Détection automatique de liens entre messages à l'aide de chaînes lexicales

Agathe Mollé

Introduction

Nous nous intéressons à la tâche de détection automatique de liens entre e-mails. Cette tâche consiste à déterminer la structure d'un thread, à savoir qui répond à qui.

Différentes techniques permettent de représenter un message, nous avons ici opté pour l'utilisation des chaînes lexicales. Ces chaînes sont des séquences de mots sémantiquement reliés au sein d'un texte.

Les chaînes lexicales peuvent être établies à l'aide de ressources linguistiques (comme un thésaurus) ou bien de mesures distributionnelles. Ce sont ces dernières que nous allons utiliser. En effet, nous nous appuyons sur un réseau de collocation pour établir nos chaînes lexicales.

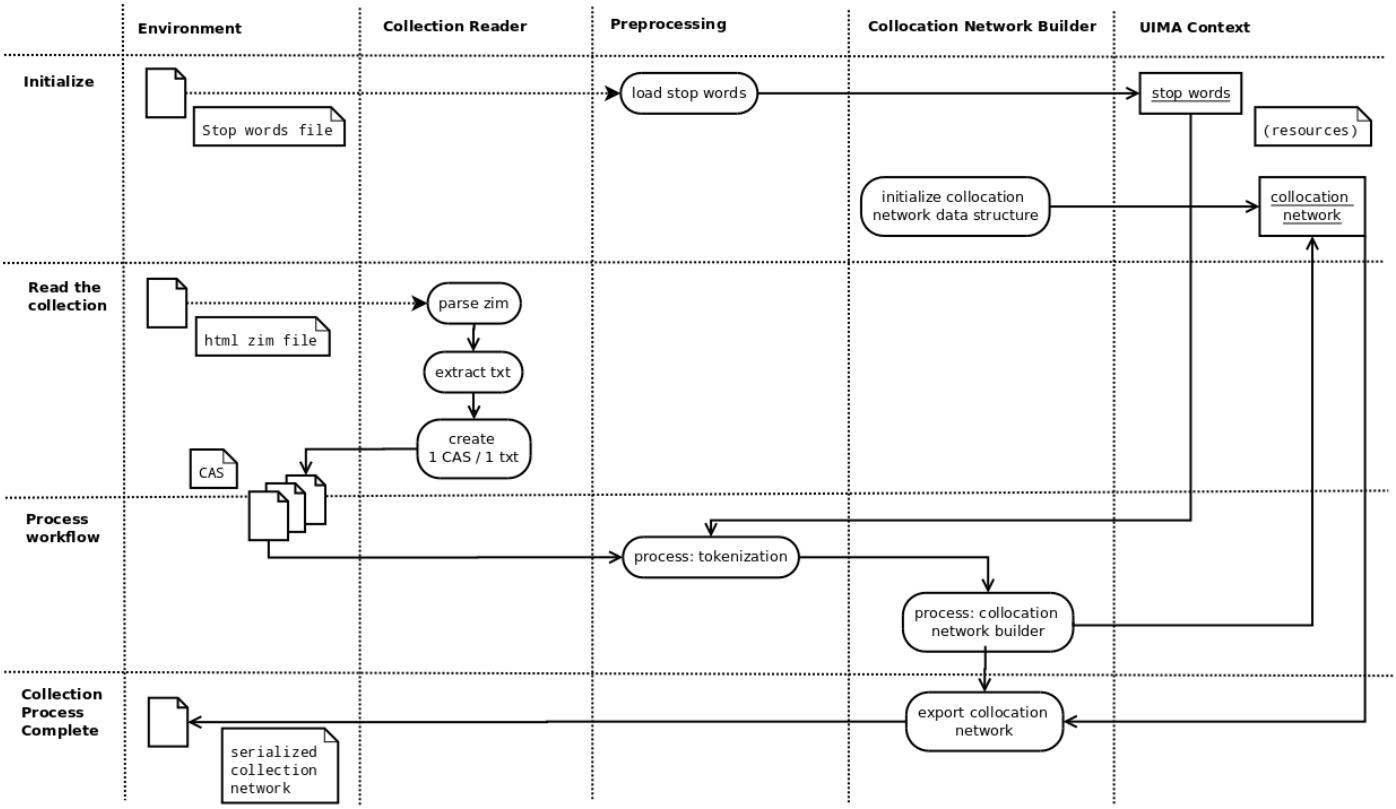
Cette tâche se réalise donc en trois temps. D'abord, nous allons créer un réseau de collocation à partir d'un corpus dédié, et traitant du même sujet. Les deux étapes suivantes utilisent par la suite un corpus composé de mails : l'extraction des chaînes lexicales puis la détection des liens entre messages.

