Générateur de
« Poèmes pour Bègues »

Table des matières

Introduction

Poésie et Contraintes

Si le poète est, dans l'imaginaire populaire, une incarnation de liberté, la poésie, elle, est la forme littéraire *contrainte* par excellence. La poésie classique caractérise volontiers la poésie par l'art de la rime. Et la rime est une contrainte (on peut, sur ce critère de la rime, valider ou invalider un vers).

Observons qu'une rime n'a d'ailleurs de sens que dans un contexte versifié. En effet les rimes sont la répétition des syllabes finales de deux vers. Et souvent la versification s'accompagne d'une contrainte sur le nombre de syllabes. Ainsi on peut voir la rime comme la répétition d'une même syllabe à intervalle régulier. Et cet intervalle - la longueur d'un vers - est une contrainte supplémentaire.

Si dans un second temps la poésie s'est *libérée* de ces contraintes, pour en revendiquer sa forme moderne, c'est souvent pour se prêter à de nouveaux exercices poétiques. Les contraintes sonores n'ont pas disparues, et prennent la forme d'allitérations - répétition d'un son -, d'assonances - celle de voyelles - etc. Inévitablement la poésie retrouve à être une forme de jeu d'écriture sonore, d'écriture rythmée. La forme plus lointaine des haïkus, respecte également quelques règles : syntagmes courts souvent incomplets, trois vers.

L'OuLiPo

L'Ouvroir de Littérature Potentielle - OuLiPo - est un groupe littéraire né dans l'après-guerre de la rencontre de Raymond Queneau, un écrivain, et de François le Lionnais, un mathématicien. La démarche oulipienne est de poser la production littéraire comme objet d'étude des mathématiques. Les œuvres sont produites sous-contrainte. La rime était un exemple de contrainte d'écriture, les oulipiens recherchent - dans les œuvres pré-oulipiennes -, imaginent de nouvelles contraintes, et s'efforcent à produire des œuvres pour les illustrer. Le tout avec un humour ubuesque, qui frise le « troll » moderne.

Cette démarche tranche notamment avec la démarche du surréalisme, courant auquel appartenait d'ailleurs Queneau, et de la méthode d'écriture automatique - écriture spontanée sans aucune censure consciente.

Deux œuvres en particulier, sont emblématiques de l'œuvre oulipienne

- *Cent-mille milliards de poèmes*, Queneau (1961)
- *La Disparition*, Perec (1969)

Littérature combinatoire

Le premier ouvrage est une œuvre *combinatoire*. L'ouvrage permet au lecteur d'assembler dix vers successifs parmi dix choix de vers à chaque fois. L'ensemble des combinaisons possibles est de 10^{10} poèmes possibles. On trouvera un précurseur de la méthode combinatoire dans la personne de Tristan Tzara qui conseille « pour faire un poème dadaïste » d'assembler aléatoirement les segments d'un journal. Dans cette catégorie d'œuvre on pourrait également ranger « les histoires dont vous êtes le héros », voir même la méthode S+N - remplacer chaque substantif d'un texte par le N-ième mot suivant dans un dictionnaire -. Toutes ces méthodes sont des algorithmes générant par combinatoire des productions poétiques. Toutes, sauf S+N, impliquent activement le lecteur dans les choix de production.

La contrainte formelle

La Disparition, œuvre de Perec est quand à elle une œuvre *contrainte*. Il s'agit d'un *lipogramme littéral* (par opposition aux lipogrammes phonétiques), et l'œuvre est écrit sans aucune apparition de la lettre 'e'.

Formellement, on pourrait périphraser mathématiquement comme ceci :

Soit A *notre alphabet,*
Soit k *le nombre de caractères de l'œuvre,*
Soit $u(n)_{n\in[0,k]\cap N}$ *la suite de* $N \rightarrow A$ *des caractères de l'œuvre,*
Une œuvre est un lipogramme en x*, si et seulement si*

$$\forall\, c \in u(n),\ c \neq x,\ \ x \in A$$

Art génératif

Dans le manifeste de l'OuLiPo : *L'Atlas de Littérature Potentielle,* un chapitre est consacré à la potentialité qui « guette » l'OuLiPo : l'informatique. Il présente trois types d'œuvres possibles incluant l'informatique : la littérature assistée par ordinateur, la littérature générée par ordinateur, la littérature générée par l'ordinateur sur les critères du lecteur.

La double articulation de la génération combinatoire et de l'évaluation de la satisfaisabilité à une contrainte formelle, offre un terrain d'implémentation privilégié à l'informatique. Les possibilités combinatoires d'un ordinateur étant démultipliées, on peut passer de la combinaison de vers à celle de mots.

