Université de Montréal

Apprendre à résoudre des analogies de forme

par
Rafik Rhouma

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique

Juillet, 2018

RÉSUMÉ

L'apprentissage analogique est un mécanisme qui permet le transfert de formes d'entrée (par exemple, des chaînes) vers des formes de sortie sans recourir à l'entrainement d'un modèle. Pour se faire, chaque analogie identifiée dans l'espace d'entrée impliquant la forme d'entrée à traiter donne lieu à une équation analogique dans l'espace de sortie dont la résolution mène à des solutions candidates. Puisque plusieurs analogies peuvent impliquer la forme d'entrée et que la résolution d'une équation génère plusieurs solutions, de nombreuses formes candidates sont typiquement produites parmi lesquelles il convient de filtrer.

Dans cette thèse, nous étudions deux solutions permettant de réduire le bruit généré par ce processus. La première consiste à apprendre à agréger les solutions candidates. Nous comparons à cet effet des classificateurs entraînés de manière supervisée à reconnaitre la solution attendue des autres solutions produites à des systèmes de reclassement d'hypothèses. La seconde solution met à profit l'apprentissage structuré pour apprendre à résoudre une équation analogique en générant moins de formes candidates, de manière à augmenter les chances de prédire en tête la solution attendue.

Nous montrons que les deux solutions permettent d'améliorer significativement les performances d'un système analogique, et ce sur plusieurs tâches de traduction.

Mots clés: analogie formelle, apprentissage analogique, prédiction structurée, réordonnancement de solutions.

ABSTRACT

Analogical learning is a mechanism that allows the transfer of input forms (eg, strings) to output forms without resorting to model training. To do this, each analogy identified in the input space involving the input form to be processed gives rise to an analogical equation in the output space whose resolution leads to candidate solutions. Since several analogies may involve the input form and the resolution of an equation generates several solutions, many candidate forms are typically produced from which to filter.

In this thesis, we study two solutions to reduce the noise generated by this process. The first is to learn how to aggregate candidate solutions. To this end, we compare classifiers trained in a supervised way to recognize the expected solution of the other solutions produced to hypothesis reclassification systems. The second solution uses structured learning to learn how to solve an analog equation by generating fewer candidate forms in order to increase the chances of predicting the expected solution.

We show that both solutions can significantly improve the performance of an analogical system on several translation tasks.

Keywords: Formal analogy, analogical learning, structured prediction, ranking of solutions.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

REMERCIEMENTS

Je souhaite adresser mes remerciements les plus sincères à toutes les personnes qui ont contribué à la réalisation de ce travail.

Cette thèse n'aurait pas été possible sans le soutien de ma famille, de mes amis et aussi de ma tendre moitié. Je tiens aussi à remercier mon directeur de recherche le professeur Philippe Langlais pour l'aide et le temps qu'il a bien voulu me consacrer. Finalement, je tiens à remercier le professeur Guy Lapalme pour son aide lors de la phase de rédaction de cette thèse.

J'adresse, pareillement, une pensée particulière à tous mes professeurs de primaire, de collège, de lycée et de l'université sans exception.

CHAPITRE 1

INTRODUCTION

1.1 Contexte

Une analogie est une relation entre quatre objets a, b, c et d notée $[a : b :: c : d]$ qui s'énonce par "a est à b comme c est à d". Il y a plusieurs types d'analogies comme les analogies numériques ($[3 : 5 :: 7 : 9]$ car par exemple $5\text{-}3{=}9\text{-}7$) ou les analogies textuelles ([*canada* : *ottawa* :: *france* : *paris*]). Ces dernières sont à leur tour divisées en deux types : les *analogies sémantiques* où la relation partagée entre les deux paires d'entités est de nature sémantique (comme dans [*paris* : *france* :: *beijing* : *china*] ou [*marteau* : *clou* :: *tournevis* : *vis*]) et les *analogies syntaxiques*[1]. Parmi les analogies syntaxiques, nous nous intéressons aux analogies dites formelles ou *analogie de forme* comme [*reader* : *unreadable* :: *doer* : *undoable*] que nous définirons dans cette thèse et qui présentent l'avantage d'être manipulables par des automates à états finis.

L'intérêt que nous portons aux analogies de forme tient à leur usage au sein de ce qui s'appelle *l'apprentissage analogique*. Il s'agit d'un paradigme d'apprentissage dit paresseux (Delhay et Miclet, 2004) en ce sens qu'il ne tente pas d'apprendre un modèle d'un corpus d'entraînement, mais utilise plutôt celui ci afin d'y trouver des formes analogues à la forme d'intérêt afin de résoudre un problème. Étant donné un corpus $\mathscr{T}$ de paires constituées chacune d'une forme d'entrée et d'une forme de sortie, un système analogique produit une solution à une forme d'entrée u inconnue en 1) recherchant dans $\mathscr{T}$ les formes d'entrée en relation analogique avec la forme inconnue u, 2) résolvant les *équations analogiques* formées en considérant les formes de sortie associées dans $\mathscr{T}$ aux formes des analogies identifiées et en 3) agrégeant les solutions produites par les

[1] Nous reprenons ici la terminologie traduite de l'article de (Mikolov et Dean, 2013).

équations ainsi construites. Ce processus est schématisé en figure 1.1 et instancié en figure 1.2 dans un cadre de translitération de noms propres du chinois vers l'anglais étudié par (Langlais, 2013).

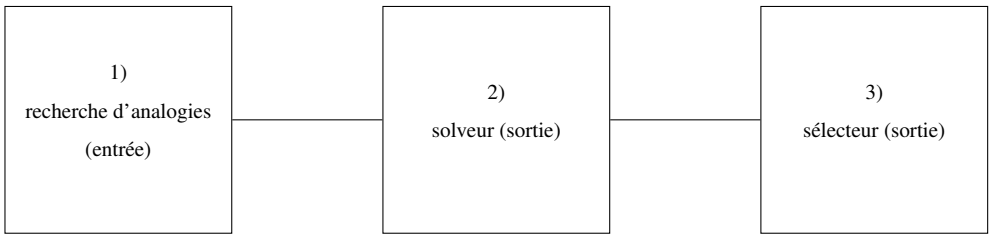

Figure 1.1 : Illustration des trois étapes du processus d'apprentissage analogique.

Cette forme d'apprentissage bien qu'assez marginale dans le paysage du traitement automatique des langues naturelles présente un certain nombre d'avantages. Ainsi, la correspondance entre une forme d'entrée et une forme de sortie est réalisée sans recourir à un modèle qui tenterait d'aligner les formes d'entrée avec les formes de sortie. En effet, les analogies lors de la recherche sont identifiées dans l'espace d'entrée uniquement, tandis que les équations sont résolues dans l'espace de sortie. Seul biais inductif qui considère qu'à une analogie dans l'espace d'entrée correspond une analogie dans l'espace de sortie, permet à l'apprentissage des correspondances. Un autre avantage de cette forme d'apprentissage est la vision paradigmatique qu'elle opère implicitement sur les données Pirelli et Federici (1994). Une analogie comme [*Elle mange une pomme* : *Ils mangent des pommes* :: *Elle avale une couleuvre* : *Ils avalent des couleuvres*] capture (passivement) plusieurs *alternances* comme l'alternance féminin/masculin ou l'alternance singulier/pluriel qui sont utilisables pour traiter d'autres formes les mettant en jeu sans qu'un modèle particulier n'apprenne (par exemple) que *Ils* est la forme masculin/pluriel du pronom *Elle* ou qu'à la troisième personne du pluriel, un verbe se termine au présent par le suffixe *ent*. Cette gestion intrinsèque des relations paradigmatiques apporte une solution élégante à des problèmes où il existe peu d'occurrences d'entraîne-

ment : une seule analogie mettant en jeu une alternance particulière est suffisante à priori pour que l'apprentissage analogique en tire profit. Obtenir des modèles performants sur des observations peu vues en corpus est un enjeu de l'apprentissage automatique pour lequel ce paradigme présente un intérêt.

Enfin, il est important de souligner que l'apprentissage analogique est en mesure de produire des formes non vues à l'entraînement, ce qui n'est pas le cas de toutes les approches d'apprentissage. Cette caractéristique a par exemple été à la base des travaux sur la prédiction de la traduction de mots non vus à l'entraînement (Denoual, 2007, Langlais et Patry, 2007b), un problème qui demeure sans solution satisfaisante en traduction automatique.

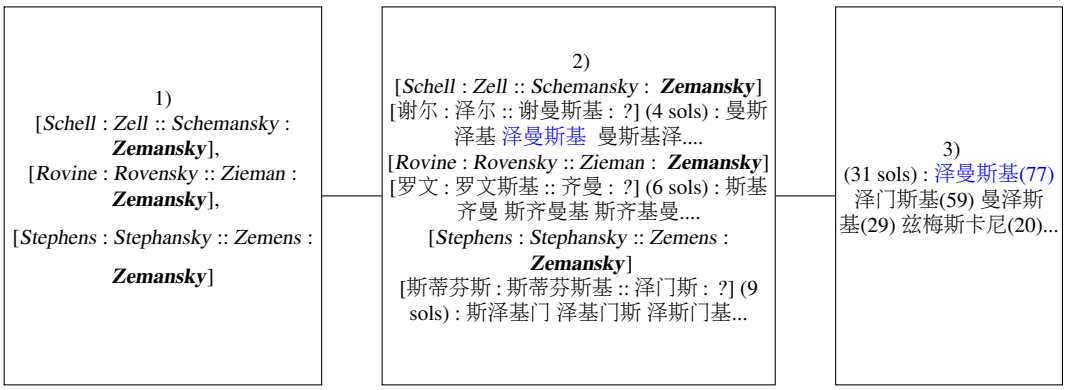

Figure 1.2 : Illustration des trois étapes de l'apprentissage analogique d'après (Langlais, 2013) pour la translitération du nom propre *Zemansky* de l'anglais vers le chinois.