1 Conception

Voici les diagrammes d'activité représentant les étapes de cette tâche de détection de liens. Ce travail a été découpé en deux workflows.

1.1 Construction du réseau de collocation

Le premier workflow consiste à établir un réseau de collocation à partir d'un corpus Zim, différent de celui avec lequel nous manipulons des mails. Le diagramme d'activité était préalablement fourni. Il s'appuie sur deux AE : le premier tokénise le texte à l'aide de la ressource contenant les mots-outils, puis le second recense toutes les paires de collocations, et les stocke dans une nouvelle ressource.



1.2 Détection des liens entre messages à l'aide des chaînes lexicales

Le second worflow manipule un corpus de mails, traitant du même sujet que le premier corpus qui nous a permis d'établir le réseau de collocation.

Le déroulement est le suivant : dans un premier temps, on extrait les messages un à un, créant ainsi un CAS par message. Ensuite, on les tokénise, cette étape nécessitant toujours la ressource de mots-outils. Les messages segmentés, on s'appuie sur le réseau de collocation afin de construire une nouvelle ressources contenant les chaînes lexicales. Enfin, on utilise cette dernière pour établir les liens entre messages (cette étape nécessite également des infos sur les threads, obtenues à l'aide d'une autre ressource).

Ce workflow aurait pu être scindé en deux :

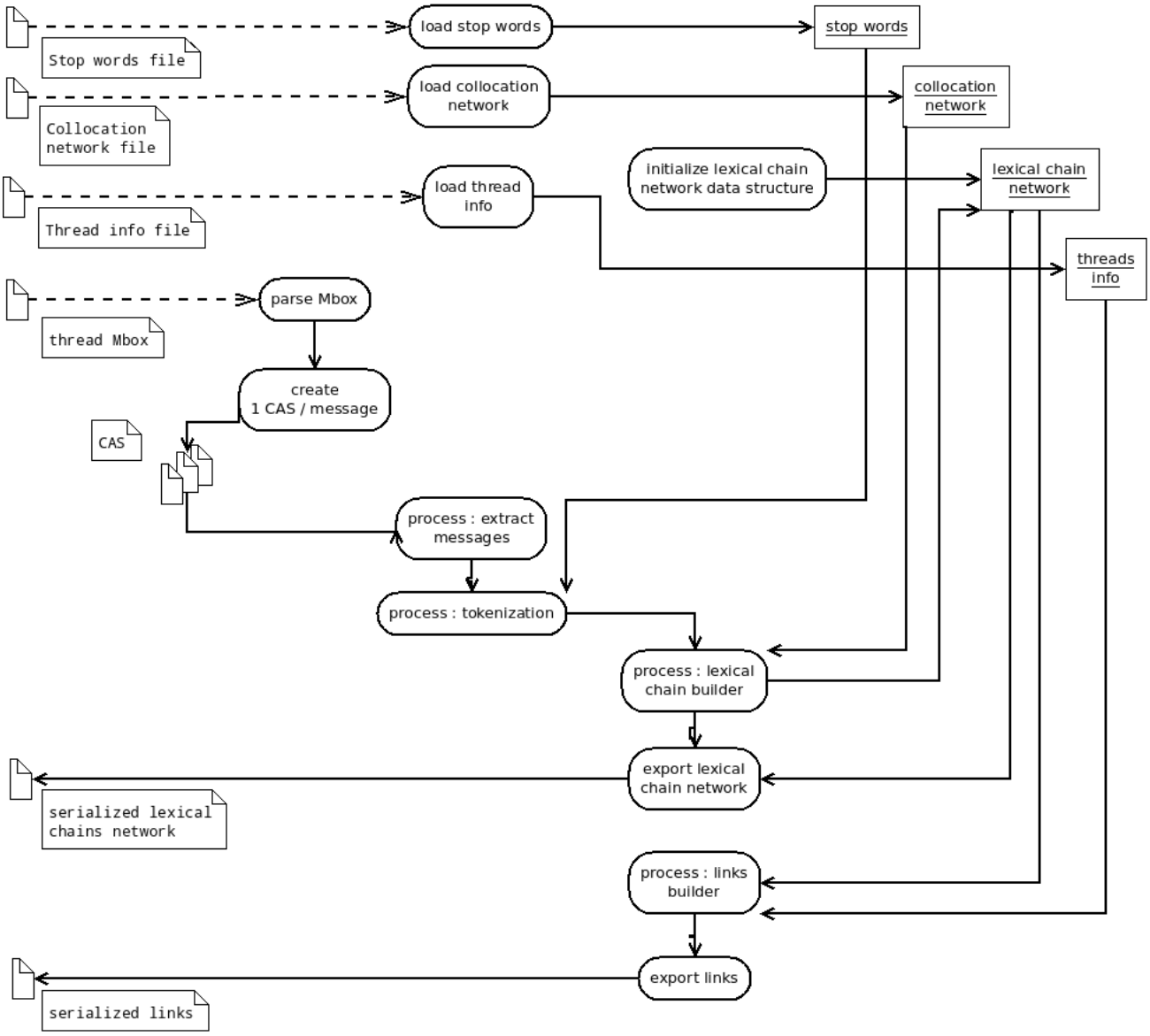
— un LexicalChainer qui utilise les trois premiers AE ainsi que les ressources de mots-outils et de collocations. Il génère alors une ressource de chaînes lexicales.

