Génération automatique de contrepèteries

Implémentation d'une baseline

Guersande CHAMINADE

Master 2 pluriTAL – année 2016 / 2017

Mémoire encadré par Kim GERDES (Sorbonne Nouvelle – Paris 3) et Sylvain KAHANE (Université Paris X – Nanterre)

Jury : Kim GERDES, Sylvain KAHANE, Damien NOUVEL (INALCO)

Table des matières

Introduction

Qu'est-ce que la contrepèterie ?

Pouvant être considéré comme un jeu de mots, la contrepèterie est un mécanisme de permutation de sons contenus dans une phrase permettant d'en produire une autre. Même si le mécanisme en lui-même ne l'exige pas, il est d'usage que le contrepet (le résultat de la permutation) ait un sens grivois ou indécent, que la phrase (ou séquence de mot) initiale ne possède pas. Le mécanisme repose uniquement sur la dimension phonétique de la phrase en input, et pas sur son orthographe. J. Martin (2000, entres autres) la définit comme « l'art de décaler les sons », ce qui est en soi une contrepèterie (solution : « l'art de dessaler les cons »).

On trouve de multiples recueils de contrepèteries (éventuellement enrichis d'autres jeux de mots), contenant souvent un historique : « La redoute des contrepèteries » (L. Perceau et J. Touchet), « Quand le français s'amuse avec ses… maux: calembours, holorimes, contrepèteries et tutti quanti » (R. Druetta), « La bible du contrepet: une bible qui compte pour décaler les sons: 2000 contrepèteries du patrimoine, 18000 contrepèteries nouvelles » (J. Martin), ou encore « Sur l'album de la comtesse: 1979-1987: 2300 contrepèteries parues dans le Canard enchaîné et 500 inédites » (J. Martin, Y. Audouard), pour ne citer qu'eux. D'autres travaux comportent une dimension analytique littéraire ou linguistique, tel que le « Que sais-je ? » rédigé par J. Martin, ou encore l'étude d'A. Rabatel qui envisage la contrepèterie, sous un angle énonciatif, comme une confrontation de points de vue.

En revanche, il n'existe encore pas, du moins à notre connaissance, d'étude menée sur la contrepèterie dans le domaine du traitement automatique des langues, et plus précisément dans la génération automatique de textes (que ce soit pour le français ou d'autres langues). Nous proposons donc un premier travail sur la question. Il s'agit, dans un premier temps, de caractériser le mécanisme de la contrepèterie à différents niveaux (phonétique, syntaxique, morphologique et sémantique) afin de déterminer comment convertir ces particularités en contraintes à poser sur la génération automatique de texte. Dans un second temps, nous avons implémenté une première ébauche de générateur automatique de contrepèteries, en prenant soin d'en analyser les limitations afin de baliser la suite des travaux à mener.

La contrepèterie en regard des jeux de mots

On peut classer les jeux de mots en plusieurs catégories, selon le mécanisme et le niveau de représentation de la phrase sur laquelle il s'applique. On peut distinguer :

- Les jeux de mots reposant sur la graphie : les acronymes récursifs (« GNU » : « GNU is Not Unix »), les acrostiches (strophe(s) à partir de laquelle ou desquelles il est possible de reconstituer un mot ou une expression si l'on conserve la première lettre ou les premiers mots de chaque vers) , les ambigrammes (mots qui, par symétrie, peuvent se lire soit à l'identique, soit former un autre mot), les palindromes, les dingbats (rébus dans lesquels la graphie est utilisée pour sa dimension visuelle, par exemple « lesnuatêteges » ↔ « la tête dans les nuages » ou encore « le chaperon » ↔ « le petit chaperon rouge »), les panagrammes (des phrases ou séquences de mots qui contiennent toutes les lettres de l'alphabet), le pendu (un mot dont on ne connaît que le nombre de lettres au départ, et le joueur doit deviner les lettre en question ; il a un nombre de tour limité, et pour chaque lettre proposée qui n'est pas dans le mot, un élément d'un pendu est dessiné ; le joueur a perdu si le pendu est dessiné en entier avant qu'il ait trouvé le mot), etc.
- Les jeux de mots reposant sur la phonétique : le calembour, les rimes, l'homophonie, la paronymie, le rébus, la charade, les virelangues (séquences réputées difficiles à prononcer, par exemple « Un chasseur sachant chasser doit savoir chasser sans son chien. »)…
- Les jeux de mots reposant sur la construction d'un troisième mot à partir des deux premiers : les mots-valises (« adulte » + « adolescent » → « adulescent ») et les duomots (deux mots concaténés en forment un autre : «pet » + « gaz » → « Pégase »), par exemple.

Ces trois catégories ne sont pas mutuellement exclusives : le jeu de mots correspond à une fantaisie de la part de celui ou celle qui le produit, en conséquence de quoi rien n'empêche le locuteur de les combiner. Plusieurs jeux de mots mentionnés ci-dessus comme jouant sur la phonétique reposent également en partie sur la construction d'un mot. C'est le cas, entre autres, des rébus, charades, et contrepèteries.

Certaines définitions de la contrepèterie proposent de poser des critères formels au niveau du mécanisme de permutations afin de distinguer la contrepèterie d'autres jeux de mots similaires.

Ici, « mot » est entendu dans une acception large, sans entrer dans le débat des mots composés sans tirets (« pomme de terre ») ou des collocations : séquence de caractères suivie et précédée par un espace typographique.
Par exemple, il est possible d'imposer que les éléments à permuter appartiennent à des « supports disjoints »[1], c'est à dire qu'on ne puisse pas exécuter plusieurs permutations à l'intérieur d'un même mot. La question de la segmentation en mots est évidemment plus complexe ; nous y reviendrons ultérieurement. Cette définition de la contrepèterie permet de distinguer entre la contrepèterie et l'anagramme. Ainsi, la paire « le groupe affine » / « le gouffre à pine » ne serait pas acceptée. Le texte produit résulterait de deux permutations au sein du substantif et du verbe.

La contrepèterie en français et dans d'autres langues

Ce mécanisme de jeu de mots existe également dans d'autres langues. Cependant, il semble que le caractère grivois soit moins important qu'il ne l'est dans la définition de la contrepèterie en français.
En anglais, on retrouve le « spoonerism », proche du lapsus, du nom du Révérend William Archibald Spooner, dans les sermons duquel on retrouvait des contrepets. En espéranto, on retrouve le « kontraŭknalo ». Quelques exemplaires sont également disponibles en espagnol, en italien et en allemand, bien que le phénomène soit plus rare[2].

Si c'est le cas dans d'autres langues (le finnois, par exemple), en français, le rapport entre les graphèmes et les phonèmes n'est pas bi-univoque : il n'y a pas de correspondance systématique.
A titre d'exemples :

- le son [o] correspond aux graphies <o>, <au>, <eau>, <aux>, <eaux>, <haut>, <hauts>, <oh>, <ô> et <os>.
- Le graphème <g> peut correspondre aux phonèmes /ʒ/ et /g/, être associé à un autre graphème pour produire un phonème (<agneau> se transcrit par /aɲo/), ou ne pas être prononcé et donc ne pas être transcrit (<étang> se transcrit par /etã/).

L'attention sera donc portée sur les transcriptions phonétiques des phrases et non sur leur orthographe. Autrement dit, les permutations seront effectuées sur ces transcriptions phonétiques (et la vérification sur les transcriptions orthographiques).

1 https://fr.wikipedia.org/wiki/Contrepèterie
2 https://fr.wikipedia.org/wiki/Contrepèterie

Humour et génération automatique de texte

Si l'étude de l'humour n'est pas particulièrement récente dans plusieurs domaines des sciences humaines, comme la psychologie, la cognition ou même la linguistique, il reste encore beaucoup de travail à faire dans les domaines du traitement automatique, que ce soit du point de vue de la détection ou de celui de la génération. D'autre part, les différentes langues semblent inégalement représentées. Une étude menée au début de novembre 2016 par Santiago Castro, Matías Cubero, Diego Garat et Guillermo Moncecchi révèle par exemple qu'il n'y avait jusque là aucune étude menée sur la reconnaissance automatique de l'humour en espagnol, mais mentionne plusieurs études menées sur l'anglais, en les classant d'après certaines caractéristiques spécifiques à l'analyse de l'humour (sans forcément être guidé par des visées d'implémentation). Ils présentent, entre autres, les paramètres suivant :

- D'après Mihalcea et Strapparava (2005), le registre argotique « pour adulte » connaît un grand succès dans le domaine de l'humour, parce qu'il touche à des domaines tabous tels que la sexualité, et que les sujets tabous donnent typiquement lieu à des stratégies de défoulement et de soulagement. Ils proposent d'utiliser des extensions de Wordnet étiquetés en domaines (par exemple celui de la sexualité) afin de repérer des mots y appartenant dans des textes. Le cas de la contrepèterie pourrait tout à fait s'inscrire dans ce cadre.

- Mihalcea et Strapparava envisagent les jeux de mots basés sur les sonorités, tels l'allitération, comme une généralisation du phénomène de rime. Ils expliquent que les propriétés structurelles et phonétiques de ces jeux de mots sont primordiaux. Là aussi, le lien avec la contrepèterie est assez fort.

- L'ambiguïté joue également un rôle prépondérant, envisagée dans la théorie de la résolution de l'incongruité (Ritchie 1999): il s'agit d'avoir, pour un même texte, plusieurs interprétations possibles. Là encore, on peut voir un lien avec la contrepèterie : le caractère grivois de certains contrepets peut être envisagée comme une ambiguïté que les participants choisissent de résoudre en choisissant systématiquement le sens grivois (ce point est développé dans la présentation du module de filtrage sémantique). L'incongruité est souhaitée, et se situe au niveau du contraste entre l'innocence de la phrase en input et le caractère tabou de son contrepet.

- La fréquence d'apparition plus élevée de certains mots dans un contexte humoristique.

> On peut rapprocher cette caractéristique de celle de l'argot « pour adultes ». Sjöbreg et Araki proposent par exemple une approche d'apprentissage automatique de classification de phrase courtes en deux classes (blagues vs. phrases banales) s'émancipant d'une compréhension sémantique. Ils proposent d'utiliser une combinaison simple de propriétés (telles que l'existence de mots en commun avec d'autres blagues, la présence de mots fréquemment relevés en contexte humoristique, l'ambiguïté et le recouvrement partiel de tournures idiomatiques) qui, en nombre suffisant permettent de classifier un input comme étant une manifestation d'humour.

La plupart des travaux de génération automatique d'humour portent sur les jeux de mots ('puns' en anglais) et s'inscrivent dans le cadre de l'interaction homme-machine (chat-bots).

Pawel Dybala, Michal Ptaszynski, Shinsuke Higuchi, Rafal Rzepka et Kenji Araki (2008) démontrent que l'ajout d'un générateur de jeu de mots (japonais) améliore l'expérience de dialogue homme-machine en comparant les évaluations de l'expérience avec et sans génération de jeux de mots. Ils utilisent le générateur PUNDA (Dybala et al. 2010), qui permet de générer des jeux de mots en japonais selon quatre patrons dans un premier temps, auxquels ils en ont ajouté trois. L'ensemble de ces patrons repose soit sur la phonétique, soit sur les transformations de mores (que l'on peut voir comme une unité intermédiaire en japonais entre le phonème et la syllabe).

Citons également une étude se focalisant sur des contraintes très similaires à celles que pose la contrepèterie. Il s'agit d'un travail mené par Alessandro Valitutti, Antoine Doucet, Jukka M. Toivanen et Hannu Toivonen (2016), dans lequel les auteurs envisagent la génération de textes humoristiques par la substitution d'un seul mot dans un input court. Ils dégagent trois types de contraintes pour cette substitution lexicale : la similitude de sonorité entre le mot original et son substitut, les contraintes sémantiques de sens ou de connotation tabou posées sur le mot candidat, et les contraintes relatives au contexte et à la position du mot sur lequel on applique une substitution. Les résultats de leur étude montrent, d'une part, que la contrainte de l'appartenance du mot à un lexique tabou est statistiquement très efficace, et d'autre part que l'interaction est très forte entre le caractère tabou du mot et la position de celui-ci.

Dans un premier temps, nous présenterons le corpus utilisé pour une analyse exploratoire du phénomène de la contrepèterie, afin de définir des propriétés phonétiques, morphosyntaxiques et

sémantiques. Nous décrivons dans un deuxième temps chacun des modules utilisés dans le prototype, de la génération des candidats au filtrage sémantique, avant de proposer une discussion sur l'architecture du système. Enfin, nous présentons les premiers résultats obtenus avec le prototype de génération de contrepèteries, en tentant d'expliquer ses limites en l'état actuel.

Constitution d'un corpus pour l'étude du mécanisme de la contrepèterie et de sa modélisation

Le corpus d'étude, fourni en annexe, du mécanisme de permutations comprend une centaine de contrepèteries. Elles ont été sélectionnées à partir de deux sources : un recueil de contrepèteries (Martin 2005), et plusieurs sites internet. En dehors de contrepèteries littéraires (ce type de jeu de mots ayant été assez populaire chez les écrivains des XVIème, XXème et XXIème siècles), ce sont souvent les mêmes paires input / contrepet que l'on retrouve. Le but de la constitution de ce corpus n'est pas l'exhaustivité, cependant il contient des paires dont le contrepet possède un sens grivois, mais également d'autres, dites « pour enfants », où seul le mécanisme de permutation de sons entre en jeu, la dimension grivoise étant laissée de côté (par exemple : « La vie des mots3 / « L'ami des veaux »).

Pour chaque paire input / contrepet, plusieurs paramètres ont été examinés :

- le nombre de permutations que comprend le mécanisme
- les paires d'éléments constituant chaque permutation
- la position dans la structure syllabique du ou des phonème(s) constituant les éléments de permutation
- la place des syllabes contenant l'élément dans le mot contenant la syllabe.

Par exemple, considérons la paire « Le lac des sapins » / « le sac des lapins » :

- Elle contient une permutation.
- La paire de sons constituant cette permutation est /l/ - /s/.
- Chacun de ces deux phonèmes représente l'attaque consonantique de la première syllabe du mot (ou la seule, étant donné que le mot « sac » est monosyllabique).

Si on se limite au mécanisme de permutations comme définition de la contrepèterie, on peut imaginer que le nombre de permutations n'est pas plus limité que ne l'est le nombre de mots d'une phrase : plus la phrase est longue, plus le nombre de possibilités augmente, et ce pour plusieurs raisons. Le nombre de mots influence le nombre de positions possibles pour chaque permutation, et donc également le nombre d'orthographes à vérifier pour chaque candidats (le nombre de mots à vérifier est multiplié par le nombre d'orthographes possibles pour chaque

séquences de phonèmes).

Néanmoins, la contrepèterie n'est pas qu'un mécanisme de permutations. Sans même tenir compte de la question de la grivoiserie du contrepet, il faut garder à l'esprit que la contrepèterie est un mécanisme qui n'existe pas en dehors de l'usage. Ce qui implique que des limites cognitives s'appliquent nécessairement sur le nombre de possibilités. De même qu'une phrase peut contenir, théoriquement, un nombre illimité d'enchâssements, subordinations et coordinations, si le contrepet est produite pour être comprise, elle ne doit pas être trop longue, et le nombre de permutations ne doit pas être trop important.

L'examen du corpus a permis de faire plusieurs observations limitant le nombre de possibilités.
Le nombre maximal de permutations observées est de 3 :

« A l'hôtel du B**lé** d'**Or**, il y a de superbes **ch**ambres au **m**ois » / « A l'hôtel du b**or**d**el**, il y a de superbes **m**embres au **ch**oix ».

- première permutation : « or » ↔ « lé »
- deuxième permutation : « m » ↔ « ch »
- troisième permutation : « l » ↔ « é »

Or le corpus ne contient qu'un seul exemple de contrepèterie à trois permutations. Cela ne signifie pas qu'un patron de permutations à trois permutations n'est pas réaliste, simplement qu'il est, du moins dans le corpus, nettement moins fréquent.

La majeure partie des contrepèteries comportent une ou deux permutations. Le corpus contient 13 contrepèteries à 2 permutations, et 99 contrepèteries à une seule permutations. De plus, la majeure partie des contrepèteries à deux permutations peuvent être décomposées en deux contrepèteries à une seule permutation, parce que les éléments ne sont pas croisés.

Par exemple :

« A force de [**m**ouiller les **f**iches]$_1$, je suis arrivé au [**f**ond de la **c**olle]$_2$ » / « A force de **f**ouiller les **m**iches, je suis arrivé au **c**on de la **f**olle ».

Dans le principe, rien n'empêche que les permutations ne se croisent, cependant notre corpus

n'en contient pas : chaque paire, à l'image de celle présentée ci-dessus, se voit appliquer une permutation à l'intérieur d'un syntagme. Dans cet exemple, il s'agit d'un syntagme verbal [1] et d'un syntagme nominal [2]. Parce que les syntagmes verbaux ou prépositionnels / nominaux constituent l'essentiel d'un texte, la majeure partie des permutations ont lieu à l'intérieur de ces syntagmes.

D'autre part, les permutations semblent s'opérer sur des mots pleins. Ce qui peut se révéler utile pour limiter le nombre de variations à produire. On entend ici par mot plein l'ensemble des mots ayant un sens propre, répartis en quatre catégories grammaticales : substantifs et noms propres, adjectifs, adverbes et formes verbales. Cette considération renvoie à la question de la segmentation des unités lexicales : si l'on examine la préposition complexe « à force de », peut-on considérer « force » comme mot plein, et subir une permutation ? Des éléments de réponse sont donnés au fil de cet étude.