Le processus analogique analogique possède en contre-partie des défis qui limitent son applicabilité en traitement des langues. Identifier les analogies dans l'espace d'entrée est une opération couteuse. De plus, puisque plusieurs analogies sont typiquement identifiées, plusieurs équations sont résolues, chacune — comme nous le verrons — produisant un nombre potentiellement important de solutions. Ceci complexifie l'étape d'agrégation permettant de sélectionner les solutions à retenir du bruit qui a été généré.

Nous décrivons dans cette thèse deux façons de filtrer ce bruit. La première consiste

à classer ou réordonnancer à postériori les solutions produites à l'issue des deux premières étapes de l'apprentissage. La seconde approche consiste à apprendre à résoudre une équation dans le but de générer moins de solutions. Nous tirons profit pour cela de l'apprentissage structuré.

1.2 Organisation de la thèse

Ce mémoire se divise en cinq chapitres. Nous décrivons les concepts nécessaires à la compréhension de l'apprentissage analogique basé sur l'analogie de forme dans le chapitre 2.

Au chapitre 3, nous choisissons des algorithmes d'apprentissage automatique de classification et de réordonnancement pour une tâche d'agrégation des résultats produits par le système de résolution analogique. Comme une application, nous choisissons différentes tâches de traduction de termes médicaux sur plusieurs paires de langues et une tâche de translittération de noms propres en deux langues différentes (chinois et anglais).

Au chapitre 4, nous présentons une technique d'apprentissage automatique appelée la prédiction structurée. Nous étudions l'algorithme de perceptron structuré moyenné pour apprendre à résoudre des équations analogiques formelles. Nous présentons deux solveurs de référence très différents que nous comparons pour un premier jeu de données appliqué sur les mots. Après la réussite de notre solveur STRUCTANALOG à dépasser les performances de ces systèmes, nous proposons (à mode de contribution) un nouveau jeu de données plus difficile, appliqué sur des segments. Nous avons réussi aussi pour cette tâche à dépasser le système de référence et nous avons obtenu des résultats satisfaisants. Nous terminons par une conclusion et des extensions pour des travaux futurs.

CHAPITRE 2

ÉTAT DE L'ART : ANALOGIE ET APPRENTISSAGE ANALOGIQUE

2.1 Introduction

Comme la quantité de données linguistiques numérisées est de plus en plus volumineuse, nous avons besoin d'approches qui ne nécessitent pas de connaissances à priori afin d'exploiter ces données. L'une d'elles est l'analogie. Nous commençons, dans ce chapitre, par la définir, puis nous introduisons le raisonnement par analogie. Nous présentons ensuite le processus d'apprentissage analogique en l'illustrant avec un exemple de translittération des noms propres de l'anglais au chinois. Nous nous intéressons à l'analogie formelle, un type spécifique d'analogie sur les séquences. Nous terminons avec un survol de l'application de l'apprentissage analogique en traitement automatique des langues naturelles (TALN).

2.2 Analogie

L'analogie a été étudiée tout au long de l'histoire de la philosophie (Lepage, 2003). Voici quelques définitions trouvées dans la littérature :

- une ressemblance perçue comme non fortuite entre deux éléments (Wikipédia).
- un rapport d'ordre physique, intellectuel ou moral qui existe à certains égards entre deux ou plusieurs choses différentes (Académie-Française, 1838).
- un rapport de ressemblance, d'identité partielle, entre des réalités différentes préalablement soumises à comparaison (trait(s) commun(s) aux réalités ainsi comparées, ressemblance bien établie, correspondance) (Pierrel, 2004).

- Il y a analogie (ou proportion) lorsque le second nom est au premier comme le quatrième est au troisième. Car on dira le quatrième à la place du second et le second à la place du quatrième. Cette définition est donnée par Aristote et tirée de (Rutten, 1983).

- Une conformité de rapports entre quatre objets de même type (Lepage, 2003).

- L'analogie prédictive est celle qui permet la résolution de problèmes en transférant les connaissances tirées de problèmes suffisamment similaires déjà résolus (Anderson et al., 1986).

Comme nous avons mentionné, les analogies textuelles sont divisées en deux types : des analogies sémantiques et des analogies syntaxiques. Pour le premier type, les quatre formes qui forment ce type d'analogies sont liées sématiquement comme les pays et leurs capitales ([$paris$: $france$ :: $beijing$: $china$]). Pour le deuxième type, ces analogies sont aussi divisées en deux sous types : soit formelles, soit non formelles.
Dans ce travail, nous définissons le concept d'analogie formelle. Il s'agit d'une relation entre quatre formes que l'on peut repérer "visuellement". Nous écrivons par exemple [$reader$: $unreadable$ :: $doer$: $undoable$] pour signifier que la relation liant la forme *reader* à *unreadable* est la même que celle liant *doer* à *undoable*. Ce concept est élaboré plus loin dans ce chapitre.

2.3 Raisonnement par analogie

Il s'agit d'une aptitude humaine qui permet de capter des similitudes entre des observations. Ces dernières peuvent être des chaînes de caractères, des données structurées (par exemple des arbres), des images, etc. Raisonner par analogie permet de comprendre et de prendre des décisions pour de nouvelles situations à partir de situations analogues déjà

vues. Ce concept est intéressant en TALN pour des applications spécifiques que nous détaillerons plus loin.

La première implémentation d'un système analogique est le système ANALOGY proposé par Evans (1968). ANALOGY est capable de résoudre des puzzles analogiques à choix multiples tels que ceux de la figure 2.1. Étant donnés trois pictogrammes, le système infère le quatrième qui est en relation analogique avec les trois premiers.

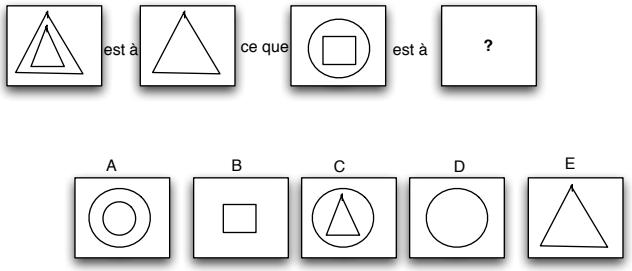

Figure 2.1 : ANALOGY ((Evans, 1968))

Un autre système qui utilise l'analogie est **copycat** (Hofstadter, 1984). Ce dernier résout des analogies au niveau des chaînes de caractères. Par exemple à l'équation $[abc : abbcc :: ijk : x]$, **copycat** produit la solution x= *ijjkk*.

Une autre implémentation d'intérêt tente de résoudre les tests de **Raven** présentés par le docteur John C.Raven en 1936. Chaque test correspond à une question où le candidat est prié de compléter une série d'images. Chaque question est présentée sous forme d'une matrice 3x3 ou 2x2. Les matrices de **Raven** 3x3 contiennent 8 images géométriques affichées dans 8 cellules avec une neuvième cellule vide. L'objectif de ces tests est de deviner l'image manquante parmi une série de candidats proposés tel qu'illustré en figure 2.2.

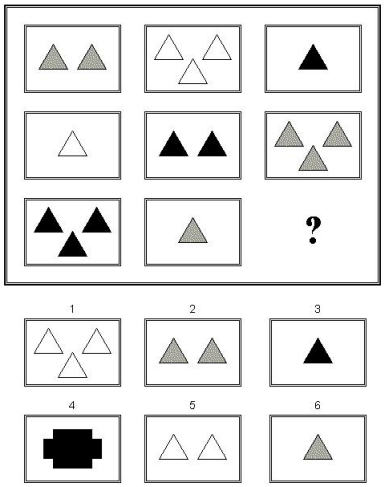

Figure 2.2 : Tests de **Raven** d'aprés (Prade et Richard, 2014), où le sujet doit sélectionner l'un des six pictogrammes proposés pour compléter la grille.

L'analogie était utilisée pour les tests de raisonnement SAT (Scholastic Assessment Test). C'est un examen standardisé utilisé sur une base nationale pour l'admission aux universités aux États-Unis (Turney et Littman, 2005). Une partie de ces tests était menée sous forme de questionnaires à choix multiples (voir exemple du tableau 2.I). L'objet de ces questions est de trouver les relations les plus similaires à celle donnée comme source.

stem		mason :stone
choix	(a)	teacher :chalk
	(b)	**carpenter :wood**
	(c)	soldier :gun
	(d)	photograph :camera
	(e)	book :word

Tableau 2.I : Exemple de question analogique dans les tests SAT, d'après (Turney et Littman, 2005). Un stem est présenté au sujet qui doit identifier un stem réponse parmi cinq proposés. Le stem en gras est la solution attendue.

2.4 Apprentissage analogique

2.4.1 Définition

Soit un ensemble d'apprentissage (ou encore mémoire) $L = \{L_1, \ldots, L_n\}$ composé d'un ensemble de n couples d'observations $< entrée, sortie >$. Nous notons $\mathscr{I}$ l'espace d'entrée, c'est-à-dire l'ensemble toutes les entrées du corpus L et $\mathscr{O}$ l'espace de sortie (ou encore l'espace cible), c'est-à-dire l'ensemble de toutes les chaînes. Soit $I(x)$ la projection dans l'espace d'entrée de x élément de L et $O(x)$ sa projection dans l'espace de sortie. La tâche d'inférence met en place trois étapes (Langlais et Patry, 2007a) pour une forme x dont seule l'entrée est connue :

1. Construire $\varepsilon_{\mathscr{I}}(x) = \{(a,b,c) \in L^3 | [I(a) : I(a) :: I(c) : I(x)]\}$.
2. Construire $\varepsilon_{\mathscr{O}}(x) = \{y | [O(a) : O(b) :: O(c) : y], \forall (a,b,c) \in \varepsilon_I(x)\}$ l'ensemble des solutions trouvées aux équations analogiques formées par projection des triplets analogiques de x dans l'espace de sortie.
3. Choisir $O(x)$ parmi les éléments de $\varepsilon_{\mathscr{O}}(x)$.