— un Linker qui extrait les messages puis utilise la ressource créée précedemment ainsi que les informations sur les threads pour détecter les liens.

Comme mentionné ci-dessus, l'étape d'extraction de message génère un CAS par message, ce qui entraîne une perte d'informations. En effet, on a besoin de connaître les autres messages du thread courant afin de pouvoir éventuellement les relier avec le message en cours d'analyse. Pour résoudre ce problème, nous avons opté pour une ressource contenant les informations nécessaires (à savoir le thread associé au message, ainsi que les autres messages associés à ce même thread). D'autres approches étaient envisageables, telles que la fusion de CAS, ou encore la création d'annotations gardant les informations requises.

2 Implémentation

Nous allons détailler les différents composants évoqués dans la présentation des workflows.



2.1 Ressources

2.1.1 Stop Words

Cette ressource était fournie, et est uniquement utilisée lors de la segmentation, afin d'éliminer les mots-outils.

2.1.2 Réseau de collocation

Cette ressource est initialisée puis remplie dans le premier workflow. Elle est constituée de paires de mots associées à leur score de collocation. Les paires ayant un score de 1.0, c'est-à-dire n'apparaissant qu'une seule fois dans le corpus fourni, ne sont pas conservées, car peu utiles pour la suite. Elle est ensuite appelée pour la tâche d'extraction de chaînes lexicales.

2.1.3 Chaînes Lexicales

Cette ressource est initialisée et remplie dans le second workflow. Elle contient tous les identifiants des messages, chacun étant associé à l'ensemble des chaînes lexicales le représentant.

2.1.4 Information sur les threads

Cette ressource s'appuie sur les informations disponibles dans un fichier thread digest, détaillant la structure des threads (qui répond à qui ?). Cette ressource stocke chaque thread avec les messages qu'il contient, ainsi que chaque message avec le thread auquel il est associé. Cette redondance permet par la suite d'accéder plus rapidement aux informations nécessaires.

2.2 Analysis Engines

2.2.1 WordSegmenterAE

Cet AE était fourni. Il a été adapté afin de ne pas prendre en compte les mots composés d'un seul caractère, qui alourdissaient les traitements ultérieurs.

2.2.2 CollocationNetworkBuilderAE

Cet AE a été préalablement réalisé lors d'un TP et se base sur un principe de fenêtre glissante. Il a été légèrement modifié pour un souci de temps d'exécution (le corpus étant ici bien plus imposant !).

2.2.3 CollocationNetworkConsumerAE

Ce second AE a également été conservé. Il aurait également pu être fusionné avec le précédent.

2.2.4 MBoxMessageParserAE

Cet AE permet de découper le corpus en messages. Un type Message a d'ailleurs été rajouté afin de garder une trace de certaines informations. Ainsi, on rajoute une vue au CAS contenant le corps texte du message. On y ajoute un paramètre stockant l'ID du message, nécessaire pour la suite.

Dans le second Workflow, cet AE est lancé avant celui de segmentation : ainsi seul le corps du message est tokénisé.

2.2.5 LexicalChainerAE

Cet AE s'appuie sur les travaux de Marathe & Hirst pour établir les chaînes lexicales. A ceci près que les chaînes ne sont pas mergées : si plusieurs chaînes obtiennent de bons scores, on n'ajoute le candidat qu'à la meilleure. Les scores de similarité permettant de déterminer si un mot appartient à une chaîne lexicale étant calculés par moyenne, il est peu probable d'obtenir plusieurs chaînes avec le score optimal.