En ce qui concerne les éléments constituant les permutations, le corpus présente plusieurs possibilités, examinées en terme de structure syllabique (ce point est développé ultérieurement). Ils peuvent être :

- une attaque consonantique simple : « le **l**ac des **s**apins » (« le **s**ac des **l**apins »);
- une attaque consonantique complexe : « Ce que votre **pl**ante me fait » (« Ce que votre fente me **pl**aît »);
- la première consonne d'une attaque consonantique complexe : « Ce coup de **b**lanc m'a grisée » (« Ce coup de gland m'a **b**risée »);
- un noyau : « L'éc**a**ille de ces m**ou**les » (« Les c**ou**illes de ces m**â**les »);
- une coda simple : « Bê**t**e à vous échauffer la bi**l**e » (« Be**ll**e à vous échauffer la bi**t**e ») ;
- la seconde consonne de la coda complexe : « Brigitte aime les ver**t**es plages » (« Brigitte aime les verges pla**tes »).**

Ces éléments peuvent être combinés :

- un noyau + coda complexe : « Ces c**âbles** ne valent pas un rond ».

Selon le contrepet que l'on vise, on peut choisir un autre patron. Si le contrepet est « ces cons ne valent pas un râble », on doit avoir recours à l'élément ci-dessus. En revanche, si le contrepet est « Ces râbles ne valent pas un con », les deux éléments utilisés par la permutation sont des attaques consonantiques simples.

Les permutations les plus fréquentes s'opèrent sur les attaques consonantiques (63,7 % des cas),

avec donc les éléments suivants, par fréquence d'apparition : attaque consonantique simple (57,5 % du total soit 85,5% des permutations sur les attaques consonantiques) , attaque consonantique complexe, une des consonnes de l'attaque consonantique complexe. Ensuite viennent les permutations sur les noyaux et les syllabes complètes. Les permutations sur les codas, qu'elles soient simples ou complexes, ne concernent que deux paires (voir le corpus en annexe).

Le corpus permet la détection d'une autre propriété de la contrepèterie : la similitude syntaxique entre l'input et le contrepet. Observons par exemple les paires :

- « Décaler les sons » / « Dessaler les cons » ;
- « Le lac des sapins » / « Le sac des lapins » ;
- « J'ai glissé dans la piscine » / « J'ai pissé dans la glycine ».

On constate que les paires input / contrepet ont la même structure, ou du moins des structures similaires :

- Vinf Det N / Vinf Det N ;
- Det N Prep N / Det N Prep N ;
- Pro Aux Vppart Prep Det N / Pro Aux Vppart Prep Det N.

Les deux structures du premier exemple témoignent d'une double ambiguïté au niveau de « cons ». La première est causée par l'homophonie / homographie (« cons » considéré comme un synonyme familier ou vulgaire de « bête », « idiot », ou bien « con » comme désignant dans un registre familier ou vulgaire le vagin). La seconde est d'ordre plus syntaxique : il s'agit de l'emploi substantivé des adjectifs (« un débile », « la folle », etc.). Cette ambiguïté est traitée plus en détails ultérieurement.

Rappelons que le nombre de possibilités de candidats à la contrepèterie pour une même phrase peut être énorme. A titre d'exemple, si l'on ne tient compte que des éléments de permutation des deux permutations en imposant des contraintes sur le nombre de syllabes des mots contenant les éléments, l'input « A force de mouiller les fiches, je suis arrivé au fond de la colle » génère plus de 1 800 candidats (dont la plupart sont des suites de mots sans aucun sens). La vérification de ces candidats, si elle reste possible, peut être très lente. Il faut donc éliminer le plus de candidats le plus tôt possible.

On peut donc se demander comment exploiter les propriétés phonétiques et de similitudes

syntaxiques imposées par la contrepèterie pour vérifier l'acceptabilité des candidats générés. Cette propriété est-elle suffisante ? Si non, quelles autres vérifications doivent être faites afin de vérifier l'acceptabilité des candidats générés ?

Présentation des différents modules

Le prototype présenté dans cette étude utilise cinq modules, présentés brièvement dans le schéma de la page suivante. En voici la légende :

- Chacun des modules est représenté par un bloc gris, et prend en input l'unité mentionnée en marron (hormis le premier, dont l'input coïncide avec celui du système).
- Les ressources utilisées sont indiquées par les rectangles bleus (sur la droite).
- L'input du système dans sa globalité est représenté par l'élément en violet (en haut à gauche) et l'output du système, qui coïncide avec l'output du module de filtrage sémantique, est en vert (en bas).

Le système est conçu pour tester plusieurs patrons de permutation de manière successive. Le schéma est donc parcouru plusieurs fois jusqu'au quatrième module (avec pour input une phrase et un patron), autant de fois que l'on a de phrases et de patrons à tester, avant d'appliquer une seule et unique fois le module de vérification sémantique.

Chacun des modules est présenté en détails dans les sous parties suivantes, dans le même ordre que dans le schéma. Une discussion est ensuite proposée sur la modularisation et l'interaction entre les différents modules de l'architecture proposée.

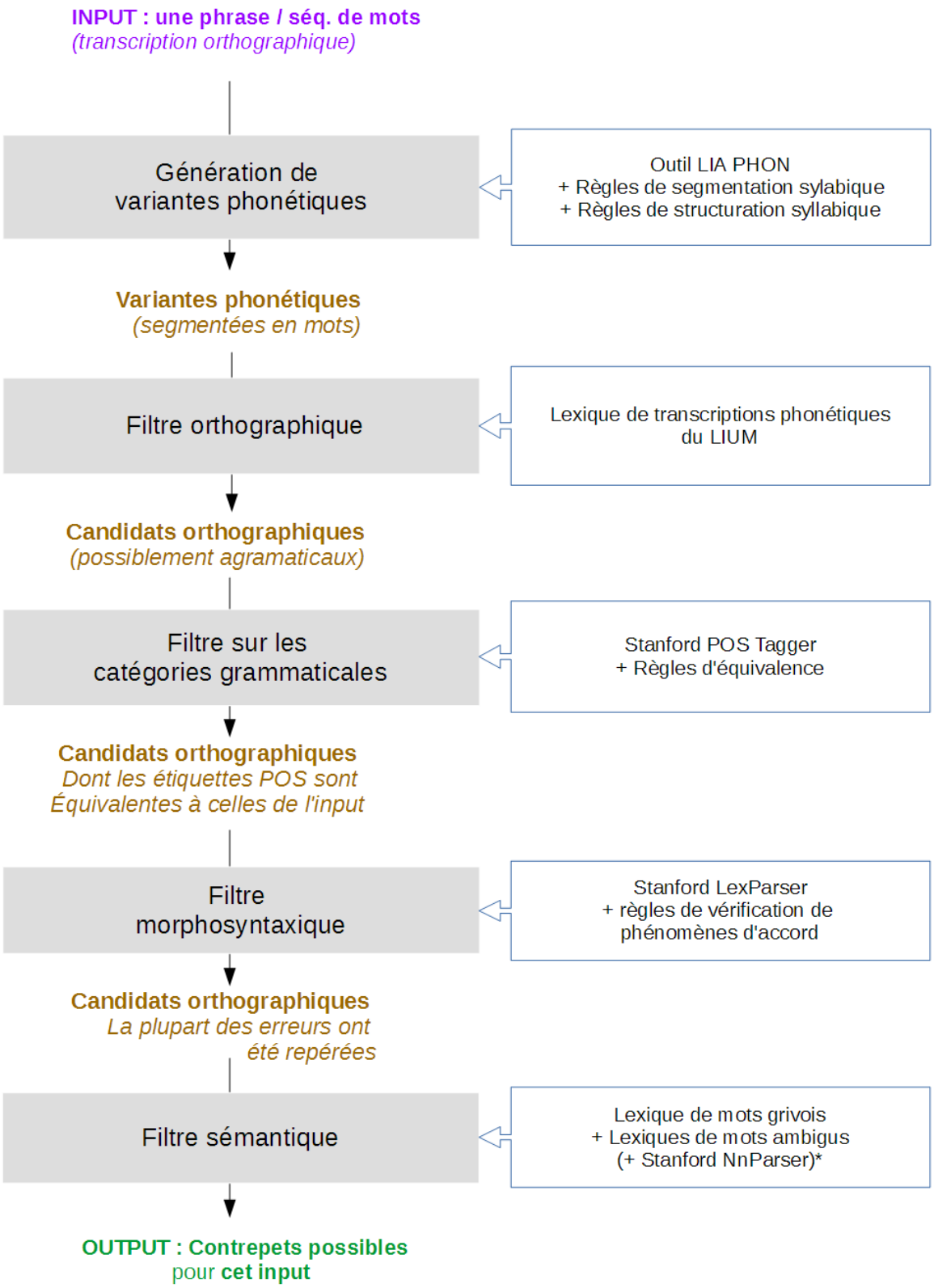

Illustration 1: Architecture du système proposé

Module 1 : Génération de variations phonétiques

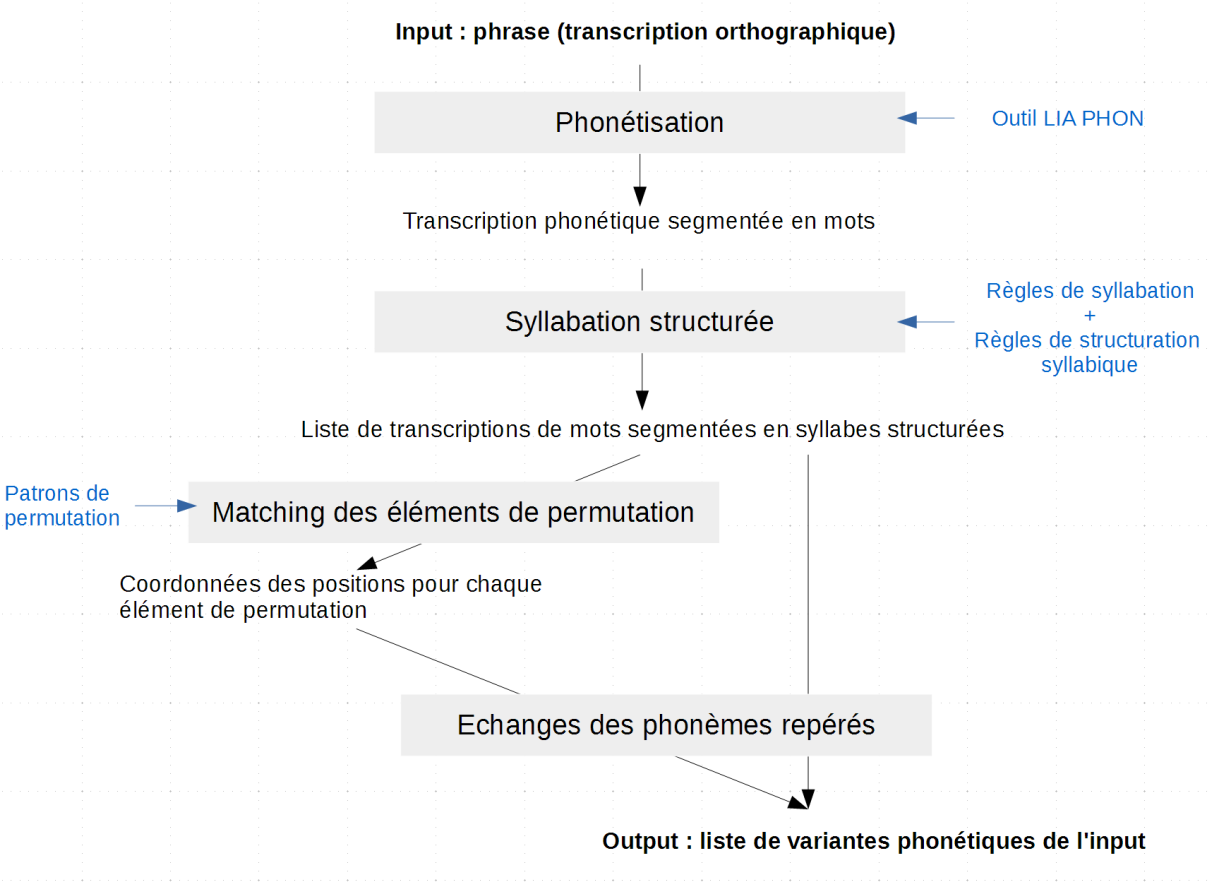

Illustration 2: Génération des variations phonétiques

Phonétisation ou phonémisation de l'input ?

La phonétisation de l'input n'est pas, dans le cadre de cette étude, une fin en soi. Si elle reste très importante parce que la qualité de sa sortie conditionne tout le reste de la chaîne de traitement, le but n'est pas de rendre compte de variations de prononciation. Ces variations pourraient être prises en compte dans une version ultérieure, notamment lorsqu'aucun contrepet n'a été trouvé pour un input.

Cependant il est tout de même souhaitable de pouvoir rendre compte de phénomènes dépassant la simple transcription graphème – phonème. On pense par exemple aux phénomènes de coarticulation tel que le dévoisement, ou les liaisons.

Considérons les deux exemples suivants :

- « Je suis une phrase de test » ;

- « Je suis absent ».

Afin que la transcription soit exploitable pour le repérage des éléments de permutation, il faut que, dans le premier cas, la liaison qui peut se faire entre « suis » et « une » soit repérable, et, dans le second cas, que l'assourdissement du « b » soit pris en compte. On rappelle que les permutations se font d'après la dimension phonétique de l'input, pas d'après son orthographe. Il faut donc que la fonction de matching reconnaisse que la séquence de caractères « absent » est prononcée /apsã/ et non /absã/.

A cela s'ajoutent, entre autres, des phénomènes tels que la phonétisation des noms propres, des sigles, ou encore la désambiguïsation entre plusieurs prononciations possibles pour une même chaîne de caractères. La solution à adopter pour la phonétisation doit donc tenir compte de ces traitements linguistiques, qui dépassent un système de règles de transcription graphème-phonème. On peut donc considérer qu'il s'agit de phonétisation de l'input.

Phonétisation de l'input

Le choix s'est porté sur l'outil de phonétisation LIA PHON. Cet outil a été développé par Frédéric Béchet à l'université d'Avignon (Laboratoire Informatique d'Avignon). Il s'agit d'un ensemble de scripts permettant différents traitements : formatage de texte, étiquetage et accentuation, transcription phonétique et conversion des transcriptions au format MBROLA.
La transcription phonétique se fait avec un ensemble de règles, en suivant ces étapes : phonétisation de l'input étiqueté en catégories grammaticales, insertion des liaisons entre les mots, décision quant à la lecture des acronymes, calcul de la prononciation des noms propres, et enfin vérification de la présence éventuelle d'exceptions aux règles de transcriptions.

Si l'on reprend les deux exemples précédents, on constate qu'avec cet outil, les transcriptions sont les suivantes (deux caractères ASCII par phonème) :

- « Je suis une phrase de test » est transcrit par *jjee ssuyii zzuunn ffrraazz ddee ttaissttee*
- « Je suis absent » est transcrit par *jjee ssuyii aappssan*

Ces exemples nous montrent plusieurs choses. Premièrement, la liaison entre les mots « suis » et « une » est faite dans le premier cas, mais pas dans le second (entre « suis » et « absent »). Ensuite, l'assourdissement du « b » de « absent » est réalisé. Ceci montre que la transcription

réalisée dépasse un système de conversion graphèmes – phonèmes. Lorsque plusieurs transcriptions sont possibles (par exemple celle pour « je suis une phrase de test », ci-dessus), l'outil n'en fournit qu'une.

En outre, l'outil propose deux échelles de segmentation intéressantes. La première est celle qui correspond au mot. Utilisée dans cette étude, elle a tout de même demandé quelques petites adaptations. La segmentation en mots tient compte de l'étiquetage en catégories grammaticales des tokens du texte. Par exemple, « au fond de » est reconnu comme une seule unité, et dans la sortie est présentée avec des underscores : « au_fond_de auffonddee ».

Cela soulève la question de la segmentation en mots. Le mécanisme de la contrepèterie ne s'embarrasse pas d'une analyse syntaxique aussi fine, et il est donc préférable de considérer « au fond de » comme trois mots séparés, étant donné que chacun des termes constituant une expression telle que celle ci, et existant par ailleurs de manière indépendante en tant que mot plein, peuvent être sujet à une permutation.

Ainsi, les tokens présentant des underscores en sortie de phonétisation ont été resegmentés, et l'étiquetage en catégories grammaticales n'a pas été conservé. Leur transcription phonétique a été retrouvée grâce à un lexique de correspondances entre formes orthographiques et formes phonétiques. Il s'agit d'un lexique établi par le LIUM (Laboratoire d'Informatique de l'Université du Maine).

Ce lexique a été choisi en raison du mode de représentation des phonèmes, identique à celui utilisé par l'outil LIA PHON : deux caractères ASCII par phonème, comme on peut le voir dans les exemples ci-dessus. Deux dictionnaires ont été construits à partir de ce lexique. Le premier permet d'aller de l'orthographe à la phonétique, et comporte 65 373 entrées, et le second, permettant d'aller de la phonétique aux orthographes correspondantes, en contient 77 262. Le fait que le second dictionnaire contient plus d'entrées découle directement des différentes possibilités de prononciations pour chaque orthographe. Cela inclue les liaisons, mais aussi les différentes possibilités de prononciation du schwa.