La première des contraintes qu'un texte respecte est la contrainte de grammaticalité. On peut la résoudre par combinaison pure - ce qui serait la démarche de la G*rammaire Générative* -, ou sur la base d'une combinaison contrainte - ce qu'est l'approche de la théorie *Lexical Functionnal Grammar* (LFG) -. À cette contrainte, nous pouvons adjoindre autant d'autres contraintes formelles que seraient la versification (alexandrins), la rime (plus ou moins riche), une figure

strophique (sonnet, quatrains, haïkus etc.), ou encore n'importe quelle contrainte oulipienne traduite en contrainte formelle.

Par ailleurs il est à noter que la démarche de l'OuLiPo s'applique aussi bien à d'autres domaines artistiques. Et des Ouvroirs équivalents existent : l'OuPeinPo s'applique à la peinture, l'OuSonMuPo au son et à la musique, l'OuTraPo au théâtre etc.… Le tout étant regroupé dans l'OuXPo dépendance du *collège de 'Pataphysique* (on vous avait dit qu'ils avaient de l'humour).

Ces domaines sont également investis par l'Art Génératif. Pour peu qu'on parle de notes musicales ou de pixels au lieu des caractères, on peut par exemple traduire un lipogramme en musique ou en photo. Il est à noter que les algorithmes de *deep learning* ont permis de sérieuses en art-génératif. Les générations musicales « à la façon de » et le *deep dream* de google en sont deux bons exemples.

Et le sens ?

Un impact important de la combinaison contrainte est qu'il faut jauger le bruit et le silence. Si le ratio combinaisons/contraintes est trop en faveur de la combinatoire le calcul devient très long (bruit), s'il est trop en faveur des contraintes il ne trouvera pas ou peu de résultats (silence).

En plus d'être difficile à implémenter, le risque d'une implémentation sémantique est de produire le silence. Le sens ne peut être manipulé qu'au travers des contraintes syntaxiques (transitivité des verbes, place des pronoms), statistiques (probabilités dans les choix des arguments des verbes par exemple) et à la limite être l'objet de descriptions arborescentes (héritage) ou encore via un graphe (les connotations entre les mots).

L'art génératif est un art du hasard. L'interprétation est laissée au lecteur (source bien plus fiable que l'ordinateur pour l'analyse sémiotique). L'implémentation du hasard est essentielle (est-elle seulement possible ?). L'intervention du lecteur dans la production est souhaitable.

Contraintes Syllabiques

Supposons que l'on dispose d'un générateur d'énoncés grammaticaux, d'un phonétiseur de texte, et d'un syllabeur de chaîne phonétique. Voici quelques contraintes qui deviendraient facilement implémentables[1] :

Poèmes pour Bègues (syllabique)

C'est la contrainte visée.
Chaque syllabe se répète au moins deux fois de suite à l'identique.

- Exemples : *« ton tonton tond ton tonton », « les murmures de l'Ararat »*
- La description ensembliste est :

$$\prod_i a_i^n,\ n \geq 2$$

Soit $u(n)_{n \in [1,k] \cap N}$ *une suite de* $N \rightarrow A$ *des syllabes*

- Le cardinal des figures de poèmes pour bègues potentiels est :

$$\sum_{i:[1;\ |u|/2]} i \text{ parmi } |u|-1$$
$$\sim 2^{|u|-1}/2$$
$$\sim 2^{|u|}$$

- Explication : au minimum toutes les syllabes sont identiques - un seul groupe -, au plus chaque syllabe est répétée exactement deux fois (|u|/2 groupes), cela justifie le découpage en *i* groupes identiques. Ce découpage revient à placer *i*-1 séparateurs. Ce qui justifie l'intervalle de la somme. Il y a |u|-1 positions possibles pour un séparateur.
- Évaluation polynomiale (heuristique) : on évalue la distance (binaire ou non) entre chaque paire de syllabe successives. On fait la moyenne de ces distances. Un poème pour bègue minimise cette moyenne. Un seuil doit être fixé. On suppose que le calcul de la moyenne est linéaire. Donc la complexité de l'évaluation est linéaire (si le calcul d'une distance se fait en temps constant).

[1] N.B. : la plupart des contraintes suivantes se traduisent facilement en équivalents syllabique, phonétiques (consonantique et/ou vocaliques) et littéraux. Leur résolutions sont assez proches.

Poèmes Baobab (consonantique)

C'est une généralisation des poèmes pour bègues. Un groupe de consonnes est dit baobab s'il est symétrique (comme les consonnes de « baobab »). La symétrie peut conserver l'ordre (symétrie axiale : /ktr|ktr/) ou le renverser (symétrie centrale : /ktr rtk/ ou /ktrtk/).

