André GIANG

Rapport de stage

Un moteur d'inférences pour la correction à partir de cas de phrases en français

Corrector

Du 9 avril au 6 juin 2018

Responsable de stage :
Bruno GUILLAUME
Yves LEPAGE
Jean LIEBER
Emmanuel NAUER

Tuteur universitaire :
Mathieu CONSTANT

Licence 3 informatique

Table des matières

Avant propos

Ce rapport est le résultat d'un stage de recherche en collaboration avec deux autres étudiants d'une durée de deux mois dans le cadre de la licence 3 Informatique à la Faculté des Sciences et Technologies.
J'ai effectué mon stage au sein du Loria, laboratoire lorrain de recherche en informatique et ses applications à Nancy.
Le sujet de recherche de base était l'étude d'une approche afin de proposer des corrections de phrases en français à l'aide d'un ensemble de correction de phrases composé par des couples (phrase incorrecte, phrase corrigée) en utilisant le principe du raisonnement à partir de cas.
Ce stage était prévu initialement pour des étudiants en Master 2 mais a été redéfini afin de l'adapter aux étudiants de Licence 3. Le stage s'est donc décomposé en trois parties distinctes :

- La conception et le développement d'une interface web et sa base de données.
- La recherche et la création de script permettant l'acquisition automatique de cas
- La recherche et le développement d'un moteur de correction de phrases en français

C'est cette troisième partie que j'ai choisi car le sujet m'intéressait d'avantage et me permettait de mettre en pratique les connaissances vues durant mon parcours.

Le stage s'est bien déroulé dans l'ensemble et le moteur de correction est fonctionnel et utilisable.

Remerciements

Mathieu CONSTANT pour ses enseignements durant la licence et sa tutelle pendant le stage, mes partenaires durant le stage Hue Nam LY et Damien LEVY pour leur aide.
Messieurs Bruno GUILLAUME et Emmanuel NAUER qui m'ont aidé à la rédaction du rapport et à la présentation.
Ainsi que Monsieur Yves LEPAGE qui m'a beaucoup aidé notamment grâce au partage de son travail.
Je tiens à remercier Monsieur Jean LIEBER qui m'a encadré et conseillé pendant toute la durée du stage.

Introduction

De nos jours, nous sommes aidés et entourés de beaucoup d'outils qui nous permettent de modifier et corriger nos fautes d'orthographe. Cependant aujourd'hui et de tout temps nous avons toujours eu de nombreuses déformations et « raccourcis » grammaticaux sans même nous en rendre compte. **La correction de phrase** est le but de ce stage.
Par exemple « *Je suis sur Nancy* » devrait être « *Je suis à Nancy* ».
Pour ce faire nous allons utiliser le raisonnement à partir de cas (RàPC) qui consiste à résoudre un problème à partir d'expériences déjà réalisés, ici nous utiliserons une base de cas qui contient un ensemble de couples (problème, solution du problème).
Voici un exemple :

- Notre phrase à corriger est « *J'aime pas les araignées.* »
- Un couple qui servira à la correction du problème est « *J'aime pas nager.* », « *Je n'aime pas nager.* »
- Notre correction à partir du cas donnée « *Je n'aime pas les araignées.* »

Le RàPC se déroule en quatre étapes :

- Remémoration : Choix d'un cas dans la base jugé ressemblant au problème à résoudre.
- Adaptation : modification du problème à résoudre en une correction à partir du cas choisi.
  Comme on peut le voir dans la figure 1.