Considérons par exemple les transcriptions phonétiques possibles pour le pronom personnel de troisième personne du pluriel, au féminin :

```
elles ai ll
```

```
elles(2) ai ll ee
elles(3) ai ll ee zz
elles(4) ai ll zz
elles(5) tt ai ll
elles(6) tt ai ll ee
```

Chaque entrée de ce lexique est construite comme ceci : forme orthographique, éventuellement numéro de l'entrée lorsqu'il y en a plusieurs pour une même forme orthographique (les formes uniques et les premières formes ne sont pas numérotées), transcription segmentée en phonèmes. On constate que dans cet exemple, pour une même graphie on dispose de cinq transcriptions possibles, selon qu'on prenne en compte ou non les liaisons avec le mot antérieur ou le schwa.

Une autre fonctionnalité intéressante de l'outil LIA PHON est la segmentation en syllabes. Or, si cette idée est tentante, la segmentation présentée en sortie est faite sur la transcription orthographique. Des tests ont montré que la concaténation de transcription de ces syllabes ne donne pas nécessairement la même séquence de phonèmes que la transcription du mot complet.

Voici la sortie de la phonétisation associé à la segmentation en syllabes pour « segmenter » :

- segmenter ssaiggmmanttei [VINF] seg men ter

Si l'on utilise le même outil pour retranscrire chacune des syllabes, on obtient :

- seg ssaigg [NMS]
- men mmainn [NMS]
- ter ttairr [VINF]

Ainsi, en concaténant les transcriptions de ces trois syllabes, on obtient *ssaiggmmainnttairr*, au lieu de *ssaiggmmanttei.* Il y a dans la concaténation des trois transcriptions des phonèmes supplémentaires, qui bloqueraient le matching des éléments de permutations, et donc l'application des patrons. En outre, les étiquettes fournies pour les catégories grammaticales sont un peu douteuses étant donné que ces syllabes n'existent pas de manière isolée.

C'est pourquoi la segmentation en syllabe proposée par l'outil n'est pas exploitable ici. Elle a donc été réalisée dans une étape suivante, indépendamment de l'outil et directement à partir des transcriptions phonétiques segmentées en mots, et ajustées.

Ce phénomène n'a pas été pris en compte dans cette étude, mais il paraît tout de même intéressant de le mentionner : les oppositions entre certaines voyelles, qui sont le siège de

variations de prononciation. Par exemple, « sénateur » est présent dans le lexique du LIUM transcrit par *ssainnaattoerr*. On peut se demander si *sseinnaattoerr* est une transcription valable. Les transcriptions 'ei' et 'ai' correspondent respectivement au /e/ mi fermé et au /ɛ/ mi ouvert. Selon les régions, la réalisation phonétique de « é » peut se faire plus ou moins ouverte. On peut donc se demander s'il ne serait pas judicieux de modéliser une neutralisation de ces oppositions. Étant donné que le but principal de cette étude n'est pas la finesse de la phonétisation, et que le cas ne s'est présenté qu'une fois pour les exemples traités, la seconde prononciation a simplement été ajoutée au lexique.

Syllabation structurée de la transcription phonétique

Comme expliqué précédemment, la transcription phonétique fournie par l'outil LIA PHON est déjà segmentée en mots, et cette segmentation a été ajustée. Il font donc maintenant, afin d'effectuer le matching des éléments de permutation, segmenter cette transcription en syllabes structurées. Ceci se fait en plusieurs temps. Premièrement, pour chaque token, chaque phonème est étiqueté selon qu'il s'agisse d'une consonne, d'une voyelle, ou d'une semi-voyelle. Ensuite, des règles de segmentation en syllabes sont appliquées. Enfin, chaque syllabe est structurée selon une structure attaque – rime (noyau – coda).

Structure syllabique

L'attaque peut être vide, la syllabe commence alors directement par le noyau. C'est cette place qui est remplie lorsque sont réalisés les phénomènes de liaison. Considérons l'exemple suivant :

- « C'est une phrase » *ssai ttuunn ffrraazz*

La transcription phonétique de « une », *ttuunn*, peut être structurée comme ceci :

- attaque : *tt*
- noyau : *uu*
- coda : *nn*

L'attaque consonantique est le résultat de la liaison entre les deux mots. Cela rejoint la présentation des transcriptions dans le lexique du LIUM, dans lequel les liaisons matérialisées sont celles en début de mot.

Étant donné que ces phénomènes de liaisons ont été traités par l'outil LIA PHON, la structure retenue pour modéliser la structure syllabique des transcriptions est attaque-noyau-coda. Le niveau intermédiaire (la rime en tant que telle et pas uniquement les éléments qui la composent)

pourrait être utile pour certaines permutations, mais le corpus n'en présente qu'une réclamant ce paramètre ; nous avons donc opté pour une structure plate dans cette étude.

Caractérisation des phonèmes et règles de segmentation

Chaque phonème est examiné pour être classé dans l'une des trois catégories suivantes :

- noyau : ce qui peut servir de noyau syllabique ; en français, les voyelles ("ii", "ei", "ai", "aa", "oo", "au", "ou", "uu", "EU", "oe", "eu", "an", "un", "on", "ee", "in"),
- consonnes occlusives et fricatives ("tt", "pp", "gg", "kk", "mm", "bb", "nn", "ff", "vv", "dd", "jj", "ss", "ch"),
- consonnes liquides et semi-voyelles ("rr", "ll", "ww", "yy").

Ce classement résulte d'une considération des clusters consonantiques du français, qui peuvent être composés de :

- une occlusive suivie d'une liquide (« prêter », « plein », « clan »...)
- une fricative suivie d'une liquide (« vraiment », « flocon », « fromage », « vrombir »…)
- deux (voire trois) occlusives si elles sont en fin de mot (« rapt », « abrupt », « cercle »)

Le dernier cas est particulièrement sensible au traitement du schwa. Le parti a été pris ici d'en tenir compte s'il est présent dans la transcription fournie par l'outil LIA_PHON. Le résultat pourra éventuellement paraître artificiel dans certains cas, mais les calculs faits par l'outil et les besoins relatifs de précisions requise devraient limiter ce problème.

A l'issue de cette catégorisation, à chaque phonème est associée une étiquette (« N », « C » ou « Cl »). Les consonnes associées pour former un cluster sont alors regroupées.

Lorsque l'ensemble des phonèmes d'un token ont été catégorisés, des règles de segmentation en syllabes sont appliquées. Il s'agit de règles proposées par Léon, Bhatt et Baligand (1992). Elles précisent que :

- Si un phonème est une voyelle, c'est un noyau (R1);
- Si deux consonnes se suivent et que C1 est une occlusive et C2 une liquide, elles forment un cluster (R2);
- Si deux consonnes se suivent sans former un cluster, alors C1 appartient à la syllabe de gauche et C2 à la syllabe de droite (R3);
- Si une consonne est entre deux voyelles, elle appartient à la syllabe de droite (= elle forme l'attaque consonantique de la syllabe dont la voyelle de droite est le noyau) (R4);

Le cas des deux (voire trois) occlusives en fin de mot est une conséquence de l'interaction entre le traitement du schwa et la règle (R4).

A l'issue de ce traitement, chaque syllabe est délimitée, et peut donc être structurée. La démarche est alors la suivante :

- Si le premier élément est un noyau, alors il n'y a pas d'attaque consonantique, et le reste des phonèmes appartiennent à la coda ;
- Si le premier élément est un cluster, alors il s'agit de l'attaque consonantique et celle-ci ne peut pas contenir d'autre élément. Parmi les éléments qui suivent, il y a forcément (dans cet ordre) au moins un noyau, et éventuellement une coda composée d'une consonne ou d'un cluster.
- Si le premier élément est une consonne, alors il s'agit de l'attaque consonantique. Le second élément est forcément un noyau, et l'élément suivant la coda, s'il existe.

A ce stade, on dispose d'une transcription phonétique de l'input, segmentée en syllabes structurées. C'est sur cet objet que l'on va matcher les éléments de permutation.

Matching des éléments de permutation

Les permutations sont organisées sous la forme de patrons, qui correspondent aux régularités dégagées lors de l'examen du corpus. Nous avons choisi de parler de patron pour rendre compte de cette régularité. L'intégralité des patrons de permutation implémentées dans la baseline ne comportent qu'une seule permutation. Ils sont disponibles en annexe de ce document.

Éléments et permutations

Rappelons que les permutations sont envisagées comme deux éléments que l'on va échanger. Chacun de ses éléments est caractérisé par trois paramètres :

- le numéro de la syllabe à l'intérieur du mot qui la contient,
- la nature de l'élément en termes de structure syllabique,
- le nombre de syllabes que doit contenir le mot.

Voici un exemple d'élément :

(1,AC,1) ↔ attaque consonantique de la première syllabe d'un mot monosyllabique.
(1,AC,2) ↔ attaque consonantique de la première syllabe d'un mot dissyllabique.

Une permutation utilisant ces deux éléments permet d'obtenir, par exemple, la paire suivante :

« Le lac des sapins » / « Le sac des lapins ».

	Le	sac	des	lapins	
	llee	ssaakk	ddei	llaappin	
	Syllabe 1/1	Syllabe 1/1	Syllabe 1/1	Syllabe 1/2	Syllabe 2/2
	Attaque : ll Noyau : ee	Attaque : ss Noyau : aa Coda : kk	Attaque : dd Noyau : ei	Attaque : ll Noyau : aa	Attaque : pp Noyau : in
Mots pleins		X		X	
(1,AC,1)		MATCH - ss			
(1,AC,2)				MATCH - ll	

Ces critères de définition des éléments peuvent paraître restrictifs, surtout le dernier. Gardons à l'esprit le besoin de limiter le nombre de candidats. L'étude du corpus a montré que certains patrons sont plus fréquents que d'autres, et que, de même, certains éléments de syllabes sont plus souvent sujet à permutation que d'autres. C'est pour ces raisons que l'idée est de cibler dans un premier temps les éléments et patrons plus fréquents (les autres patrons sont testés ensuite). La vérification est plus rapide en procédant ainsi plutôt qu'en générant toutes les possibilités de permutation.

Cette étude ayant une visée préliminaire, seuls certaines permutations ont été modélisées. L'attention a été concentrée sur les combinaisons les plus fréquentes : patrons à une permutation portant sur les attaques consonantiques. Les éléments repérés sont donc tous de nature consonantique (les noyaux peuvent être matchés également, mais aucun patron n'incluant le noyau n'a été implémenté). Il peut s'agit soit d'une attaque consonantique simple, soit d'une attaque consonantique complexe (le cluster en entier), soit une des deux consonnes du cluster (nous n'avons pas de cluster à trois consonnes dans le corpus).

Matching des éléments

Le matching se fait en parcourant chaque token transcrit phonétiquement et segmenté en syllabes structurées. Dans un premier temps, il faut vérifier si le token en question comporte le bon

nombre de syllabes (i.e. celui précisé par les éléments). Si c'est bien le cas, sa composition est examinée, sinon on passe au suivant.

La fonction de matching renvoie pour chaque élément l'ensemble des positions de l'input auxquelles l'élément recherché se trouve. La position comporte cinq coordonnées :

- l'index du token dans la phrase
- le numéro de la syllabe dans ce mot (rapporté à une notation commençant par zéro)
- la position dans la structure syllabique de l'élément matché
- l'index de début de l'élément matché
- l'index de fin de l'élément matché.

Si l'on reprend l'exemple précédent, on obtient :

	Le	sac	des	lapins	
	llee	ssaakk	ddei	llaappin	
	Syllabe 1/1	Syllabe 1/1	Syllabe 1/1	Syllabe 1/2	Syllabe 2/2
	Attaque : ll Noyau : ee	Attaque : ss Noyau : aa Coda : kk	Attaque : dd Noyau : ei	Attaque : ll Noyau : aa	Attaque : pp Noyau : in
Mots pleins		X		X	
(1,AC,1)		MATCH - ss			
(1,AC,2)				MATCH - ll	
positions		[1,0,'a',0,2]		[3,0,'a',0,2]	

Si l'on détaille, [1,0,'a',0,2] signifie 'les deux caractères transcrivant l'attaque de la première syllabe du deuxième mot', et [3,0,'a',0,2] signifie 'les deux caractères transcrivant l'attaque de la première syllabe du quatrième mot'. Voici le second set de coordonnées, légendé coordonnée par coordonnée :

- l'index du token dans la phrase : 3
- le numéro de la syllabe dans ce mot (rapporté à une notation commençant par zéro) : 0
- la position dans la structure syllabique de l'élément matché : 'a' (pour attaque)
- l'index de début de l'élément matché : 0
- l'index de fin de l'élément matché. 2

Les coordonnées de début et de fin d'élément ne sont pas utiles dans cet exemple. Cependant, la

longueur de la chaîne de caractères a son importance. Premièrement, elle permet de distinguer entre les attaques consonantiques simples (leur longueur est de deux) et les attaques consonantiques contenant des clusters (leur longueur est de 4). La fonction de matching va vérifier le contenu de l'attaque et déterminer si la longueur de la chaîne est la bonne. Par ailleurs, il est nécessaire de pouvoir sélectionner une partie de cette chaîne lorsque l'on veut matcher seulement la première ou seulement la seconde consonne du cluster.

C'est le cas pour la paire « Un **p**lan de **G**ap. » / « Un gland de pape ». Il faut matcher la première consonne du cluster en attaque de la première syllabe du second mot, et la consonne unique de l'attaque de la première syllabe du quatrième mot. Les coordonnées des positions seraient alors respectivement [1,0,'a',0,2] et [3,0,'a',0,2].
Tandis que pour la paire «**gl**isser dans la **p**iscine » / « pisser dans la glycine », il faudrait matcher le cluster entier de l'attaque de la première syllabe du premier mot, et la consonne unique de l'attaque du cinquième mot. Les coordonnées des positions seraient alors [0,0,'a',0,4] et [4,0,'a',0,2].[3]

Réalisation des échanges

Lorsque l'ensemble des positions pour chacun des éléments de la permutation ont été récupérés, on peut procéder aux échanges. Toutes les possibilités d'échanges sont examinées les unes après les autres. L'ensemble des échanges possibles est égal au produit cartésien des deux ensembles de positions. Autrement dit, il s'agit de l'ensemble des paires dont la première composante appartient à la première liste de positions et la seconde composante à la seconde liste. Si l'on a deux listes ['a','b','c'] et ['d','e'], le produit cartésien donnera [('a','d'),('a','e'),('b',d'),('b',e'),('c','d'),('c','e')].

Pour chaque possibilité d'échange, les syllabes subissant les permutations sont reconstruites. Admettons que l'on aie deux syllabes σ_1 et σ_2, et deux éléments e_1 et e_2. Les phonèmes de σ_1 sont matchés par e_1, et ceux de σ_2 par e_2, donnant respectivement phonèmes$_1$ et phonèmes$_2$. Les phonèmes1 remplacent les phonèmes de σ_2 aux positions matchées par e_2. Les phonèmes$_2$ remplacent les phonèmes de σ_1 aux positions matchées par e_1.

3 La coordonnée de fin de l'élément matché n'est pas l'index de son dernier élément, mais cet index plus un. Cela s'explique par l'utilisation qui est faite de ces coordonnées dans les échanges auxquels on procède ensuite. Elles servent comme bornes de découpage et doivent donc correspondre aux contraintes de python, qui veut que la sélection s'arrête à la borne finale moins un.

Toutes ces possibilités sont stockées, et pour chacune un candidat-variante est généré. Il contient l'ensemble des transcriptions phonétiques reconstruites pour tous les tokens du candidat. Tous les candidats produits sont ensuite retranscrits en orthographe, lorsque cela est possible.

A ce stade, nous disposons d'un ou de plusieurs candidats à la contrepèterie. Chaque candidat est une liste de tokens transcrits phonétiquement. Ils ont été reconstruits sous forme de tokens transcrits (par opposition à la segmentation en syllabes structurées) afin de faciliter l'interaction avec le module de vérification orthographique.

Voici un exemple de sortie de ce module pour les patrons générant des candidats (pour l'input « Beethoven détestait les sons courts ») :

```
Ce patron permet de générer 4.0 candidats phonos :
['bbeissauvvainn', 'ddeittaissttai', 'llei', 'tton', 'kkourr'],
['bbeikkauvvainn', 'ddeittaissttai', 'llei', 'sson', 'ttourr'],
['bbeittauvvainn', 'ddeissaissttai', 'llei', 'tton', 'kkourr'],
['bbeittauvvainn', 'ddeikkaissttai', 'llei', 'sson', 'ttourr'],

Ce patron permet de générer 1.0 candidat(s) phono.
[['bbeittauvvainn', 'ddeittaissttai', 'llei', 'kkon', 'ssourr'],
```

Voici la sortie de ce module pour les patrons ne générant pas de candidat :

```
Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.
Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.
```

Le module de génération de variantes phonétique n'implémente que certains patrons et s'il est possible d'en implémenter des supplémentaires (d'autres éléments de la structure syllabiques, par exemple), il n'est pas encore capable de modéliser des permutations qui changent la segmentation en mots ou en syllabes à l'intérieur d'un mot. Il ne peut donc pas gérer les transformations qui amènent à ces paires, par exemple :

- « La vie des mots » / « l'ami des veaux »
- « J'en perds mon latin » / « J'enterre mon lapin »
- « Le scorpion est malade » / « Le morpion escalade ».