Cette forme d'apprentissage analogique s'apparente donc à une approche de type "plus proche voisin" (Beyer et al., 1999), mais possède plusieurs caractéristiques particulières. En particulier, il n'existe pas de distance mise à profit pour identifier le plus proche voisin, mais une recherche d'analogies impliquant la forme incomplète (x). Cette relaxation opératoire s'accompagne cependant d'une recherche à priori cubique dans l'espace d'entrée, tandis que la recherche du plus proche voisin est une opération à priori linéaire dans cet espace.

2.4.2 Exemple

Pour illustrer ce processus, nous prenons empruntons un exemple de (Langlais, 2013) où l'auteur l'applique à une tâche de translittération de noms propres anglais vers le chinois. Le corpus L est constitué de 7 paires de noms propres (anglais,chinois) en relation de translittération. La figure 2.3 illustre les trois étapes de l'apprentissage analogique pour la translittération du nom propre **Zemansky** (qui n'appartient pas au corpus d'entrainement). Les étapes se déroulent comme suit :

1. Nous cherchons l'ensemble de triplets des noms propres en anglais qui forment une analogie avec **Zemansky**. Plusieurs analogies sont identifiées comme [*Schell* : *Zell* :: *Schemansky* : ***Zemansky***], [*Rovine* : *Rovensky* :: *Zieman* : ***Zemansky***], ou [*Stephens* : *Stephansky* :: *Zemens* : **Zemansky**].

2. Nous construisons l'ensemble des équations en projetant les formes des analogies de l'espace d'entrée (les noms propres en anglais) dans l'espace de sortie (les noms propres en chinois). Un nombre d'équation au moins égal[1] au nombre d'analogies sources identifiées est donc résolu.

3. Nous sélectionnons des candidats parmi toutes les solutions de translittération ainsi produites. Ici, les solutions sont triées en ordre décroissant de fréquence. Cette fréquence reflète le nombre d'analogies cibles qui donnent cette solution. Nous obtenons dans cet exemple 31 solutions, la bonne (soulignée) est au premier rang avec une fréquence d'apparition de 77.

[1] Une forme d'entrée peut être associée à plusieurs paires dans L.

L = (Schell, 谢尔), (Zemens, 泽门斯), (Zell, 泽尔), (Schemansky, 谢曼斯基), (Clise,克莱斯), (Rovine, 罗文), (Rovensky, 罗文斯基)

1. Recherche :

   [*Schell* : *Zell* :: *Schemansky* : ***Zemansky***],
   [*Rovine* : *Rovensky* :: *Zieman* : ***Zemansky***],
   [*Stephens* : *Stephansky* :: *Zemens* : ***Zemansky***].

2. Projection et résolution :

   - [*Schell* : *Zell* :: *Schemansky* : ***Zemansky***]
     [谢尔 : 泽尔 :: 谢曼斯基 : ?] (4 sols) : 曼斯泽基 泽曼斯基 曼斯基泽....

   - [*Rovine* : *Rovensky* :: *Zieman* : ***Zemansky***]
     [罗文 : 罗文斯基 :: 齐曼 : ?] (6 sols) : 斯基齐曼 斯齐曼基 斯齐基曼....

   - [*Stephens* : *Stephansky* :: *Zemens* : ***Zemansky***]
     [斯蒂芬斯 : 斯蒂芬斯基 :: 泽门斯 : ?] (9 sols) : 斯泽基门 泽基门斯 泽斯门基...

3. Agrégation :

   (31 sols) : 泽曼斯基(77) 泽门斯基(59) 曼泽斯基(29) 兹梅斯卡尼(20)...

Figure 2.3 : Extrait d'une session de translittération du nom propre anglais **Zemansky** d'après (Langlais, 2013).

Dans cet exemple, la bonne solution est proposée en tête, mais ça n'est bien sûr pas toujours le cas. (Langlais, 2013) montre par exemple que seulement 43% des candidats produits en tête sont des translittérations de référence, malgré le fait que 80% des solutions de référence figurent parmi les 100 premiers candidats produits. Un tel exemple est illustré en figure 2.4. Il est donc important en pratique d'intervenir dans la phase d'agrégation des résultats, ce qui sera abordé au chapitre 3.

rang	*Abbell*	fréquence
1	阿布贝尔	889
2	巴贝尔	842
3	巴布尔	830
4	阿贝尔尔斯	611
5	尔阿贝尔	572
6	阿贝尔	401
7	阿尔贝尔	383
8	布阿布尔	341
9	阿布布尔	276
10	阿贝尔尔	188

Figure 2.4 : Extrait des candidats de translittération du nom propre *Abbell* où la bonne translittération est proposée au 6 ème rang.

2.4.3 Analogie formelle

L'analogie formelle est une analogie de forme, il en existe plusieurs définitions dans la littérature. Une définition est proposée par Hathout (2002) qui s'intéresse aux couples de formes qui partagent le même préfixe par exemple [*reader* : *readeable* :: *doer* : *doable*] où *er* commute en *able*. Dans (Moreau et al., 2007), les auteurs capturent des analogies autorisant à la fois une opération de préfixation et de suffixation comme dans [*republishing* : *unpublished* :: *rediscovering* : *undiscovered*] où *re* commute avec *un* et *ing* commute avec *ed*.

La première définition de l'analogie formelle à caractère général est celle de Lepage (1998). L'auteur exploite certaines propriétés de l'analogie afin d'élaborer un algorithme capable de reconnaitre si quatre chaînes sont en relation analogique (formelle). Cet algorithme permet de résoudre des analogies plus complexes impliquant par exemple plusieurs suffixations et plusieurs préfixations et sert ainsi de définition d'une analogie formelle. Aussi, d'autres opérations morphologiques plus complexes sont permises avec cette définition comme la double infixation dans l'exemple (pris de l'auteur) [*arsala* : *mursilun* :: *aslama* : *muslimun*].

Une seconde définition à caractère général est due à Yvon et al. (2004) et a ensuite été reprise dans (Stroppa et Yvon, 2005). Cette définition met en œuvre le concept de *factorisation* que nous détaillons ci-après. Il n'existe pas de comparaison des deux définitions. Selon Langlais (2009), la seconde reconnait plus d'analogies que la première, aussi c'est celle que nous utilisons dans ce travail.

Définition 1 Soit $f_a = (f_a^1, ..., f_a^n)$ la factorisation de a en n facteurs $f_a^i \in A$, où A est l'alphabet. $[a : b :: c : d]$ est une analogie formelle si et seulement s'il existe une factorisation (f_a, f_b, f_c, f_d) de (*a*,*b*,*c*,*d*) telles que $\forall i \in [1, e] : (f_b^i, f_c^i) \in \{(f_a^i, f_d^i), (f_d^i, f_a^i)\}$, où e est la longueur de la factorisation.

Nous illustrons cette définition à l'aide de [*this guy drinks too much* : *this boat sinks* :: *these guys drank too much* : *these boats sank*] discutée dans (Langlais et Yvon, 2008). La figure 2.4.3 illustre une factorisation de cette analogie, où ε représente la chaîne vide et à des fins de lisibilité _ représente un espace. Les facteurs de chaque forme sont alignés sous forme de colonne de manière à faire ressortir l'élément constitutif de la définition. On peut ainsi facilement vérifier que pour la première colonne ($i = 1$), les facteurs de la première et quatrième forme (*this* et *these*) sont identiques aux facteurs de la seconde et troisième forme (*this* et *these*). Cette égalité ensembliste peut être vérifiée pour chacune des colonnes, ce qui démontre qu'il s'agit bien d'une analogie de forme.

$$
\begin{array}{lcccccc}
f_a^1 & \equiv & this & \_guy & \varepsilon & \_dr & inks & \_too\_much \\
f_b^1 & \equiv & this & \_boat & \varepsilon & s & inks & \varepsilon \\
f_c^1 & \equiv & these & \_guy & s & \_dr & ank & \_too\_much \\
f_d^1 & \equiv & these & \_boat\_ & s & s & ank & \varepsilon
\end{array}
$$

Figure 2.5 : Une factorisation en 6 facteurs des formes de l'analogie [*this_guy_drinks_too_much* : *this_boat_sinks* :: *these_guys_drank_too_much* : *these_boats_sank*].

Il existe plusieurs factorisations prouvant qu'il s'agit d'une analogie, dont celle de la figure 2.4.3 impliquant 9 facteurs. Parmi l'ensemble des factorisations, nous nous intéressons souvent à celles impliquant le moins de facteurs.

$$
\begin{array}{llllllllll}
f_a^2 & \equiv & thi & s & \_gu & y & \varepsilon & \_dr & inks & \_too & \_much \\
f_b^2 & \equiv & thi & s & \_bo & at & \varepsilon & s & inks & \varepsilon & \varepsilon \\
f_c^2 & \equiv & the & se & \_gu & y & s & \_dr & ank & \_too & \_much \\
f_d^2 & \equiv & the & se & \_bo & at\_ & s & s & ank & \varepsilon & \varepsilon
\end{array}
$$

Figure 2.6 : Une factorisation en 9 facteurs de l'analogie [*this_guy_drinks_too_much* : *this_boat_sinks* :: *these_guys_drank_too_much* : *these_boats_sank*].

