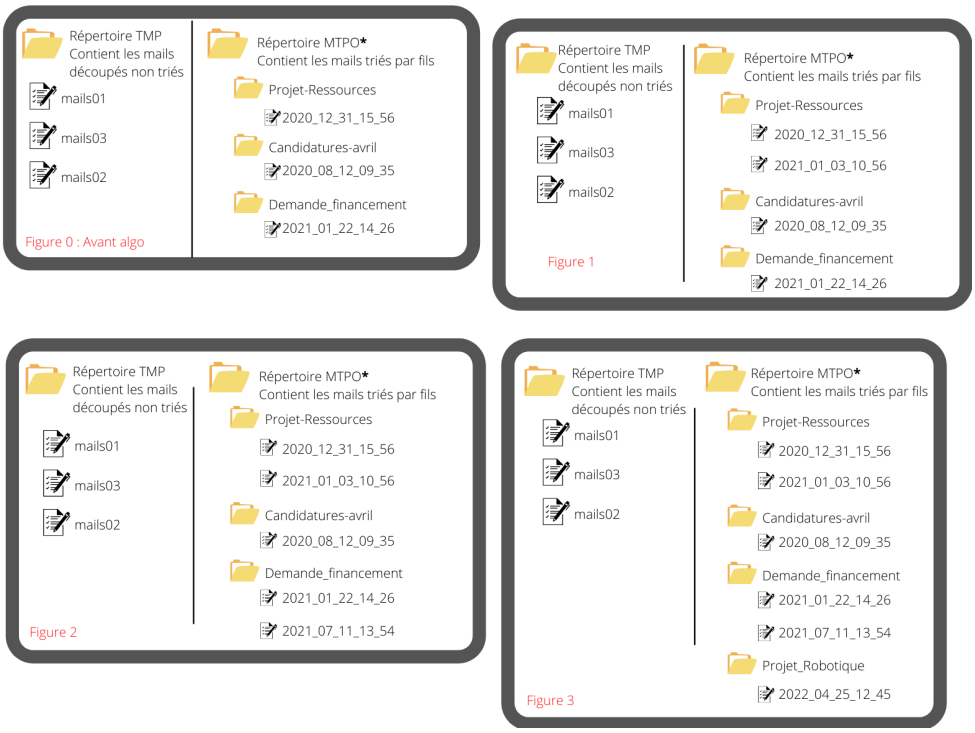


FIGURE 6 – Trace d'exécution de la fonction threader

1. L'algorithme récupère mails01 et son objet qui est : "Projet-Ressources"
2. Il voit que l'objet correspond à un répertoire déjà créé et commence donc à copier le contenu du mail dans un fichier texte
3. On nomme le fichier avec la date d'envoi du mail, ici "2021_01_03_10_56" soit le 3 janvier 2021 à 10h56

   On obtient l'arborescence MTPO de la figure 1

4. L'algorithme récupère mails02 et son objet qui est : "Demande_financ"
5. Il voit que l'objet ne correspond pas à un répertoire déjà créé, il vérifie alors si l'objet correspond à 75% à l'un des noms de répertoires déjà créés
6. L'objet correspond, il faut donc copier le contenu du mail dans un fichier texte et le placer dans ce répertoire
7. On nomme le fichier avec la date d'envoi du mail, ici "2021_07_11_13_54"

   On obtient l'arborescence MTPO de la figure 2.

8. L'algorithme récupère mails03 et son objet qui est : "Projet-Robotique"
9. Il voit que l'objet ne correspond pas à un répertoire déjà créé, il vérifie alors si l'objet correspond à 75% à l'un des noms de répertoires déjà créés
10. Aucun répertoire ne correspond au minimum à 75%, ce mail fait donc partie d'un autre fil de messages

11. On créée le répertoire avec l'objet du mail
12. On créée un fichier texte dans ce répertoire et on y copie le contenu de mails03
13. On nomme le fichier avec la date d'envoi du mail, ici "2022_04_25_12_45"

    On obtient l'arborescence MTPO de la figure 3.

Les différents tests effectués permettent de ne pas créer plusieurs fois le même fil de messages et avec un taux à 75% on évite aussi de corréler des mails qui n'ont rien à voir. Si l'objet d'un mail est tronqué par un système ou une fonction, selon l'étendue de la coupe de la chaîne de caractère, on pourra tout de même retrouver avec de fortes chances la file de messages correspondante.