Cette limitation résulte d'une volonté, pour une première étude, de limiter le nombre de possibilités. Il s'avère que la limitation est probablement trop forte, parce qu'elle bloque

l'application de certains patrons.

Module 2 : Filtre orthographique

La majorité des modules de vérification dans systèmes mentionnés en introduction consistent plutôt en une vérification orthographique plus complexe que ce qui est présenté ici. Principalement parce que ces systèmes sont conçus pour recevoir en entrée des segments linguistiques plus longs qu'un mot isolé, et doivent donc remplir des fonctions plus avancées de validation morphosyntaxique.

La plupart de ces systèmes sont utilisés dans des architectures dans lesquelles le processus de génération est également différent : plusieurs niveaux de représentation abstraite sont impliqués, et la production orthographique vérifiée est alors considérée comme une réalisation de surface. L'ambition de ce module est moindre : il s'agit simplement de vérifier au moyen d'un lexique de transcriptions phonétiques la possibilité de retranscrire en orthographe une transcription phonétique. Le lexique utilisé est celui du LIUM, présenté plus haut. Le principe général est le suivant.

Pour chaque token, plusieurs possibilités sont récupérées, et une variante orthographique est générée pour chaque possibilité. Cette étape de retranscription permet de limiter le nombre de candidats dont on doit réaliser la vérification syntaxique et sémantique. Si l'un des tokens ne trouve pas d'orthographe, le candidat est rejeté. En procédant de cette manière, on peut se heurter aux limites du lexique : un candidat peut être rejeté même s'il est correct uniquement parce que le lexique ne contient pas un de ses tokens. Cependant le lexique du LIUM, organisé en dictionnaire dans le sens phonétique → orthographe contient, rappelons le, 77 262 entrées. S'il contient la plupart des mots du registre standard présents dans le corpus, il a été nécessaire de l'enrichir de quelques termes grivois, voire franchement vulgaires.

Intégrer ce petit module dans le prototype de génération de contrepèteries a demandé deux ajustements. Le premier concerne la sélection des phonétisations pour lesquelles il est nécessaire de chercher les orthographes possibles. En effet, étant donné que chaque candidat n'a subi qu'une (voire deux) permutations, il y a de grandes chances pour que la majorité des candidats partagent un bon nombre de transcriptions communes avec l'input. Ainsi, avant de rechercher dans le dictionnaire les orthographes possibles pour une phonétisation, on vérifie en premier lieu s'il existe dans la transcription phonétique de l'input au même index. Si c'est le cas, l'équivalent orthographique est récupéré. Rechercher les orthographes de ces phonétisations dans le

dictionnaire ne prendrait pas plus de temps, cependant le résultat de la 'requête' serait susceptible, de par des phénomènes d'homophonie, de retourner plusieurs résultats. Sachant que le nombre de candidats orthographiques est le résultat du produit du nombre d'orthographes possibles par mot, le nombre de candidats serait rapidement trop important.

L'étape de vérification de l'existence des tokens dans le lexique constitue une étape charnière entre la génération et la validation, parce qu'elle constitue un premier tri sans entrer dans des considérations syntaxiques ou morphologiques. Au niveau de l'exécution du programme, on peut identifier le problème en considérant les candidats phonétiques, et il suffit d'ajouter l'entrée dans le lexique (ainsi que dans les autres, de taille plus conséquente et décrits plus bas, s'il en est également absent).

La principale limitation concerne les contrepèteries pour lesquelles la permutation produit un contrepet contenant un nombre de mots différent de celui de l'input. C'est le cas quand la permutation implique un pronom clitique faible :

- « Un lien vaut mieux que deux tutorats » / « Un tien vaut lieux que deux tu l'auras »
- « C'est lui qui sert l'amande ! » / « C'est lui qui sent la merde ! »

Pour traiter ces cas également, il faudrait d'une part, ajouter une étape dans la reconstruction des candidats phonétiques afin de changer leur structure, et d'autre part, ajuster la comparaison des structures de contrepèteries (voir plus loin).
En outre, il faudrait adapter la méthode de comparaison des phonétisation présentes à la fois dans l'input et dans les candidats : il faudrait alors parcourir la transcription phonétique de l'input pour chaque phonétisation considérée plutôt que de se limiter à un seul index (de mot).
Il pourrait y avoir des cas d'ambiguïtés à résoudre : imaginons que l'input contienne des homophones. Il y aurait alors plusieurs fois la même phonétisation, mais celle-ci correspondrait à plusieurs orthographes présentes dans la transcription orthographique de l'input. Étant donné que si l'ambiguïté se pose, c'est que la même séquence phonétique est présente à la fois dans l'input et dans le candidat, et qu'elle n'a pas subi de permutation, on peut imaginer poser une condition qui stipule que l'ordre des transcriptions orthographiques doit être le même dans l'input et dans le candidat.
Il est évidemment également possible de ne pas tenir compte de cela et de simplement générer toutes les possibilités, au risque de conserver en sortie du filtre un nombre important de candidats.

S'il est d'un fonctionnement très simple, ce filtre trouve son utilité dans le fait que plusieurs patrons de permutations sont appliqués à un même input. En l'état actuel du prototype, il est possible que plusieurs patrons de permutations produisent des candidats phonétique : il suffit pour cela que chacun des éléments de la permutation matche une position. Cela n'implique pas que tous ces candidats passent le filtre orthographique. Utilisé comme dans cette étude, le filtre permet de rejeter les candidats sans avoir à appliquer une analyse plus fine. Et si aucun candidat n'est retenue après l'application de ce filtre, le prototype passe au patron de permutation suivant.

Considérons ces extraits numérotés du log pour l'input « En voyant les nippons, la Chine se souleva. », dans lequel chaque application de patron est numérotée :
(solution : « En voyant les nichons, la pine se souleva »)

```
[…]
(1) Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.
Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

(2) Application du patron de permutations...
Ce patron permet de générer 1.0 candidat(s) phono.
['an', 'vvwwaayyan', 'llei', 'chiippon', 'llaa', 'nniinn', 'ssee',
'ssoulleevvaa']
Génération des orthographes possibles...
Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

(3) Application du patron de permutations...
Ce patron permet de générer 1.0 candidat(s) phono.
['an', 'nnaayyan', 'llei', 'vvwwiippon', 'llaa', 'chiinn', 'ssee',
'ssoulleevvaa']
Génération des orthographes possibles...
Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

(4) Application du patron de permutations...
Ce patron permet de générer 1.0 candidat(s) phono.
['an', 'chaayyan', 'llei', 'nniippon', 'llaa', 'vvwwiinn', 'ssee',
'ssoulleevvaa']
Génération des orthographes possibles...
Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

(5) Application du patron de permutations...
```

```
Ce patron permet de générer 1.0 candidat(s) phono.
['an', 'vvwwaayyan', 'llei', 'nniippon', 'llaa', 'lliinn', 'ssee',
'ssoucheevvaa']
Génération des orthographes possibles...
Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

(6) Application du patron de permutations...
Ce patron permet de générer 2.0 candidat(s) phono.
['an', 'vvwwaachan', 'llei', 'nniippon', 'llaa', 'yyiinn', 'ssee',
'ssoulleevvaa']
['an', 'vvwwaayyan', 'llei', 'nniichon', 'llaa', 'ppiinn', 'ssee',
'ssoulleevvaa']
Génération des orthographes possibles...
o ['en', 'voyant', 'les', 'nichon', 'la', 'pine', 'se', 'souleva']
o ['en', 'voyant', 'les', 'nichons', 'la', 'pine', 'se', 'souleva']
les candidats : en voyant les nichons la pine se souleva.

[…]
```

L'extrait n°1 montre ce qui se passe lorsqu'aucune position n'a été trouvée pour un élément de permutation. Les extraits 2 à 5 illustrent l'utilité du filtre : ces quatre patrons de permutation produisent des candidats phonétiques, mais ceux ci ne passent pas le filtre. Dans chacun de ces quatre cas (de 2 à 5), au moins une des phonétisations (soulignées) n'a aucune transcription orthographique dans le lexique.
Inversement, l'extrait n°6 montre le succès d'un patron de permutation, du moins jusqu'à cette étape. On voit que le patron permet de générer deux candidats phonétiques, mais que seulement l'un de ces candidats n'est composé que de phonétisations de mots existant (dans le lexique).

La sortie de ce filtre est une liste de transcriptions orthographiques, reconstruites en chaînes de caractères. Elles peuvent être grammaticales ou agrammaticales.

Module 3 : Filtre sur les étiquettes catégorielles

Le but de ce filtre est de conserver, parmi un ensemble de phrases (ou séquences de mots), celles dont la structures est sinon identique, du moins similaire à la structure de la phrase passée en input. Cette démarche peut être considéré comme une mesure de similarité entre des phrases. Utiliser ce module dans cette étude pour vérifier l'acceptabilité des phrases suppose que la phrase en input soit grammaticale. Étant donné que les étiquettes catégorielles de l'input doivent être comparées à celles de plusieurs phrases, la vérification de la grammaticalité de l'input se fait en dehors de ce troisième module.

La plupart des travaux menés sur la similarité entre des phrases se concentrent sur leur aspect sémantique, en utilisant différentes mesures, répertoriées par Palakorn Achananuparp , Xiaohua Hu et Xiajiong Shen (2008). Parmi celles-ci, on trouve tout de même des mesures de recoupement (overlap), ne tenant pas compte de la sémantique :

- Le coefficient de Jacquard, qui, lorsque l'on compare deux phrases, est défini comme « la taille de l'intersection entre les mots de chacune des deux phrases, comparée à la taille de l'ensemble des mots des deux phrases » (ma traduction, depuis l'anglais).
- Le recouvrement de mots simples et la mesure IDF, deux baselines définies par Metlzer et al. (2005) Le recouvrement de mots simples est défini comme « la proportion de mots qui apparaissent dans les deux phrases, normalisée par la longueur des phrases », et la mesure IDF est définie par « la proportion de mots qui apparaissent dans les deux phrases, pondérée par leur fréquence inverse dans les documents » (mes traductions, depuis l'anglais).

Rien n'empêche de remplacer les mots des paires de phrases à comparer par leur étiquette catégorielle, et ainsi utiliser des mesures de recoupement pour classer les séquences examinées par ordre de similarité avec la séquence « modèle ». Cependant les méthodes de recouvrement posent un problème : elles fonctionnent sur des ensembles, qui par définition sont des paquets d'éléments dans lesquels aucun ordre n'est maintenu lorsque la mesure d'éléments en commun se fait. Si le filtre était basé sur ces mesures, il laisserait donc potentiellement passer des structures dont les étiquettes sont identiques, mais ordonnancées de manières complètement différente. Cette solution n'est donc pas envisageable.

L'article de Archananuparp, Hu et Shen présente un autre type de mesure de similarité, basée sur

l'ordre des mots et développée par Li et al. ; elle est définie comme « la différence (normalisée) entre l'ordre des mots de deux phrases » (ma traduction, depuis l'anglais). Pour chaque phrase, l'ordre des mots est modélisé par un vecteur, dont les coordonnées sont fournies par les traits de l'ensemble des mots. Il s'agit donc encore une fois d'une méthode basée sur les mots en eux-mêmes plus que sur leur catégorie grammaticale.

D'autres mesures sont proposées par Li et al., cependant elles utilisent des mesures de similarité sémantique. Or cela n'a pas de sens dans le cadre de la génération de contrepèteries : la différence de sens entre l'input et le contrepet fait partie de la définition même du jeu de mots.

En conséquence de quoi, la méthode de comparaison utilisée dans ce module ne se base que sur les catégories grammaticales des phrases (ou séquences de mot) à comparer, ainsi que sur leur ordre d'apparition.

Dans le cadre de leur générateur de paroles de chanson de rap, Hieu Nguyen et Brian Sa (2009) proposent une piste utilisant la catégorie grammaticale d'un mot pour calculer sa probabilité d'apparition. La mise en place se ferait en deux temps. L'idée est d'analyser d'abord un corpus d'apprentissage parsé avec le parseur de Stanford pour en extraire les structures les plus fréquentes, ainsi que les appariements mots / catégorie grammaticale les plus fréquents. Ils proposent de générer ensuite des arbres (de leur racine vers leurs feuilles) selon ces probabilités d'apparition, afin d'obtenir une structure plate consistant en une suite de catégories grammaticales, à remplir selon les probabilités d'appariement mot / catégorie dégagées dans le corpus d'apprentissage.

Dans le cadre de cette étude, les contraintes posées sur la génération sont plus fortes : les mots constituant les candidats générés ne sont pas tous libres : seuls ceux correspondant à des positions subissant les permutations le sont.

Nous proposons donc de vérifier la similitude entre la phrase correcte en input et chacune des phrases à évaluer en comparant les étiquettes catégorielles qui les composent. Le degré de similitude peut être parfait si les deux suites comparées sont identiques, et sinon rend compte d'équivalences (voir ci-dessous).

Étiquetage en catégories grammaticales avec l'étiqueteur de Stanford

Le parseur de Stanford propose d'appliquer les principes suivants :

- l'utilisation du contexte en termes d'étiquettes (précédente et suivante) via une représentation en réseau des dépendances,

- l'utilisation de traits lexicaux, en tenant compte de l'influence réciproque de mots qui se suivent,
- la prise en compte des résultats précédents dans des modèles statistiques (modèles log-linéaires, permettant des approximations)
- une gestion raffinée des traits des mots inconnus.

La sortie de cet outil est l'input étiqueté, dans lesquels les syntagmes sont séparés par des parenthèses. Voici le résultat pour « Je suis une phrase de test. » :

```
(ROOT
  (SENT
    (NP (DET Le) (NC con)
      (PP (P de)
        (NP (DET la) (ADJ folle))))
    (PUNC .)))
```

La prise en compte du contexte et des influences réciproques des traits et étiquettes du contexte immédiat de l'unité considérée permet d'améliorer la fiabilité des probabilités calculées pour chaque possibilité d'étiquette pour cette unité. Combiner propriétés lexicales et propriétés syntaxiques permet de rendre compte d'usages (dans les étiquettes de syntagmes) en limitant l'influence de ces usages sur l'étiquetage des unités. Voici, pour comparaison, la sortie du parseur de Berkeley pour l'input « Le con de la folle » : ((SENT (NP (D Le) (N con) (PP (P de) (NP (D la) (N folle.))))))). La différence entre les deux sorties se situe au niveau de l'étiquette attribuée à « folle ». Le parseur de Stanford considère cette unité comme un adjectif mais reconnaît qu'elle est utilisée comme le serait un nom : précédée d'un déterminant. Le parseur de Berkeley reconnaît également cet usage, puisque le syntagme est également étiqueté comme syntagme nominal, mais considère l'unité comme étant un substantif. Retrouver ces deux résultats n'est pas étonnant, et confirme nos intuitions sur les équivalences, développées plus bas.

L'influence de la prise en compte des mots inconnus est en revanche négligeable dans l'utilisation du parseur dans cette étude, étant donné qu'ils ont été évacués précédemment.

Enfin, le parseur tentera de générer une sortie, peu importe le nombre d'erreurs présentes dans l'input. Si les sorties un peu trop exotiques ont été éliminées précédemment, les phénomènes d'accord non réalisés dans les syntagmes ne sont donc pas bloquants et doivent être repérés. Le

choix a été fait de parseur tous les candidats en une seule fois afin de minimiser le temps de traitement (en évitant d'appeler l'outil à chaque nouveau candidat).

Première élimination de candidats : propriété de similitudes syntaxiques

Comme mentionné en introduction, la structure syntaxique de l'input et de son contrepet sont sinon identiques, fortement similaires. Pour cette raison, il est possible de représenter l'input et les candidats par l'ensemble des étiquettes des mots qui les composent, et de les comparer. Les candidats trop différents sont éliminés. Les candidats à la structure identique sont conservés. Les candidats ayant une structure similaire sans être identique sont comparer selon un système d'équivalences.

Degrés de similitude et équivalences

Outre dans le cas de catégories très proches comme les substantifs et les noms propres, il n'est pas question de considérer qu'une étiquette catégorielle est équivalente à une autre, dans l'absolu. Des catégories peuvent en revanche partager certaines propriétés, qui permettent de retrouver ces catégories dans des usages et constructions similaires. C'est le cas, par exemple, des substantifs et des adjectifs. Les deux varient en genre et en nombre, et peuvent être précédés d'un déterminant :

- « Les voitures sont garées dans le parking. »
- « Les derniers seront les premiers . »

Ces propriétés communes expliquent que l'on retrouve des unités étiquetées par les parseurs comme syntagmes nominaux, mais contenant un mot étiqueté comme un déterminant et un mot étiqueté comme adjectif : (NP (D la) (Adj folle)). Notons cependant que l'équivalence n'est pas forcément valable dans le sens inverse :

- Cette fille est folle.
- C'est une folle.
- * Cette folle est fille.