Nous soulignons qu'une analogie de forme peut être identifiée sans pour autant qu'elle n'ait de sens, comme par exemple l'analogie [*faire* : *taire* :: *foire* : *toire*], où *toire* n'est pas un mot du français. Nous qualifions ces analogies de *fortuites*. L'identification de telles analogies ne pose pas nécessairement de problème lors de l'apprentissage analogique : il est en effet rare qu'une analogie fortuite identifiée dans l'entrée donne lieu par projection à une équation cible admettant des solutions.

On désigne enfin par *degré* d'une analogie le nombre minimum de facteurs que l'on peut trouver à une factorisation vérifiant la définition. Dans notre exemple, on peut montrer que le degré de l'analogie est 6. Intuitivement, une analogie de degré faible met en jeu peu de commutations et a à priori moins de chance d'être fortuite.

2.4.4 Équation analogique

Pour chaque définition d'analogie formelle, il existe un algorithme capable de produire les solutions d'une équation analogique. Les définitions les plus générales mettent en œuvre des procédés plus complexes. Nous recensons deux façons génériques de résoudre une équation analogique [$a : b :: c : ?$].

2.4.4.1 Algorithme de Lepage

La méthode de résolution proposée par Lepage (1998) consiste à synchroniser deux tables d'édition ; la première entre la forme a et la forme b et la deuxième entre a et c. L'algorithme trouve la solution d en concaténant des séquences dans un ordre pres-

crit (et guidé par les tables d'édition). Tout d'abord, les séquences communes entre les formes a et b et entre les formes a et c sont recherchées. La solution est alors produite en concaténant :

1. les sous-séquences communes à a, b et c.
2. les sous-séquences qui sont dans b et qui ne figurent pas dans a.
3. les sous-séquences qui sont dans c et qui ne figurent pas dans a.

Ce procédé est illustré par l'équation analogique [*copieur* : *incopiable* :: *faiseur* : *?*] prise de (Lepage). Les éléments clés de l'algorithme sont :

1. la séquence commune entre les trois chaînes est la séquence *i*
2. les sous-séquences présentes dans *incopiable* mais pas dans *copieur* sont *in* et *able*,
3. les sous-séquences dans *faiseur* mais pas dans *copieur* sont *fa* et *s*.

L'algorithme (Lepage, 1998) tente de mettre ces sous-séquences dans le bon ordre. L'algorithme produit ici *infaisable* comme solution. Il est important de souligner que puisque plusieurs alignements de coût minimal entre deux chaînes peuvent exister (amenant à des sous-séquences potentiellement différentes) cette procédure doit normalement être appliquée plusieurs fois, amenant le solver à produire plusieurs solutions.

2.4.4.2 Algorithme de Yvon et al.

Pour la résolution d'une équation analogique, Yvon et al. (2004) ont proposé une solution qui repose sur la construction d'un automate à états finis. Cette construction nécessite l'introduction de deux notions : la complémentarité et le mélange de deux chaînes.

Mots complémentaires : si v est un sous mot de w, l'ensemble des *complémentaires* de v par rapport à w, noté $w \backslash v$, est l'ensemble de sous mots obtenus en supprimant de w les symboles de v (de gauche à droite).
Par exemple, *spondyondontilalgiatis* \ *ondontalgia* définit un language qui contient 60 formes dont :

spondyiltis car spondyondontilalgiatis \ ondontalgia= spondyiltis.

spydoniltis car *spondyondontilalgiatis* \ *ondontalgia*= *spydoniltis*.

Produit de mélange : le produit de mélange de u et v ($u \bullet v$) contient tous les mots formés des symboles de u et v avec la contrainte que l'ordre relatif des symboles dans chaque forme est respecté dans le mélange. Les règles suivantes sont respectées :

- $u \bullet \varepsilon = \{u\}$
- $u \bullet v = v \bullet u$
- $(u \bullet v) \bullet w = u \bullet (v \bullet w)$

Par exemple, **spondylagia** $\circ$ **ondontitis**, le mélange des mots *spondylagia* et *ondontitis* est l'ensemble constitué de 59 279 mots dont **spondyondontilalgiatis** et **ondspondonylaltitisgia**.
L'étude de (Yvon et al., 2004) montre que les deux opérations peuvent être implémentées par des automates à états finis. Ils proposent alors la définition suivante qui établit que les solutions à une équation analogique sont reconnus par un automate à états finis que l'on peut construire en composant les opérations rationnelles décrites ci-dessus.

Définition

d est une solution de $[a : b :: c : ?]$ **ssi** $d \in \{b \circ c\} \backslash a$

Il convient de noter que la construction d'un tel automate peut cependant faire face à des problèmes combinatoires, ce qui rend cette approche viable pour des analogies impliquant des formes assez courtes.

Une représentation possible de l'espace parcouru est un cube à trois dimensions. Chaque dimension indique une forme parmi les trois entrées (a, b, c) comme l'indique la figure 2.7. Résoudre une équation analogique nécessite la consommation des trois chaînes *a*, *b* et *c*. Ceci construit le chemin d'un "état initial" (aucune forme consommée) vers un "état terminal"(les 3 chaînes sont entièrement consommées). L' état terminal est une solution de l'équation analogique. C'est pour cela que (Langlais et al., 2009) ont proposé un algorithme permettant de parcourir aléatoirement les chemins dans l'automate, sans requérir la construction de cet automate.

Cet algorithme de résolution d'une analogie $[x : y :: z : ?]$ peut être conceptualisé par des parcours dans un cube où les arrêtes sont les trois chaînes de l'équation. Un pas dans l'une des trois directions correspond à une action associée à l'une des deux opérations décrite précédemment. La figure 2.7 illustre l'espace de recherche de cet algorithme. Il est important de souligner que tous les chemins dans le cube ne sont pas atteignables en composant les deux opérations rationnelles décrites, car l'opération de complémentarité requière la correspondance d'un symbole lu avec le dernier symbole du mélange. Nous reviendrons sur cet aspect dans le chapitre 4.

L'algorithme décrit par (Langlais et al., 2009) (appelé `alea` dans la suite) échantillonne des chemins dans ce cube selon le procédé :

1. échantillonner un chemin dans l'automate du produit de mélange, ceci peut être conceptualisé comme le parcours du plan dans le cube constitué en se déplaçant seulement dans les directions associés à *y* et *z*. Le nombre de chemins considérés est contrôlé par un taux d'échantillonnage ρ.

2. appliquer les opérations de complémentarité aux chaines du mélange échantillon-

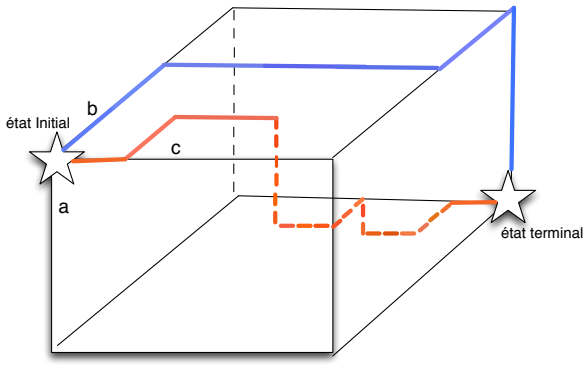

Figure 2.7 : Résolution d'une équation par parcours dans un cube. Deux chemins peuvent mener à deux solutions différentes, la même, voire aucune.

nées à l'étape précédente. Cette opération correspond à se déplacer dans la direction associée à x, dans le but de consommer la chaîne x au complet, ce qui n'est pas toujours possible.

Ces étapes sont formalisées dans l'algorithme 1 qui fait appel aux deux fonctions de mélange et de complémentarité qui sont décrites dans les algorithmes 2 et 3 respectivement où le point désigne l'opération de concaténation, $x[a,b]$ désigne la sous-chaîne formée des symboles $x[a]\ldots x[b]$. (lorsque b est omis il est égale à $|x|$ et $x[:]$ désigne donc x au complet).

Plus on augmente ρ, plus le nombre de solutions générées augmente, augmentant ainsi la chance de générer la bonne solution. Étant donné que plusieurs combinaisons d'opérations de mélange et de complément peuvent conduire à la même solution, nous pouvons classer les solutions en ordre décroissant de leur fréquence.

Pour mettre ce dernier point en perspective, Letard (2017) en se basant même sur l'algorithme (Lepage, 1998) rapporte que l'équation [*advenait* : *advient* :: *contenait* : ?], génère 33590 alignements permettant de produire 126 solutions distinctes (dont les trois premières étant : *contient* *11410*, *conitent* *4780*, *cointent* *3040*). Comme le nombre de solutions est nettement inférieur au nombre d'alignements, il semble raisonnable de trier les

solutions en ordre décroisant de fréquence. Cette approche est aussi utilisée dans (Langlais et al., 2009, Lepage et Denoual, 2005, Stroppa et Yvon, 2005).

Algorithm 1: Solveur `alea` proposé par (Langlais et al., 2009).

Input : 3 formes a, b and c, ρ taux d'échantillonnage
Output : *sol* ensemble de solutions de $[a : b :: c : ?]$

$sol \leftarrow \varnothing$
for i $\leftarrow$1 to ρ **do**
    $m \leftarrow mélange(b, c)$
    $com \leftarrow complémentarité(m, a, \varepsilon, \{\})$
    $sol \leftarrow sol \cup com$
return *sol*

Algorithm 2: Fonction de *mélange* décrite dans (Langlais et al., 2009).

