Faculté des Sciences et Technologies de Nancy

Licence Informatique 2018 - 2019

Rapport de stage

Étude et implémentation d'un outil de remémoration de cas pour un système de raisonnement à partir de cas de correction de phrases en français

---

Auteur
Julien Romary

Maître de stage
Jean Lieber

Encadrant universitaire
Emmanuel Jeandel

Date
8 avril 2019 - 9 juin 2019

Lieu
Loria – Campus Scientifique
Vandoeuvre-lès-Nancy

Avant-Propos

Ce rapport décrit le déroulement du stage de fin de troisième année de licence informatique. Le stage s'est déroulé au Loria [4], sous la tutelle de MM. Jean Lieber, Emmanuel Nauer, Bruno Guillaume et Yves Lepage.

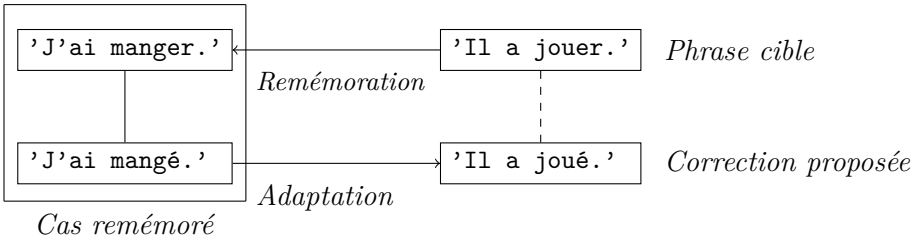
Le sujet initialement formulé est le suivant :

> Ce stage participe au développement de l'outil *Corrector* dont l'objet est l'application du raisonnement à partir de cas à la correction de phrases en français.
>
> Un cas est donné par un couple (PI, PC) où PI est une phrase incorrecte et PC une correction de cette phrase, par exemple PI = `'Il mange pas de pain.'` et PC = `'Il ne mange pas de pain.'` Un tel cas est associé à une explication, dans l'exemple, cela serait le fait que le mot de négation `'pas'` doit être accompagné du mot `'ne'`.
>
> Étant donné une nouvelle phrase supposée incorrecte telle que `'Tu ménages pas tes efforts.'`, l'idée est alors (1) de chercher un cas, dans la base de cas, proche (dans un certain sens) de cette phrase et (2) de modifier ce cas pour la correction de cette nouvelle phrase. L'étape (1) est appelée remémoration, l'étape (2) est appelée adaptation. L'objectif de ce stage sera de concevoir, développer et évaluer une méthode de remémoration. Cette méthode s'appuiera sur la notion d'explication associée à un cas.

Le complément au sujet proposé durant le stage a été le suivant :

> Création d'un programme d'adaptation simple en utilisant les résultats de la remémoration créée préalablement pour avoir un logiciel complet et fonctionnel.

Le programme final fonctionne de manière satisfaisante et a été implémenté par l'intermédiaire d'un site réalisé par M. Sies [8]. Voici un exemple de correction réalisée par le logiciel développé :



Remerciements

Au terme de ce travail, je tiens à remercier mon maître de stage M. Jean Lieber de m'avoir permis d'effectuer mon stage au Loria. Merci aussi pour ses conseils et son aide, tout au long du stage.

Je remercie aussi Emmanuel Nauer et Bruno Guillaume, qui m'ont guidé dans mes recherches et développements.

Merci aux précédents stagiaires, Isabelle Mornard, André Giang, Hue-Nam Ly, Damien Levy, ayant travaillé sur le projet. Ceux-ci ont proposé des pistes de recherche et des informations sur le sujet, et des bases de cas déjà réalisées sur lesquelles j'ai pu travailler.

Je remercie enfin Reynault Sies, stagiaire de L3 dans la même équipe que moi, avec qui j'ai pu travailler en binôme de façon efficace ainsi que tous mes camarades qui m'ont relu ou aidé lors de ce stage, et que toute l'équipe encadrante de la Faculté des Sciences de Nancy qui ont rendu ce stage possible.

Sommaire

Présentation de l'entreprise

Le stage s'est déroulé sur le site du Loria, Laboratoire Lorrain de Recherche en Informatique et ses Applications, situé à Nancy. Cet établissement étant une unité mixte de recherche, il est en lien avec le CNRS, l'Université de Lorraine et Inria. Le Loria est un organisme de recherches créé en 1997 et comportant à ce jour 28 équipes de chercheurs [10].

Les 5 départements de ce laboratoire sont :

— Algorithmique, calcul, image et géométrie ;

— Méthodes formelles ;

— Réseaux, systèmes et services ;

— Systèmes complexes, intelligence artificielle et robotique ;

— Traitement automatique des langues et des connaissances.

C'est dans ce dernier, et plus particulièrement dans l'équipe $\mathcal{K}$ en lien avec l'équipe *Sémagramme* que se sont déroulés les 2 mois de stage.

Introduction

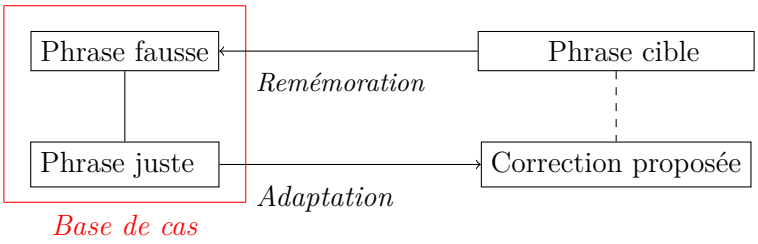
Le but de ce stage est d'effectuer une étape de remémoration de phrases à partir d'une phrase donnée. Les phrases ainsi remémorées pourront, dans un second temps, être utilisées dans une étape d'adaptation, le tout dans le but de l'implémenter dans un logiciel de correction orthographique en français.
Dans le raisonnement à partir de cas (RàPC), on possède une base de cas BC, où chaque cas est constitué de plusieurs attributs :

— un problème x ;
— une solution y (de x) ;
— une explication e (du passage de x à y).