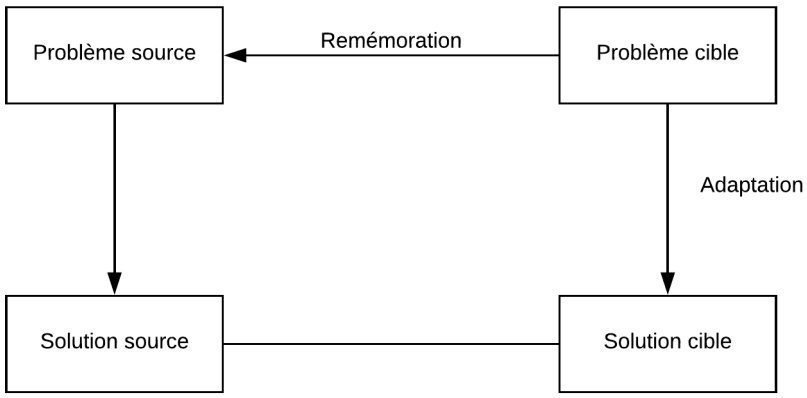

Figure 1 : RàPC

- Révision : Une fois qu'une correction est générée par le moteur, elle doit être vérifiée et validée. Cette étape est externe au moteur et sera réalisée manuellement par les administrateurs de notre site.

- L'apprentissage : Une fois l'étape de révision faite, il nous faut ajouter le nouveau cas dans la base avec les informations importantes permettant ainsi de nouvelles ou meilleures corrections.

C'est un outil qui peut donc 'apprendre' très vite si beaucoup de cas lui sont donnés. Dans notre groupe ce sera Monsieur Levy qui se chargera d'alimenter la base de cas, manuellement au départ puis en automatisant avec un script permettant la récupération de cas trouvé sur le Web. Monsieur Ly se chargera de créer la base de données ainsi qu'un site vitrine donnant l'accès au moteur. Mon rôle est donc le développement du moteur permettant la correction de phrase en français. Nous verrons plus particulièrement les deux premières étapes du RàPC : la remémoration et l'adaptation.

Dans le cadre de ce stage, nous avons reçu une maquette des travaux d'Yves Lepage, professeur à l'Université Waseda au Japon, et de ses collaborateurs. En effet ces derniers ont développés un outil qui utilise le raisonnement à partir de cas permettant de traduire une phrase d'une langue à une autre. Nous allons nous servir de ces travaux afin de développer en Python un outil capable de proposer une correction d'une phrase cible en français, en s'appuyant sur le principe du raisonnement à partir de cas.
Je vais vous présenter le déroulement du développement du moteur de correction de phrases en plusieurs parties :

- La description de l'entreprise ;
- L'explication et le choix des outils utilisés ;
- La phase d'analyse et de compréhension des travaux d'Yves Lepage.

Les parties suivantes seront les différentes étapes du développement du moteur :

- L'adaptation ;
- La remémoration.

Pour finir seront présentées les difficultés rencontrées et apports éventuels du moteur :

- Les améliorations éventuelles et futures ;
- La conclusion.

1 - Description de l’entreprise

Le stage s’est déroulé au Loria[1], Laboratoire lorrain de recherche en informatique et ses applications situé à Nancy.
Ce laboratoire est une Unité Mixte de Recherche (UMR), il est commun à plusieurs établissements le CNRS, l’université de Lorraine et l’Inria.
Créé depuis 1997 son domaine est la recherche fondamentale et appliquée en sciences informatiques.
Cet UMR possède un effectif total de plus de 400 personnes et est composée de 28 équipes répartis en 5 départements, dont 15 sont communes avec Inria. Le Loria est un des plus grands laboratoires de la région lorraine.
Notre stage s’est déroulé au sein du département 4 TALC[2] (traitement automatique des langues et des connaissances) qui comme son nom l’indique est spécialisé dans 3 domaines, la recherche et l’étude :

- *Des langues qui vont de l’analyse à sa reconnaissance et d’autres secteurs encore.*
- *Des connaissances ou plus particulièrement sa représentation et les méthodes d’apprentissage et la classification.*
- *Des documents, la reconnaissance de formes, la reconnaissance de symboles et la modélisation de l’écriture manuscrite.*

Nous étions dans l’équipe Orpailleur qui compte 12 membres permanents ainsi que de nombreux ingénieurs, doctorants et étudiants en doctorat.
Parmi les nombreux travaux qu’ils ont réalisés ils ont conçu un système du nom de Taable dont sa fonction est de résoudre des problèmes de cuisine en se basant sur un livre de recette. Dans le cadre de ce projet beaucoup de thèmes ont été abordés notamment le raisonnement à partir de cas.