Input : b,c deux formes
Output : un mot aléatoire dans $b \circ c$

function $mélange(b, c)$
  if $(b{=}\varepsilon)$ **then**
    return c
  else
    $n \leftarrow rand(1, |b|)$
    return $b[1 : n].mélange(c, b[n+1 :])$
end function

Afin de mieux apprécier la complexité de la résolution d'une équation par le solveur `alea`, considérons l'équation [*arsala* : *mursilun* :: *aslama* : *?*]. Les mélanges considérés résultant d'un tirage aléatoire, nous avons pour une valeur de ρ donnée, lancé dix fois notre solveur, ceci afin de mettre en relief la variabilité du solveur. Le tableau 2.II résume les résultats obtenus en nombre de solutions produites et le rang de la bonne solution dans la liste triée en ordre décroissant de fréquence, lorsqu"on fait varier le taux ρ entre 1 et 100k.

Nous observons que nous pouvons produire au plus 41 solutions distinctes. Même avec un taux $\rho = 50k$, nous ne pouvons pas produire toutes les solutions possibles (dans l'es-

Algorithm 3: Fonction de *complémentarité* proposée dans (Langlais et al., 2009).

Input : $m \in b \circ c$, a a form
Output : ensemble $m \setminus a$

function *complémentarité* (m,a,r,s)
  if (m=ε) **then**
    if (a=ε) **then**
      $s \leftarrow s \cup \{r\}$
  else
    if (a=ε) **then**
    complémentarité$(m[2:], a, r.m[1], s)$
    if ($m[1]$=$a[1]$) **then**
      complémentarité$(m[2:], a[2:], r, s)$
end function

sai E10). En augmentant le taux d'échantillonnage ρ, le nombre de solutions augmente, mais la bonne solution n'apparaît pas toujours en première position. Nous n'observons pas de corrélation forte entre le taux d'échantillonnage et le rang de la bonne solution. Ceci contribue à montrer que la fréquence aide à agréger les résultats, sans pour autant être un indicateur parfait. On observe également que la résolution d'une équation analogique peut donner zéro solution. Ceci arrive le plus souvent lorsque trop peu de mélanges sont échantillonnés. Il convient de souligner qu'un mélange n'amène pas toujours à une solution. Enfin, même lorsque le solveur produit des solutions, la solution de référence n'est pas nécessairement produite. Il semble globalement qu'il est nécessaire d'échantillonner suffisamment de mélanges pour s'assurer de la présence de la bonne solution dans la liste. Ainsi, avec $\rho = 50$, aucune résolution ne mène à la bonne solution.

ρ	**1**	**10**	**20**	**50**	**100**	**200**	**500**	**1k**	**5k**	**10k**	**20k**	**50k**	**100k**
E1	0_{ϕ}	1_{ϕ}	3_{2}	5_{ϕ}	2_{ϕ}	11_{1}	17_{5}	24_{8}	36_{5}	39_{3}	40_{3}	41_{4}	41_{4}
E2	0_{ϕ}	1_{ϕ}	1_{ϕ}	0_{ϕ}	5_{3}	9_{ϕ}	16_{ϕ}	26_{3}	34_{2}	38_{5}	39_{4}	41_{4}	41_{4}
E3	0_{ϕ}	0_{ϕ}	3_{ϕ}	1_{1}	5_{ϕ}	9_{1}	13_{ϕ}	27_{ϕ}	34_{2}	38_{4}	40_{4}	41_{4}	41_{4}
E4	0_{ϕ}	1_{ϕ}	2_{2}	2_{ϕ}	5_{4}	8_{3}	12_{2}	22_{1}	35_{3}	41_{2}	41_{5}	41_{3}	41_{4}
E5	0_{ϕ}	0_{ϕ}	1_{1}	2_{ϕ}	8_{ϕ}	4_{ϕ}	14_{4}	20_{5}	36_{1}	39_{5}	39_{4}	41_{4}	41_{4}
E6	0_{ϕ}	0_{ϕ}	2_{2}	4_{ϕ}	2_{ϕ}	3_{1}	10_{ϕ}	25_{1}	37_{1}	38_{2}	39_{4}	41_{4}	41_{3}
E7	0_{ϕ}	2_{ϕ}	1_{ϕ}	1_{1}	2_{ϕ}	10_{ϕ}	19_{ϕ}	24_{1}	37_{5}	39_{4}	41_{4}	41_{4}	41_{4}
E8	0_{ϕ}	0_{ϕ}	2_{ϕ}	3_{ϕ}	5_{ϕ}	4_{ϕ}	17_{ϕ}	18_{1}	36_{3}	39_{5}	39_{4}	41_{4}	41_{4}
E9	0_{ϕ}	1_{1}	0_{ϕ}	3_{ϕ}	7_{ϕ}	10_{4}	17_{ϕ}	21_{2}	33_{5}	39_{1}	40_{4}	41_{4}	41_{4}
E10	0_{ϕ}	0_{ϕ}	0_{ϕ}	3_{ϕ}	4_{ϕ}	10_{1}	14_{ϕ}	25_{8}	36_{5}	40_{2}	40_{3}	40_{4}	41_{4}

Tableau 2.II : Dix résolutions de l'équation analogique [*arsala* : *mursilun* :: *aslama* : ?] pour différents valeurs de ρ. Chaque case indique le nombre de solutions différentes produites (à gauche en noir) ainsi en bleu que le rang de la bonne solution. Lorsque la solution attendue n'est pas produite, ce rang est ϕ.

Le tableau 2.III montre les cinq premières solutions (les cinq plus fréquentes) à l'équation de notre exemple pour dix lancements du solveur `alea` avec un taux d'échantillonnage $\rho = 5k$. Nous constatons que la fréquence d'une même solution change pour les dix essais et la solution référence bascule entre les cinq premiers rangs malgré que le nombre des solutions produites est pratiquement le même (entre 33 et 37 solutions distinctes). Aussi, nous avons enregistré un taux de variations de 50% avec seulement ce taux d'échantillonnage, ce qui est important.

Pour appuyer la complexité de la tâche de résolution d'une équation, nous avons fixé un taux ρ élevé ($\rho = 10^6$) et nous avons résolu 1000 analogies de séquences de mots d'un corpus qui sera détaillé dans le chapitre 4. La taille des séquences est plus grande (16 symboles en moyenne) ce qui, nous le verrons plus tard, rend la résolution plus difficile. Le solveur `alea` a généré une moyenne de 963 367 solutions distinctes avec la bonne solution classé en moyenne en 58è position. La figure 2.8 illustre le cas particulier d'une équation où `alea` produit plus de 14.7 millions de solutions distinctes, avec la bonne solution au rang 8. Nous constatons que la bonne solution a une fréquence de 73 158,

36	5	*muslmiun 36*, *slmuimun 35*, *musilmun 31*, *muislmun 24*, ***muslimun** 21*
34	2	*muslmiun 40*, ***muslimun** 30*, *slmuimun 25*, *musilmun 24*, *muislmun 19*
34	2	*muslmiun 36*, ***muslimun** 30*, *slmuimun 28*, *musilmun 25*, *smuilmun 22*
35	3	*slmuimun 39*, *muslmiun 35*, ***muslimun** 25*, *musilmun 24*, *smuilmun 20*
36	1	***muslimun** 30*, *slmuimun 28*, *musilmun 21*, *muslmiun 21*, *smuilmun 19*
37	1	***muslimun** 41*, *slmuimun 36*, *musilmun 28*, *smuilmun 19*, *muislmun 18*
37	5	*slmuimun 30*, *muslmiun 29*, *musilmun 27*, *muislmun 23*, ***muslimun** 14*
36	3	*muslmiun 30*, *slmuimun 28*, ***muslimun** 27*, *musilmun 24*, *smuilmun 22*
33	5	*slmuimun 26*, *smuilmun 21*, *slmumiun 19*, *musilmun 17*, ***muslimun** 16*
36	5	*slmuimun 32*, *muslmiun 31*, *musilmun 28*, *muislmun 19*, ***muslimun** 18*

Tableau 2.III : 5 premières solutions produites par `alea` à l'équation [*arsala* : *mursilun* :: *aslama* :?] pour 10 essais avec le même taux d'échantillonnage $\rho = 5k$. La première colonne indique le nombre de solutions différentes générées, la seconde indique le rang de la solutioon attendue et chaque solution dans la liste est accompagnée de la fréquence avec laquelle elle est générée.

soit trois fois moins que la solution la plus fréquemment générée.

sthe_comm_ission_proposal_and	*208595*	*sthe_commission_and_proposal*	*163307*
sthe_commission_proposaln_ad	*158277*	*s_thecommission_and_proposal*	*96809*
the_commission_propossal_and	*81642*	*s_thecommission_proposal_and*	*80102*
the_commission_propossaln_ad	*73715*	***the_commission_proposals_and***	*73158*
the_commission_proposalsn_ad	*73158*	*smission_andthe_com_proposal*	*64100*

Figure 2.8 : Les dix solutions les plus fréquemment produites par le solveur `alea` à l'équation [*position_of_the_commission* : *the_commission_proposal* :: *positions_of_the_commission_and* : *?*] avec un taux d'échantillonnage $\rho = 10^6$. 14 720 129 solutions différentes sont produites, la bonne (en gras) est au rang 8. Chaque solution est accompagnée de la fréquence avec laquelle elle a été générée.

Ces observations contribuent à montrer que la tâche de résolution d'une équation analogique est difficile et que s'en remettre à l'aléatoire n'est pas suffisant. Ceci justifie un pan de notre thèse où nous étudions comment on peut apprendre à résoudre une équation à l'aide de l'apprentissage structuré. Un autre pan de notre recherche vise à gérer le bruit généré par le solveur lors de l'agrégation.

2.5 Application de l'apprentissage analogique en traitement automatique du langage naturel (TALN)

L'apprentissage analogique est atypique dans le paysage du TALN. L'une de ses particularité tient, comme nous en avons discuté, à l'absence de modèle alignant les entrées aux sorties. Il a cependant été mis à contribution dans différentes tâches du TALN, nous en discutons quelques unes dans la suite.