En général dans un système de RàPC, la base de cas est de taille limitée (de l'ordre de quelques centaines de cas) contrairement à certaines techniques d'apprentissages numériques récentes utilisant des bases d'exemples de très grande envergure. La recherche des cas similaires peut alors être une tâche complexe car le principe est de pouvoir expliquer la démarche, et pourquoi un cas a été choisi et pas un autre. Le but étant qu'à partir d'un problème, on cherche dans la base de cas un problème similaire dont on connaît la solution (remémoration). À partir de ce problème similaire on cherche à résoudre le problème cible (adaptation).

Le RàPC peut donc se définir de la manière suivante : la remémoration cherche à trouver un cas similaire dans la base de cas, tandis que l'adaptation utilise le cas trouvé pour générer une solution au problème cible.

Le fonctionnement général de l'application se fait en deux étapes, la remémoration d'une phrase fausse similaire et l'adaptation pour corriger la phrase cible.



Dans ce projet, nous allons nous focaliser sur l'étape de remémoration dans le logiciel *Corrector*. Étant donné une phrase fausse x^c (phrase cible), on cherche à identifier un cas source appartenant à la base de cas telle que son adaptation à x^c permette de générer une phrase corrigée (ou du moins avec une faute en moins) de x^c.

Le but de la remémoration dans ce stage est que la recherche de phrase se fasse sans devoir effectuer une analyse lexicale ou grammaticale. En effet cette méthode bien que plus appropriée en théorie, est difficile à mettre en pratique au vu de la complexité des langues. De plus, la durée du stage ne permet pas cette approche. Cela permet également, une approche indépendante de la langue, qui, est susceptible d'être réutilisable simplement du français vers une autre langue alphabétique.

Par exemple, soit le problème cible x^c = `'Je danse pas la valse.'` et soit la base de cas suivante composée de deux cas X^1 et X^2 :

X^1	X^2
x^1 = `'Tu ne danse pas la valse.'`	x^2 = `'Ils partent pas loin.'`
y^1 = `'Tu ne danses pas la valse.'`	y^2 = `'Ils ne partent pas loin.'`

(1)

L'étape de remémoration va ainsi chercher, dans la base de cas, le cas le plus proche, dans un certain sens, de la phrase cible x^c. Ici, il semble difficile d'identifier le cas à choisir, on remarque que les deux choix peuvent avoir une justification.

Dans le cas du choix de X^1, les deux phrases, x^1 et x^c, sont très semblables en terme de mots, et ainsi l'adaptation ajouterait un `'s'` au verbe de x^c, ce qui rendrait la phrase fausse puisque `'Je danses pas la valse.'` , n'est pas correcte. Dans le cas du choix de X^2 les deux phrases, x^2 et x^c, possédant la même erreur, mauvaise utilisation de la négation, un `'ne'` serait ajouter. Cela aurait produit la phrase `'Je ne danse pas la valse.'` qui correspond à la correction attendue. On remarque donc la complexité de ces choix qui peuvent nécessiter des explications supplémentaires. Le but est donc de mettre en avant de nouvelles méthodes de remémoration en s'appuyant sur des travaux déjà réalisés.

Dans ce projet, l'adaptation consiste, intuitivement, à appliquer la différence dans le cas source (entre la phrase fausse et la phrase juste) sur la phrase cible.

Exemple :

`cas :` x `= 'Je danses.'` y `= 'Je danse.'`
`Différence : 's'`
`Phrase cible :` c `= 'Je nages.'`
`Addaptation : 'Je nage.'`

La différence ici dans le cas est le retrait du `'s'`. L'application de celle-ci sur la phrase cible donne l'adaptation sans le `'s'`.

L'adaptation est implémentée par un système de résolution d'équations analogiques entre chaînes de caractères conçu et développé par M. Yves Lepage.

Lors de ce stage je me suis servi des travaux de stagiaires ayant travaillé sur le projet avant moi : Isabelle Mornard et André Giang.

Dans un premier temps, j'ai réalisé une analyse des différentes idées trouvées par monsieur Giang [2]. En effet, en plus de leurs résultats, j'ai effectué une critique et des précisions sur leur fonctionnement afin d'affiner les recherches déjà faites. J'ai aussi proposé des pistes d'améliorations possibles.

Suite à cette étape, j'ai cherché d'autres moyens de trouver les cas similaires pour la remémoration, en expliquant leur fonctionnement, en effectuant une implémentation et ensuite en émettant une critique sur les résultats obtenus.

Ensuite, après réunion avec l'équipe encadrante, un des algorithmes a été choisi, l'indexation, et j'en ai donc réalisé une implémentation. Par la suite, j'ai développé un algorithme simple d'adaptation afin de créer un logiciel indépendant pouvant être utilisé dans le site développé pour cela.

1 Analyse de l'existant

Le document de monsieur Giang nous fait part de deux façons de remémorer et de leur mise en place, la première liée à la distance d'édition et la seconde liée à un niveau d'indexation. Afin de tester complètement les idées développées, j'ai recréé les algorithmes en suivant les instructions fournies dans son rapport.

1.1 Distance d'édition

Définition Le principe de l'analyse sur la distance d'édition se porte sur le nombre de changements à effectuer pour passer d'une chaîne de caractères à une autre. L'algorithme LCS (*Longest Common Subsequence*) permet de trouver la plus longue sous-séquence, et ainsi à partir de celle-ci, y ajouter les caractères manquants pour obtenir la chaîne de caractères voulue. Cette méthode permet de quantifier le nombre d'étapes de transformation (addition ou suppression d'un caractère) entre deux mots.
Soit c_1 et c_2 deux chaînes de caractères :
c_1 = `'chats'`
c_2 = `'chiens'`
La plus longue sous-séquence commune est `'chs'`, ainsi, pour partir de c_1 et obtenir c_2, il suffit de supprimer les caractères jusqu'à obtenir `'chs'`, puis d'ajouter à celle-ci les caractères pour obtenir c_2.
chats $\Longrightarrow$ chs : 2 étapes
chs $\Longrightarrow$ chiens : 3 étapes
On obtient donc une distance de 5 entre les deux chaînes c_1 et c_2.

De manière formel, soit deux phrases c_1 et c_2 soit L la plus longue sous-séquence commune entre c_1 et c_2. On a alors :

$$d(c_1, c_2) = (|c_1| - |\mathrm{L}|) + (|c_2| - |\mathrm{L}|)$$

La distance entre deux chaine de caractères. Avec $|c_1|$, $|c_2|$, $|\mathrm{L}|$ les longueurs des chaines respectives c_1, c_2 et L

Analyse Afin de tester cette méthode, j'ai créé un script récupérant une phrase et regardant dans la base de cas réalisée par Isabelle Mornard [5] quel cas s'en approche le plus. Le script prend en paramètre une chaîne de caractères et cherche dans la base de cas la chaîne la plus proche en suivant l'algorithme LCS [1]. Il retourne la phrase fausse du cas identifié.