- Exemples : « une toge comique agitée » → /(tʒkmkʒt)/ : *baobab* strict ;
  « les murmures de l'Ararat » → /(mr mr)dl(r r)/ : *poème pour bègue* syllabique ;
- Description informelle : là où les poèmes pour bègue ne permettaient la répétition que d'un seul élément (syllabe), les poèmes baobabs permettent la répétition d'un groupe d'éléments (plus d'une syllabe), la répétition dans l'ordre inverse, et l'enchâssement.
- Description formelle : une suite de consonnes est dite baobab s'il existe au moins un centre de symétrie[2]. La qualité dépendra alors de la longueur de la symétrie, du degré d'enchâssement de cette symétrie dans une autre etc...
- Le cardinal des figures de poèmes pour bègues potentiels est :

  $\sum_{i:[1;\,2|u|]} ((i \text{ parmi } 2|u|) * (|u|/2)^i)$
  $< |u| \sum_{i:[1;\,2|u|]} (i \text{ parmi } 2|u|)$
  $\sim |u|\ 2^{|u|}$
- Explication : pour ce problème, on simplifie en admettant que chaque caractère et intervalle inter-caractère peut être le centre d'une symétrie de longueur maximale : |u|/2 (y compris aux bornes). Le nombre de positions potentielles pour un centre de symétrie est |u|-2+|u|-1 ~ 2|u|. On ramène le problème à choisir k indices pour des symétries parmi les 2|u| possibles. Ensuite, pour chaque chaîne de k indices, on a : |u|/2 longueurs possibles. Cela fait $(|u|/2)^k$ choix pour chaque chaîne.
- Évaluation polynomiale (heuristique) : un centre de symétrie de taille n minimise la moyenne des distances entre les paires de consonnes indicées identiquement de part et d'autre du centre de symétrie (à l'inversion près de l'ordre d'indiçage). On fixe un seuil à partir duquel on considère qu'il y a une symétrie. Il reste à parcourir les ~2|u| positions possibles de failles de taille au plus |u|/2. Ce qui fait 2|u|*2|u| = $4|u|^2$ moyennes à calculer. On a admi que le calcul de la moyenne est linéaire. Donc la complexité est cubique (si le calcul d'une distance se fait en temps constant). Un poème baobab contient au moins une faille. Mais une évaluation plus fine peut être mise au point.

[2] *NB. à* un centre de symétrie donné on associe la plus grande symétrie réalisée seulement.

Vers fixe (syllabique)

Le nombre des syllabes d'un vers est fixé arbitrairement (à 12 pour les alexandrins). L'évaluation se fait trivialement dans un temps polynomial, et la nature de l'évaluation est binaire (respect ou non-respect de la contrainte).

Rime (syllabique)

On identifie un vers à sa dernière syllabe. Et on cherche à enchaîner deux syllabes identiques. En imaginant un dictionnaire de vers rangé dans l'ordre de leur dernière syllabe, le temps de l'évaluation devient logarithmique.

Strophe (syllabique)

Une strophe est un enchaînement particulier de vers rimés qui réalisent une figure. Formellement, soit n paires de vers rimés, une figure est une permutation particulière des 2n vers. La taille d'une strophe est fixée avec la figure (quatrains, sonnets, haïkus). Le sonnet classique est typiquement « abba cdcd ». Selon l'exemple choisi, il suffit de générer quatre paires de vers rimés et de leur attribuer une position pertinente dans le sonnet. L'évaluation est triviale.

Remarquons que sont encore baobabs les figures distinguées classiquement des rimes simples ((a|a)(b|b)), des rimes croisées (ab|ab) et rimes embrassés (ab ba). Ce qui ouvre la voie à la contrainte combinée des poèmes en strophe baobab.

Vers ambigus (syntaxique)

Un cas particulier (extrême) de poème-baobab sont les vers ambigus. Admettons qu'un vers, dans sa prononciation, possède deux significations possibles (un mot ayant des homonymes, deux analyses syntaxiques différentes possibles etc...). En accolant la forme écrite de ces deux vers, celles-ci forment une symétrie parfaite. Il s'agit donc de générer des poèmes-baobab parfait ayant obligatoirement une symétrie (axiale) de taille maximale en son milieu. En général, il faut que les deux lectures possibles soient des phrases bien formées.

Exemple : « L'essence de tout et rien ensemble nous rirons, la nuit | Les sens de tout Terrien en sang-bleu nourriront la nuit ». La symétrie entre les deux vers est parfaite.

Le problème est particulièrement difficile en ce sens qu'il ne peut supporter aucune entorse phonologique. Il n'est plus question de minimiser les moyennes des distances mais de rendre cette moyenne parfaitement nulle (pour deux sons distinctifs du moins).