2.5.1 Traduction automatique

Plusieurs travaux relatent de l'usage de l'apprentissage analogique en traduction automatique. Les premiers à en montrer l'intérêt sont Lepage et Denoual (2005) qui présentent ALEPH. Ce système novateur obtient de bonnes performances dans la tâche partagée IWSLT'05 où il a fini au deuxième rang, en compétition avec des systèmes pour la plupart statistiques. ALEPH a également été utilisé dans un contexte d'alignement sous-phrastique lors de la campagne d'évaluation IWSLT'07 avec le premier rang parmi treize systèmes.

La traduction des mots inconnus est un problème important de la traduction qui a été étudié à l'aide de l'apprentissage analogique par Langlais et Patry (2007b) ainsi que par Denoual (2007). Les premiers auteurs ont montré que leur approche permet de traduire environ 60% des mots inconnus.

Langlais et al. (2009) proposent de traduire les termes médicaux par apprentissage analogique. Ils montrent sur un jeu de tests de termes médicaux extraits de `MESH`, que 64% d'entre eux recoivent une traduction candidate, et que la traduction de référence est présente dans 80% des cas. Nous montrons sur cette même tâche, qu'en appliquant des classifieurs lors de l'étape d'agrégation, on obtient de meilleures performances. Ceci sera décrit dans le chapitre 3.

2.5.2 Translittération

La translittération est la traduction phonétique des noms à travers les langues. Elle est considérée comme une technologie essentielle dans des domaines comme la traduction automatique ou l'extraction d'information (Dandapat et al., 2010). (Langlais, 2013) utilise l'apprentissage analogique pour une tâche de translittération des noms propres de l'anglais au chinois. Les données sont prises de la campagne d'évaluation NEWS 2009 (Li et al., 2009). Le système analogique qu'il a développé obtient des performances qui l'auraient placé troisième dans cette évaluation. Nous avons aussi amélioré ces performances en appliquant d'autres agrégateurs, ceci sera décrit dans le chapitre 3.

2.5.3 Recherche d'information

Pour la recherche d'information, un utilisateur doit écrire sa requête avec des mots qui correspondent le plus aux documents qu'il recherche. Il s'agit d'une tâche difficile, car il faut bien choisir ces mots pour obtenir les documents répondant au besoin. Ceci justifie l'expansion des requêtes pour minimiser les déséquilibres de vocabulaire entre les requêtes et les documents (Selvaretnam et Belkhatir, 2012).
Moreau et al. (2007) ont étudié l'expansion de requête à l'aide d'une approche reconnaissant des analogies simples (préfixation ou suffixation). Ils ont étudié une telle approche pour six langues (allemand, anglais, espagnol, français, italien et portugais). Cette expansion donne de très bons résultats. Elle surpasse presque toujours les résultats obtenus avec les outils existants (entre 10 et 20% de précision) et fournit également des performances plus stables.

2.5.4 Transfert de langage

Plus récemment, (Letard et al., 2016) montre que l'apprentissage analogique peut être mis à profit pour traduire des requêtes en langage naturel comme "supprime le fichier

a.txt" en langage formel comme `rm a.txt` en langage `bash`. Les auteurs ont montré que leur système analogique était à même de traduire correctement 62% des exemples d'un jeu de test.

2.6 Conclusion

Dans ce chapitre, nous avons défini les notions d'analogie, principalement l'analogie formelle que l'on traite tout au long de cette thèse, et d'apprentissage analogique. Nous avons aussi présenté quelques utilisations récentes de l'apprentissage analogique dans différentes tâches du traitement automatique des langues.

Nous avons souligné que le processus analogique génère du bruit, c'est-à-dire des solutions qui sont fausses. Une technique d'agrégation basée sur techniques d'apprentissage statistique peut améliorer les performances existantes. Nous détaillerons ces techniques et les performances obtenues dans le chapitre 3.

Une autre piste consiste à réduire le nombre de solutions candidates produites par le solveur. Nous faisons appel à cet effet à la prédiction structurée. Nous expliquons ces notions et ces techniques avant de détailler notre solveur au chapitre 4.

CHAPITRE 3

AGRÉGATION DES RÉSULTATS PRODUITS PAR ANALOGIE

3.1 Introduction

Nous nous intéressons, dans ce chapitre, à la troisième étape du système global (Sélecteur) décrit dans le chapitre 1. Nous allons introduire des algorithmes d'apprentissage machine pour l'agrégation incluant des algorithmes de classification et de réordonnancement (*ranking*). Nous utilisons la sortie (les candidats produits) d'un solveur analogique (`alea`) décrit dans le chapitre 2.

Deux propositions pour l'agrégation des solutions produites sont évoquées. Une première est de classifier les solutions en bonnes ou mauvaises. L'autre consiste à réordonnancer les hypothèses candidates afin de "hisser" la bonne solution au premier rang.

Nous testons nos solutions sur deux tâches : l'une de translittération des noms propres de l'anglais vers le chinois (et vice versa), l'autre de traduction des termes médicaux en plusieurs langues.

3.2 Les systèmes

3.2.1 Système de référence : `Moses`

Nous comparons un certain nombre de systèmes analogiques avec l'état de l'art[1] en traduction statistique (`Moses`) (Koehn et al., 2007).

Ce système cherche à trouver la meilleure traduction $\widehat{e}$ d'une phrase f en utilisant une combinaison log linéaire de modèles (h_i) y compris un modèle de langue $p(e)$ qui marque la probabilité d'une hypothèse dans la langue cible et un modèle de traduction $p(f|e)$ qui

[1] Du moins au moment où nous avons réalisé ces travaux.

prédit la probabilité que deux phrases soient des traductions :

$$\widehat{e} = argmax_e p(f|e)p(e) = argmax_e exp(\sum_i \lambda_i h_i(e,f))$$

Nous avons entrainé `Moses` au niveau des caractères. Cela a été fait en séparant chaque caractère dans l'entrainement par un espace ; les vrais espaces étant précédemment remplacés par un caractère spécial n'appartenant pas à l'alphabet. Nous avons utilisé la configuration par défaut de `Moses`. Nous avons entrainé un modèle de langue 5-grammes basé sur les caractères sur chaque corpus d'entrainement `TRAIN` (pour chaque langue). Nous avons utilisé chaque corpus de développement `DEV` pour ajuster les coefficients (λ) donnés à chaque modèle. Un extrait de la table de traduction apprise par `Moses` sur la tâche de traduction des termes médicaux est proposé à la figure 3.1.

eckos	|||	echos	|||	0.303	0.006	0.303	0.002
, _ kvinn	|||	, _ fema	|||	0.101	8.3e-09	0.303	2.5e-11
eckrina	|||	eccrine	|||	0.151	0.009	0.303	0.001
edel	|||	ator	|||	0.002	4.6e-06	0.002	1.9e-06

Figure 3.1 : Table de traduction *sw-en*. Les 4 colonnes de score sont des estimations de la vraisemblance d'un couple donné.

3.2.2 Système analogique

Ce système est basé sur les trois étapes que nous avons déjà décrites à la section 2.4 du chapitre 2. Nous avons contrôlé notre générateur analogique afin de traduire chaque corpus de développement `DEV`, en utilisant le `TRAIN` comme mémoire. Les traductions proposées ont été utilisées pour l'entrainement de nos agrégateurs de manière supervisée. Ensuite nous avons généré la traduction des termes de `TEST` avec notre système analogique, en utilisant le `TRAIN` et le `DEV` comme une mémoire. L'ajout du corpus `DEV` à la mémoire utilisée par le générateur est acceptable, car le système analogique

n'opère pas d'entrainement. Nous ne considérons que les cent premiers candidats les plus fréquemment générés pour chaque entrée. Ceci diminue certainement le rappel du système analogique, mais simplifie le processus global. Ces candidats sont transmis à l'agrégateur et un candidat est finalement sélectionné. Le comportement par défaut du système consiste à garder la solution candidate la plus souvent générée. Nous appelons cet agrégateur `Freq`.Langlais et al. (2009) ont entrainé un classificateur binaire pour reconnaître les bons exemples des mauvais.

3.3 Agrégations

Nous présentons deux techniques d'agrégation des résultats : une technique de classification et une autre de réordonnancement.

3.3.1 Classification

L'objectif principal de la classification est de prédire une catégorie ou une classe. Pour notre tâche nous allons prédire si une traduction (ou translittération) est correcte ou non. Nous parlons donc d'une classification binaire qui classifie un exemple donné dans l'une de deux classes. Nous avons essayé plusieurs algorithmes, mais nous décrivons seulement les deux approches qui ont obtenu les meilleurs résultats.

Séparateurs à vaste marge (`Svm`)

Les séparateurs à vaste marge `Svm` (Vapnik, 2013) sont des modèles d'apprentissage supervisés. Un modèle `Svm` est une représentation des exemples en points dans l'espace, cartographié de manière à ce que les exemples de catégories distinctes soient divisés par un écart (la marge) aussi large que possible. `Svm` sont employées dans de nombreux problèmes où les données sont non équilibrées (Chang et Lin, 2011) comme le nôtre.

Voted perceptron

La méthode de voted perceptron (VP) est basée sur l'algorithme du perceptron (Rosenblatt, 1957). L'algorithme tire parti des données qui sont séparables linéairement avec de grandes marges. L'algorithme peut également être utilisé dans des espaces dimensionnels très élevés en utilisant les fonctions noyau. Sassano (2008) a effectué une comparaison expérimentale du VP avec Svm sur les tâches de classification de TALN et a observé que le VP est comparable à Svm en termes de précision. En ce qui concerne le temps d'apprentissage et la vitesse de prédiction, un VP est considérablement plus rapide que Svm.