Résultat 1

```
1 phrase cible : Je vais rangé.
2 phrase fausse remémorée : "Je vais mangé."
```

Résultat 2

```
1 phrase cible : Tu va travaillé dehors.
2 phrase fausse remémorée : "Toute travaillent."
```

Les résultats sont sous la forme phrase fausse fournie par l'utilisateur et phrase fausse retournée par le logiciel.

Comme on peut le remarquer, le premier exemple donne satisfaction. Il trouve bien le cas le plus proche et, lorsqu'on applique la correction en utilisant l'adaptation, on récupère une phrase juste et satisfaisante. Malheureusement, les limites de cette méthode sont mises en avant par l'exemple 2. En effet, la phrase trouvée, bien que la plus proche en terme de sous-séquence, n'est pas du tout représentative de l'erreur présente, et donc on obtiendrait une correction fausse et non satisfaisante.

Conclusion Le résultat de cette méthode peut être très efficace si la base de cas est très grande, et donc contient soit directement la phrase à corriger, soit une phrase très proche et avec la même erreur. Comme observé dans le premier résultat où un cas très proche existait déjà, une solution a été trouvée, cependant, pour le résultat 2, une petite modification entraîne une perte totale du bon résultat.

Avoir tous les cas possibles de la langue française est impossible de manière générale, plus encore avec la limite de taille assez restreinte de la base de cas. De plus, on cherche une erreur en fonction de tous les caractères de la phrase, sans les mettre en lien entre eux. Cela ne permet pas de relier l'erreur à son contexte, et donc de savoir le type d'erreur à chercher au lieu de la ressemblance à la chaîne. Cette approche de la remémoration n'est donc pas très satisfaisante.

1.2 Niveau d'indexation

Définition Le principe de l'analyse sur les niveaux d'indexation se porte sur la proximité des mots avec la correction effectuée dans chaque cas. Pour chaque cas, on récupère la position de la correction apportée dans la chaîne de caractères. Ainsi, on peut repérer la correction apportée dans son contexte (seulement avec les mots).
Explication de la récupération du niveau d'indexation, soit c^1 un cas de la base de cas :

c^1
x^1 = `'Tu ne danse pas la valse.'`
y^1 = `'Tu ne danses pas la valse.'`
e^1 = (11, add, `'s'`)

(2)

Si l'explication est notée de cette façon, on a alors l'information de la position (11), le type de modification effectuée sur la chaîne de caractères (add) qui correspond à un ajout de chaîne, et la chaîne à ajouter ou retirer (`'s'`). À partir de cela, on peut chercher les mots adjacents pour former la table d'indexation du cas. Celle-ci contient chaque niveau correspondant au nombre de mots adjacents cherchés, et un couple correspondant aux mots à droite et à gauche.
On part de la position 11 dans la chaîne :
`'Tu ne danse| pas la valse.'`
On cherche ensuite pas à pas les mots adjacents et on crée ainsi les indexations :

Niveau d'indexation	Couple du niveau
0	`([], [])`
1	`(['danse'], ['pas'])`
2	`(['ne', 'danse'], ['pas', 'la'])`
3	`(['Tu', 'ne', 'danse'], ['pas', 'la', 'valse.'])`

(3)

Grâce à la table ainsi créée, on peut comparer, pour chaque caractère d'une phrase donnée, les niveaux d'indexation avec chaque cas et ainsi récupérer le taux de compatibilité avec un cas. Cela ce fait de la manière suivante :

Algorithme de comparaison d'indexation

```
1 Pour chaque cas c^i de la base de cas
2   Pour chaque caractère de la chaîne x^c à corriger
3     Pour chaque niveau d'indexation de c^i
4       On compte le nombre de mots similaires
```

On a ainsi une quantification de la proximité d'un cas avec une phrase à corriger. À partir de cela, on peut utiliser le cas le plus semblable pour effectuer la correction.

Analyse Afin de tester cette méthode, il faut changer la méthode d'explication et donc recréer une base de cas, plus petite que la première base. Cette base sera orientée principalement côté programmation. Trois classes (en programmation orientée objet) ont été créées :

`Explication` Une classe comportant 3 champs comme expliqué dans l'analyse (position, type, chaîne de caractères).

`Cas` Une classe correspondant à un cas (chaîne de caractères fausse, chaîne de caractères juste, Explication).

`BaseDeCas` Classe principale de cette mise en place de base de cas contenant les Cas.

Une fois ces classes créées, il était facile de tester avec des exemples. La base de cas a été remplie avec 7 cas, presque tous tirés de la base de cas utilisée pour le test de LCS. Pour faire ce test, les mêmes phrases fausses ont été utilisées. *insertion dans la base de cas avec explication* (voir annexe).

Résultat 1

```
phrase cible: Je vais rangé.
Cas identifié :
x: Je vais mangé.
y: Je vais manger.
e: (12, add, er)
Endroit le plus proche dans la phrase cible : 8
```

Résultat 2

```
phrase cible: Je vais travaillé.
Cas identifié :
x: Je vais mangé.
y: Je vais manger.
e: (12, add, er)
Endroit le plus proche dans la phrase cible : 8
```

Comme on peut le remarquer, les deux exemples sortent un cas identique qui correspond bien à l'erreur dans la phrase cible. On peut en effet voir que le programme a de plus identifié que l'erreur se trouvait dans le dernier mot. Les résultats obtenus par cette méthode sont donc satisfaisants.

Résultats experimentaux

Soit :
$x^c =$ `'Je vais pas manger.'`
Deux cas :
$x^1 =$ `'Tu vas pas jouer.'` $y^1 =$ `'Tu ne vas pas jouer.'` $e^1 = (3, \text{add}, \text{`ne '})$.

x^2 = 'Je ne vais pas regardé.' y^2 = 'Je ne vais pas regarder.' e^1 = (21, add, 'er').
Si le niveau d'indexation est de 2, alors x^1 aura le mot 'pas' en commun et l'erreur sera placée après le premier mot de x^c. x^2 aura en commun le mot 'pas', et placera l'erreur à la fin de la phrase x^c. Cependant avec indexation de niveau 3, x^1 trouvera 'pas' avec l'erreur au même endroit, alors que x^2 trouvera 'vais' et 'pas'. Le choix du cas correspondant sera alors porté sur le deuxième, qui ne correspond pas à une correction de x^c.