Générateur Syntaxique

Lexique

La base de données employée est celle constituée par l'Université d'Annecy[3]. La base contient 135 000 mots du français avec leurs représentations orthographiques et phonémiques, la syllabation, la catégorie grammaticale, le genre et le nombre, les fréquences, les lemmes associés, etc. Nous n'avons extrait de cette base que les représentations orthographiques, et les informations grammaticales. Nous avons distingué chaque catégorie grammaticale avec les informations morphosyntaxiques, exemple Nom.masc.sing, Nom.fem.sing, et listé ces lexèmes dans des fichiers séparés pour chaque catégorie. Le phonétiseur de texte nous a permis de générer à nouveau la prononciation de chaque lexème. Afin de connaître les éventuelles prononciations de liaisons, pour un lexème donné, on teste sa prononciation suivie d'une consonne et suivie d'une voyelle. Si les prononciations diffèrent, on considérera qu'il y a deux lexèmes distincts dont le contexte phonétique, phone suivant, est contraint.

Une entrée dans notre base de donnée est de la forme :

Forme écrite	[Prononciation-Associée]	Expression Régulière du Contexte Phonétique
abeilles	[abEj]	[bpmtdnNfvszSZkgrl6wj]
abeilles	[abEjz]	[@aeiouyEO123475]

Ainsi, le premier lexème ne se réalisera que devant les consonnes et le second seulement devant les voyelles.

Un enjeux crucial de la catégorisation lexicale en vue d'obtenir des résultats satisfaisant en termes de grammaticalité des énoncés est la sous-catégorisation. Par exemple les adjectifs se divisent entre ceux qui surviennent à gauche du nom qu'ils qualifient et ceux qui surviennent à droite de celui-ci. Les verbes eux se distinguent entre verbes impersonnels, intransitifs, transitifs,

[3] disponible en fichier Excel sur les site lexique.org

bi-transitifs etc… Ces catégorisations plus fines ne sont pas compilées dans la base d'Annecy. Et l'exploration de celles-ci pourrait faire l'objet d'améliorations ultérieurs ; par exemple, VerbeNet et LVF+1 constituent deux bases de données possibles pour la sous-catégorisation verbale.

Chaîne Catégorielle

Munis de notre lexique catégorisé morphosyntaxiquement, il s'agit désormais de les combiner afin de produire des énoncés. Soit N_1 et N_2 les tailles respectives de deux catégories que l'on veut combiner pour produire des fragments lexicaux, le nombre de combinaisons possibles est $N_1 * N_2$.

On appellera chaîne catégorielle, le schéma combinatoire imposé lors d'un enchaînement de combinaisons. Soit un énoncé dont la chaîne catégorielle comporte n catégories de taille N_1, $N_2 \ldots N_n$ le nombre de combinaisons possibles est :

$$\prod_{i:[1;\,n]} N_i$$

Evidemment cette explosion exponentielle du temps de calcul est problématique. C'est pourquoi l'idée est d'élaguer les résultats d'une combinaison avant de les combiner avec un nouvelle catégorie. La capacité maximale des résultats d'une combinaison est un paramètre supplémentaire. Bien sûr, en possession d'une évaluation quantifiée, il s'agit de garder seulement les meilleurs résultats[4]. Dans un second temps, il peut être intéressant d'ajouter, en plus d'une limite en quantité de résultats, une limite en temps de calcul par combinaison de catégorie.

Le programme doit donc connaître le schéma catégoriel à respecter afin de produire un résultat. Il serait amusant à ce stade de pouvoir "souffler" un mot au programme. Cela correspondrait à une catégorie constituée d'un seul élément, le mot imposé. Tel quel, pour un schéma donné, le programme combine les catégories deux par deux de la gauche vers la droite. Un second défi qu'il serait pertinent de relever est de permettre des combinaisons dans d'autres ordres, éventuellement plus que deux par deux, même si là encore une explosion exponentielle est à prévoir.

Exemple de chaîne catégorielle simple :

DET_fs NOM_fs ADJ_fs VERBE_ind3s DET_fp NOM_fp ADJ_fp

Exemple de chaîne catégorielle avec le mot "âmes" imposé comme Objet :

DET_fs NOM_fs ADJ_fs VERBE_ind3s DET_fp âmes **ADJ_fp**

Exemple de chaîne catégorielle avec le mot "âmes" imposé comme Objet, et une combinaison dans un ordre différent que Gauche → Droite :

(((DET_fs NOM_fs ADJ_fs) (VERBE_ind3s (DET_fp âmes **ADJ_fp)))**

[4] Dans cette optique, l'implémentation d'une liste ordonnée efficace, pourrait améliorer grandement le temps d'exécution. Nous nous sommes contentés des algorithmes de tris offert par la bibliothèque standard de java. Mais un AVL aurait été sans doute souhaitable.