Vous pouvez retrouver le code de cette fonction dans l'annexe
Après ces deux fonctions, nous nous retrouvons donc avec des mails nettoyés et classés prêts à être analysés.

4.3 Le temps de réponse

L'une des fonctionnalités attendues de l'outil est de pouvoir calculer les temps de réponse au sein d'un fil de discussion. La tâche s'annonçait compliquée vue la quantité de métadonnées contenues dans les mails. Heureusement, grâce aux étapes de nettoyage et de recréation des fils réalisées en amont, cette étape fut simplifiée. Chaque mail traité commence donc par `"__Date__ jj/mm/aaaa hh:mm:ss"` suivi de `"__From__ adresse@mail"`, ainsi l'essentiel des données nécessaires pour le calcul peuvent être récupérées pour être interprêtées à la chaîne. Les fonctions utilisées sont regroupées dans le fichier `responseTime.py` et sont appelées par le main lorsque l'utilisateur choisit l'option (3) pour analyser des threads. Une fois le fil de discussion sélectionné, le dossier correspondant est passé à la fonction suivante.

Retrouvez le code dans l'annexe

Cette dernière va parcourir chaque fichier du dossier qui correspond à chaque mail du fil de discussion. Elle va récupérer l'émetteur et l'heure, en créer un tableau et ajouter ce dernier à un autre tableau qui sera renvoyé trié par date d'envoi croissante. Il va donc ensuite être utilisé par les autres fonctions du fichier, pour afficher à l'utilisateur le temps mis entre les réponses et peut être exporté dans un fichier texte.

4.4 Les statistiques

La fonction statistique peut être considérée comme la finalité des scripts précédents ; c'est elle qui permet d'extraire depuis les mails des données exploitables pour pouvoir ensuite les traiter et les annalyser.
Lors de son lancement, l'utilisateur choisit s'il veut générer un rapport sur un corpus de mails (qu'il soit filtré ou qu'il provienne d'un thread), dans les deux cas, la fonction va appeler le script `rapport` (dont le code se trouve dans l'annexe) qui va fournir des statistiques sur quatre grand axes :

— Les périodes
— Les adresses
— La longueur moyenne des mails
— Les pièces jointes

Dans un premier temps, on a les statistiques concernant des périodes. Cela inclus la répartition des mails selon les années, les mois, ainsi que les jours de la semaine. Pour les deux premier, il suffisait de récupérer la date du mail dans l'en-tête de ce dernier, tandis que pour le jour de la semaine, il fallait convertir la date qui était au format JJ/MM/AAAA en un jour de la semaine. Pour ce faire, nous avons donc utilisé la bibliothèque `calendar`.

	A	B
1	Années	Pourcentage
2	2015	2,20%
3	2016	25,90%
4	2017	31,00%
5	2018	40,90%

12	Mois	Pourcentage
13	Mars	13.96%
14	Janvier	13.29%
15	Novembre	9.59%
16	Mai	9.39%
17	Février	8.6%
18	Decembre	8.27%
19	Septembre	7.8%
20	Avril	7.74%
21	Octobre	7.54%
22	Juin	7.34%
23	Juillet	5.09%
24	Aout	1.39%

26	Semaine	Pourcentage
27	Mardi	22.09%
28	Lundi	19.11%
29	Mercredi	18.58%
30	Jeudi	18.39%
31	Vendredi	15.34%
32	Dimanche	3.97%
33	Samedi	2.51%

FIGURE 7 – Exemple de répartition des années, des mois et des jours de la semaine

Pour les adresses, nous avons deux fonctions : la première récupère l'ensemble des adresses présentes dans le corpus (ou dans l'ensemble des mails pour un rapport complet). Une fois ces adresses récupérées, la seconde fonction parcourt cette fois-ci l'ensemble des mails et compte combien la contiennent. Comme nous avons accès au nombre de mails totaux, il suffit alors de transformer le quotient en pourcentage pour plus de lisibilité.