1 http://www.loria.fr/fr/presentation/

2 http://www.loria.fr/fr/la-recherche/departements/4-traitement-des-langues-et-des-connaissances/

2 - Choix et analyse

Dans le cadre du stage, j'ai développé le moteur avec le langage Python[3] dans sa version 2. Ce choix s'est fait assez naturellement, en effet les travaux d'Yves Lepage étaient en Python 2.7 et dans un souci de cohérence nous avons décidé de garder le même langage de programmation. N'ayant encore jamais développé sous Python il a fallu que j'apprenne ce nouveau langage. Une fois cela décidé et fait j'ai récupéré et analysé le code de la maquette qu'Yves Lepage a développé qui utilise des analogies.

Une analogie est une association d'idées entre deux ou plusieurs objets de nature différente, voici une analogie : *a : b :: c: d* qui se lit « a est à b ce que c est à d »

Voici quelques analogies de sens avec des mots en français :
avion : voler :: voiture : rouler
faim : manger :: soif : boire

Et quelques analogies en terme de chaîne de caractères qui sont celles que je vais utiliser :
manger : manges :: ranger : ranges
J'aime pas les marshmallow. : Je n'aime pas les marshmallow. ::
J'aime pas nager dans le lac du Connemara. : Je n'aime pas nager dans le lac du Connemara.

Voici un autre exemple, cette fois-ci d'équation analogique mathématique :

6 / 9 = 8 / x
On cherche à déterminer x

En connaissant le rapport des 2 premiers termes, on arrive à trouver le rapport entre le 3ème et 4ème.

si 6 / 9 = 8 / x alors x = 12.

3 https://openclassrooms.com/courses/apprenez-a-programmer-en-python

Mon but sera donc de résoudre ces équations analogiques sur des chaînes de caractères. À partir d'un problème cible et d'un cas choisi dans la base, je dois retourner une solution au problème cible.

Problème source : solution source :: problème cible : x

Je vais maintenant vous montrer un exemple du fonctionnement de la maquette d'Yves Lepage. Nous avons une phrase à traduire de l'anglais au français qui est « *They love apples. »*

La remémoration va chercher un cas ressemblant dans la base de données

- *They love pears. Ils aiment les poires.*

Il va ensuite passé à l'adaptation où il va chercher et traduire pears par apples et poires par pommes. Il va ensuite remplacer poires par pommes :

- *Ils aiment les poires. Ils aiment les pommes.*

Et finalement retourner le résultat :

- *They love apples. Ils aiment les pommes.*

3 - Travail réalisé

Je me suis appuyé sur le code de la maquette d'Yves Lepage et l'ai modifié puis adapté selon nos besoins. Une interface en ligne de commande a été ajoutée afin d'être plus facile à comprendre et mieux indiquer les marches à suivre pour tester le moteur comme montré avec la Figure 2.

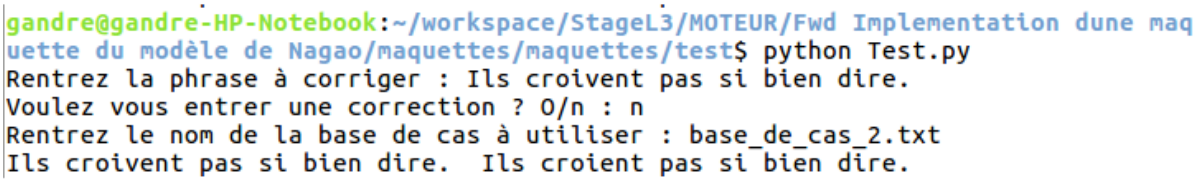

Figure 2 : Capture d'écran du test du moteur

Je vais expliquer en détails les deux premières étapes du raisonnement à partir de cas la remémoration et l'adaptation qui sont celles que j'ai développées.

3.1 – La remémoration

La remémoration est l'étape qui permet de choisir le/les meilleures cas pour la résolution d'un problème cible.
Pour un exemple comme celui-ci :

Salut je manges.

Comment retrouver le bon cas pour avoir la meilleure correction ?