Retour sur résultat Suite à une réunion réalisée avec les encadrants du projet, M. Jean Lieber, M. Emmanuel Nauer et M. Bruno Guillaume, une refonte de la remémoration de cas a été faite. En effet, l'utilisation des différentes classes n'est plus utile. Une seule structure correspondant à un cas contient la phrase juste, la phrase fausse et le niveau d'indexation à partir de l'erreur. L'erreur est trouvée suite au parcours des deux chaînes en partant du début puis de la fin. On génère ensuite 10 niveaux d'indexation, le niveau 0 correspondant à la chaîne de caractères qui à été enlevée. Suite à cela, on prend la phrase cible, et on effectue l'algorithme suivant :

Algorithme de comparaison

```
1  Pour chaque cas C de la base de cas
2    Pour chaque occurrence d'indexation de niveau 0 dans la phrase cible
3      On génère les niveaux d'indexation à l'endroit de l'occurrence
4    S'il n'y a pas d'occurrence de ce niveau 0 on ne prend pas le cas

6  Pour chaque cas retenu
7    Pour chaque indexation généré en fonction de l'occurrence
8      Pour chaque niveau d'indexation généré
9        On compte le nombre de mots identiques entre cible et cas
10     On a un ratio pour chaque occurrence
11     On sélectionne le meilleur pour chaque cas
12   On a un ratio par cas on sélectionne le meilleur
13     On stocke le cas optimal
14 On retourne le cas optimal identifié
```

L'indexation de niveau 0 dans un cas est la chaine de caractères corespondant à la différence entre dans la phrase fausse et phrase juste du cas source. C'est la chaine qui change entre les deux phrases.

Conclusion

Première conclusion Les résultats de cette méthode sont beaucoup plus satisfaisants que la première. En effet, on arrive même à identifier où se trouve l'erreur.
Contrairement à la première méthode, on se focalise sur les mots et leur place dans la phrase, on peut alors mieux placer les erreurs dans leur contexte. Il reste toujours des erreurs comme montré dans les résultats expérimentaux ; avec des phrases très proches. Il est aussi possible que les erreurs ne soient pas identifiées si on ne prend une phrase qu'avec des synonymes d'un cas de la base de cas, alors les mots ne seront pas identiques et donc, on ne pourra pas trouver le cas en question.

Deuxième conclusion Cette approche est plus efficace que la méthode par distance d'édition, mais il reste encore des possibilités d'améliorations. En effet, cette méthode est très prometteuse mais la mise en place n'est pas optimal. On verra par la suite sa remise en place pour une version définitive.

Amélioration possible Le principe de base de cette méthode fonctionne plutôt bien, on peut donc chercher à l'améliorer, voici quelques idées :

- Combiner les deux méthodes et chercher dans les niveaux d'indexation avec la méthode LCS pour avoir des mots proches et non identiques, cependant on risque de se trouver avec les mêmes biais que ceux expliqués ci-dessus.
- Ajouter à chaque cas une liste de mots importants, qui, s'ils sont trouvés, renforce la possibilité de choisir le cas. Par exemple dans le cas :
  x^1 = `'Tu vas pas jouer.'` y^1 = `'Tu ne vas pas jouer.'` $e^1 = (3, \text{add}, \texttt{'ne '})$
  Si lors de l'indexation on se place a un endroit de la phrase cible et que l'on trouve le mot pas a droite. Exemple :
  x^c = `'Je nage pas.'`
  Ici, l'indexation de niveau 2 entre `'Je'` et `'nage'` ressemblera a cela :
  `(['Je'], ['nage', 'pas'])`
  La présence ici du mot `'pas'` est importante, et correspond a une information que le cas devrait porter. On pourait alors garder pour chaque cas cette information, et l'utiliser pour favoriser un choix plus qu'un autre.
- Analyser la phrase pour identifier les mots, ainsi, lors de la comparaison on cherche par exemple à identifier des verbes, car il n'est pas utile d'avoir le même verbe. En revanche, on laisse le `'pas'` en toutes lettres dans une négation, car il est important de retrouver ce mot dans la phrase. On se retrouverait donc avec des niveaux d'indentation, pour la négation, de la forme `([`⟨ *Sujet* ⟩`],[`⟨ *Verbe* ⟩`,'pas'])`. On retrouve la structure générale de droite et gauche du `'ne'` avec le sujet à gauche et, un verbe suivi d'un `'pas'` à droite.

2 Approfondissement et recherche

Suite au travail de l'état de l'art, je me suis mis à rechercher de mon coté de nouvelles pistes de dévelopements ainsi que des améliorations possible sur celles déjà existants. Je suis revenu sur mes implémentations et j'ai réalisé des tests et fait des critiques concernant les avancées que j'ai pu réaliser. Ainsi le travail poura être réutilisé sans passer par les mêmes chemins et erreurs que j'ai pu faire.

2.1 Indexation squelettique

Définition Le principe est d'ajouter à la méthode des niveaux d'indexations une fonctionnalité supplémentaire. Le but est d'ajouter une liste de mots qui sont représentatifs de l'erreur corrigée dans le cas en question, et donc de n'avoir que les mots *utile* de la phrase, on va donc créer un squelette de la phrase. Par exemple :
x^1 = `'Tu vas pas jouer.'` y^1 = `'Tu ne vas pas jouer.'` $e^1 = (3, \text{add}, \texttt{'ne '})$
On peut ajouter une information :
`[None, None, 'pas', None]`
Cela correspond a une forme simplifier de la phrase ou juste les éléments utile pour comprendre l'erreur sont affichés.

Résultats Pour les cas vu dans l'indexation simple cela fonctionne toujours aussi bien. En revanche, certains cas ont quelques problèmes :

Résultat 1

```
choix du test[I1: IndexationV1; I2: IndexationV2; L: LCS]: I2
Chaîne de caractere: Je vais pas ranger.
Cas identifié :
x: Je vais mangé.
y: Je vais manger.
i: (12, add, er, ['Je', 'vais', None])
```

`Endroit le plus proche dans la chaine cible : 8`
Après différents tests réalisés, les retours sont presque identiques aux niveaux d'indexation.