Grammaire sur-générée

Une grammaire effective de la langue, serait la liste des chaînes catégorielles valides grammaticalement. La constitution de cette grammaire pourrait être la production d'un module supplémentaire. Une simple expression régulière pourrait décrire beaucoup de chaînes catégorielles valides. Il faudrait alors produire le langage reconnu par un tel automate.

Exemple d'expression régulière pour décrire des chaînes catégorielles avec Sujet en nombre singulier, et Objet quelconque optionnel :

((DET_fs NOM_fs (ADJ_fs)?)|(DET_ms NOM_ms (ADJ_ms)?)) VERBE_ind3s ((DET_fs NOM_fs (ADJ_fs)?)|(DET_ms NOM_ms (ADJ_ms)?)|(DET_fp NOM_fp (ADJ_fp)?)|(DET_mp NOM_mp (ADJ_mp)?))?

Une autre possibilité est de décrire une grammaire transformationnelle classique avec des règles de réécriture. Puis de générer le langage reconnu par cette grammaire. A notre connaissance, il nous semble que la librairie NLTK sur python permet ce genre de "sur-génération".

Exemple de grammaire transformationnelle décrivant la même grammaire que l'expression régulière :

S_ind3s	**→**	**GN_s GV**
GV_ind3s	**→**	**VERBE_ind3s (GN)?**
GN	**→**	**GN_s | GN_p**
GN_s	**→**	**DET_fs NOM_fs (ADJ_fs)? | DET_ms NOM_ms (ADJ_s)?**
GN_p	**→**	**DET_fp NOM_fp (ADJ_fp)? | DET_mp NOM_mp (ADJ_mp)?**

Syllabation

Une des étapes importantes du projet est de découper en syllabes le texte afin de pouvoir l'analyser en fonction de la contrainte imposée.

Il faut dans un premier temps phonétiser les chaînes de caractère. Nous avons pour cela utilisé le logiciel espeak, pour cela nous avons fait appel depuis java à la commande à exécuter sur le terminal.

Dans un second temps nous avons créé des règles pour analyser chaque cas de figure qui se présente à nous lorsque l'on veut découper en syllabe un texte. Ainsi, en parcourant de gauche à droite le texte et en s'aidant de regex pour chaque différent cas, nous avons séparé les syllabes en fonction des voyelles, des noyaux et des successions de consonnes :

- Si on trouve deux voyelles à la suite, alors on ajoute un séparateur entre les deux voyelles ;
- Si on trouve une seule consonne avant une voyelle, alors le séparateur est placé à gauche de la consonne ;
- Si on trouve deux consonnes à la suite, plusieurs cas se présentent à nous :
    - si les deux consonnes apparaissent dans la regex des éléments enchaînables, le séparateur est alors placé avant la première consonne ;
    - sinon, on place le séparateur entre les deux consonnes
- Si on trouve trois consonnes à la suite, on effectue les mêmes conditions que pour deux consonnes mais pour deux paires de consonnes.
  Par exemple, [str] → [st] et [tr].

Distances Syllabiques

Afin de maximiser le nombre de répétitions de syllabes, il faut au programme un moyen de comparer deux syllabes. En se contentant d'une comparaison binaire (identiques vs différentes), nous ne pourrions produire que des suites d'exactement la même syllabe. Pour introduire une plus grande souplesse dans la contrainte, il faut instaurer des distances non binaires entre les syllabes. Pour ce faire nous devrons prendre en compte de manière également non-binaire la distance entre deux consonnes et celle entre deux voyelles.

Nous avons ainsi créé une nouvelle métrique évaluant les syllabes en fonction des consonnes et des voyelles qu'elles contiennent. Pour ce qui est des glides de la langue française, nous avons pris la décision de les considérer à la fois comme des consonnes et comme des voyelles.

Consonnes

Nous avons suivi le tableau de l'API afin de calculer les distances entre consonnes en fonction de quatre critères phonétiques :

- La nasalité, un critère binaire, une consonne étant soit nasale soit orale
- La frication, critère spécifié verticalement sur le tableau des consonnes, une consonne pouvant être plus ou moins fricative, de l'occlusion à la latéralité.
- La position de la langue, critère spécifié horizontalement sur le triangle vocalique.
- Le voisement

Instinctivement, nous avons coefficienté chaque dimension en fonction de leur importance et de leur pertinence pour calculer la distance entre deux voyelles. Ainsi, la frication est plus importante que la nasalité, elle-même plus importante que la position de la langue.