On pourrait naturellement se dire, on détecte l'utilisation du pronom personnel 'je' + verbe et rechercher un cas similaire dans la base.
Ce n'est pas le cas pour notre moteur. En effet celui-ci a été conçu avec le principe le plus simple possible, en travaillant sur les chaînes de caractères sans analyse lexicale ou sémantique.

3.1.1 - Distance d’édition

Sans analyse du contexte de la phrase notre première piste a été d’utiliser la distance d’édition qui s’appuie sur l’algorithme LCS[4] afin de classer les phrases par ordre de ressemblance.
L’algorithme LCS (Longest common subsequence, plus longue sous-séquence commune) sert à trouver la plus longue chaîne de caractères commune entre deux chaînes de caractères.
Soit C1 et C2 deux chaînes de caractères avec :
C1 = « *bonjour* »
C2 = « *tondre* »
la plus longue sous-séquence commune retournée par l’algorithme LCS est « onr ».
La spécificité est que les caractères doivent être dans le même ordre mais n’ont pas besoin d’être consécutif, à l’instar d’une autre distance qui s’appelle aussi LCS (longest common substring) où les caractères doivent se suivre.

La distance d’édition que j’utilise quand à elle calcule le nombre :

- de caractères manquants
- de caractères à rajouter

entre une chaîne de référence et une deuxième chaîne à comparer. Chaque action coûte 1. Donc, plus la distance d’édition est petite, plus les chaînes de caractères se ressemblent.
Reprenons l’exemple précédent avec nos deux chaîne de caractères C1 et C2.
Voici la liste de tous les actions à effectuer pour transformer tondre en bonjour :

- Suppression du t → ondre
- Ajout du b à la place du t → bondre
- Suppression du d → bonre
- Ajout du j à la place du d → bonjre
- Ajout du o après le j → bonjore
- Ajout du u après le o → bonjoure
- Suppression du e → bonjour

Il a fallu sept étapes pour transformer C2 en C1 la distance d’édition est donc égal à sept.
distance_édition(*« bonjour », « tondre »*) = 7
Après quelques tests, je me suis rendu compte que la distance d’édition n’était pas la bonne méthode pour la remémoration. Je vais vous montrer un exemple :

4 https://fr.wikipedia.org/wiki/Plus_longue_sous-s%C3%A9quence_commune

Problème cible « *Tu peux appelé ?* »

Cas1 = (*« Tu peux demandé.», « Tu peux demander. »*)
Cas2 = (*« Il peux apeler. », « Il peux appeler. »*)

distance_édition(*« Tu peux appelé? » , « Tu peux demandé. »*) = 11
distance_édition(*« Tu peux appelé? », « Il peux apeler. »*) = 10

On peut voir dans notre problème cible que la correction attendue est le passage à l'infinitif du mot « *appelé* » en « *appeler* ». Or, comme on peut le voir, le moteur choisi le Cas2 où la correction est l'ajout d'un p manquant.

Voici un exemple plus flagrant :

Problème cible « *Tu peux appelé ?* »

Cas1 = (*« J'aime les Litchys.», « J'aime les Litchis. »*)
Cas2 = (*« Salut j'ai un problème tu peux appelé chez moi. », « Salut j'ai un problème tu peux appeler chez moi . »*)

distance_édition(« *Tu peux appelé? » , « J'aime les Litchys. »*) = 27
distance_édiiton(« *Tu peux appelé ? », « Salut j'ai un problème tu peux appelé chez moi. »*) = 39

Ici, le moteur choisirait le Cas1, qui n'a rien à voir. Alors que le Cas2 contient le même problème que le problème cible et serait celui qu'on doit choisir. On voit qu'il peut y avoir beaucoup de caractères 'parasites' qu'on ne voudrait pas prendre en compte. Cela est d'autant plus problématique car la base dont les cas proviennent de WiCoPaCo peuvent contenir plusieurs phrases dans un problème. Il a donc fallu réfléchir à un autre moyen de choisir les cas dans la base.