Conclusion Cette méthode n'apporte pas d'amélioration majeure à la recherche de cas actuel avec le niveau d'indexation, il ne change pas non plus la recherche de mot similaire dans la phrase. Il n'accélère pas non plus le temps de calcul. Cette méthode n'apporte donc rien au problème actuel.

2.2 Le poids des petits mots

Définition Le principe de cette méthode se fonde sur l'heuristique suivante :

> Les petits mots portent plus d'importance que les grands mots. En effet, on peut remarquer que les sujets (`'je'`, `'tu'`, ...) ou les négations (`'ne'`, `'plus'`, `'pas'`) sont souvent de petite taille, et informent beaucoup sur la construction de la phrase. Les grands mots comme `'amélioration'` ne sont pas forcement dans toutes les phrases, ainsi si deux phrases on ce mot, leurs sens a plus de chance d'être différent. Ainsi en se focalisant sur les mots de petite taille on oriente vers des cas ayant la même structure de phrase.

Ainsi on va trier les mots en fonction de leurs taille, et plus un mot est petit dans un cas plus, si on le trouve dans la phrase cible, il sera important. Exemple :
Soit le cas,
x^1 = `'Tu ne danse pas la valse.'` y^1 = `'Tu ne danses pas la valse.'` i^1 = None (l'information ici n'est pas utile).
Et le tableau suivant correspondant au aux mots de la phrase fausse trié en fonction de leurs longueur (je ne cherche que les mots de longueur au plus 26 lettres) :
`[[], [], ['Tu', 'ne', 'la'], ['pas'], [], ['danse'], ['valse.'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]`.

Implémentation Une implémentation de l'algorithme a été effectuée en python (voir annexe), dans celle-ci *res* contient alors le cas identifié, si *res* est égale à `None` cela veut alors dire que aucun cas satisfaisant n'a été trouvé dans la base de cas. Ce qui veut dire que l'on ne trouve pas forcement de solution, ce qui n'est pas forcement ce que l'on veut.

Résultats

Résultat 1

```
phrase cible: Tu ne nage pas le crawl ?
cas remémoré:
x: Tu ne danse pas la valse.
y: Tu ne danses pas la valse.
e: None
```

Résultat 2

```
phrase cible: J'aime mangé.
cas remémoré:
x: Je vais mangé.
y: Je vais manger.
e: None
```

Résultat 3

```
phrase cible: Tu aime les jeux vidéos.
cas remémoré:
x: Tu vas pas jouer.
y: Tu ne vas pas jouer.
e: None
```

Un test de plus grande échelle a été réalisé comme nous n'utilisons pas ici les phrases corrigées, j'ai pu réutiliser les phrases de Isabelle Mornard en réalisant une petite extraction depuis le fichier.

Conclusion Contrairement aux attentes de cette méthode, celle-ci renvoie des résultats plutôt satisfaisant, bien qu'il reste des erreurs majeures comme mis en évidence dans le résultat 3. Pour les problèmes comme celui ci, cela provient du fait que les phrases ont beaucoup de mots en commun et donc que se soit de l'indexation ou de la recherche sur les mots en général, on se retrouve avec les mêmes problèmes. On ne peut pas mettre dans leurs contextes les mots.

Les tests réalisés par la base de cas plus grande, ont en effet apporté plus de résultats mais aussi plus d'erreurs dans la correction, on peut se demander si trop de cas ne créerait pas une perte d'informations et on se focaliserait pas trop sur la phrase avec les mots en commun et non avec le même sens. Toutefois cette méthode plutôt simple fournit de bons résultats sur des exemples simples.

Le poids des petits mots pondéré Une amélioration possible étant de rajouter des mots important à considérer dans la recherche de mots en commun. Comme pour certaine corrections des mots sont plus important on les cherche en priorité en ajoutant un score sur certains mots qui augmente le score de la phrase.

Résultats Une amélioration dans certains cas quand deux phrases sont proches mais ont des corrections différentes, les mots importants tendent à choisir la bonne phrase. Cependant il est plutôt lourd d'ajouter à chaque cas les mots importants et cela demande une intervention humaine, ce qui n'est donc pas automatisable.

2.3 Approche de Gestalt

Définition L'approche de Gestalt [3] se fonde sur les sous-chaînes et non sur les sous-séquences comme pourrait le faire LCS. Pour avoir plus de chance de réussite les tests ont été faits avec la base de cas de Isabelle MORNARD. En effet cette méthode demande beaucoup d'exemples pour rechercher les cas les plus proches.

Résultats

Résultat 1

```
phrase cible: Je mange pas de pain.
phrase fausse remémoré: "Je viendrais demain."
```

Résultat 2

```
phrase cible: J'aime pas dormir.
phrase fausse remémoré: "J'ai peus de temps."
```

On remarque que dans aucun des cas présentés l'algorithme ne propose de bons résultats. Et dans le cas général, il ne propose pas non plus d'erreur pouvant se rapprocher du problème.

Conclusion Cette méthode ne fonctionne pas. En effet même en limitant la taille des sous-chaînes pour ne pas être inférieure à 2 celle-ci ne permet pas de trouver efficacement les cas proches. La méthode n'est donc pas recommandée seule.

Dans l'implémentation faite, il est possible de récupérer un ratio de proximité entre les deux phrases. Ainsi il serait possible de restreindre la base de cas dans un premier temps en ne prenant que les phrases assez proches, dans un sens. Dans un deuxième temps utiliser les cas trouvé pour des algorithmes plus complexes qui demandent plus de calculs, cela permettrait d'augmenter l'efficacité générale de l'implémentation.

3 Indexation de phrase

Suite à une réunion, une revue de l'implémentation du niveau d'indexation a été faite, et son fonctionnement refait en python2 [6].

3.1 Définition

Le fonctionnement reste plus ou moins le même ; chaque cas se voit attribuer un tableau de niveau d'indexation correspondant à l'endroit où se trouve l'erreur dans le cas. Pour cela on identifie l'erreur dans le cas en parcourant la phrase dans un sens puis dans l'autre. Pour le niveau d'indexation 0, si on enlève une chaîne de caractères de la phrase fausse celle-ci est recopié dans le niveau 0 sinon, celui-ci reste vide.