En ce qui concerne la variable MIN_QUALITY, présente dans les deux classes, EspaceVocalique et EspaceConsonantique, elle nous permet de distinguer fortement les consonnes et voyelles qui se ressemblent et celles qui ne se ressemblent selon un seuil dans les distances choisi arbitrairement. Ainsi, si leur distance est supérieure à la variable, elle est directement augmentée à 1.00, la distance maximale.

On trouve ci-dessous l'exemple pour les consonnes de chacune de leur distance si l'on change la qualité minimale à 0.2 (premier tableau) ou à 1.0 (deuxième tableau).

	b	p	t	d	f	v	s	z	S	Z	k	g	l	m	n	r	w	j	5	0
b	0,00	0,03	0,31	0,28	0,37	0,33	0,65	0,61	0,69	0,65	0,63	0,60	0,81	0,77	1,00	1,00	1,00	0,81	1,00	1,00
p	0,03	0,00	0,28	0,31	0,33	0,37	0,61	0,65	0,65	0,69	0,60	0,63	0,85	0,80	1,00	1,00	1,00	0,85	1,00	1,00
t	0,31	0,28	0,00	0,03	0,61	0,65	0,33	0,37	0,37	0,41	0,32	0,35	0,57	1,00	0,80	0,89	0,89	0,57	0,89	1,00
d	0,28	0,31	0,03	0,00	0,65	0,61	0,37	0,33	0,41	0,37	0,35	0,32	0,53	1,00	0,77	0,85	0,85	0,53	0,85	1,00
f	0,37	0,33	0,61	0,65	0,00	0,03	0,28	0,31	0,32	0,35	0,93	0,97	0,51	0,47	0,75	0,83	0,83	0,51	0,83	1,00
v	0,33	0,37	0,65	0,61	0,03	0,00	0,31	0,28	0,35	0,32	0,97	0,93	0,48	0,43	0,71	0,80	0,80	0,48	0,80	1,00
s	0,65	0,61	0,33	0,37	0,28	0,31	0,00	0,03	0,04	0,07	0,65	0,69	0,23	0,75	0,47	0,55	0,55	0,23	0,55	1,00
z	0,61	0,65	0,37	0,33	0,31	0,28	0,03	0,00	0,07	0,04	0,69	0,65	0,20	0,71	0,43	0,52	0,52	0,20	0,52	1,00
S	0,69	0,65	0,37	0,41	0,32	0,35	0,04	0,07	0,00	0,03	0,61	0,65	0,27	0,79	0,51	0,51	0,51	0,27	0,51	1,00
Z	0,65	0,69	0,41	0,37	0,35	0,32	0,07	0,04	0,03	0,00	0,65	0,61	0,24	0,75	0,47	0,48	0,48	0,24	0,48	1,00
k	0,63	0,60	0,32	0,35	0,93	0,97	0,65	0,69	0,61	0,65	0,00	0,03	0,89	1,00	1,00	0,57	0,57	0,89	0,57	1,00
g	0,60	0,63	0,35	0,32	0,97	0,93	0,69	0,65	0,65	0,61	0,03	0,00	0,85	1,00	1,00	0,53	0,53	0,85	0,53	1,00
l	0,81	0,85	0,57	0,53	0,51	0,48	0,23	0,20	0,27	0,24	0,89	0,85	0,00	0,51	0,23	0,32	0,32	0,00	0,32	1,00
m	0,77	0,80	1,00	1,00	0,47	0,43	0,75	0,71	0,79	0,75	1,00	1,00	0,51	0,00	0,28	0,83	0,83	0,51	0,83	1,00
n	1,00	1,00	0,80	0,77	0,75	0,71	0,47	0,43	0,51	0,47	1,00	1,00	0,23	0,28	0,00	0,55	0,55	0,23	0,55	1,00
r	1,00	1,00	0,89	0,85	0,83	0,80	0,55	0,52	0,51	0,48	0,57	0,53	0,32	0,83	0,55	0,00	0,00	0,32	0,00	1,00
w	1,00	1,00	0,89	0,85	0,83	0,80	0,55	0,52	0,51	0,48	0,57	0,53	0,32	0,83	0,55	0,00	0,00	0,32	0,00	1,00
j	0,81	0,85	0,57	0,53	0,51	0,48	0,23	0,20	0,27	0,24	0,89	0,85	0,00	0,51	0,23	0,32	0,32	0,00	0,32	1,00
5	1,00	1,00	0,89	0,85	0,83	0,80	0,55	0,52	0,51	0,48	0,57	0,53	0,32	0,83	0,55	0,00	0,00	0,32	0,00	1,00
0	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00