3.1.2 - Niveau d'indexation

Nous nous sommes rendus compte que prendre la totalité des phrases n'était pas efficace. Ce qui importait en plus de la chaîne à substituer était aussi ce qu'il y avait autour : le contexte de la phrase.
Par exemple, pour la négation « *Je veux pas.* » → « *Je ne veux pas.* » Le « pas » est important.
Pour une conjugaison au participe passé « Elle est allé. » → « Elle est allée. » L'utilisation d'un pronom « *Elle* », implique l'ajout d'un accord au féminin ici le « *e* ».
On a donc décidé de récupérer les mots adjacents à gauche et à droite de la chaîne à substituer.
À l'indice 0 index_0 on récupère la chaîne qui va être substituer.
À l'indice 1 on récupère la chaîne qui va être substituer et 1 mot à gauche et à droite de la chaîne.
À l'indice 2 on récupère 2 mots à gauche et à droite de la chaîne en plus de la chaîne à substituer.
Et ainsi de suite.
Pour l'instant, on a jugé qu'un indice maximum de niveau 2 était suffisant.
Je vais détailler son fonctionnement à travers un exemple :
On a un problème cible « *Je suis sur Paris.* » et deux cas ainsi que leur différent niveau d'index pour la résolution de celui-ci :

Cas1 = (« *Je suis sur Nancy.* », « *Je suis à Nancy.* »)
Modification droite et gauche : (sur, à)
Index_cas1[0] = « *sur* »
Index_cas1[1] = « *suis sur Nancy* »
Index_cas1[2] = « *Je suis sur Nancy.* »

Cas2 = (« *Le monde et triste.* », « *Le monde est triste.* »)
Modification droite et gauche : (, s)
Index_cas2[0] = « »
Index_cas2[1] = « *et* »
Index_cas2[2] = « *monde et triste* »

Une fois ces index récupéré, on va partir de zéro jusqu'à deux. À chaque niveau, on va compter le nombre de mots différents entre les index et le problème cible. Le moteur choisira le minimum. S'ils ont le même nombre de mots différents, il prend les index de niveau supérieur et recommence. Si on n'arrive toujours pas à départager les deux cas au niveau maximum, on utilisera la distance d'édition pour choisir.

Algorithme niveau d'indexation :

Indexation
entrées :

- cible : chaîne de caractère
- cas1 : chaîne de caractère
- cas2 : chaîne de caractère
- Index_cas1 : tableau chaîne de caractère
- index_cas2 : tableau chaîne de caractère
- k : entier, niveau maximum d'indexation

sortie :

- cas_choisi : chaîne de caractère

Début

Pour i de 0 à k faire
cas1_nb = nombre_de_mot_différent(cible, *index_cas1[i])*
cas2_nb = nombre_de_mot_différent(cible, index_cas2[i])
si i == 2 et cas1_nb == c*as2_nb alors*
si distance_d'édition(cible, index_cas1[i]) <=
distance_d'édition(cible, index_cas2[i])
retourne cas1
sinon
retourne cas2
sinon
si aucun des 2 index n'est une chaîne vide alors

si cas1_nb < cas2_nb alors
retourne cas1
sinon
si cas1_nb > cas2_nb
retourne cas2

Fin

3.1.3 – Comparatif

Voici les cas (source, solution (source)) présent dans la base de cas initiale créée par Monsieur Levy, j'y ai moi même ajouter quelques cas:

- *J'aime pas les pommes. Je n'aime pas les pommes.*
- *Je suis sur Nancy. Je suis à Nancy.*
- *Au jour d'aujourd'hui. Aujourd'hui.*
- *J'ai été au cinéma. Je suis allé au cinéma.*
- *J'amènerai une bouteille. J'apporterai une bouteille.*
- *Elle apporte son enfant. Elle amène son enfant.*
- *Ils croivent. Ils croient.*
- *Tu es trop nul. Tu es très nul.*
- *Nous avons été à Paris. Nous sommes allés à Paris.*
- *Après qu'il ait dîné. Après qu'il a dîné.*
- *Après qu'il soit arrivé, il est allé chez sa mère. Après qu'il est arrivé, il est allé chez sa mère.*
- *Lui aussi n'a pas compris pourquoi elle chante. Lui non plus n'a pas compris pourquoi elle chante.*
- *Ce n'est pas de ma faute. Ce n'est pas ma faute.*
- *J'ai manger. J'ai mangé.*
- *Je m'arrete. Je m'arrête.*
- *Je manges. Je mange.*
- *Tous le monde. Tout le monde.*
- *She don't do anything. She doesn't do anything.*