Ensuite le niveau 1 correspond au contexte direct, c'est-à-dire le mot dans lequel la modification est effectuée, et sa position dans le mot, au début, à la fin ou au milieu (a ajout, f fin, d début, m milieu). Les niveaux suivants correspondent au couple de mots de part et d'autre de la modification dans la phrase fausse, de plus en plus loin.
Exemple :
Cas : `'Il va mangé tard.' 'Il va manger tard.'`

Niveau d'indexation	Contenu
0	`'é'`
1	`('mangé', 'f')`
2	`('va', 'tard')`
3	`('Il', '.')`
4	`(", ")`

(4)

Une fois cela fait, on prend la phrase cible, pour chaque cas, pour chaque occurence du niveau 0 (si celui-ci n'est pas présent on ignore le cas) on crée les niveaux d'indexations (et on stocke la position dans le niveau 0). Ensuite on regarde les niveaux d'indexation au fur et à mesure, et on garde ceux où la partie droite ou gauche du couple correspond, pour ne garder que les plus grands. On a alors pour chaque cas un niveau d'indexation, et les endroits où ce niveau est atteint dans la phrase cible. On trie ensuite les cas retenus en fonction de leurs modifications phrase fausse / phrase juste.

3.2 Éxécution et résultat

L'exécution du programme permet de récupérer les cas avec le plus grand niveau d'indexation avec la phrase cible. Sans moyen de départager les cas après le tri, on prend tout les cas avec un niveau d'indexation supérieur à 1 (on élimine tous les cas où le niveau 1 ou moins est présent car ceux-ci ne sont pas utile au vu des expérimentations). L'algorithme ne retourne pas

que un cas, mais plutôt un groupe de cas (souvent 10% de la base de cas initiale) avec pour chacun le niveau d'indexation le plus haut obtenue et la position dans la phrase.

Exemple d'exécution sur la base de cas de Isabelle Mornard :

```
Phrase cible : Je vais nagé.
Nombre de cas initiaux: 209   Nombre de cas retenus: 15
Nombre de cas de niveau 5: 15
Nombre de cas de niveau 4: 0
Nombre de cas de niveau 3: 0
Nombre de cas de niveau 2: 0
Remémoration de niveau: 5
Enlever: 'é' position dans mot: f
Nombre de cas: 1
Enlever: 'e' position dans mot: f
Nombre de cas: 2
Enlever: " position dans mot: a
Nombre de cas: 11
Enlever: 'a' position dans mot: m
Nombre de cas: 1
```

Ici on remarque que sur les 209 cas initiaux, 15 sont retenus. Tous sont de niveau 5, et, dedans se trouve un cas permettant une correction satisfaisante de la phrase (celui où on enlève `'é'` et ajoute `'er'`). Cependant il n'est pas possible de le départager par rapport aux autres car juste avec un travail sur les chaîne de caractères rien ne les différencie.

Exemple d'exécution de la récupération du plus haut niveau d'indexation :

N°	Cas remémoré	Phrase cible		
	`Il va mangé tard.`	`Hélène ira marché dans le pré.`		
0	`'é'`	`('é', 1)`	`('é', 16)`	`('é', 28)`
1	`('mangé', 'f')`	`('Hélène', 'm')`	`('manché', 'f')`	`('pré', 'f')`
2	`('va', 'tard')`	`(", 'ira')`	`('ira', 'dans')`	`('le', '.')`
3	`('Il', '.')`	`(", 'marché')`	`('Hélène', 'le')`	`('dans', ")`
4	`(", ")`	`(", 'dans')`	`(", 'pré')`	`('marché', ")`

(5)

Ici on remarque que le niveau 1 marche pour le `'é'` de `'marché'` et `'pré'` jusqu'au niveau d'indexation 1. En effet le `'é'` et dans les 2 cas à la fin, ce qui correspond au cas remémoré. On va donc récupérer ce cas avec les différentes positions dans la phrase cible (ici [16, 28]) et le niveau maximum (ici 1).

Cette méthode permet aussi de remémorer plusieurs cas et donc de facilement se rabattre sur un cas alternatif si une correction ne correspond pas, cela peut aussi servir à la correction, car on peut utiliser plusieurs cas dans l'hypothèse d'un bug ou problème lors de l'adaptation.

3.3 Conclusion

L'outil se fondant sur l'indexation des cas permet un bon premier filtrage mais il n'est pas toujours suffisant pour discriminer les cas restants. En effet, dans la plupart des cas testés dans les cas choisies se trouve un cas qui permet de corriger de manière satisfaisante la phrase cible. On peut donc considérer cette algorithme comme satisfaisant. Celui-ci s'inscrit dans l'approche MAC/FAC (*Many Are Called, Few Are Chosen*), on part d'une grande base de cas et on applique un filtre pour générer une sous base de cas.

Formellement cela s'explique de la façons suivante. Soit une base de cas BC, soit un filtre f, on peut appliquer f sur BC. On obtient alors BC_f qui est un sous ensemble de cas de BC, dont les elements verifient le critère f.

Il est possible d'appliquer une succession de filtre sur une base de cas initiale, pour avoir une sélection de plus en plus fine.

Cette méthode est très pratique, en effet l'algorithme est plutôt rapide, on peut alors appliquer un algorithme complexe sur les cas retournés, est ainsi augmenter la rapidité de l'ensemble du programme. De plus on peut appliquer plusieurs filtres à la suite, et ainsi obtenir des sous-ensembles de la base de cas. Ici on retrouve dans la plupart des tests, un cas permettant une bonne correction dans le groupe remémoré. De plus une réduction à 10% de la base de cas est un ratio plutôt important. On peut donc utiliser cette méthode.