Par ailleurs, d'autres catégories peuvent être concurrentes pour un même token orthographique. Considérons les exemples suivant :

- output du parseur pour « Manger des fruits est bon pour la santé. » :

```
(ROOT
  (SENT
    (NP (NC Manger)
      (PP (P des)
        (NP (NC fruits))))
    (VN (V est))
    (AP (ADJ bon)
      (PP (P pour)
        (NP (DET la) (NC santé))))
    (PUNC .)))
```

- output du parseur pour « Les lapins mangent des carottes » :

```
(ROOT
  (SENT
    (NP (DET Les) (NC lapins))
    (VN (V mangent))
    (NP (DET des) (NC carottes))
    (PUNC .)))
```

Dans les deux cas, « des +N » est objet direct du verbe manger. Dans les deux cas, 'des' peut être considéré comme un déterminant partitif. Si l'on ramène ces deux structures en gras à des listes plates d'étiquettes grammaticales, on obtient 'P NC' et 'DET NC'. Les structures arborescentes desquelles sont extraites les deux séquences d'étiquettes sont certes différentes, mais la comparaison s'effectue dans ce module au niveau des listes d'étiquettes. On peut donc considérer que dans ces cas, la présence d'une étiquette 'P' est équivalente à celle d'une étiquette 'D'.

Que ce soit dans le cas des paires de catégorie substantif/adjectif ou préposition/déterminant, il ne s'agit que d'équivalences partielles : on ne peut pas systématiquement interchanger les étiquettes. Rappelons que les paires à comparer dans ce module sont un input et le résultat de permutations appliquées à cet input. Les deux phrases (ou séquences de mots) ont donc nécessairement des structures similaires, tant qu'elles n'ont pas subi trop de permutations. Les divergences entre les deux suites d'étiquettes catégorielles sont donc restreintes. Ainsi, la mesure d'équivalence peut s'entendre comme « Si les autres étiquettes sont identiques, alors la différence

[entre l'étiquette 'Adj' et l'étiquette 'N'] / [entre l'étiquette 'P' et l'étiquette 'D'] peut être ignorée ». De plus, cette simplification des listes d'étiquettes n'est faite que lors de la comparaison, les listes ne sont pas modifiées en place.

Voici un exemple d'output du module pour la comparaison de la suite d'étiquettes obtenues pour « Mouiller les fiches » et l'ensemble des candidats :

```
nouveau candidat :  ['fouillée', 'les', 'miches', '.']
avant simplification :  ['ADJ', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification : ['N', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N', 'PUNC']

nouveau candidat :  ['fouillé', 'les', 'miche', '.']
avant simplification :  ['VPP', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification :  ['VPP', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N',
'PUNC']

nouveau candidat :  ['fouiller', 'les', 'miche', '.']
avant simplification :  ['VINF', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification :  ['VINF', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N',
'PUNC']

nouveau candidat :  ['fouiller', 'les', 'miches', '.']
avant simplification :  ['VINF', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification :  ['VINF', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N',
'PUNC']

nouveau candidat :  ['fouillé', 'les', 'miches', '.']
avant simplification :  ['VPP', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification :  ['VPP', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N',
'PUNC']

nouveau candidat :  ['fouillés', 'les', 'miche', '.']
avant simplification :  ['VPP', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification :  ['VPP', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N',
'PUNC']

nouveau candidat :  ['fouillés', 'les', 'miches', '.']
avant simplification :  ['VPP', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC',
'PUNC']
après simplification :  ['VPP', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N',
'PUNC']

nouveau candidat :  ['fouillée', 'les', 'miche', '.']
```

```
avant simplification : ['ADJ', 'DET', 'NC', 'PUNC'] ['VINF', 'DET', 'NC', 'PUNC']
après simplification : ['N', 'DET', 'N', 'PUNC'] ['VINF', 'DET', 'N', 'PUNC']
```

Les candidats ayant passé ce filtre sont donc ceux dont la suite d'étiquettes grammaticales simplifiée est identique à la suite d'étiquettes simplifiée de l'input. Il s'agit de :

- ['fouiller', 'les', 'miche', '.'] (['VINF', 'DET', 'N', 'PUNC'] et ['VINF', 'DET', 'N', 'PUNC'])
- ['fouiller', 'les', 'miches', '.'] (['VINF', 'DET', 'N', 'PUNC'] et ['VINF', 'DET', 'N', 'PUNC'])

On constate que, même si un certain nombre d'erreurs on été éliminées (on passe de huit candidats à deux), les candidats ayant passé ce filtre peuvent toujours être agrammaticaux : on constate dans le premier résultat une violation d'accord en nombre (déterminant au pluriel et substantif au singulier).

En l'état actuel du prototype, l'ensemble des équivalences doit être partiel et limité afin d'éviter de maintenir l'efficacité du filtre en ne neutralisant pas des différences de manière absurde. Pour étendre les possibilités d'applications de ce module, plusieurs paramètres devront être pris en compte.

L'échelle d'application de la comparaison peut être changée : plutôt que de comparer des étiquettes catégorielles de mots, il serait possible de comparer des étiquettes catégorielles de syntagmes. En effet, si l'on examine la contrepèterie « Marco Polo est arrivé à pieds par la Chine » / « Marco Polo est arrivé à chier par la pine », on constate que les structures attendues pour les syntagmes en gras sont les suivantes :

- « Marco Polo est arrivé **à pieds** par la Chine » :
  (PP (P à) (N pieds))

- « Marco Polo est arrivé **à chier** par la pine »
  (PP (P à) (Vinf chier))

Dans une telle perspective, il faudrait pouvoir remonter d'un niveau pour faire la comparaison : si l'on compare 'P N' et 'P Vinf', l'équivalence ne se fait pas, tandis que si l'on compare les deux syntagmes prépositionnels, elle est valide. Par ailleurs, remonter d'un niveau aurait également le mérite de contourner les problèmes liés à la ponctuation éventuellement présente dans l'input, et qui n'est actuellement pas encore prise en compte dans la génération des candidats.

Dans d'autres travaux proposent également de remonter d'un niveau afin de réaliser une comparaison d'ordre syntaxique. Par exemple, le système FILTRAR-S (Campion et al. 2009), un

projet d'outil de cyber sécurité, présente un module s'appuyant entre autres sur la structure syntaxique du contenu examiné. Le premier module de FILTRAR-S réalise un filtrage sémantique par « extraction automatique de connaissances sémantiques et factuelles ». Le second module extrait des relations entre entités nommées ainsi que des faits reliés à ces entités. Ces relations sont classées de manière à rassembler celles considérées comme sémantiquement équivalentes. C'est là que les auteurs proposent, au moyen d'algorithmes de clustering, de tenir compte d'obstacles posées par les catégories grammaticales des unités dans la modélisation d'une mesure de similarité : « *Une attention spécifique sera portée à ce niveau sur la définition d'une mesure de similarité prenant en compte les problèmes de normalisation prédicative afin de pouvoir apparier par exemple les verbes et les nominalisations. ».* Ce n'est certes pas le cœur du système, mais il ne peut s'en affranchir.

Influence des outputs du parseur

En réalité, l'output du parseur pour « à chier » est (PP (P à) (N chier)). L'explication la plus probable de cette sortie est que le mot « chier » n'est répertorié nulle part comme un verbe dans les ressources utilisées par le parseur. Le modèle du parseur pour le français a été appris à partir du French Tree Bank (Abeillé et al. 2003), lui même constitué d'après un corpus d'un million de mots, composés d'articles du journal « Le Monde « sur une période s'étendant de 1989 à 1995. On peut donc penser que des mots appartenant à un registre trop familier ou clairement vulgaire ne soient pas très fréquents, voire absents, de ce corpus. Dans cette hypothèse, « chier » (et autres) est traité comme un mot inconnu et a donc de grandes chances de se voir considéré comme un substantif. La qualité de l'output du parseur est donc à prendre en compte et peut éventuellement expliquer certains dysfonctionnements du filtre, d'autant que la plupart des phrases examinées sont erronées.

D'autre part, à nouveau se pose la question de la segmentation en mots. Le jeu d'étiquettes du parseur, repris dans l'étude menée par Spence Green, Marie-Catherine de Marneffe, John Bauer et Christopher D. Manning, se focalise sur les expressions et autres unités polylexicales : MWN (pour les noms), MWADV (adverbes), MWA (adjectifs), MWP (prépositions), MWC (conjonctions), MWV (formes verbales), MWD (déterminants), MWPRO (pronoms), MCL (clitiques), etc. Ces cas ne sont pas du tout pris en compte dans le module. Cependant on pourrait imaginer un système d'équivalences entre la forme polylexicale et l'étiquette monolexicale : MWN ↔ N, MWADV ↔ ADV, etc.

Filtrer sur les étiquettes catégorielles a le mérite d'éliminer un grand nombre de candidats générés à cause de l’homophonie. Cependant le filtre est trop restrictif : il serait véritablement intéressant de remonter d'un niveau, comme mentionné précédemment.

Module 4 : Filtre sur la réalisation des phénomènes d'accord

Comme mentionné dans l'introduction, de nombreux travaux de génération de texte utilisent un certain nombre de structures intermédiaires, plus ou moins abstraites, avant de parvenir au texte généré, alors envisagé comme réalisation de surface.

Le système TEMSIS (Transnational Environmental Management Support and Information System) (Denzer et al. 1998) utilise également une représentation intermédiaire, qui ne contient finalement que très peu d'indications sur la façon dont le contenu informationnel doit ou peut être mis en mots, dans la mesure où le lien sémantique entre ces éléments d'information est majoritairement implicite. Le réalisateur de surface utilisé dans leur étude est TG/2. Il s'agit d'un système intégrant des messages pré-enregistrés, des patrons, et un ensemble de règles (grammaire hors-contexte) dans un même formalisme.

SURGE (Systemic Unification Realization Grammar of English, Elhadad 1996) est un réalisateur de surface syntaxique développé par Michael Elhadad et Jacques Robin pour l'anglais, ayant pour principe d'intégrer différentes théories linguistiques offrant ainsi une complémentarité qui a rendu possible son intégration à « huit générateurs d'architectures assez différentes » (ma traduction). En ce qui concerne les phénomènes d'accord, il est divisé en deux étapes : une propagation des traits à unifier (ce qui est considéré comme une sous-tâche), qui sert d'input au module morphologique qui va appliquer les marques de flexion appropriées.

La principale différence entre ces méthodes et celle proposée dans ce quatrième module réside principalement dans la nature de l'input. Si le module de la présente étude utilise également une représentation abstraite dans une certaine mesure, c'est bien la phrase qui est donnée en entrée. Il n'est donc pas question d'application de contraintes pendant la génération (la réalisation de surface, plus précisément) comme c'est le cas dans les travaux mentionnés ci-dessus, ainsi que dans le troisième module, mais de vérification d'unification de traits à posteriori.

Il est possible d'utiliser certaines des fonctionnalités de ce module afin de construire un ensemble de règles de réécritures pour vérifier la grammaticalité de la structure d'une phrase (pour des questions de présentation, le processus est expliqué à la fin de cette partie). Cependant, tel qu'utilisé dans le prototype, ce module ne vérifie que les accords dans les phrases qu'il examine. La raison en est la suivante : un premier examen de ces phrases a déjà été fait par un autre module, vérifiant la similitude syntaxique entre ces phrases et un modèle accepté comme grammatical. Cette démarche résulte d'une volonté d'optimiser le temps de traitement de

l'architecture de manière globale.

Hiérarchisation des syntagmes par niveaux d'enchâssement

Le but de cette démarche est de parcourir les syntagmes et de vérifier les phénomènes d'accord de manière interne, en remontant du syntagme le plus enchâssé à la phrase. Cet ordre présente plusieurs avantages. Premièrement, les vérifications d'accord sont plus simples à faire étant donné que les structures de traits à comparer sont encadrées par les bornes du syntagme, on sait donc quoi comparer, et les éléments sont moins nombreux. Par ailleurs, les accords à vérifier ont été grandement filtrés par la première élimination des candidats, et les cas restants sont principalement des accords en genre et en nombre dans des syntagmes nominaux / en nombre dans des accords prépositionnels. Restent également des accords en nombre et en personne dans les syntagmes verbaux.

La hiérarchisation des syntagmes par niveau se fait en deux temps. Premièrement, il faut traiter la sortie du parseur de Stanford pour déterminer quel syntagme est enchâssé à quel niveau. Pour ce faire, on considère que cette sortie n'est autre qu'une chaîne de caractères, et que chaque niveau est matérialisé par un couple de parenthèses ouverte/fermante.

La chaîne de caractères est parcourue, et, pour chaque parenthèse ouvrante, la position de début de syntagme est stockée dans une liste L. Dès qu'une parenthèse fermante est trouvée, elle est associée avec la dernière parenthèse ouvrante examinée, et la position de celle-ci est supprimée de la liste L. Les index de ces deux parenthèses sont les bornes de la sous-chaîne constituant le syntagme enchâssé. La profondeur de ce syntagme dans l'arbre est équivalente au nombre de parenthèses ouvrantes déjà examinées (soit la longueur de la liste L contenant les positions de début de syntagmes).

Considérons un exemple simple : (NP (D le) (N chat)).

- Positions des parenthèses ouvrantes (L) : 0,4
- Position de la première parenthèse fermante trouvée : 9
- → Bornes du premier syntagme : 4 et 9
- Position des parenthèses ouvrante (L): 0 → la longueur est de 1

On a donc un premier syntagme (D le), de niveau 1.

Si l'on continue de parcourir la chaîne :

- Position des parenthèses ouvrante (L): 0, 11
- Position de la prochaine parenthèse fermante trouvée : 18
- → Bornes du second syntagme : 11 et 18
- Position des parenthèses ouvrantes (L) : 0 → la longueur est de 1

On a donc un second syntagme (N chat), de niveau 1 également.

Finissons de parcourir la chaîne :

- Position des parenthèses ouvrante (L): 0
- Position de la prochaine (et dernière) parenthèse fermante trouvée : 19
- → Bornes du troisième syntagme : 0 et 19
- Position des parenthèses ouvrante (L): → la longueur est de 0

On a donc un syntagme global ((NP (D le) (N chat)), de niveau 0.

Ce qui peut être représenté par des tuples comme ceci :[(1, 'D le'), (1, 'N Chat'), (0, 'NP (D le) (N Chat)')]. Le dictionnaire construit à partir de cette liste ressemble à ceci :

syntagmes de niveau 1 : (D le) ; (N chat)

syntagmes de niveau 0 : ((NP (D le) (N chat)).

On veille ensuite à parcourir les niveaux d'enchâssement de syntagmes dans le bon ordre.

Vérification des accords

La motivation derrière le parcours des syntagmes par ordre décroissant de profondeur de l'arbre est la vérification des phénomènes d'accords, et sa facilitation : l'examen commence par les feuilles, dont il est facile de récupérer les structures de traits puisque cela correspond à les piocher dans un lexique. De plus, cela permet d'examiner les syntagmes par complexité croissante, et par là même d'éliminer la phrase à la première erreur (donc le plus rapidement possible).

Observons cet exemple : « Le chats de la voisine dort ». L'output du parseur pour cette phrase est :

'(ROOT (SENT (NP (DET Le) (NC chat) (PP (P de) (NP (DET la) (NC voisines)))) (PUNC .)))'

Avec la méthode proposée, le module ne doit faire que deux vérifications pour détecter que cette phrase est agrammaticale. La première consiste en réalité à récupérer les structures de traits des mots constituant la phrase. La seconde permettra de détecter la violation de l'accord en genre dans le syntagme '(NP (DET la) (NC voisines))' et ainsi parcourir moins de deux niveaux sur cinq.

Étant donné que le module est conçu pour vérifier la grammaticalité de plusieurs phrases à la

fois, et que ces phrases sont susceptibles d'être plusieurs à contenir la même erreur, le module stocke chaque syntagme qu'il détecte comme erroné, et teste à chaque vérification de syntagme si celui-ci a déjà été jugé comme incorrect. Cela a le mérite d'accélérer la vérification des phrases lorsqu'elles sont similaires. En revanche, l'utilité est moindre si les phrases analysées contiennent des mots complètement différents, il semble alors plus rapide et pertinent de se limiter à détecter de nouvelles erreurs plutôt que de les stocker.

La récupération des traits dans le lexique a par ailleurs été légèrement filtrée. A cause de phénomènes d'homographie d'une part, et de plusieurs possibilités d'usages de certains mots d'autre part, il est possible de trouver dans le lexique plusieurs structures de traits pour une même orthographe .

Le lexique utilisé dans ce module est la version 'mlex' du Lefff[4] : il s'agit de la partie morphologique de la version extensionnelle. Benoît Sagot explique (Sagot 2010) que ce lexique est compilé automatiquement à partir de la version intensionnelle : « ce processus de génération comporte une phase de flexion, en fonction de la classe morphologique associée à l'entrée intensionnelle, puis une phase de construction de la structure syntaxique associée à chacune des formes fléchie obtenues (les informations syntaxiques variant d'une forme à une autre, en particulier pour les formes infinitives et participiales, et en fonction de chaque construction associée à l'entrée) ».

Voici un exemple d'entrées de ce lexique pour la graphie « bout »:

```
bout nc   bout ms
bout v    bouillir  P3s
```

Chaque ligne comporte un mot-forme, la catégorie grammaticale, le lemme et les traits morphologiques. Ils sont présentés selon « un encodage proche du LADL », où une lettre représente un mode et un temps, ou un nombre, genre, et un chiffre une personne.

La graphie « bout » correspond dont à deux possibilités grammaticalement très différentes, dans les contraintes d'accord à satisfaire (/ positions d'apparition). Combinées aux traits que l'on obtient, par exemple, pour la graphie « un » :

```
un   det  un   ms
un   nc   un   ms
un   pro  un   ms
```

On obtient six possibilités de structures à comparer (le nombre de possibilités de combinaisons de structures à vérifier est le résultat de la factorielle de chacun des nombres de possibilités pour

4 http://alpage.inria.fr/~sagot/lefff.html

chacune de ces orthographes). Or ces six possibilités peuvent être réduites : le module ne renvoie qu'un seul exemplaire de chaque structure. De plus, il sélectionne les structures de traits qui sont associées à la catégorie grammaticale indiquée par le parseur. On passe donc, pour cet exemple, de six (deux fois trois) possibilités à une seule comparaison à faire. La différence serait encore pus flagrante si les syntagmes n'étaient pas examinés par ordre croissant.