A partir de cette base, j'ai testé diverses phrases pour voir les cas que me retournait les différents algorithmes :

	Problème cible	Niveau d'indexation	distance d'édition
•	*J'aime pas nager.*	*J'aime pas les pommes.*	*J'ai manger*
•	*C'est pas de ma faute.*	*ce n'est pas de ma faute.*	*Ce n'est pas de ma faute.*
•	*J'ai danser.*	*J'ai manger.*	*J'ai manger.*
•	*Dis lui que je suis surveiller.*	*Tous le monde.*	*Je suis sur Nancy.*
•	*Dis lui que je suis sur Paris.*	*Je suis sur Nancy.*	*Je suis sur Nancy.*
•	*C"est pas de ta faute.*	*Ce n'est pas de ma faute.*	*Ce n'est pas de ma faute.*
•	*Désolé je m'arete, je suis trop crevé là.*	*Tu es trop nul.*	*Je m'arrete.*

Les cas en rouges sont ceux qui ne sont pas bon pour la correction. On peut voir que les solutions choisies par les deux algorithmes retourne des solutions plus ou moins juste dans la base de cas initiale. Dans ce test, les cas étant de taille relativement petite, il y a moins de caractères non voulus. Cependant, dans le corpus WiCoPaCo, les cas peuvent être composés de plusieurs phrases. Cela réduirait l'efficacité de l'algorithme des distances d'éditions. Je suis en train de réfléchir aux possibles améliorations.

3.1.4 - Récupération des données

Au départ Monsieur Levy a créé manuellement une dizaine de cas d'erreurs courantes en français dans un fichier au format CSV. C'est ce dont je me suis servi pendant le stage pour tester le moteur et les différentes étapes. Une base de données contenant les cas récupéré automatiquement est prévue. La base ayant été développée sous Mysql, j'ai utilisé une librairie capable d'exécuter des requêtes sur cette base.

3.2 – Adaptation

3.2.1 - Raisonnement

Après avoir choisi un cas ressemblant, grâce à la remémoration, je dispose du problème cible et d'un cas jugé ressemblant me permettant de réaliser une équation analogique comme vu précédemment.

problème source : solution source :: problème cible : x

Pour détecter la chaîne à remplacer et sa position dans la phrase j'utilise des algorithmes permettant de récupérer les préfixes et suffixes communs entre deux chaînes de caractères. Ici, nous avons un problème cible «*J'aime pas nager.*». Accompagné du cas récupéré grâce à la remémoration cas = (*«J'aime pas les serviettes.», «Je n'aime pas les serviettes.»).* Tout d'abord, je récupère la modification avant, sa position dans la chaîne et la modification après du cas *(« », « e n », 1), ainsi que l*es préfixes et suffixes communs :

préfixe commun (« *J* »)
suffixe commun (« *'aime pas les serviettes* »)
Une fois que j'ai toutes les informations sur le changement, il faut que je l'applique à mon problème cible. Cette fois, j'utilise aussi le préfixe commun mais sur la cible et le problème source. Si toutefois je ne trouve pas de préfixe commun, j'essaie de trouver une chaîne de caractère dans le problème source qui correspondrait au début du problème cible. Pour cela, j'utilise un autre algorithme LCS (longest common substring, plus longue sous chaîne commune). Cet algorithme permet de retrouver la plus longue sous chaîne de caractère commune consécutive entre deux chaînes. Par exemple, nous avons deux chaînes C1 et C2 avec :
C1 = « Coucou, je suis pas en retard j'espère. »
C2 = « Hey je suis là, par ici. »
La plus longue sous chaîne consécutive ici est « je suis ».
Je n'ai plus qu'à appliquer la correction et cela me donne :

J'aime pas les serviettes. : Je n'aime pas les serviettes. :: J'aime pas nager. : Je n'aime pas nager.