3.4 Utilisation

Pour utiliser le programme de démonstration, exécuter le programme de remémoration `Indexation_V2.py`. Stocker le résultat dans un fichier permet de sortir un cas en particulier en le passant en paramètre de `lcs_pour_result.py`.
Il y a aussi un programme de test qui permet l'éxécution dans l'ordre avec sauvegarde intermédiaire dans deux fichiers (`res1` et `res2`).

```
jromary … Rapport-Julien_Romary Test_Rememoration Test_Indexation python
 test.py "Je vais sur Paris."
Je vais à Paris.
```

4 Adaptation

4.1 Description

Suite au développement de la partie remémoration, et devant intégrer le projet dans le site développé par monsieur Sies [8] nous devions faire fonctionner l'étape d'adaptation. Celle-ci devais se baser sur le travail de monsieur Lepage. Cependant comme nous n'arrivions pas à faire fonctionner le code, j'ai dù refaire une version simple de l'adaptation pour une démonstration complète du projet. Le developpent d'une classe pyton pour le lien entre python [7] et le site a été realisé en plus.

4.2 Fonctionement

Je récupère de ma remémoration un cas avec la position du meilleur appariement dans la phrase cible. Je récupère alors les modifications effectuées dans le cas, et je les adapte à la phrase cible. dans un premier temps je retire de la phrase fausse du cas ce qui doit être enlevé, puis je compare avec la phrase juste ce qui doit être ajouté, puis je retire / ajoute à la phrase cible ce que j'ai alors récupéré.
Exemple :

Cas remémoré	
`phrase fausse`	`'Je vais sur Nancy.'`
`phrase juste`	`'Je vais à Nancy.'`

Phrase cible : `'Je vais sur Paris.'`
Transformation de `'Je vais sur Nancy.'` en `'Je vais Nancy.'`
Récupération de la différence avec la phrase juste : `'à'`
Retrait puis ajout dans la phrase cible des modification : `'Je vais à Paris.'`.

4.3 Résultat

Les résultats sont plutôt satisfaisants pour des phrases où un retrait et ajout de caractères est effectué. Il reste cependant beaucoup d'améliorations possible. En effet, pour le cas où plusieurs appariements apparaissent, on ne peut pas choisir où se fait la modification.

5 Prolongement

5.1 Réalisation

Le travail réalisé répond aux attentes faites par le sujet de stage, des améliorations ont aussi été apportées. Une réalisation suplémentaire a été réalisée (adaptation) pour l'utilisation de l'outil dévelopé.

5.2 Documentation

Ce document peut sevir de documentation pour réutilisé le travail réalisé, de plus, tout le code réalisé se trouve sur G*it*L*ab* de Inria. Il est accompagné d'un README en plus de commentaire dans le code. En utilisant les résultats et la documentation, il devrait être rapide de reprendre le travail réalisé.

5.3 Perspectives

Il reste plusieurs pistes d'amélorations à partir du travail réalisé.

— L'outil de remémoration se fait en deux étapes, une première sur le principe de l'indexation. La deuxième utilise les résultats de la première et utilise l'algorithme LCS pour en extraire un cas en particulier. Il serait judicieux d'utiliser ici une analyse syntaxique par des outils plus développés afin d'avoir de meilleurs résultats.

— L'adaptation réalisée est très simpliste, de plus elle ne fonctionne pas / très mal si il existe plusieurs bons appariements dans la phrase cible. Le choix devrait ne pas être réalisé arbitrairement.

— Le fonctionnement de l'algorithme en général est plus efficace sur les cas où ont doit ajouter et retirer quelque chose de la phrase cible. Il serait donc bon de repasser sur les différentes étapes et regarder où améliorer le fonctionnement.

— Pour le moment lors de la remémoration, on recalcule à chaque fois les niveaux d'indexation des cas, alors qu'il serait plus intelligent de les stoker dans la base de cas.

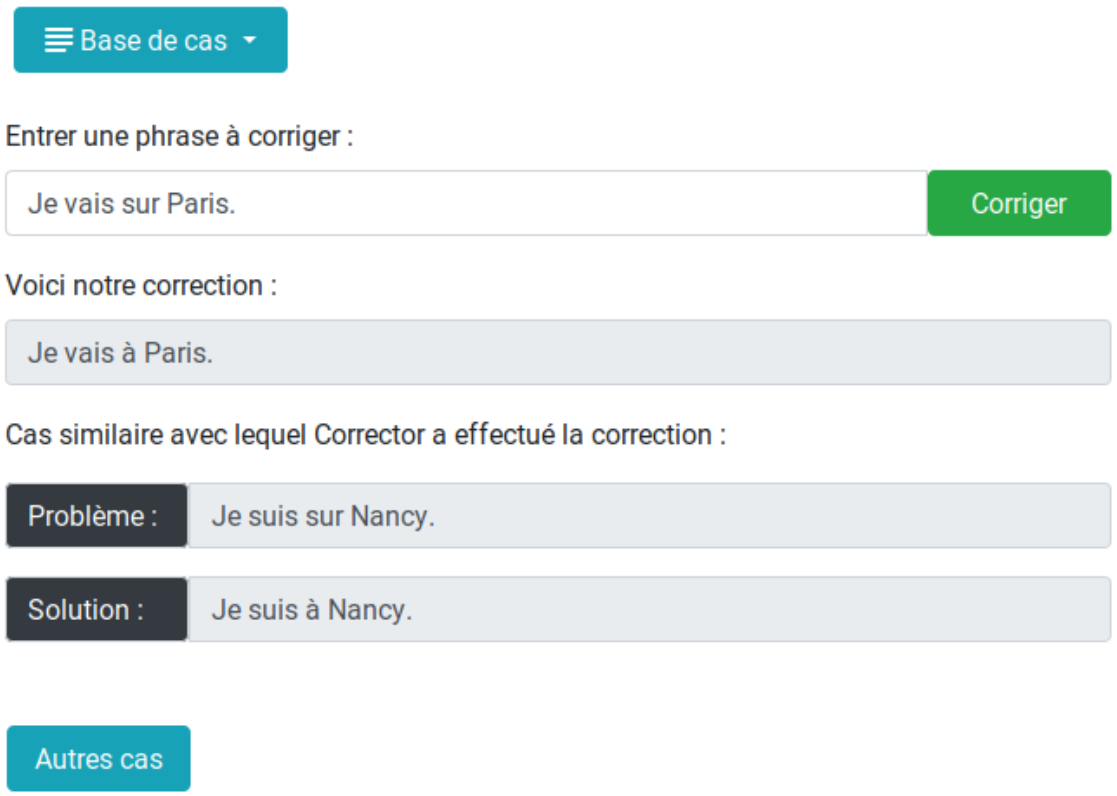
— En fonction de la provenance d'une correction, il est possible de la considérer plus important / fiable qu'une autre. Il faudrait alors prendre la provenance en compte.