Les phénomènes d'accord vérifiés dans ce module sont les accords en genre et en nombre à l'intérieur des syntagmes nominaux, l'accord en genre et en personne dans les syntagmes verbaux (à l'exception des infinitifs). Le parti est pris de chercher à détecter des violations d'accord plus que des réalisations réussies, à la fois pour éliminer des possibilités erronées rapidement, mais également pour limiter la sur-généralisation des vérifications.

Par exemple, la vérification des accords dans les syntagmes prépositionnels est difficilement réalisable en ne se basant que sur les étiquettes catégorielles de l'output du parseur. Cela provient du fait que lorsque les prépositions sont contractées avec des articles définies, le parseur ne décompose pas cette contraction. Par exemple, « le chat des voisins », que l'on peut décomposer comme « le chat de+les voisins » donnera la suite d'étiquettes D N P N. Comment alors détecter qu'un syntagme nominal tel que « le chat des voisin » est agrammatical ? Il n'est pas possible d'imposer que le nombre de la préposition et celui du substantif qui en dépend doivent être identiques, puisque on peut avoir des suites comme « une prise de risques », « un matelas de plumes », « un ensemble de personnes ». Il faut donc détecter des cas de violation, comme 'si la préposition est au pluriel, et que le substantif (ou le syntagme nominal) qu'elle introduit est au singulier, il y a violation'. Le cas des noms propres est un peu plus souple, cependant le lexique leur attribue des traits morphosyntaxiques, ils sont donc traités comme des noms communs.

Un phénomène d'accord est considéré comme correctement réalisé s'il n'y a pas de violation. Selon la nature du syntagme examiné, une structure de traits est produite pour les syntagmes nominaux, adjectivaux et verbaux, une structure vide dans le cas des syntagmes prépositionnels et adverbiaux. Cette structure est alors stockée dans un dictionnaire du module, ce qui permet de la récupérer pour d'éventuelles autres vérifications au fur et à mesure qu'on remonte vers la racine de l'arbre.

Cette approche élimine la plupart des phrases agrammaticales, mais néglige certaines erreurs lorsqu'elles sont dans des structures de compléments du nom, confondues avec des syntagmes prépositionnels. Ainsi, la vérification des candidats de certains contrepets laisse passer quelques

erreurs. Voici les candidats filtrés par les quatre premiers modules :

- pour l'input « Aimer le blanc est une histoire de goût » :

```
aimer le gland est une histoire de Bou.
aimer le gland est une histoire de bouts.
aimer le gland est une histoire de boue.
aimer le gland est une histoire de boues.
aimer le gland est une histoire de bout.
```

- Pour l'input « Ici on peut voir un vieux plan de Gap. » :

```
ici on peut voir un vieux gland de pape.
ici on peut voir un vieux gland de papes.
ici on pleut voir un vieux pan de Gap.
ici on pleut voir un vieux paon de Gap.
```

On constate que dans les deux cas, le filtre laisse passer des phrases que l'on souhaiterait pourtant éliminer. Dans les deux cas, ce n'est pas la syntaxe qui permet de détecter cette erreur, mais la sémantique - bien que de deux manières différentes.
Dans le premier cas, les phrases ne sont pas agrammaticales à proprement parler, parce qu'il est tout à fait possible qu'un existe une histoire qui parle effectivement de Bou, de boues, de boue ou même de bouts. Mais le lecteur reçoit une première orientation, un indice, vers une interprétation grivoise avec le mot « gland », et aura tendance à sélectionner le candidat avec « boût ».
Dans le second cas, c'est encore la sémantique qui permet de savoir que cette partie du corps ne peut appartenir à plusieurs personnes en même temps.

Cela montre les limites de cette approche : il n'est pas possible de modéliser tous les phénomènes d'accord uniquement à partir d'étiquettes catégorielles, de mots et de syntagmes. Il semble nécessaire d'inclure des considérations fonctionnelles et sémantiques.

Utilisation du module pour l'extraction de règles de réécriture depuis un corpus

Comme annoncé plus haut, il est possible d'utiliser certaines des fonctionnalités de ce module pour construire une grammaire de réécriture d'après un corpus. La méthode est la suivante. Chaque phrase du corpus est analysée avec le parseur de Stanford. Pour chaque phrase :

- La sortie du parseur est transformée en liste de syntagmes enchâssés (il n'est pas

nécessaire de les hiérarchiser dans le cadre de cette utilisation).

- Pour chaque syntagme, on extrait son sigle (« NP », « SN », « SENT », « ROOT », etc.). Si celui ci contient d'autres syntagmes, on extrait le sigle de chacun de ses dépendants directs.

Le tout est stocké dans un dictionnaire dans lequel on a pris soin de supprimer les doublons. Il serait également possible de construire des probabilités de réécriture : ordonner les réécritures constatées, pour même syntagme, selon leur fréquence d'apparition dans le corpus. Ce n'est pas le cas ici, étant donné qu'on a juste besoin de vérifier si une réécriture existe. Mais il serait possible de former un module indépendant à partir de celui-ci, qui serait utilisable dans une autre architecture.

Il existe cependant un risque d'enregistrer des structures erronées, selon le contenu du corpus. Il faut donc valider le corpus d'apprentissage en amont. Ceci dit, on peut supposer qu'en utilisant par exemple un corpus composé d'articles de presse, le risque soit limité. En l'occurrence, les structures ont été extraites depuis le corpus d'étude présenté en début de mémoire.

Module 5 : Filtre sémantique

L'usage, en français, veut que le contrepet généré pour une phrase innocente passée en entrée comporte une dimension définie tantôt comme grossière, grivoise, salace, coquine, équivoque… Il faut que d'une manière ou d'une autre, il y ait un élément qui touche à un tabou. Détecter un contenu grivois peut également être utile dans des contextes complètement différents de la génération de la contrepèterie. On peut imaginer un outil d'assistance, qui lorsqu'il détecte une connotation tabou dans un texte, prévient l'utilisateur (on pense notamment à des apprenants ou à des personnes souffrant de troubles du langage). Cela pourrait être appliqué aussi bien à la rédaction du texte qu'à sa lecture. On peut donc rapprocher cette vérification des systèmes de détection de spam, ou de modération automatique de forums de discussion et de réseaux sociaux.

Quasiment tous les systèmes de modération de contenu sur internet sont semi-automatiques : un premier travail de repérage du contenu sensible est effectué automatiquement, et le contenu repéré est ensuite vérifié par des humains. De nombreux systèmes de modération dépendent d'annotations faites par les utilisateurs : ils peuvent signaler du contenu offensant ou violant les règles d'utilisation, voire les lois en vigueur. Momeni (2012) explique que les évaluations de contenu par les utilisateurs, dans le cadre de la modération automatique, sont pondérées, le plus souvent en étudiant la disparité de votes positifs et négatifs. Elle propose d'effectuer la pondération en tenant compte de l'influence du contexte sur ces annotations d'utilisateurs.

Delort et al. (2014) confirment que la détection de contenu inapproprié est encore bien souvent une tâche manuelle, quand bien même il existe des techniques de prévention automatiques, qu'ils classent en deux catégories principales : les systèmes à base de règles et ceux à base de calculs statistiques (généralement plus robustes). Ils expliquent cependant que pour adopter ces techniques statistiques, il est nécessaire de disposer de corpus dans lesquels le contenu inapproprié a été préalablement annoté, afin d'entraîner des classifieurs. De telles ressources sont coûteuses, puisqu'il s'agit d'annotation manuelle. Ils présentent un classifieur pouvant être entraîné à partir d'un corpus partiellement annoté, pouvant répartir des textes en deux classes : la première pour les documents avec du contenu inapproprié, et l'autre pour les documents au contenu estimé comme acceptable. Par « partiellement annoté », il faut comprendre que seul les exemples d'une des classes ont été annotés.

On pourrait imaginer utiliser une méthode similaire : annoter le contenu grivois du corpus de contrepèteries, afin de classifier les candidats selon deux classes, 'grivois' et 'non grivois', et à

l'inverse des démarches de modération, ne garder que les textes classés comme inappropriés, grivois. Cependant, par manque de temps, nous avons plutôt opté pour une approche lexicale. Si elle ne couvre pas l'intégralité des manifestations de la grivoiserie (voir infra), cette méthode constitue une première approche, comptant sur le fait que le sens grivois des contrepets du corpus est le plus souvent peu ambigu, voire totalement explicite.

Les contours de la contrainte sémantique posée sur la contrepèterie sont assez flous. Cependant, sans même chercher à les préciser, on se rend bien compte que l'emploi d'un mot grossier n'implique pas une connotation grivoise / sexuelle, et l'inverse non plus. De même, en examinant le corpus, on se rend compte que la contrainte sémantique posée sur le contrepet peut être :

- grivoise mais pas grossière « la route des vins / la voûte des reins », « fouiner dans le coin / couiner dans le foin », « passez donc vos lèvres sur mon nombril »...
- grossière mais pas grivoise : « il sert l'amende / il sent la merde »

Dans le second cas, on a plutôt affaire au registre scatologique. Par ailleurs, on note le recours au champ lexical de la religion (« pape », « nonne », « abbé »..).

On peut chercher à caractériser cette connotation réclamée par la contrepèteries par plusieurs champs lexicaux : celui de la sexualité, celui des parties du corps, celui des fonctionnalités corporelles, celui de la religion, et celui de la scatologie, et des animaux principalement. Il est donc possible de faire un rapprochement sémantique entre la contrepèterie et les jurons et insultes, même si l'usage des jurons est plus discursive, plus proche de la décharge émotionnelle (Benveniste 1974). Pour des raisons de commodité de rédaction, cette connotation sera ci-après qualifiée de « grivoise » ; la connotation sexuelle est la plus présente dans le corpus.

Lors de la phase de développement, un double lexique a été créé à partir des exemples du corpus. Ce lexique sommaire est divisé en deux sous-parties. La première contient des termes grivois non ambigus « couille », « pute », « nichon... » et la seconde des mots qui permettent plusieurs interprétations, dont au moins une est grivoise et une autre non : « gland », « chatte »… Cette séparation a été voulue pour marquer la différence entre plusieurs cas de manifestation de la grivoiserie.

L'étude du corpus a montré qu'elle peut se manifester de plusieurs manières et à différents degrés :

- la phrase contient un terme grivois (/ grossier / vulgaire / scatologique) non ambigu (ex :

« pisser dans la glycine », « un gland de pape »)

- la phrase contient un terme ambigu - tel que défini ci-dessus (« un gland de pape », « fouiller les miches »)
- la phrase contient une expression grivoise, qui n'est pas repérable par un examen mot-à-mot (« la voûte des reins », « elle a perdu sa fleur »)
- la phrase contient une connotation grivoise pour l'interprétation de laquelle il faut des connaissances extra-linguistiques (« couiner dans le foin »).

Le recours à ce double lexique permet de résoudre les deux premiers cas. En fonction des termes trouvés ou non dans les deux lexiques, un score est attribué à chaque phrase examinée selon la pondération suivante :

- un point par mot de la phrase présent dans le lexique ambigu,
- deux points par mot de la phrase présent dans le lexique grivois.

Cette pondération a été choisie pour rendre compte des points suivants :

- un sens non ambigu est plus saillant qu'un sens ambigu,
- la nature de la contrepèterie amoindrit l'ambiguïté parce son usage implique une transformation aboutissant à du texte grivois. Ce qui veut dire qu'on peut considérer que s'il y a ambiguïté, c'est qu'il y a au moins un sens grivois.

Le module renvoie toutes les phrases dont le score d'évaluation de la grivoiserie est supérieur à zéro, classées du score le plus élevé au plus faible.

Considérons l'extrait de log suivant :

```
Voici le(s) contrepet(s) générés pour 'Aimer le blanc est une histoire de
goût.' :
(9 patrons testés)

aimer le gland est une histoire de boue.
aimer le gland est une histoire de bout.
aimer le gland est une histoire de bouts.
aimer le gland est une histoire de Bou.
aimer le gland est une histoire de boues.


Vérification sémantique de la grivoiserie...
```

```
Il reste (en ordre décroissant si plusieurs contrepets) :

aimer le gland est une histoire de bout.
aimer le gland est une histoire de bouts.
aimer le gland est une histoire de boue.
aimer le gland est une histoire de Bou.
aimer le gland est une histoire de boues.
```

On voit dans cet exemple que toutes les phrases ont été conservées, mais ont été ordonnées. Les deux premières phrases ont un score de deux points (gland : 1 ; bout/bouts : 1) et les trois suivantes ont un score de un point (elles ne contiennent que « gland »).
Voici un autre exemple :

```
Voici le(s) contrepet(s) générés pour 'Avez vous vu le bond de la crue ?' :
(9 patrons testés)

avez vous vu le con de la bru.
avez vous crus le bond de la vue.
avez vous cru le bond de la vue.
avez vous crû le bond de la vue.
avez vous crues le bond de la vue.

Vérification sémantique de la grivoiserie...
Il reste (en ordre décroissant si plusieurs contrepets) :

avez vous vu le con de la bru.
```

On voit cette fois que le module n'a conservé qu'une seule phrase. On note au passage l'agrammaticalité des candidats avant filtrage sémantique (le phénomène d'accord n'a pas été implémenté, il faudrait pour cela ajouter une série de conditions sur l'auxiliaire – après l'avoir détecté - et sur la position de l'objet direct).

Dans cet état, le module peut produire des faux négatifs : les expressions comme « la voûtes des reins » (input : « la route des vins ») ne sont pas reconnues. Il y a au moins deux possibilités pour corriger cela : soit parcourir l'input par groupes de mots plutôt que par mot avant de consulter le lexique, soit utiliser une dernière étape de parsing. Nous avons tenté d'utiliser une autre version du parseur de Stanford (utilisant un réseau neuronal), qui renvoie en output des triplets relation – tête – dépendant. Voici un exemple d'application.

```
Voici le(s) contrepet(s) générés pour 'Ici on peut voir un vieux plan de
Gap.' :
(9 patrons testés)

ici on peut voir un vieux gland de pape.
ici on peut voir un vieux gland de papes.
ici on pleut voir un vieux pan de Gap.
ici on pleut voir un vieux paon de Gap.


Vérification sémantique de la grivoiserie...
Il reste (en ordre décroissant si plusieurs contrepets) :

ici on peut voir un vieux gland de pape.
ici on peut voir un vieux gland de papes.


Troisième vérification...
Il reste :
Tagging completed in 0,69 sec.
Parsed 2 sentences in 0,29 seconds (6,87 sents/sec).

ici on peut voir un vieux gland de pape.
```

S'il serait certainement plus rapide de parcourir l'input par groupes de mots afin de retrouver les expressions multi-mots (une liste plus conséquente de termes étiquetés comme « vulgaires » a été extraite de Wikipédia pour essais futurs, mais n'a pas été utilisée dans cette étude), l'utilisation d'une analyse par type de relation a le mérite de permettre de vérifier des phénomènes d'accord qui n'auraient pas été vérifiés précédemment en même temps qu'elle peut reconnaître des expressions si elles sont modélisées en terme de relations de dépendance. L'exemple ci-dessus utilise un patron modélisant les expressions du type 'partie du corps' + 'complément du nom avec un trait +humain'. Pensée initialement pour proposer une désambiguïsation des termes évoquant soit une partie du corps dans un registre très familier / vulgaire ou un autre référent sans connotation sexuelle (« gland »), on peut imaginer que cette fonctionnalité pourrait exploiter des modélisations de lexique pour vérifier des phénomènes d'accord qui ne découlent pas de la syntaxe (l'exemple ci-dessus), ou des phénomènes d'accord ne pouvant être vérifiés qu'analysés à travers une approche fonctionnelle (on pense par exemple aux accords des participes passés).

Plusieurs combinaisons possibles des modules

Reiter et Dale (2000) expliquent que les problèmes posés par la génération automatique de langage naturel peuvent être regroupés en six catégories principales :

- la prise en compte du contexte : ce qui doit être inclus ou omis,
- la structuration du document à produire,
- la lexicalisation : les phrases / mots / structures requises pour construire le message,
- l'agrégation : comment le contenu informationnel du message doit ou peut être décomposé en segments phrastiques,
- la génération d'entités référentielles : quelles propriétés des entités utiliser,
- la réalisation de surface : comment présenter le contenu du message sous la forme d'une séquence de phrases grammaticalement correctes.

Ils précisent par ailleurs que ces catégories ne recouvrent pas l'intégralité des problèmes que l'on peut rencontrer dans une architecture de génération de texte, d'une part, et qu'il est possible de discuter l'appartenance de certains problèmes à telle ou telle catégorie, d'autre part. Par ailleurs, cette classification n'impose pas que l'architecture du système développé contienne un module par catégorie de problèmes à résoudre. Ce qui signifie que, selon la dimension priorisée par les différents travaux, qu'il s'agisse de la structuration du texte, de la lexicalisation ou encore de la réalisation de surface, l'architecture sera conçue différemment.