	b	p	t	d	f	v	s	z	S	Z	k	g	l	m	n	r	w	j	5	0
b	0,00	0,03	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
p	0,03	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
t	1,00	1,00	0,00	0,03	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
d	1,00	1,00	0,03	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
f	1,00	1,00	1,00	1,00	0,00	0,03	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
v	1,00	1,00	1,00	1,00	0,03	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
s	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,03	0,04	0,07	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
z	1,00	1,00	1,00	1,00	1,00	1,00	0,03	0,00	0,07	0,04	1,00	1,00	0,20	1,00	1,00	1,00	1,00	0,20	1,00	1,00
S	1,00	1,00	1,00	1,00	1,00	1,00	0,04	0,07	0,00	0,03	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
Z	1,00	1,00	1,00	1,00	1,00	1,00	0,07	0,04	0,03	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
k	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,03	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
g	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,03	0,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
l	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,20	1,00	1,00	1,00	1,00	0,00	1,00	1,00	1,00	1,00	0,00	1,00	1,00
m	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	1,00	1,00	1,00	1,00	1,00	1,00
n	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	1,00	1,00	1,00	1,00	1,00
r	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,00	1,00	0,00	1,00
w	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,00	1,00	0,00	1,00
j	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,20	1,00	1,00	1,00	1,00	0,00	1,00	1,00	1,00	1,00	0,00	1,00	1,00
5	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,00	0,00	1,00	0,00	1,00
0	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00

Voyelles

Nous avons créé à partir des transcriptions phonétiques d'espeaks de nouvelles transcriptions phonétiques pour certaines voyelles, notamment les voyelles nasales, afin de faciliter les transcriptions et leurs lectures..

@	ə	2	ɛ̃	j	j
E	ɛ	3	ɔ̃	y	y
O	ɔ	4	œ̃	a	a
1	ɑ̃	w	w	e	e
i	i	o	o	u	u

Nous avons suivi le schéma du triangle vocalique afin de calculer les distances entre voyelles en fonction de quatre critères phonétiques :

- La nasalité, un critère binaire, une voyelle étant soit nasale soit non nasale
- L'arrondissement de la bouche
- La position de la langue, critère spécifié horizontalement sur le triangle vocalique.
- L'aperture, critère spécifié verticalement sur le triangle vocalique

Instinctivement, nous avons coefficienté chaque dimension en fonction de leur importance et de leur pertinence pour calculer la distance entre deux voyelles. Ainsi, l'aperture est plus importante que la nasalité, elle-même plus importante que l'arrondissement de la bouche.

Au vu des quantités de coefficients choisis arbitrairement, nous avons conscience qu'un des enjeux du programme soit de les définir à partir du machine learning.

Syllabes

Une syllabe étant composée d'une attaque, d'un noyau et d'une coda, nous avons calculé la distance entre les attaques (succession de consonnes), celles entre les noyaux (successions de voyelles) et celle entre les codas (succession de consonnes). Pour ce qui est des coefficients, nous avons estimé que le bégaiement se faisait entendre essentiellement par les consonnes, moins par les voyelles, c'est pourquoi les noyaux ont un coefficient plus faible que les attaques et les codas.

Nous avons introduit un phone 0, à distance 1 de toutes les voyelles et consonnes afin de pouvoir comparer des successions de consonnes n'ayant pas la même taille. En effet, on insère autant de phone 0 qu'il est nécessaire pour qu'elles aient la même taille pour pouvoir ensuite les comparer.

Aller plus loin...

Le schéma Générateur - Évaluateur a de nombreuses qualités : il permet notamment l'amélioration indépendante de l'un ou l'autre de ces deux modules. Il permettrait également l'accumulation de plusieurs évaluations, et donc l'accumulation de contraintes.

Mais l'indépendance de ces deux modules est dans une certaine mesure un problème. En effet, le générateur n'a pas la moindre idée de ce qu'il fait. Il ne développe aucune heuristique particulière si ce n'est accumuler les mots petit à petit en gardant les meilleurs résultats. Par exemple, pour une suite de mots finissant par la syllabe /ba/, le programme pourrait consulter un dictionnaire pour rechercher un mot commençant par cette même syllabe.