7	Adresses	Pourcentage
8	XXX@univ-montp2.fr	24.87%
9	XXX@univ-montp2.fr	24.8%
10	XXX@yahoo.fr	21.63%

FIGURE 8 – Pourcentage d'expédition des mails des trois premières adresses

Vient ensuite la longueur des mails. Dans le rapport des statistiques, trois colonnes sont présentées comme suit : une colonne avec les adresses, une colonne avec le nombre de mail envoyés et une colonne avec la longueur moyenne de chaque adresse (exprimée en nombre de lignes).

180	Longueur des mails		
181	Adresse	Longueur moyenne	Nombre de mails
182	XXX@yahoo.fr	59.3	327
183	XXX@umontpellier.fr	36.7	132
184	XXX@univ-montp2.fr	18.7	375

FIGURE 9 – Exemple de longueur moyenne des mails

Pour finir, on retrouve également le pourcentage de pièces jointes par rapport à l'ensemble des mails du corpus (ou des mails totaux pour un rapport complet).

347	Pourcentage de pièces jointes
348	32.47%

FIGURE 10 – Exemple de pourcentage pour les pièces jointes

Une fois toutes les données calculées, le fichier est écrit au format `CSV`, pour qu'il puisse être ouvert dans un tableur et facilement exploiter les données.
Avec en moyenne 150 adresses, les rapports complets de statistiques contiennent environ 350 lignes d'informations. Pour les rapports sur un coprus, leur taille dépend fortement de la taille de ce dernier, mais le fichier excède rarement les 30 lignes pour un corpus issu d'un fil de messages.

5 Résultats

5.1 Performances

Afin d'analyser la puissance de notre programme, nous avons effectué des tests sur plusieurs fichiers bruts de plusieurs centaines de `Mo`.
Pour chaque fichier nous avons calculé le temps mis par chaque fonction pour se terminer, voici les résultats :

```
   Fichier 2015 : 2,8Mo
Temps découpage : 0,06s
Temps nettoyage : 2,29s
Temps threader : 2,35s
Temps création rapport : 0,04s


   Fichier 2016 : 625Mo
Temps découpage : 13,83s
Temps nettoyage : 107,06s
Temps threader : 129,82s
Temps création rapport : 29,77s


   Fichier 2018 : 676
Temps découpage : 14,93s
Temps nettoyage : 117,02s
Temps threader : 148,72s
Temps création rapport : 31,40s

...
```

Cela nous a permis de générer un graphique et de calculer la puissance de calcul de notre programme :

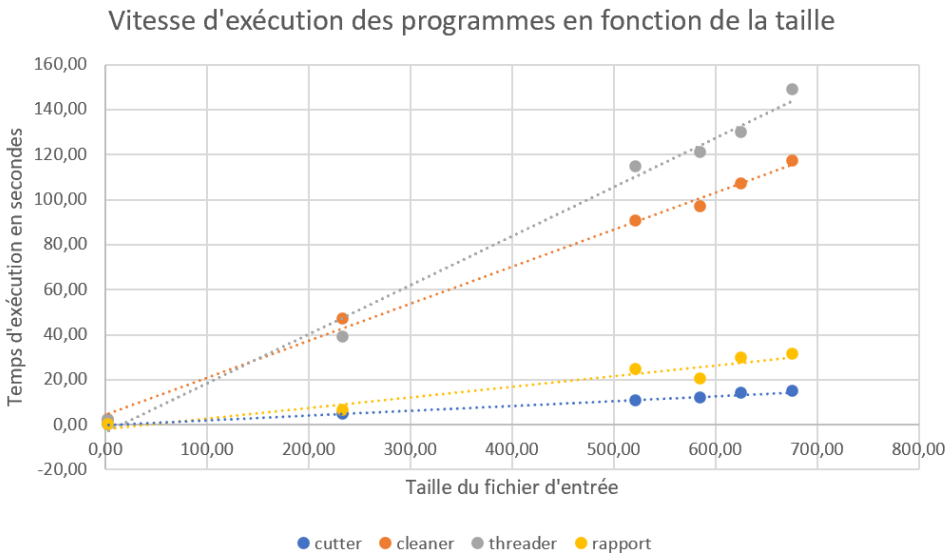


FIGURE 11 – Temps de calcul par fonction et par taille du fichier d'entrée

Nous pouvons conclure que notre programme a une vitesse moyenne pour l'analyse de 4,9Mo/s et une vitesse de calcul moyenne pour la création des fils de 4,4Mo/s.