3.2.2 - Algorithme d’adaptation

Adaptation

entrées :

- cible : chaîne de caractères
- source : chaîne de caractères
- solution source : chaîne de caractères

sorties :

- préfixe : chaîne de caractères
- modification : chaîne de caractères
- suffixe : chaîne de caractères

Le triplet en sortie forme la solution cible

Début

Calcul des préfixe suffixe commun (problème cible , problème source)

si oui (ressemblance dans la fin et/ou le début)

Calcul préfixe suffixe commun problème source, solution source

récupération de la sous chaîne à remplacer et sa position dans le problème source

Découpage dans problème cible

Constitution d'un triplet préfixe, sous chaîne, suffixe

sinon

Utilisation de LCS (détection d'une sous chaîne consécutive ressemblante dans la chaîne)

si oui

Calcul préfixe suffixe commun problème source, solution source

récupération de la sous chaîne à remplacer et sa position dans le problème source

Découpage dans problème cible

Constitution d'un triplet préfixe, sous chaîne, suffixe

sinon

retourne le problème cible

retourne le triplet

Fin

4 - Travaux futurs et améliorations

Le sujet est extrêmement vaste et le fait qu'on ait commencé à un niveau très bas, au niveau des chaînes de caractères, nous laisse beaucoup de voies d'explorations disponibles. Pour commencer sur du cours terme :

- Il faudrait tester le moteur de manière plus complète et approfondie avec la base de cas filtrée ou non et dans d'autres langues. Avec ces tests, je pourrais réaliser des statistiques afin de comparer les différentes méthodes de remémoration et d'adaptation.
- Je pourrais revoir l'accès à la base de cas, les classer et les catégoriser dans un arbre permettant une recherche moins coûteuse et plus rapide.

Sur du plus long terme je pourrais :

- Modifier ou chercher une approche différente de l'adaptation, toujours en travaillant sur les chaînes de caractères.
- Chercher d'autres approches d'adaptation, comme par exemple utiliser une analyse plus précise et poussée en analysant la structure sémantique et lexicale des phrases.
- Chercher d'autres méthodes de remémoration.

5 - Conclusion

Le stage s’est globalement bien déroulé. Le moteur est utilisable et les points principaux du stage ont été respectés. Une des difficultés a été l’apprentissage d’un nouveau langage de programmation. Le sujet en lui même, l’analyse de phrase est tellement vaste qu’il y a énormément de choses à faire de différentes manières. La reprise d’un code existant a apporté son lot d’avantages et d’inconvénients :

- D’un côté l’analyse et la compréhension d’un autre code est toujours plus ou moins fastidieuse. Dans mon cas le code était clair et bien commenté, il contenait toutefois des notions plus poussées que mes connaissances.

- L’existence d’une base déjà développé a permis d’éviter de commencer à partir de rien.

Les divers projets développés en groupe et seul durant la licence m’ont permis une rapide mise en route au sein du projet.

Le développement d’un projet concret et utilisable ainsi que le travail en groupe a été très motivant. La fonctionnalité du moteur dans une première version peut être la base d’un projet plus poussé car il y a de nombreuses voies d’améliorations.

Bibliographie / Webographie

- Analogy for Natural Language Processing and Machine Translation – Yves Lepage

- Plus longue sous séquence commune
  https://fr.wikipedia.org/wiki/Plus_longue_sous-s%C3%A9quence_commune
  dernière consultation : mai 2018

- Apprenez à programmer en Python - Vincent Le Goff
  https://openclassrooms.com/courses/apprenez-a-programmer-en-python
  dernière cosultation : mai 2018

- 5. Data structures – Python
  https://docs.python.org/3/tutorial/datastructures.html
  dernière cosultation : mai 2018

- how to exit from python without traceback ?
  https://stackoverflow.com/questions/1187970/how-to-exit-from-python-without-traceback
  dernière consultation : mai 2018

- loria.fr
  http://www.loria.fr/fr/presentation/
  dernière consultation : juin 2018

- loria.fr http://www.loria.fr/fr/la-recherche/departements/4-traitement-des-langues-et-des-connaissances/ dernière consultation : juin 2018

- D4.pdf
  http://www.loria.fr/wp-content/uploads/2016/05/D4.pdf
  dernière consultation : juin 2018