Ils proposent de décrire l'architecture selon trois paramètres :

- la modularisation : il s'agit de la description des modules que comprend l'architecture et de comment l'ensemble des tâches à exécuter viennent s'inscrire dans ces différents modules ;
- l'interaction : il s'agit de décrire la façon dont les modules agissent les uns sur les autres. Cela comprend typiquement l'ordre d'intervention des modules et les entrées et sorties de ceux-ci ;
- la représentation : il s'agit de la représentation utilisée pour la communication entre les modules.

A l'aide de ces trois paramètres, ils dessinent deux extrêmes entre lesquels peuvent être situées les architectures de génération de texte en langue naturelle. L'un des extrêmes est représenté par une architecture composée d'un seul module exécutant toutes les tâches. Dans ce cas, il n'y a pas

d'interaction à étudier, ni de représentation à construire. A l'opposé, on trouve une architecture décomposée au maximum, dans laquelle on peut imaginer un module par catégorie de problèmes. Dans cette hypothèse, l'output d'un module constitue l'input du module suivant, à l'exception du module de réalisation de surface qui correspond à l'output du système global.
Reiter et Dale proposent quelques critiques de ce deuxième extrême. Ils expliquent qu'une telle organisation se base sur certaines suppositions pouvant induire en erreur :

- les modules constituent l'intégralité des tâches à accomplir pour la génération de texte en langage naturel,
- les fonctionnalités des différents modules ne connaissent aucune superposition : ils remplissent des fonctions totalement différentes et il n'y a aucun double emploi ;
- il est possible de déterminer un ordre d'application des modules tenant compte des influences respectives d'un module sur les autres.

Ils préconisent donc d'opter pour un compromis prudent, qui doit être motivé par les besoins pour lesquels le système de génération est développé et le domaine d'application de ce système.

L'approche adoptée par le prototype présenté dans cette étude est en effet un compromis. L'architecture est présentée comme comportant cinq modules intervenant de manière successive, mais il serait également possible de concevoir la modularisation de ce système autrement, et donc également prévoir une interaction différente.

Le premier module sert à générer des variantes phonétiques. Il pourrait être subdivisé en deux modules correspondant aux deux tâches principales réalisées : la phonétisation de l'input, et la syllabation structurée de la transcription phonétique.
Le deuxième module effectue la vérification d'existence possible d'une variante orthographique. Ce module pourrait être intégré au premier module, à l'étape des échanges de phonétisations de mots ayant subi les permutations. Ainsi, il servirait de filtre aux candidats phonétiques à l'intérieur même de leur génération, plutôt que de filtrer les outputs du premier module.
Le troisième module, qui consiste en la vérification de bonne formation d'un candidat par examen des catégories grammaticales des unités qui le composent, pourrait alors également être intégré au premier module si le second l'est aussi, ou bien uniquement être intégré au second module. Il permettrait de ne conserver que les variantes orthographiques dont les unités sont associées aux catégories grammaticales contenues par le « modèle » (la phrase en input du générateur de contrepèteries).
Le quatrième module réalise la vérification des phénomènes d'accord. On peut se demander s'il

serait possible de l'intégrer lui aussi au module de génération, le premier. Cette intégration nécessiterait de disposer de structures de traits considérées comme correctes en amont de la vérification, afin de ne rechercher que les orthographes associées à des structures de traits compatibles. On peut imaginer que les mots restés inchangés par rapport à l'input pourraient remplir ce rôle. Cependant une telle démarche demanderait de vérifier la réalisation des phénomènes d'accord à chaque fois où ceux-ci devraient se réaliser. Il faudrait donc appeler le parseur au minimum une fois par candidat examiné. Le temps de calcul serait alors démultiplié. Pour donner un ordre d'idée, lors d'une premier tentative d'implémentation ce choix avait été fait. La vérification de la réalisation des phénomènes d'accord dans les candidats générés avec un seul patron, pour une phrase composée d'une quinzaine de mots, demandait une bonne vingtaine de minutes. La solution proposée ici permet de récupérer les contrepets produits après l'examen de neuf patrons à une permutation en moins d'une minute.

Comme nous l'avons vu, certaines fonctionnalités du quatrième module peuvent être utilisées pour extraire d'un corpus l'ensemble des réécritures possibles des syntagmes, et construire une grammaire. Il serait possible de créer un module totalement indépendant à partir de ces fonctionnalités. Dans le cadre du prototype proposé, l'utilisation du quatrième module a lieu deux fois : une première fois lors de l'examen de la phrase passée en input de l'architecture (qui coïncide avec l'input du premier module) afin d'en vérifier la grammaticalité. Il est ensuite utilisé pour vérifier les phénomènes d'accords au sein des candidats.

En revanche, la question reste ouverte en ce qui concerne le cinquième module. Il est chargé de réaliser une vérification sémantique très simple : détecter combien contient le candidat de termes grivois ou ambigus. Étant donné que cette tâche repose sur l'utilisation d'un lexique, il pourrait être soit intégré au module de vérification orthographique, soit placé avant celui-ci, soit entre celui-ci et le module de vérification des phénomènes d'accord. La dimension encore basique du cinquième module nous porte à croire qu'il ne serait pas profitable de l'intégrer au premier module ou au module de vérification orthographique : étant donné qu'il parcourt le candidat token par token, il pourrait générer des faux positifs. Considérons le mot « cul ». Isolément, il est assurément grossier (le CNRTL le qualifie de « très familier ou trivial »), mais dans une expression comme « cul de bouteille », il perd son sens grivois. D'autre part, il paraît préférable de conserver ce module en tant que module indépendant : cela permet plus de possibilités de réutilisation des autres modules pour des taches différentes. Pour reprendre la conception de Reiter et Dale, intégrer le module de vérification sémantique rendrait l'architecture trop dépendante au domaine (celui de la grivoiserie).

Évaluation du système

La liste des inputs utilisés pour tester le système est en annexe. Il s'agit de quarante deux phrases, pour certaines simplifiées par rapport au corpus. En effet, telles quelles, peu de contrepèteries sont réalisées : le système est très sensible à la ponctuation et aux erreurs de parsing. Nous avons donc transformé les structures interrogatives inversées, par exemple, et enlevé toute ponctuation non finale. Définir des critères d'évaluation pour l'ensemble du système n'est pas simple. A l'idéal, il faudrait pouvoir rendre compte de la responsabilité de chaque module. Pour ce faire, les échecs sont classés par explication de l'échec.

Une autre question se pose : comment considérer les cas dans lesquels la sortie produite contient une contrepèterie mais pas celle prévue ? Pour résoudre ce problème, nous choisissons de considérer chaque contrepet grammatical sorti comme une réussite.

Voici les résultats obtenus pour le test. La phrase affichée par défaut est la phrase en input, et les contrepets trouvés sont entre parenthèses. Les résultats ont été classés selon qu'ils présentent un échec pour telle ou telle raison, ou une réussite.

- problème d'étiquetage en catégories grammaticales

Bernard Pivot en a vu défiler des bouquins sur la deux. ([])
Allons ma fille essuie la vite et bien. ([])
A quoi bon me pousser pour que je vous trompe? ([])
Vous chantez quand je vous laisse ? ([])
Adélaïde rue de La Paix. ([])

- patrons de permutations manquants

Appréciez le tout de mon cru. ([])
Assez de disputes tendues ! ([])
Après de pareils faits vous pouvez vous permettre . ([])
Apprendre à calculer en cent leçons. ([])
Ce n'est pas avec la branche qu'on fait des boulettes . ([])
Adjudant faites bisser l'appel ! ([])
Admirez donc l'écaille de ces moules ! ([])

Ah ma sœur vous avez bêché trois allées ! ([])
Ça pue dans le car. ([])
Boude pas c'est pas ton genre ! ([])
À Wimbledon le juge de touche s'est fait acculer derrière Sanchez. ([])
Brigitte aime les vertes plages. ([])
Au Zambèze les filles sont belles et gentilles. ([])
A l'Armée le lieutenant veut défiler et le général m'engueule. ([])
Attention aux pannes de micro ! ([])
Ce gant m'empêche de broder! ([])
Auberge de Vendée. ([])
Ce camp a été victime d'un coup de semonce. ([])
A vouloir aller plus vite que le son vous risquez de vous briser la nuque. ([])

- problème de neutralisations d'oppositions phonémiques

Ce pignon a royale mine. ([])
Avoir des cheveux ne vous empêche pas de nier. ([])

Voici les cas de réussite :

Ce couvent de femmes a été fondé par les Saluces. (['ce couvent de Sam a été fondé par les phallus.'])
Aimer le blanc est une histoire de goût. (['aimer le gland est une histoire de Bou.', 'aimer le gland est une histoire de bout.', 'aimer le gland est une histoire de bouts.', 'aimer le gland est une histoire de boues.', 'aimer le gland est une histoire de boue.'])
Ce que votre plante me fait ! (['ce que votre fente me plaît.'])
Ce beau maillot excitait les foules. (['ce faux maillot excitait les boules.'])
En voyant les nippons la Chine se souleva. (['en voyant les nichons la pine se souleva.'])
Visiter le salon des vins ? (['visiter le vallon des sein.', 'visiter le vallon des seins.',])
Ici on peut voir un vieux plan de Gap. (['ici on peut voir un vieux gland de papes.', 'ici on peut voir un vieux gland de pape.'])
Ça sent la frite de Malbat ! (['Ça sent la bite de malfrat.', 'Ça sent la bite de malfrats.'])
Après un apéritif les penseurs aiment dîner. (['après un apéritif les danseurs aiment piner.'])
En voyant la Chine la jeune fille est envahie par une étrange pâleur. (['en voyant la pine la jeune fille est envahie par une étrange chaleur.'])

Beethoven détestait les sons courts. (['Beethoven détestait les cons sourds.'])
Dans ce clip il aime les sons. (['dans ce slip il aime les cons.'])
Ce jeune homme a la mine piteuse. (['ce jeune homme a la pine miteuse.'])
Achète que je rie ! (['arrête que je chie.'])
Ce qui me plaît quand je dîne est la purée. (['ce qui me plaît quand je pine est la durée.'])
Avez vous vu le bond de la crue. (['avez vous vu le con de la bru.'])

Il y a donc 16 cas de réussite, soit 38 % des cas évalué) , et 26 cas d'échec (soit 62 % des cas évalués). Parmi ces échecs, 73 % sont dus à un patron de permutation non implémenté, 19 % à des erreurs générés par la vérification des similitudes entre les suites de catégories grammaticales et 8 % à des problèmes d'oppositions entre phonèmes ('oe' vs. 'eu', ou 'ai' vs 'ei', par exemple). Dans le cas de réussite, on constate que 3 (soit 18,75%) contiennent des contrepets qu'on peut considérer comme des faux positifs : ils sont agrammaticaux.

Il est difficile de donner un sens à l'évaluation de ce système dans l'absolu. D'une part, parce que les caractéristiques des inputs sont vraiment très variées (nombre et nature des mots, constructions plus ou moins canoniques, nombre de permutations à effectuer…). D'autre part, parce que le résultat est, comme mentionné précédemment, très sensibles à plusieurs facteurs :

- la présence ou l'absence des mots dans les lexiques,
- l'absence d'un patron de permutation adéquat,
- les étiquettes grammaticales fournies par le parseur, qui ne sont pas toujours fiables,
- les phénomènes d'accord pas encore modélisés.

Les deux premiers problèmes sont simples à corriger : il suffit d'enrichir les lexiques (fichiers texte brut) et d'implémenter les patrons manquants. Les deux derniers en revanche sont plus complexes, et demandent des adaptations de modélisation.

De manière générale, le système fonctionne bien pour les inputs courts, à condition que les patrons de permutations soient les bons : de préférence des syntagmes nominaux ou des syntagmes verbaux, mais il est également possible de tester des phrases complètes (voir les logs en annexe). En revanche, dès que l'on cherche à générer des contrepèteries pour des inputs plus complexes, le système se heurte à des obstacles qu'il faudra résoudre.

Conclusions et perspectives d'amélioration

L'ensemble des systèmes de génération de texte mentionnées dans ce mémoire peuvent être rangées en deux grandes catégories.

D'un côté, ceux qui cherchent, en gros, à mettre en mots, de manière grammaticale, un contenu informationnel. Dès le départ, il y a du sens matérialisé en mots. La génération de contrepèteries ne fonctionne pas comme ça : s'il y a bien une contrainte sémantique sur le texte à générer, les éléments lexicaux de la phrase ne dépendent pas de sens à réaliser. Dans le prototype présenté, la sémantique intervient à l'opposé, au dernier moment. En cela, on peut rapprocher la génération de contrepèteries de la génération de poésie ou du générateur de paroles de rap : le sens de la phrase ou du texte généré n'est pas connu d'avance, mais une contrainte s'exerce tout de même sur ce sens : critères du poétique, spécificités syntaxiques du au modèle appris, ou dimension grivoise.

Un autre point, qui découle du précédent, explique que les contraintes d'unification de traits morphologiques (ou phénomènes d'accord) n'interviennent pas au même moment. A l'intérieur de ces architectures, l'objet intermédiaire (la représentation) transformé d'un module à l'autre est une structure abstraite, plus ou moins lexicalisée, jusqu'à la sortie du réalisateur de surface, qui correspond souvent à la sortie de l'architecture. C'est sur l'un des états de cet objet intermédiaire qu'interviennent les structures de traits morphologiques : elles servent de contraintes de génération. Le module de vérification des accords de ce prototype effectue une vérification a posteriori, parce qu'il ne s'agit pas de choisir un mot-forme en accord avec des traits, le choix a été fait par un autre module en amont.

Le système proposé constitue un point de départ. D'autres patrons de permutation ont été repérés et peuvent être implémentés. Il apparaît clair que la propriété de similitude des séquences d'étiquettes grammaticales de l'input et de son (ses) contrepet(s), si elle a le mérite de limiter considérablement le nombre de candidats, implique des développements supplémentaires. D'une manière générale, le prototype demande à être étoffé, et le grain d'analyse des modules mieux paramétré. La comparaison de l'input et du candidat basé sur les suites de catégories grammaticales est trop restrictif (grain trop fin), il vaut mieux remonter d'un niveau. Des ajustements devront tout de même être faits pour le traitement des clitiques élidés. La

vérification des accords doit en revanche être paramétrée avec un grain plus fin : selon les catégories que l'on retrouve dans un syntagme, les accords ne se réalisent pas de la même manière (par exemple '(VN (CLS xxx) (V xxx))' vs. '(VN (CLS xxx) (v xxx) (VPP xxx))' ou '(VN (Vinf xxx))'). Il faut donc modéliser ces phénomènes d'accord. En ce qui concerne le filtre sémantique, comme admis précédemment, il faut en premier lieu améliorer la reconnaissance des expressions multi-mots (qu'ils soient ambigus ou non). En terme de réalisation de surface, il serait également possible de répliquer la ponctuation de l'input sur le contrepet. Enfin, il faut enrichir les lexiques en termes grivois, grossier, vulgaires. La plupart de ces lexiques ont été appris ou générés à partir de corpus ne contenant que très peu de ces registres / champs lexicaux.

Ressources bibliographiques et webographiques

https://fr.wikipedia.org/wiki/Contrepèterie

https://fr.wikipedia.org/wiki/More_(linguistique)#L.27exemple_du_japonais

Sagot (2010). The Le*fff*, a freely available and large-coverage morphological and syntactic lexicon for French. In Proceedings of the 7th international conference on Language Resources and Evaluation (LREC 2010), Istanbul, Turkey

Le petit livre des contrepèteries, MARTIN, J. Éditions Générales First, 2005

Abeillé, A., Clément, L., & Toussenel, F. (2003). Building a treebank for French. Treebanks, 165-187.

Aberer, K., Cudré-Mauroux, P., & Hauswirth, M. (2003, May). The chatty web: emergent semantics through gossiping. In Proceedings of the 12th international conference on World Wide Web (pp. 197-206). ACM.

Bechet F., 2001, "LIA_PHON - Un systeme complet de phonetisation de textes", revue Traitement Automatique des Langues (T.A.L.) volume 42, numero 1/2001, edition Hermes

Bigi, B., Péri, P., & Bertrand, R. (2012, June). Influence de la transcription sur la phonétisation automatique de corpus oraux. In Journées d'études sur la parole(pp. 449-456).

Campion, N., Closson, J., Carcenac, J., Ferret, O., Grau, B., & Shin, J. (2009). Modélisation de la mémoire sémantique et compréhension des messages par Filtrar-S: Un cyber-outil pour la sécurité globale. In Actes du troisième Workshop Interdisciplinaire sur la Sécurité Globale (WISG 2009).

Castro, S., Cubero, M., Garat, D., & Moncecchi, G. (2016, November). Is This a Joke? Detecting Humor in Spanish Tweets. In Ibero-American Conference on Artificial Intelligence (pp. 139-150). Springer International Publishing.

Castro, S., Cubero, M., Garat, D., & Moncecchi, G. (2016, November). Is This a Joke? Detecting Humor in Spanish Tweets. In Ibero-American Conference on Artificial Intelligence (pp. 139-150). Springer International Publishing.

Denzer, Ralf and Gerald Schimak. “TEMSIS - A Transnational System for Public Information and Environmental Decision Support.” Environmental Modelling and Software 15 (1998): 235-243.

Druetta, R. (2007). Quand le français s’ amuse avec ses… maux: calembours, holorimes, contrepèteries et tutti quanti. Publif@ rum, 6.