5.2 Etude des résultats

Notre programme se base sur de longs et lourds fichiers de mails bruts exportés de `Thunderbird`. Ces derniers regorgeant d'informations, ils ne sont pas pour autant exploitables de manière brute par un humain. C'est pour cela que nous avons trois étapes : le découpage, le nettoyage et la création des fils qui retire les informations inutiles puis recrée les discussions. Pour des raisons de confidentialité du projet, les mails ont été anonymisés. Les exemples ci-après auront donc des parties grisées.

Une fois prêts à l'emploi, les mails passeront dans les fonctions de filtre et de traitement répondant aux choix de l'utilisateur afin de pouvoir exploiter les informations qu'ils conservent. Nous obtenons ainsi des résultats intéressants pour étudier l'interdisciplinarité et il est possible d'exporter ces derniers dans des fichiers `CSV` pour les ré-exploiter ailleurs.

[redacted]@yahoo.fr a écrit le 08 Décembre 2015 à 12h05m27s
| [redacted]@msh-m.org a répondu 1 jour 20 heures 27 minutes 26 secondes plus tard
| | [redacted]@univ-montp3.fr a répondu 3 jours 3 heures 12 minutes 12 secondes plus tard
| | | [redacted]@yahoo.fr a répondu 4 heures 26 minutes 05 secondes plus tard
| | | | [redacted]@yahoo.fr a répondu 17 heures 46 minutes 24 secondes plus tard
| | | | | [redacted]@univ-montp3.fr a répondu 9 heures 16 minutes 02 secondes plus tard

FIGURE 12 – Exemple de rapport sur un fil de discussion, avec pour chaque mail le temps mis pour répondre

sep=;
Années;Pourcentage
2015;100.0%

Adresses;Pourcentage
[redacted]@msh-m.org;17.39%
[redacted]@msh-m.org;17.39%
[redacted]@ies.univ-montp2.fr;13.04%
[redacted]@yahoo.fr;13.04%
[redacted]@msh-m.org;8.7%
[redacted]@orange.fr;8.7%
[redacted]@univ-montp3.fr;8.7%
[redacted]@msh-m.org;4.35%
[redacted]@hotmail.com;4.35%
[redacted]@gmail.com;4.35%

Mois;Pourcentage
Decembre;47.83%
Novembre;26.09%
Janvier;13.04%
Mars;13.04%

Semaine;Pourcentage
Jeudi;26.09%
Mardi;21.74%
Lundi;17.39%
Vendredi;13.04%
Mercredi;8.7%
Dimanche;8.7%
Samedi;4.35%

Longueur des mails
Adresse;Longueur moyenne;Nombre de mails
[redacted]@msh-m.org;9.1;4
[redacted]@msh-m.org;7.5;4
[redacted]@ies.univ-montp2.fr;27.3;3
[redacted]@yahoo.fr;10.5;3
[redacted]@orange.fr;21.0;2
[redacted]@msh-m.org;19.0;2
[redacted]@univ-montp3.fr;10.5;2
[redacted]@msh-m.org;10;1
[redacted]@gmail.com;8;1
[redacted]@hotmail.com;0;1

Pourcentage de pièces jointes
65.22%

FIGURE 13 – Exemple de fichier rapport sur un corpus de mail

6 Gestion du projet

6.1 Organisation

Afin de mener à bien le projet, nous nous sommes réunis grâce à `Discord` pour communiquer entre nous quatre sur l'avancée du projet. Chaque semaine, nous envoyions un mail aux encadrants afin de leur faire un point sur notre progression. Certaines étaient plus productives que d'autres, notamment à cause des examens que nous avions. Nous faisions aussi des réunions sur `Zoom` avec les chercheurs pour discuter des fonctionnalités, échanger sur des questions qu'ils nous posaient ou que nous leur posions et aussi pour faire des démonstrations des différents scripts réalisés. Au sein du groupe, nous répartissions les tâches régulièrement en fonction des avancées et des objectifs à atteindre pour la prochaine réunion. Chacun travaillait de son côté, parfois ensemble sur des sessions et nous mettions en commun les progrès avant chaque mail hebdomadaire.