— La partie remémoration peut ne pas retourner de cas correspondant à la phrase cible, cela peut poser problème, car cela veut dire que ce n'est pas résoluble pour le moment.

6 Conclusion

Professionel

Le sujet initial a été fini, j'ai par la suite développer un système d'adaptation pour parfaire le projet. Puis avec monsieur Sies, j'ai pu implémenter mon travail sur le site. Voici un visuel du site avec un exemple de correction :



Les principales difficultés ont été sur la recherche, en effet, le travail n'était pas guidé, il fallait avancer par moi-même. Contrairement à presque touts les projets qui ont été réaliser par avant à la faculté. De plus l'étape d'état de l'art ma permis de développer mon esprit critique.

Formation

Comme le projet était orienté recherche et que le travail se fessais de façon itérative, la méthode de travail et de conception qui a été enseigné à la faculté a été utile. Mais dans la plus grosse partie du projet, celui-ci étant en python et sur du travail de chaîne de caractères. Peut de connaissance autres que personnel ne pouvais m'être utile.

Bilan Personel

Le stage ma permit de découvrir l'environnement qu'est la recherche, celui-ci correspondait tout à fait à l'idée que je m'en étais fait. J'ai appris à travailler de façon totalement autonome, avec des pistes d'avancement crée par moi même où lors de réunion. De développer des programmes de manière plus soigné et dans l'optique de pouvoir être réutilisé plus tard par d'autre personne ayant a travailler sur le projet.

Les différentes difficultées que j'ai pu rencontré ont principalement été résolues par de la recherche et correction de bug [9]. Les autres difficultées, ont été résolues lors de réunion et discutions avec les autres membres des équipes participant au projet.

Dans l'ensemble le projet m'a beaucoup plu.

Bibliographie

Références

[1] GEEKSFORGEEKS. *Python Program for Longest Common Subsequence*. https://www.geeksforgeeks.org/python-program-for-longest-common-subsequence/. [Online ; accessed avril-2019].

[2] André GIANG. *Un moteur d'inférences pour la correction à partir de cas de phrases en français*. Rapp. tech. Université de lorraine, faculté des sciences et des technologies, troisième année de licence informatique, Du 9 avril au 6 juin 2018.

[3] David E. Metzener JOHN W. RATCLIFF. *PATTERN MATCHING : THE GESTALT APPROACH*. https://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/1988/8807/8807c/8807c.htm. [Online ; accessed avril-2019].

[4] LORIA. *Site officiel du loria*. http://www.loria.fr/fr/presentation/. [Online ; accessed avril-2019].

[5] Isabelle MORNARD. *Base de cas Corrector*. Rapp. tech. Université de lorraine, Du 21 janvier au 15 février 2019.

[6] PYTHON. *Documentation officielle de Python*. https://docs.python.org/3/. [Online ; accessed avril-2019].

[7] QUENNEC. *Python : MySQL*. https://www.quennec.fr/trucs-astuces/langages/python/python-mysql. [Online ; accessed avril-2019].

[8] Reynault SIES. *Intégration et déploiement d'un système de raisonnement à partir de cas pour la correction de phrases françaises*. Rapp. tech. Université de lorraine, faculté des sciences et des technologies, troisième année de licence informatique, Du 9 avril au 6 juin 2019.

[9] STACKOVERFLOW. *Site stackoverflow*. https://stackoverflow.com/. [Online ; accessed avril-2019].

[10] WIKIPEDIA. *Page wikipedia du loria*. https://fr.wikipedia.org/wiki/Laboratoire_lorrain_de_recherche_en_informatique_et_ses_applications. [Online ; accessed avril-2019].

Annexes

Code d'insertion dans la base de cas

```
    bdc.append(Cas.Cas("Tu ne danse pas la valse.",
                       "Tu ne danses pas la valse.",
                       Explication.Explication(11, "add", "s")))
    bdc.append(Cas.Cas("Il a trop de bruit pour travailler.",
                       "Il y a trop de bruit pour travailler.",
                       Explication.Explication(3, "add", "y ")))
    bdc.append(Cas.Cas("Je crois qu'on a rien à cacher.",
                       "Je crois qu'on n'a rien à cacher.",
                       Explication.Explication(15, "add", "n'")))
    bdc.append(Cas.Cas("Il a essayer.",
                       "Il a essayé.",
                       Explication.Explication(10, "add", "é")))
    bdc.append(Cas.Cas("Pierre a mangée la tarte.",
                       "Pierre a mangé la tarte.",
                       Explication.Explication(14, "def", "e")))
    bdc.append(Cas.Cas("Ils son fous.",
                       "Ils sont fous.",
                       Explication.Explication(7, "add", "t")))
    bdc.append(Cas.Cas("Je vais mangé.",
                       "Je vais manger.",
                       Explication.Explication(12, "add", "er")))
```

Dans le dossier `/Rapport-Julien_Romary/Test_Rememoration/Test_Indexation/` se trouve un programme python pour la démonstration du code

Implémentation de la comparaison des petit mots

```
# Creation d'un cas fictif a partir de la phrase cible
casTemp = Cas.Cas(str1, "", None)
# recuperation de mot de la phrase cible
tab_mot_cible = casTemp.longMots()
scoreMax = 0
# cas resultat de la rememoration
res = None
# pour chaque cas de la base de cas
for cas in bdc.get():
    score = 0
    # on recupere les mots du cas
    tab_mot_cas = cas.longMots()
    for i in range(26):
        # on parcours les mot dans l'ordre de leurs longueur
        for motCible in tab_mot_cible[i]:
            for motCas in tab_mot_cas[i]:
                # on cherche des mot identique dans la phrase cas
                if motCible == motCas:
                    # plus le mot est petit plus il vaut de points
                    score += (26 - i)
                else:
                    # Si le mot ne correspond pas on baisse le score
                    # pour ne pas avoir un cas qui contiens tout les
                    # mots
                    score -= 1
    if score > scoreMax:
        scoreMax = score
        res = cas
```

`res` Contient alors le cas identifier, si `res` est égale a `None` cela veut alors dire que aucun cas satisfaisant a été trouver dans la base de cas.