3.3.2 Réordonnancement

Comme contribution, nous avons pensé, non seulement à utiliser des classificateurs, mais aussi à utiliser des modèles de réordonnancement comme une autre technique d'apprentissage automatique supervisée. Le réordonnancement d'hypothèses par apprentissage machine a connu un essor notamment grâce aux travaux menés en recherche d'information (Liu et al., 2009). Cette technique est également utilisée pour d'autres applications de TALN telles que la traduction (Podelski et Rybalchenko, 2004) et le forage de données (Croft et al., 2010). La technologie learning to rank (Trotman, 2005) est un domaine de recherche qui a émergé dans la dernière décennie.

Il existe diverses formes de problèmes de réordonnancement dans la littérature d'apprentissage machine, y compris le réordonnancement ordinal et le réordonnancement bipartite. Le premier cherche à trier toutes les hypothèses selon un score de pertinence pour trouver un ordre optimal de pertinence en attribuant une réponse ordinale à chaque hypothèse. Ce type de réordonnancement est utilisé souvent en recherche d'information. Pour une requête donnée, un bon classement consiste à retourner les bons documents triés selon un ordre de pertinence. Le classement bipartite est une forme spéciale de

classement, où les hypothèses n'ont que des étiquettes binaires (par exemple positives ou négatives). Une fonction de classement souhaitée attribuerait des scores plus élevés aux instances positives que les instances négatives (Agarwal et Niyogi, 2005).
Nous avons entrainé plusieurs algorithmes de réordonnancement bipartie. Nous avons utilisé une gamme d'algorithmes de la boîte à outils RANKLIB[2], une librairie *open source* pour les algorithmes de réordonnancement. Nous avons expérimenté tous les algorithmes dans cette librairie. Nous décrivons les deux modèles ayant obtenu les meilleurs résultats dans nos tâches :

- `RankSvm` est une variante de l'algorithme machine à vecteur de support, qui est utilisée pour résoudre certains problèmes de réordonnancement. L'algorithme `RankSvm` emploie des méthodes de réordonnancement `pairewise` pour trier les résultats. Il a été publié par (Joachims, 2002). L'objectif initial de l'algorithme était d'améliorer les performances d'un moteur de recherche sur le web.

- `RandomForest`, ou les forêts aléatoires, est une méthode d'apprentissage d'ensemble pour la classification, la régression et d'autres tâches, qui fonctionne en construisant un ensemble d'arbres de décision lors de l'entraînement et en sortant la classe qui est le mode des classes (classification) ou prédiction moyenne (régression) des arbres individuels.

  Le premier algorithme pour les forêts de décision aléatoire a été proposé par (Ho, 1995). Une extension de l'algorithme a été développée par (Breiman, 2001). L'extension combine l'idée du "bagging" de Breiman (2001) et la sélection aléatoire des fonctionnalités, présentée d'abord par Ho (1995) afin de construire une collection d'arbres de décision avec une variance contrôlée.

[2] `https://sourceforge.net/p/lemur/wiki/RankLib/`

3.3.2.1 Les traits caractéristiques

Nous pouvons diviser les traits que nous avons utilisés en trois familles.

ANA la première famille contient environ cinquante traits extraits du système analogique comme : la fréquence de la solution, le rang de la solution dans la liste des candidats, le nombre de couples d'analogies (source, cible) permettant de générer chaque solution, les probabilités du modèle de langue tel que la probabilité minimale d'un caractère étant donnés les deux précédents et le degré ou le nombre de factorisations de l'analogie source ayant servi a produire la solution.

MOS une autre famille des traits est calculée par le système de traduction statistique `Moses` : le score donné par `Moses` à la solution, le rang de la solution dans les *n* meilleures solutions produites par `Moses`, le rapport entre la taille de la séquence à traduire et la taille de chaque traduction selon `Moses`.

IBM la dernière famille est celle que nous avons calculée selon les deux modèles de traduction à base de mots IBM1 et IBM2 (Brown et al., 1993). Nous avons calculé des traits de probabilité lexicale et de probabilité d'alignement entre chaque source et sa traduction cible pour tous les tâches. Nous avons calculé 16 traits de probabilité alignement et de probabilité lexicale dans les deux sens de traduction et l'ordre inverse pour les deux modèles.

3.4 Mesures

Exactitude

L'exactitude mesure la justesse de la première traduction dans la liste candidate pour chaque mot à traduire. Cette mesure vaut 1 si tous les mots ont des traductions correctes et vaut 0 si aucun des meilleurs candidats ne sont corrects.

Rappel au rang n (R@n)

Le rappel au rang n (R@n) est une mesure qui calcule la performance d'un système donné en cherchant parmi les n résultats retournés une bonne réponse. En parcourant les *n* solutions proposées pour chaque exemple, nous mettons 1 si la référence est trouvée parmi ces *n* solutions, 0 sinon. Nous calculons la moyenne à la fin. Cette mesure permet de nous donner une idée de la performance maximale que peut atteindre un système.

Silence

Cette mesure est liée à la tâche d'analogie. Elle caractérise les équations analogiques sans aucune solution trouvée par le solveur. C'est le pourcentage des exemples sans solution produite par rapport au nombre total d'exemples ou d'équations analogiques dans le corpus.

3.5 Données

Nous utilisons deux familles de tâches dans cette étude. La première concerne la traduction de termes médicaux, la deuxième consiste à transcrire des noms propres. Les deux tâches sont importantes dans la pratique, bien qu'elles soient très spécifiques.

3.5.1 Traduction des termes médicaux

Nous utilisons les deux jeux de données[3] décrits dans (Langlais et al., 2009) :

– **MESH :** cette première partie des données vient des terminologies médicales MESH (Medical Subject Headings en anglais). Ce dictionnaire est utilisé par la bibliothèque nationale de médecine du canada pour indexer la littérature scientifique biomédicale

[3] http://rali.iro.umontreal.ca/rali/?q=en/12-medical-translation-tasks.

dans la base de données MEDLINE. Cette base concerne cinq couples de langues avec trois langues européennes relativement proches (anglais-français, anglais-espagnol, anglais-suédois), un plus éloigné (anglais-finnois) et un dernier couple très différent (anglais-russe).

Les données ont été divisées en trois parties choisies aléatoirement (entrainement, développement et test), de sorte que le développement et le test contiennent exactement 1000 termes chacun. Environ un tiers des exemples sont des couples de termes d'un seul mot. Chaque corpus est détaillé dans le tableau 3.I. Des exemples sont donnés en tableau 3.II.

	TRAIN		TEST		DEV	
	nb	*moy*	*nb*	*oov%*	*nb*	*oov%*
Finnois	19787	19.3	1000	65.0	1000	63.8
Francais	17230	21.5	1000	35.8	1000	36.8
Russe	21407	38.5	1000	42.3	1000	45.1
Espagnol	19021	21.5	1000	37.4	1000	34.9
Suédois	17090	17.3	1000	69.3	1000	70.0

Tableau 3.I : Les caractéristiques principales de nos données : *nb* indique le nombre de couples de termes dans un bitexte, *moy* la longueur moyenne en nombre de caractères et *oov%* indique le pourcentage de type hors-vocabulaire (*out-of-vocabulary* types), c'est-à-dire le pourcentage des vocabulaires de DEV ou TEST non vu dans le TRAIN.

Source	**Cible**
en : adenoviridae infections	*fi :* adenovirus infektiot
en : aid to families with dependent children	*sw :* bidrag till barnfamiljer

Tableau 3.II : Quelques paires d'exemples du corpus MESH.

– **MEDDRA :** cette partie aussi contient des termes médicaux espagnol-anglais pris du thésaurus sur les activités réglementaires sur les médicaments MEDDRA (*Medical Drug*

Regulatory Activities en anglais). Cet ensemble de données contient environ trois fois plus de termes que le matériel espagnol-anglais de MESH. Les formes de cet ensemble de données sont généralement plus longues et le pourcentage d'exemples qui sont des couples de termes de mot unique est de seulement 5,6%. Cet ensemble est utilisé pour étudier comment le taux de silence de l'apprentissage analogique évolue avec la taille de l'ensemble d'entrainement.

Ces données ont été réparties en TRAIN, DEV et TEST, tel qu' indiqué en tableau 3.III.

	TRAIN		TEST		DEV	
	nb	*moy*	*nb*	*oov%*	*nb*	*oov%*
Espagnol	65276	34.6	1000	7.1	1000	7.1

Tableau 3.III : Les caractéristiques principales des données MEDDRA.

Source	**Cible**
en : poor urinary stream	*es :* chorro de orina debil
en : intrinsic asthma with status asthmaticus	*es :*asma intrínseca con estatus asmático

Tableau 3.IV : Quelques paires d'exemples du corpus MEDDRA.

3.5.2 Translittération des noms propres (NEWS)

Cette partie contient un corpus de translittération des noms propres en deux directions : la première direction est chinois-anglais (*zh-en*) de l'année 2009 de la campagne d'évaluation NEWS (*Named Entities Workshop Shared Task on Transliteration*) (Li et al., 2009) et nous considérons également la direction de la translittération inverse (*en-zh*). Cela a été fait en changeant simplement les langues source et cible dans le jeu de données NEWS. Ce corpus est le plus grand avec 31961 exemples d'entraînement et 2896 exemples de

déploiement et de test. La longueur moyenne des formes est de 9.5 caractères. Nous détaillons ces données dans le tableau 3.V. Des exemples sont donnés en tableau 3.VI

	TRAIN		TEST		DEV	
	nb	*moy*	*nb*	*oov%*	*nb*	*oov%*
Chinois	31961	9.5	2896	0.0	2896	0.0

Tableau 3.V : Les caractéristiques principales des données NEWS.