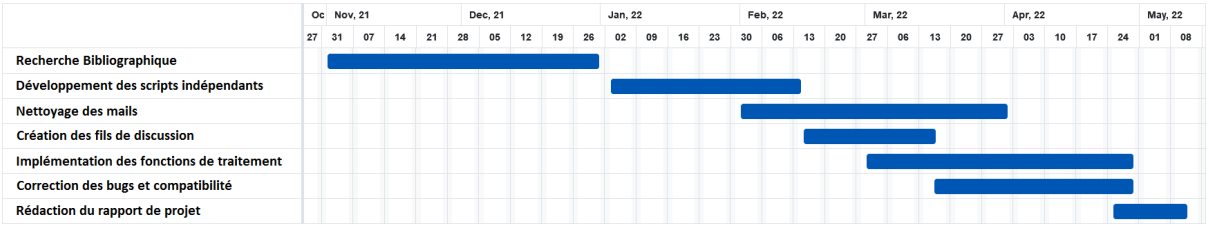


FIGURE 14 – Diagramme de Gantt pour la planification des tâches et l'organisation du projet

La progression au fil des mois s'est effectuée comme montrée ci-dessus, tout d'abord en 2021 avec la recherche bibliographique sur le sujet de l'interdisciplinarité. Une fois nos examens en janvier passés, nous avons commencé à rédiger des scripts indépendants réalisant différentes tâches qui pourraient nous être utiles par la suite : calcul de la longueur d'un mail, détection de présence de pièce jointe, calcul du temps de réponse...Début février nous avons récupéré l'ordinateur HUT contenant les mails à exploiter pour notre recherche. Ainsi nous avons pu commencer à implémenter des fonctions pour nettoyer les données brutes fournies par `Thunderbird` et ne garder que les métadonnées utiles. Par la suite, nous avons pu recréer les fils de discussion pour regrouper les mails ayant le même objet dans des dossiers. Ensuite nous avons réutilisé les scripts faits en début d'année pour créer les fonctions traitant les mails nettoyés. Enfin nous avons corrigé les bugs persistants, assemblé le tout dans un programme fonctionnel et rédigé ce rapport.

6.2 Changements au cours du projet

Sur le projet, nous n'avons pas eu de changement drastique à faire. Néanmoins, il nous a fallu nous adapter aux situations et repenser notre vision du programme. Au début du développement, nous avons commencé à utiliser la base de données des mails de `Enron` pour créer nos algorithmes d'analyse. Or le formatage était relativement daté, nous avons donc décidé d'utiliser les mails que nous échangions avec nos encadrants. Nous savions tout de même qu'il s'agissait d'une solution temporaire avant de recevoir la base de données de *HUT*. Une fois les mails officiels à notre disposition, nous avons modifié

une dernière fois nos fonctions de nettoyage pour qu'elles s'adaptent au formatage de `Thunderbird`. Une fois face à la taille imposante des conversations à notre disposition, nous avons compris qu'il fallait optimiser nos fonctions pour réduire le temps d'exécution. Ainsi, au lieu de passer en paramètre un tableau contenant les mails entiers, nous avons allégé la mémoire en passant uniquement un tableau contenant les id des mails. Enfin, une fois le programme fonctionnel, nous avons refactorisé le code afin de le simplifier et d'améliorer les performances.