Dybala Pawel, Ptaszynski Michal, Higuchi Shinsuke, Rzepka Rafal, Araki Kenji, Wobcke Wayne, Zhang Mengjie - Humor Prevails! - Implementing a Joke Generator into a Conversational System, AI 2008: Advances in Artificial Intelligence: 21st Australasian Joint Conference on Artificial Intelligence Auckland, New Zealand, December 1-5, 2008.

Dybala, P., Ptaszynski, M., Maciejewski, J., Takahashi, M., Rzepka, R., & Araki, K. (2010). Multiagent system for joke generation: Humor and emotions combined in human-agent conversation. Journal of Ambient Intelligence and Smart Environments, 2(1), 31-48.

Elhadad, M., & Robin, J. (1996). An overview of SURGE: A reusable comprehensive syntactic realization component. Technical Report 96-03, Ben Gurion University, Dept. of Computer Science, Beer Sheva, Israel.

Etienne, L. (1971). L'Art du contrepet: petit traité à l'usage des amateurs pour résoudre les contrepèteries proposées et en inventer de nouvelles (Vol. 3392). JJ Pauvert.

Hieu Nguyen, B. (2009). Rap lyric generator.

Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003 pages 252-259

Lavoie, B., & Rambow, O. (1997, March). A fast and portable realizer for text generation

systems. In Proceedings of the fifth conference on Applied natural language processing (pp. 265-268). Association for Computational Linguistics.

Léon, Bhatt et Baligand (1992)

Liste des étiquettes utilisées dans LeFFF, Lionel Clément & Benoît Sagot – INRIA, version 0.1.2

Martin, J., & Audouard, Y. (1988). Sur l'album de la comtesse: 1979-1987: 2300 contrepèteries parues dans le Canard enchaîné et 500 inédites. France Loisirs.

Metzler, D., Bernstein, Y., Croft, W., Moffat, A., Zobel, J.: Similarity measures for tracking information flow. In: Proceedings of CIKM, pp. 517–524 (2005)

Mihalcea, R., and Strapparava, C.: Making Computers Laugh: Investigations in Automatic Humor Recognition. In: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing. HLT ’05, pp. 531–538. Association for Computational Linguistics, Vancouver, British Columbia, Canada (2005)

Momeni, E. (2012, April). Semi-automatic semantic moderation of web annotations. In Proceedings of the 21st International Conference on World Wide Web (pp. 167-172). ACM.

Perceau, L., & Touchet, J. (1934). La redoute des contrepéteries. G. Briffaut.

Rabatel, A. (2015). Points de vue en confrontation substitutifs ou cumulatifs dans les contrepèteries (in absentia). Enjeux du jeu de mots. Perspectives linguistiques et littéraires, 31-64.

Reiter, E., & Dale, R. (2000). Building natural language generation systems. Cambridge university press.

Ritchie, Graeme. Developing the incongruity-resolution theory. 1999.

Sjöbergh, J., & Araki, K. (2007, July). Recognizing humor without recognizing meaning.

In WILF (pp. 469-476).

Skillicorn, David. "Beyond keyword filtering for message and conversation detection." Intelligence and Security Informatics (2005): 231-243.

Spence Green, Marie-Catherine de Marneffe, John Bauer, and Christopher D. Manning. 2010. Multiword Expression Identification with Tree Substitution Grammars: A Parsing tour de force with French.. In EMNLP 2011.

Valitutti, A., Doucet, A., Toivanen, J. M., & Toivonen, H. (2016). Computational generation and dissection of lexical replacement humor. Natural Language Engineering, 22(5), 727-749.

Annexes

Annexe 1 : Correspondances d'encodages de transcription de phonèmes

API	LIA / LIUM	EXEMPLE
/i/	ii	il
/e/	ei	blé
/ɛ/	ai	colère
/a/	aa	plat
/ɑ/	aa	pâte
/ɔ/	oo	mort
/o/	au	mot
/u/	ou	genou
/y/	uu	rue
/ø/	eu	peu
/œ/	oe	peur
/ə/	ee	le
/ɛ̃/	in	plein
/ɑ̃/	an	sans
/ɔ̃/	on	bon
/œ̃/	un	brun
/j/	yy	yeux
/w/	ww	oui
/ɥ/	ui	lui
/p/	pp	père
/t/	tt	terre
/k/	kk	cou
/b/	bb	bon
/d/	dd	dans
/g/	gg	gare
/f/	ff	feu
/s/	ss	sale
/ʃ/	ch	chat
/v/	vv	vous
/z/	zz	zéro
/ʒ/	jj	je
/l/	ll	lent
/ʁ/	rr	rue
/m/	mm	main
/n/	nn	nous
/ɲ/	gn	agneau
/ŋ/	ng	camping

Annexe 2 : Log complet pour un input, illustrant la multitude de candidats

```
Chargement du lexique phonétique du LIUM depuis lexique_lium_utf8.txt (sens = o2p) ...
Le lexique o2p contient 65382 entrées.

Chargement du lexique phonétique du LIUM depuis lexique_lium_utf8.txt (sens = p2o) ...
Le lexique p2o contient 77269 entrées.

Application du patron de permutations...
On va échanger vv avec vv
None
On va échanger vv avec bb
((2, 'bbuu'), (4, 'vvon'))
On va échanger bb avec vv
((4, 'vvon'), (2, 'bbuu'))
On va échanger bb avec bb
None
Ce patron permet de générer 1.0 candidat(s) phono.
[['aavvei', 'vvou', 'bbuu', 'llee', 'vvon', 'ddee', 'llaa', 'kkrruu']]

Génération des orthographes possibles...
['aavvei', 'vvou', 'bbuu', 'llee', 'vvon', 'ddee', 'llaa', 'kkrruu']
o ['avez', 'vous', 'bu', 'le', 'vont', 'de', 'la', 'crue']
o ['avez', 'vous', 'bus', 'le', 'vont', 'de', 'la', 'crue']
o ['avez', 'vous', 'but', 'le', 'vont', 'de', 'la', 'crue']
o ['avez', 'vous', 'buts', 'le', 'vont', 'de', 'la', 'crue']

 Première vérification syntaxique des candidats...

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
On va échanger kk avec vv
((7, 'vvrruu'), (2, 'kkuu'))
On va échanger kk avec bb
((7, 'bbrruu'), (4, 'kkon'))
Ce patron permet de générer 2.0 candidat(s) phono.
[['aavvei', 'vvou', 'kkuu', 'llee', 'bbon', 'ddee', 'llaa', 'vvrruu'], ['aavvei',
'vvou', 'vvuu', 'llee', 'kkon', 'ddee', 'llaa', 'bbrruu']]

Génération des orthographes possibles...
```

```
['aavvei', 'vvou', 'kkuu', 'llee', 'bbon', 'ddee', 'llaa', 'vvrruu']
['aavvei', 'vvou', 'vvuu', 'llee', 'kkon', 'ddee', 'llaa', 'bbrruu']
o ['avez', 'vous', 'vu', 'le', 'com', 'de', 'la', 'Bru']
o ['avez', 'vous', 'vu', 'le', 'com', 'de', 'la', 'bru']
o ['avez', 'vous', 'vu', 'le', 'con', 'de', 'la', 'Bru']
o ['avez', 'vous', 'vu', 'le', 'con', 'de', 'la', 'bru']
o ['avez', 'vous', 'vu', 'le', 'cons', 'de', 'la', 'Bru']
o ['avez', 'vous', 'vu', 'le', 'cons', 'de', 'la', 'bru']
o ['avez', 'vous', 'vu', 'le', 'kong', 'de', 'la', 'Bru']
o ['avez', 'vous', 'vu', 'le', 'kong', 'de', 'la', 'bru']


 Première vérification syntaxique des candidats...

 Seconde vérification des candidats...
avez vous vu le con de la bru.
avez vous vu le cons de la Bru.
avez vous vu le cons de la bru.
avez vous vu le con de la Bru.

Application du patron de permutations...
On va échanger kkrr avec vv
((7, 'vvuu'), (2, 'kkrruu'))
On va échanger kkrr avec bb
((7, 'bbuu'), (4, 'kkrron'))
Ce patron permet de générer 2.0 candidat(s) phono.
[['aavvei', 'vvou', 'kkrruu', 'llee', 'bbon', 'ddee', 'llaa', 'vvuu'], ['aavvei',
'vvou', 'vvuu', 'llee', 'kkrron', 'ddee', 'llaa', 'bbuu']]

Génération des orthographes possibles...
['aavvei', 'vvou', 'kkrruu', 'llee', 'bbon', 'ddee', 'llaa', 'vvuu']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'VU']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'cru', 'le', 'bond', 'de', 'la', 'vus']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'VU']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'crue', 'le', 'bond', 'de', 'la', 'vus']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'VU']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'crues', 'le', 'bond', 'de', 'la', 'vus']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'VU']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'crus', 'le', 'bond', 'de', 'la', 'vus']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'VU']
```

```
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'crut', 'le', 'bond', 'de', 'la', 'vus']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'VU']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'crû', 'le', 'bond', 'de', 'la', 'vus']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'VU']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'VUE']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'Vu']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'Vue']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'Wu']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'vu']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'vue']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'vues']
o ['avez', 'vous', 'crût', 'le', 'bond', 'de', 'la', 'vus']

['aavvei', 'vvou', 'vvuu', 'llee', 'kkrron', 'ddee', 'llaa', 'bbuu']

 Première vérification syntaxique des candidats...

 Seconde vérification des candidats...
avez vous cru le bond de la Wu.
avez vous crues le bond de la Vue.
avez vous crus le bond de la Vue.
avez vous crues le bond de la vues.
avez vous crû le bond de la Wu.
avez vous cru le bond de la vue.
avez vous cru le bond de la Vue.
avez vous crus le bond de la vues.
avez vous crû le bond de la vue.
avez vous cru le bond de la vues.
avez vous crû le bond de la Vue.
avez vous crû le bond de la vues.
avez vous crus le bond de la vue.
avez vous crues le bond de la Wu.
avez vous crus le bond de la Wu.
avez vous crues le bond de la vue.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
On va échanger vv avec vv
None
On va échanger vv avec bb
((0, 'aabbei'), (4, 'vvon'))
Ce patron permet de générer 1.0 candidat(s) phono.
[['aabbei', 'vvou', 'vvuu', 'llee', 'vvon', 'ddee', 'llaa', 'kkrruu']]

Génération des orthographes possibles...
```

```
['aabbei', 'vvou', 'vvuu', 'llee', 'vvon', 'ddee', 'llaa', 'kkrruu']
o ['Abbey', 'vous', 'vu', 'le', 'vont', 'de', 'la', 'crue']
o ['Abbé', 'vous', 'vu', 'le', 'vont', 'de', 'la', 'crue']
o ['abbé', 'vous', 'vu', 'le', 'vont', 'de', 'la', 'crue']

 Première vérification syntaxique des candidats...

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
On va échanger kkrr avec vv
((7, 'vvuu'), (0, 'aakkrrei'))
Ce patron permet de générer 1.0 candidat(s) phono.
[['aakkrrei', 'vvou', 'vvuu', 'llee', 'bbon', 'ddee', 'llaa', 'vvuu']]

Génération des orthographes possibles...
['aakkrrei', 'vvou', 'vvuu', 'llee', 'bbon', 'ddee', 'llaa', 'vvuu']

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
On va échanger vv avec vv
None
On va échanger vv avec bb
((0, 'aabbei'), (4, 'vvon'))
Ce patron permet de générer 1.0 candidat(s) phono.
[['aabbei', 'vvou', 'vvuu', 'llee', 'vvon', 'ddee', 'llaa', 'kkrruu']]

Génération des orthographes possibles...
['aabbei', 'vvou', 'vvuu', 'llee', 'vvon', 'ddee', 'llaa', 'kkrruu']
o ['Abbey', 'vous', 'vu', 'le', 'vont', 'de', 'la', 'crue']
o ['Abbé', 'vous', 'vu', 'le', 'vont', 'de', 'la', 'crue']
o ['abbé', 'vous', 'vu', 'le', 'vont', 'de', 'la', 'crue']


 Première vérification syntaxique des candidats...

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.

Application du patron de permutations...
Ce patron permet de générer 0.0 candidat(s) phono.

Il n'y a pas de contrepèterie possible pour cette phrase avec ce patron.


####################################



Voici le(s) contrepet(s) générés pour 'Avez vous vu le bond de la crue ?' :
(16 patrons testés)

avez vous vu le con de la bru.
avez vous cru le bond de la vue.
avez vous crû le bond de la vue.
avez vous crus le bond de la vue.
avez vous crues le bond de la vue.
```

```
Vérification sémantique de la grivoiserie…
Il reste (en ordre décroissant si plusieurs contrepets) :

avez vous vu le con de la bru.
```

Annexe 3 : Corpus d'étude

Pour des raisons de présentation, le corpus est joint à ce document sous la forme d'un classeur excel.

Annexe 4 : Extrait du corpus utilisé pour l'évaluation

Les solutions sont dans le corpus d'étude...

"Ce n'est pas avec la branche qu'on fait des boulettes . ",
"Avez-vous entendu parler de ces mites qui courent sur les biches ? ",
"Appréciez le tout de mon cru. ",
"Avoir des cheveux ne vous empêche pas de nier. ",
"Bernard Pivot en a vu défiler des bouquins sur la deux. ",
"Beethoven détestait les sons courts. ",
"Allons ma fille essuie la vite et bien. ",
"À Wimbledon le juge de touche s'est fait acculer derrière Sanchez. ",
"A vouloir aller plus vite que le son vous risquez de vous briser la nuque. ",
"En voyant les nippons la Chine se souleva. ",
"Ce couvent de femmes a été fondé par les Saluces. ",
"Apprendre à calculer en cent leçons. ",
"Ce gant m'empêche de broder! ",
"Boude pas c'est pas ton genre ! ",
"Admirez donc l'écaille de ces moules ! ",
"Ici on peut voir un vieux plan de Gap. ",
"Achète que je rie ! ",
"Allons les filles ne cousez pas les robes ! ",
"Ce pignon a royale mine. ",
"Auberge de Vendée. ",
"Ça pue dans le car. ",
"Visiter le salon des vins ? ",
"Ce jeune homme a la mine piteuse. ",
"Que j'envie votre brasse ! ",
"Au Zambèze les filles sont belles et gentilles. ",
"A quoi bon me pousser pour que je vous trompe? ",
"Ce beau maillot excitait les foules. ",
"Ce qui me plaît quand je dîne est la purée. ",
"Adélaïde rue de La Paix. ",
"Dans ce clip il aime les sons. ",
"Vous chantez quand je vous laisse ? ",
"Assez de disputes tendues ! ",
"A l'Armée le lieutenant veut défiler et le général m'engueule. ",
"Ce camp a été victime d'un coup de semonce. ",
"Brigitte aime les vertes plages. ",

"Adjudant faites bisser l'appel ! ",
"Avez vous vu le bond de la crue. ",
"Après de pareils faits vous pouvez vous permettre . ",
"Ah ma soeur vous avez bêché trois allées ! ",
"En voyant la Chine la jeune fille est envahie par une étrange pâleur. ",
"Après un apéritif les penseurs aiment diner. ",
"Ce que votre plante me fait ! ",
"Attention aux pannes de micro ! ",
"Aimer le blanc est une histoire de goût. ",
"Ca sent la frite de Malbat !".

Annexe 5 : Elements de permutation et permutations

Définition des éléments de permutation :

element = (numero_syllabe, nature_element, nb_syllabes_dans_mot)

e0 = Element(2,"AC",3)
e1 = Element(1,"AC",1)
e2 = Element(1,"AC",2)
e3 = Element(2,"AC",2)
e4 = Element(1,"ACC",1)
e5 = Element(1,"ACC",2)
e6 = Element(2,"AC+N",2)
e7 = Element(1,"AC+N",2)
e8 = Element(1,"AC1",1)
e9 = Element(1,"AC1",2)
e10 = Element(2,"AC1",2)
e11 = Element(1,"AC2",1)
e12 = Element(1,"AC2",2)
e13 = Element(1,"AC2+N",1)
e14 = Element(1,"AC2+N",2)

Légende :
AC = attaque consonnantique simple (vs. Cluster)
ACC = attaque consonnantique complexe (= cluster)
AC1 = première consonne de l'attaque consonantique complexe (cluster)
AC2 = deuxième consonne de l'attaque consonantique complexe (cluster)
AC + N = attaque consonantique simple + noyau (de la même syllabe)
AC1(/2) = première (/ deuxième) consonne de l'attaque consonantique complexe + noyau (de la même syllabe)

Définition des permutations

permutation = (premier_élément, second_élément)
NB : l'ensemble des patrons de permutations implémentés dans ce prototype ne contiennent qu'une seule permutation, choisie dans la liste ci-dessous. On rappelle que, dans le cadre de

cette étude, un patron de permutation est un ensemble de permutations (minimum une, maximum trois selon le corpus) à apporter sur une transcription phonétique pour passer de l'input au contrepet.

permut0 = [e1,e1]
permut1 = [e1,e2]
permut2 = [e5,e2]
permut3 = [e2,e2]
permut4 = [e1,e11]
permut5 = [e2,e3]
permut6 = [e8,e1]
permut7 = [e4,e1]
permut8 = [e5,e1]
permut9 = [e0,e1]
permut10 = [e3,e1]
permut11 = [e2,e8]
permut12 = [e4,e3]
permut13 = [e3,e1]
permut14 = [e2,e2]
permut15 = [e2,e4]
permut16 = [e0,e2]