La méthode de Marathe & Hirst semble s'appuyer sur tout le texte pour déterminer les chaînes lexicales, donc pour un mot donné, toutes les chaînes déjà calculées sont candidates. Nous avons limité le problème en ne gardant pour candidats que les 20 dernières chaînes.

Une fois le message parcouru, nous agrémentons la ressource contenant les chaînes lexicales avec ces nouvelles informations (l'ID du message + ses chaînes lexicales), puis ce n'est qu'une fois tout le corpus parcouru que nous sauvegardons la ressource.

2.2.6 LinkerAE

Cet AE utilise la ressource préalablement obtenue. Il utilise également les informations du thread. Comme nous n'utilisons qu'un workflow pour effectuer les deux dernières tâches, la détection de liens est effectuée message par message, sans connaître à l'avance les chaînes

lexicales des messages suivants. Cela n'est pas très important puisque les messages sont triés par date. Or, nous recherchons à quel message répond celui que l'on est en train de lire, donc nous n'avons besoin que des messages précédents. Si l'on voulait détecter les liens sans prendre en compte les dates, il faudrait alors séparer le workflow en deux. Cela entraînerait par contre une double extraction de messages ainsi que segmentation.

Le principe de détection de liens est le suivant : pour un message donné, on regarde les autres messages de son thread et on choisit celui qui est le plus proche. La similarité entre deux message est calculée d'après la moyenne des similarités entre les chaînes lexicales les composant (coefficient de Dice). Celles-ci sont calculées d'après les mots en commun.

3 Utilisation

Les corpus étant lourds, ils ne sont pas fournis avec le projet. Il faut donc aller changer manuellement les adresses appelées par les CollectionReader.

Le projet peut être utilisé via Eclipse, en lançant dans un premier temps le workflow linkInterMessageDetector.wf.CollocationNetworkBuilderWF.java. puis ensuite le workflow linkInterMessageDetector.wf.MboxWF.java.

Il peut également être utilisé via Maven, à l'aide des commandes suivantes :

```
mvn package

java \
  -cp target/linkInterMessageDetector-0.0.1-SNAPSHOT-jar-with-dependencies.jar \
   linkInterMessageDetector.wf.CollocationNetworkBuilderWF

java \
  -cp target/linkInterMessageDetector-0.0.1-SNAPSHOT-jar-with-dependencies.jar \
  linkInterMessageDetector.wf.MboxWF
```

4 Evaluation

Ce travail avait pour principal objectif une manipulation des outils UIMA, UIMAFit, etc. Nous n'avons donc pas fait diverses expériences afin d'améliorer la méthode proposée initialement.

Il est toutefois possible d'évaluer les résultats obtenus à l'aide d'un script Perl fourni. La commande est la suivante :

```
perl -CDS eval/compare-sets-inTermsOf-PrecisionRecallFScore.pl -r eval/in-reply-to-list.
```

Dans un souci de gain de temps, les tests effectués l'ont été sur une version allégée du corpus, à savoir 10Mo au lieu des 160Mo d'origine. Les résultats obtenus ne sont donc pas représentatifs : 46% de précision contre seulement 3% de rappel...

5 Discussion

Le principal problème lié à ce travail, comme dans quasiment toutes les tâches de Traitement Automatique du Langage, est la qualité du corpus. En effet, celui-ci souffre de problèmes d'encodages et de manques d'information. Par ailleurs, certaines données cryptées ou mal extraites alourdissent énormément le corpus et rajoutent un bruit non négligeable pour le reste des opérations.

Cependant, l'objectif n'était pas d'obtenir les meilleurs scores pour cette tâche, nous nous intéressions seulement à une approche basique, s'appuyant sur des mesures distributionnelles. Les méthodes de comparaison sont simples et les paramètres ne sont pas optimisés. L'intérêt de l'exercice résidait essentiellement sur l'utilisation de UIMA. Cette approche assez complète nécessitant plusieurs workflows ainsi que l'utilisation de ressources était donc intéressante. La modularité et la possibilité d'intégration de composants existants sont des points forts qui ont été soulignés par ce sujet, ainsi que la gestion des ressources. Il a également permis de souligner certains inconvénients, tels que la difficulté à fusionner des CAS, ou encore la complexité à appréhender le framework.