7 Bilan

L'étendue des différents objectifs que nous avons accomplis a été rendue possible grâce à plusieurs étapes successives.
Tout d'abord, la synthèse bibliographique, écrite dans les mois de novembre à décembre, nous a permis de cerner le sujet dans sa globalité et de comprendre le concept d'interdisciplinarité. C'est à partir de cette maîtrise de la partie théorique que nous avons pu envisager la façon dont nous répondrions au cahier des charges.
Ainsi, nous nous sommes concentrés sur trois objectifs principaux qui regroupaient différentes sous-tâches. Premièrement, l'exploitation des données brutes de mails afin de constituer des statistiques exploitables. Ces dernières regroupaient entre autres la longueur des mails, leur fréquence d'envoi, le temps de réponse moyen...
Deuxièmement, nous avons pris en compte la nécessité de portabilité de notre outil. Nous avons ainsi implémenté un programme léger capable de traiter de très vastes corpus de mails. Troisièmement, nous avons fait en sorte que l'utilisateur puisse exporter les données traitées vers un format texte utilisable dans d'autres contextes.
Si nous avions eu plus de temps à notre disposition, nous aurions pu mettre en place plus de fonctionnalités. D'une part la compatibilité des données avec d'autres boîtes mails que `Thunderbird`. D'autre part, nous aurions voulu réaliser une interface graphique afin de faciliter l'expérience utilisateur. Enfin, il aurait été possible d'optimiser les fonctions afin de minimiser le temps pris par le nettoyage sur des fichiers de mails extrêmement volumineux.
Malgré ces quelques points à approfondir, les résultats que nous avons fournis coïncident avec les attentes du cahier des charges. En arrivant au terme de ce projet, nous avons acquis de nouvelles capacités de gestion de groupe et d'objectifs, ou encore de recherche qui nous seront bénéfiques à l'avenir.

8 Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué à l'aboutissement de ce projet de programmation et qui nous ont accompagnées tout au long de sa réalisation.

Nous voudrions dans un premier temps remercier, Mme. Anne LAURENT, notre encadrante, pour nous avoir acceptés dans cette mission et nous avoir accompagnés durant ces six mois de projet.

Nous remercions également toute l'équipe de chercheurs et enseignants de l'Université

de Montpellier avec qui nous avons collaboré pour mener à bien ce projet : Mme. Thérèse LIBOUREL, Mme. Déborah NOURRIT et M. Guillaume ALEVEQUE. Nous avons travaillé avec plaisir sur ce projet et espérons que le résultat sera à la hauteur de vos attentes.

Nous tenons à témoigner toute notre reconnaissance aux représentants du projet HUT pour nous avoir fait confiance afin de réaliser ce sous-projet et nous avoir permis de collaborer à leurs côtés dans un projet de si grande ampleur.

Enfin, nous remercions Mme. BAERT pour nous avoir mis en relation avec Mme. LAURENT afin de réaliser ce projet de programmation.

9 Bibliographie

[Mor94] Edgar MORIN. *Sur l'interdisciplinarité*. 1994. URL : https://ciret-transdisciplinarity.org/bulletin/b2c2.php.

[Köl04] Micha KÖLLERWIRTH. *Enron Email Dataset*. 2004. URL : https://www.cs.cmu.edu/~enron/.

[Tan13] Guanting TANG. *Tasks, common techniques, and tools*. 2013. URL : https://www.researchgate.net/publication/257482113_Email_mining_Tasks_common_techniques_and_tools.

[Gra14] Marie De GRAZIA. *Thunderbird-email-parser*. 2014. URL : https://github.com/mdegrazia/Thunderbird-email-parser.

[CNR15] CNRTL. *INTERDISCIPLINAIRE*. 2015. URL : https://www.cnrtl.fr/definition/interdisciplinarit%5C%C3%5C%A9.

[Köl15] Micha KÖLLERWIRTH. *UTF-8 encoding table and Unicode characters*. 2015. URL : https://www.utf8-chartable.de/.

[Mar15] Nathalie Reveyzaz MARIE-FRANÇOISE OLIVIER. *Une définition de l'interdisciplinarité*. 2015. URL : https://www.reseau-canope.fr/notice/une-definition-de-linterdisciplinarite.html.

[Ale22] Laurent Libourel Nourrit ALEVEQUE. *JIMS*. 2022.

[W3s22] W3SCHOOLS. *Python File Open*. 2022. URL : https://www.w3schools.com/python/python_file_handling.asp.

10 Annexe

10.1 Manuel d'utilisation

Comment l'utiliser ?

Il vous faut avoir Python3 d'installé sur votre machine !