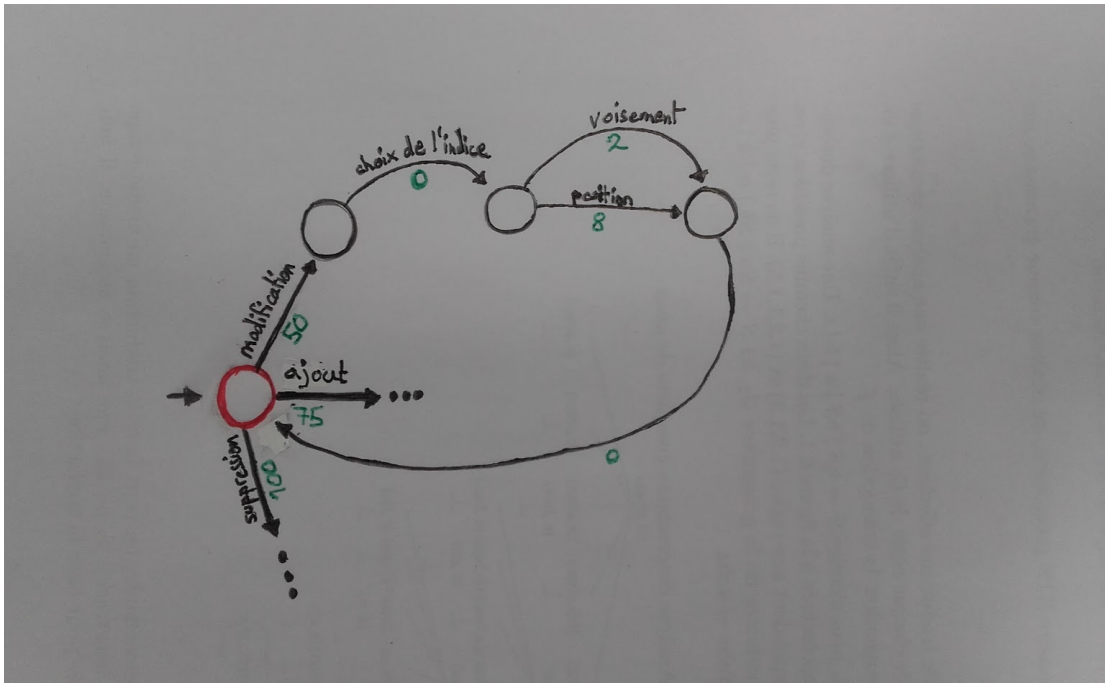
Le problème d'une telle heuristique est qu'il ne rend pas compte de la mesure des syllabes telles que nous l'avons pu constituer pour l'évaluateur. Autrement dit, si aucun mot ne commence pas par /ba/, le programme devrait chercher une syllabe "proche" de /ba/. Or à l'heure actuelle aucune fonctionnalité ne nous le permet.

Réseau de Transitions Augmentées (RTA)

Il faut un modèle permettant au programme de connaître les transformations qu'il peut opérer sur la syllabe. Il faut aussi qu'il privilégie les opérations qui minimisent la distance avec la syllabe résultante.

Formellement, un réseau de transitions augmentées est un graphe orienté, valué, avec une entrée et optionnellement des sorties. Chaque arête comporte aussi une suite d'instructions à exécuter lorsque cette arête est empruntée. Ces instructions peuvent impacter l'objet qui le traverse ou le graphe lui-même. Optionnellement chaque arête peut également être impactée d'une probabilité d'être empruntée et/ou de clauses en interdisant le passage.

Les instructions seront bien-sûr des transformations littérales : ajouter une lettre, transformer une lettre existante, supprimer une lettre etc... Il faut, à intervalle régulier rechercher la syllabe transformée dans le dictionnaire. Si elle s'y trouve, on lui attribue la distance parcourue. Et puisque la distance doit être minimisée, on peut utiliser par exemple une version légèrement modifiée de l'algorithme de Djikstra pour le parcourir, autorisant notamment à emprunter un chemin déjà parcouru.



Voici un exemple de ce à quoi pourrait ressembler un tel RTA. Le nœud rouge est central. À chaque fois qu'il est parcouru, il faut rechercher le résultat provisoire dans le dictionnaire et, s'il est trouvé, attribuer à ce mot la distance parcourue. Une liste ordonnée des mots trouvés pour une exécution bornée dans le temps permet de garder une partie des meilleurs résultats.

Informellement, on pourrait tirer l'analogie suivante : il s'agit de parcourir un labyrinthe avec un mot en poche. Chaque couloir a une certaine longueur, et chaque couloir emprunté ordonne certaines opérations à faire sur le mot. Une fois la sortie trouvée, on attribue au mot transformé la distance parcouru dans les couloirs du labyrinthe.

Cette métaphore naïve nous permet de dérouler mentalement l'algorithme. Précisions tout de suite que notre labyrinthe est plus vicieux encore puisqu'il ne comportera probablement pas de sortie. En effet, ayant un algorithme pour trouver des mots proches. Il sera probablement utilisé pour en trouver plus qu'un seul. Il faudra donc plutôt limiter le parcours du graphe en temps, et rester vigilant à une éventuelle explosion exponentielle en complexité.

On prête d'ailleurs à Raymond Queneau d'avoir défini les oulipiens comme des « rats qui construisent eux-mêmes le labyrinthe dont ils se proposent de sortir ». Cette implémentation serait un hommage amusant à cette image.