1. Tout d'abord téléchargez le package sur notre Github.
2. Depuis Thunderbird exportez vos mails dans des fichiers.
   — Pour ce faire, sélectionnez les dossiers dans l'arborescence à gauche (ou Boîte de réception). Clic droit, Enregistrez-sous, Format texte brut.
   — Attention, plus il y a de mails dans un fichier plus il sera long à analyser. Nous vous conseillons d'exporter chaque année séparement voire chaque mois en créant de nouveaux dossiers.
3. Si vous êtes sur `Windows` vous pouvez lancer le fichier "launcher.bat" puis suivre les insctructions.
4. Si vous êtes sur `MacOS` ou `Linux` vous pouvez dans, l'invite de commande (terminal), entrer la commande : python3 main.py depuis le répertoire du programme.
5. Comme indiqué par le programme, glissez les mails dans le dossier __MAIL_DEPOT__ et tapez "ok".
6. Laissez le programme analyser vos mails puis suivez les instructions à l'écran.

Les possibilités :

Pour chaque fonction il est possible au préalable de filtrer les mails par adresse, date (précise, avant, après, entre), mot contenu, présence de pièces jointes ou bien d'utiliser les mails regroupés par fils de discussion

Sur un corpus de mails vous pouvez :

— Calculer le temps de réponse
— Calculer la longueur de moyenne des mails par adresses
— Générer des rapports précisant la part d'adresses par corpus, le nombre de mails par mois ou jour de la semaine et le pourcentage de pièces jointes
— Exporter vos données précédemment calculées

10.2 Code

Code de la fonction de découpage du parser simplifiée

```
def cutter() :
for bigFile in __MAIL_DEPOT__": # Pour chaque fichiers bruts
    os.mkdir("tmp/" + str(bigFile)) # On crée le dossier
    with open("__MAIL_DEPOT__" + slash + bigFile) as currentBigFile:
    # On ouvre le fichier
```

```
        mailCount = 0 #On comtpe le nombre de mails
        for line in currentBigFile:
            if(line.startswith("From - ")):
            # Si la ligne commence par "From -" alors c'est un nouveau mail
                try:
                    mailFile.close()
                except:
                    pass
                mailCount += 1 # On crée le nouveau fichier
                mailFile = open("tmp/" + str(bigFile) + str(mailCount), "w")
            mailFile.write(line)
        mailFile.close()
    currentBigFile.close()
```

Code de la fonction de nettoyage du parser simplifiée

```
def cleaner() :
#Début du nettoyage des fichiers
for dossier in os.listdir("tmp"): # Pour chaque dossier
    for file in os.listdir("tmp" + slash + dossier): #Pour chaque fichier
        dateEnvoi = "__Date__ " # On prépare les variables
        expediteur = "__From__ "
        destinataire = "__To__ "
        objet = "__Object__ "
        pieceJ = "__PJ__ "
        content = "__CONTENT__ "
        # On ouvre le fichier
        currentFile = open("tmp" + slash + dossier + slash + file).readlines()
        for i in range(len(currentFile)): # Pour chaque ligne
            ligne = currentFile[i]
            for key, value in utf8Dico.items(): # Réencodage UTF-8
                if key in ligne:
                    ligne = ligne.replace(key, value)
            # Récupération de la date du mail
            if (ligne.startswith('Date:')) and (dateEnvoi == "__Date__ "):
                dateEnvoi += dateTranslation(ligne[6:])
            # Récupération du sujet
            if (ligne.startswith('Subject:')) and (objet == "__Object__ "):
                objTMP = ligne[9:]
            # ...
            # Récupération de l'expéditeur
            if (ligne.startswith('From:')) and (expediteur == "__From__ "):
                expediteur = expediteur + ligne[ligne.find("<") +
                1:ligne.find(">")]
            # Récupération du destinataire
            if (ligne.startswith("To:")):
                desti = []
```

```
            destiJ.append(ligne[4:])
        # ...
        # Récupération des pièces jointes
        if (("name=\"") in ligne):
            pieceJ = pieceJ + ligne.split('=')[1].strip("\n") + " - "

        # Récupération du contenu
        if ligne.startswith("Content-Type: text/plain;"):
            j = i + 1
            # ...
                # Suppression des signatures
                elif (("PENSEZ A L'ENVIRONNEMENT AVANT D'IMPRIMER
                CE MESSAGE !"
                in currentFile[j])):
                    j += 21
                elif (currentFile[j].startswith("### NOTICE LEGALE ###")):
                    j += 11
                else:
                        content += currentFile[j][:-1]

# On met les variables dans un tableau
mailArray = []
mailArray.append(dateEnvoi + "\n")
mailArray.append(expediteur + "\n")
mailArray.append(destinataire + "\n")
mailArray.append(objet + "\n")
mailArray.append(pieceJ + "\n")
mailArray.append(newContent)
# On ouvre le fichier en écriture
newMail = open("tmp" + slash + dossier + slash + file, "w")
for line in mailArray:
    # On écrit les nouvelles variables dans le fichier
    newMail.write(line)
newMail.close()
```

Code de la fonction `threader` simplifiée

```
def threader():
for dossier in tmp: # Pour chaque dossier de mails
    for file in (tmp + dossier): # Pour chaque mail
        currentFile = open("tmp" + dossier + file, "r", encoding="utf-8")
        # Ouverture du fichier
        objet = currentFile[3][11:] # récupération de l'objet
        try:
            os.mkdir("threads" + slash + str(objet.replace("\n","")))
            # On tente de créer le dossier s'il n'existe pas déjà
        except:
            pass # le dossier existe
        fullDateTime = currentFile[0][9:] # Récupération de la date
        date = fullDateTime[0]
        time = fullDateTime[1]
        newFile = open(dateTime, "w", encoding="utf-8")
        # Création d'un nouveau fichier ayant sa date pour nom
        newFile.write(line) # écriture du nouveau fichier
```

Code de la fonction `getTimesFolder` simplifiée

```
#Renvoie un tableau de tableaux sous la forme [@,time]
def getTimesFolder(folder) :
    resTab = []
    #Pour chaque mail du dossier ...
    for mail in os.listdir(folder):
        tempTab = []
        currentFile = open(folder + str(mail)).readlines()
        #... on récupère son expéditeur et sa date ...
        sSender = currentFile[1][9:-1]
        sDate = currentFile[0][9:-1]
        dtDate = datetime.strptime(sDate, "%d/%m/%Y %H:%M:%S")
        #... et on l'ajoute au tableau final !
        tempTab.append(sSender)
        tempTab.append(dtDate)
        resTab.append(tempTab)
    return sorted(resTab, key=itemgetter(1))
```

Code de la fonction de création du rapport simplifiée

```
def rapport_total(corpus):
    #On convertit le corpus pour le rendre compatible avec les fonctions
    IDs = [mail for mail in (corpus.replace("-",slash))]
    nb_total = total_mail()
    addrs = all_adr(IDs) #toutes les adresses
    ans = all_dates(IDs) #toutes les dates
    nb_corpus = len(IDs) #taille du corpus
    #Création du fichier du rapport
    f = open("rapport_corpus.csv", "w")
    #Rapport pour les années
    tab = {}
    for an in ans:
        nb_an = nb_mail_annee(an, IDs)
        tab[an] = round(100*nb_an/nb_corpus,2)
    #Rapport pour les adresses
    tab = {}
    for addr in addrs:
        nb_adr = nb_mail_adresse(addr, [])
        tab[addr] = round(100*nb_adr/nb_corpus,2)
    #Rapport pour les mois
    tab = {}
    tab_mois = nb_mail_mois(IDs)
    for i in range(len(tab_mois)):
        if tab_mois[i] != 0:
            tab[num_to_mois(i+1)] = round(100*tab_mois[i]/sum(tab_mois),2)
    #Rapport pour les jours de la semaine
    tab = {}
    tab_semaine = nb_mail_semaine(IDs)
    for i in range(len(tab_semaine)):
        if tab_semaine[i] != 0:
            tab[num_to_jour(i+1)] = round(100*tab_semaine[i]/sum(tab_semaine),2)
    #Rapport pour la longueur
    tab_adresses = longueur(IDs)
    #Rapport pour les pièces jointes
    nb_pj = nb_mail_pj(IDs) #nombre de mails avec PJ

    #écriture dans le fichier
    ...
```

10.3 Sources

— Logo HUT, "cropped-logoHUTrond-blc-2048x1325.png", 2021, via hut-occitanie.eu

— Logo UM, "LOGO_original_RVB_WEB-1.png", 2020, via umontpellier.fr

— Logo FdS, "Logo_FDS_Quadri_Grand_2392x2950.png", 2016,
via sciences.edu.umontpellier.fr