Source	**Cible**
en : Abberley	*zh :* 阿伯利
en : Schemansky	*zh :* 谢曼斯基

Tableau 3.VI : Quelques paires d'exemples du corpus NEWS.

3.6 Résultats

Nous représentons nos résultats pour les deux tâches que nous avons étudiées.

3.6.1 News

Nos résultats sont obtenus en exécutant le script officiel d'évaluation de NEWS 2009 avec l'exactitude comme mesure d'évaluation. Le Tableau 3.VII montre les résultats du système analogique (Freq), qui consiste à garder la solution candidate la plus souvent produite, et ceux du système de traduction statistique (Moses).

	Freq		Moses	meilleur	moins bon
	exac	R@100	exac	News2009	News 2009
Anglais-Chinois	43.3	81.5	66,6	73,1	19,9
Silence	3,7	18,5	0,0	-	-
Chinois-Anglais	17.2	64.9	15,4	-	-
Silence	2,5	32,9	0,0	-	-

Tableau 3.VII : Comparaison entre la variante Freq du système analogique et Moses pour les deux directions de translittération en mesurant l'exactitude (exac).

Nous constatons que le système analogique Freq a des bonnes performances sur la direction *zh-en*, mais moins bonne sur l'autre direction *en-zh*. L'agrégation en utilisant la fréquence n'est pas la bonne puisque la R@100 du système analogique dépasse le système Moses dans les deux directions. La mauvaise performance de Moses pour la direction *zh-en* pourrait s'expliquer par le fait que la langue chinoise se base sur des séquences de symboles qui peuvent représenter plusieurs prononciations en anglais. Aussi, la bonne translittération peut contenir des suites de caractères de probabilité faible, ce qui réduit les chances d'obtenir la traduction complète correcte. Le contraire est constaté pour la direction *en-zh* où Moses a des bonnes performances car il y a moins d'ambiguïté. Le meilleur système ayant participé à News2009 est supérieur à Moses mais inférieur à la R@100 du système analogique. Le moins bon système à News2009 est largement inférieur à Moses et même à la version *adhoc* du système analogique Freq. Comme nous avons mentionné, la R@100 permet de renseigner la capacité d'exactitude du système analogique en parcourant les 100 premières solutions proposées.

Nous constatons que le silence augmente avec R@100 par rapport à Freq pour les deux directions de translittération. Le silence pour la version Freq indique le pourcentage des mots sans aucune translittération mais pour R@100 il indique que la bonne solution n'apparaît pas parmi les 100 premières triées par fréquence. Avec la mesure R@100, nous constatons qu'il y a une marge de manœuvre que nous pouvons allez chercher à

travers d'autres types d'agrégateurs.

À l'instar des travaux de Langlais (2013), nous avons utilisé des algorithmes d'apprentissage automatique pour classifier ces résultats.

Nous avons pris 100 solutions produites par le système analogique pour chaque mot à translitérer dans les deux directions. Parmi ces 100 solutions, il en existe éventuellement une qui est correcte, les autres sont fausses. Dans 18% des cas (*en-zh*) et dans 33% des cas pour *zh-en*, il n'existe aucune hypothèse correcte. Nous sommes donc en présence d'un corpus non balancés. Pour cela, nous avons choisi un type de réordonnancement *bipartite*. Ce qui nous intéresse est seulement la bonne solution et non pas l'ordre de toutes les solutions produites. Les résultats obtenus sont détaillés dans le tableau 3.VIII. Nous avons utilisé tous les traits caractéristiques en section 3.3.2.1 (`ANA+IBM+MOS`).

	`Freq`		**classification**		**réordonnancement**		`Moses`
Métrique	`exac`	`R@100`	`exac`	`exac`	`exac`	`exac`	`exac`
Algo			`Svm`	`VP`	`RankSvm`	`RF`	
en-zh	43,3	81,5	62,3	**64,2**	45,0	**64,1**	66,6
Sil	3,7	18,5	3,7	3,7	3,7	3,7	0,0

Tableau 3.VIII : Mesure d'exactitude (`exac`) entre les différents algorithmes d'apprentissage pour la classification et le réordonnancement des solutions produites par analogie. Nous rappelons aussi la performance de `Freq` et `Moses` en précisant le silence (*Sil*) pour chaque variante.

Nous constatons que les meilleures performances sont retournées par `VP` pour la classification et l'algorithme `RandomForest` pour le réordonnancement. Ici encore, nous observons que l'agrégation par classification et par réordonnancement est préférable choisir la solution la plus fréquente. Il n'y a aucune différence significative entre `RandomForest` et `voted perceptron` ici. Nous observons que nous sommes loin des performances de la `R@100` et `Moses` reste encore meilleur (mais de peu).

Le tableau 3.IX rapporte les résultats obtenus en faisant varier les traits utilisés par les modèles pour les deux directions de translittération. Nous nous sommes concen-

trés sur `voted perceptron` et `RandomForest` qui ont montré les meilleures performances. En utilisant les traits caractéristiques d'analogie (`ANA`) seulement, les deux algorithmes d'apprentissage (`VP` et `RF`) ont des bonnes performances et dépassent largement la version `Freq`. En ajoutant les deux autres familles de traits (`IBM` et `MOS`), l'exactitude augmente. Donc, les meilleures performances sont enregistrées avec tous les traits caractéristiques (`ANA+IBM+MOS`) pour les deux algorithmes et aussi pour les deux directions. L'algorithme `RandomForest` est meilleur en moyenne sur les deux directions de translittération. Pour cette tâche de translittération, le réordonnancement est meilleur que la classification.

	classification			**réordonnancement**		
Config	`voted perceptron`			`RandomForest`		
Traits	`ANA`	`ANA+IBM`	`ANA+IBM+MOS`	`ANA`	`ANA+IMB`	`ANA+IBM+MOS`
zh-en	20,0	20,9	21,4	**20,9**	**21,6**	**22,3**
en-zh	57,3	**59,5**	**64,2**	**57,8**	59,2	64,1

Tableau 3.IX : Mesure d'exactitude entre les différentes configurations en fonction des traits caractéristiques utilisés. La meilleure mesure d'exactitude entre la même configuration (mêmes traits) de deux techniques (classification vs réordonnancement) est mise en gras.

3.6.2 MESH

Pour cette tâche, nous avons entrainé nos modèles sur les mêmes données que (Langlais et al., 2009).

1. Il est à noter que notre mise en œuvre de l'apprentissage analogique avec l'agrégateur `Freq` surpasse la configuration équivalente de (Langlais et al., 2009) d'environ 10 points pour l'exactitude.

2. Nous observons une légère réduction du taux de silence, qui demeure élevé, puisqu'en moyenne 54,6% des exemples ne reçoivent aucune solution candidate.

3. Nous observons que Moses, malgré son taux de silence nul, surpasse seulement la variante Freq en moyenne (Freq est meilleur dans la moitié des directions de traduction) . Le tableau 3.X détaille ces résultats.

	(Langlais, 2009)	Freq		Moses
Métrique	exac	exac	R@100	exac
fr-en	18,1(61,5)	**27,3**(59,3)	31,3(68,7)	22,3(0,0)
en-fr	14,6(58,8)	**21,8**(56,0)	28,2(71,8)	20,0(0,0)
ru-en	20,8(57,9)	29,1(56,7)	34,0(66,0)	**33,4**(0,0)
en-ru	18,7(53,8)	29,0(52,5)	35,7(64,3)	**30,5**(0,0)
fi-en	16,4(55,2)	**28,5**(53,7)	34,9(65,1)	27,0(0,0)
en-fi	14,9(52,9)	24,7(50,9)	33,2(66,8)	**26,4**(0,0)
sp-en	20,3(57,4)	**30,5**(55,6)	35,2(64,8)	29,0(0,0)
en-sp	19,5(53,0)	**29,8**(51,6)	37,3(62,7)	28,6(0,0)
sw-en	18,2(55,4)	28,3(54,3)	34,9(65,1)	**38,8**(0,0)
en-sw	15,4(57,2)	26,3(55,2)	33,3(66,7)	**37,0**(0,0)

Tableau 3.X : Mesure d'exactitude sur la tâche MESH en fonction de la direction de traduction. Le silence est indiqué entre parenthèses. La meilleure configuration entre Freq et Moses est mise en gras.

De la même façon que dans la première tâche (NEWS), nous avons appliqué les deux agrégateurs voted perceptron et RandomForest en faisant varier en fonction des traits caractéristiques. Nous observons que ces agrégateurs offrent une meilleure performance que de choisir la forme la plus fréquemment générée. Les gains ne sont pas particulièrement élevés, mais sont cohérents dans toutes les directions de traduction. Dans l'ensemble, il semble que l'algorithme de réordonnancement RandomForest offre la meilleure performance. Cela représente 92% d'exactitude atteignable selon la colonne R@100 qui implique un classificateur parfait. Cela valide l'utilité des traits caractéristiques que nous avons conçus.

	classification			réordonnancement			Freq(R@100)
	voted perceptron			RandomForest			
Traits	A	A+I	A+I+M	A	A+I	A+I+M	
fr-en	28,4*	28,8*	29,2*	28,3*	29,1*	**29,4***	31,3*$_{(68,7)}$
en-fr	23,2*	24,5*	**25,0***	23,0*	24,4*	24,9*	28,2*$_{(71,8)}$
ru-en	29,8	31,8	**32,3**	29,8	31,6	31,8	34,0*$_{(66,0)}$
en-ru	31,0*	32,3*	**32,6***	31,2*	32,4*	32,5*	35,7*$_{(64,3)}$
fi-en	29,8*	31,6*	31,6*	30,7*	31,8*	**32,2***	34,9*$_{(65,1)}$
en-fi	27,2*	28,4*	28,8*	27,4*	28,7*	**29,9***	33,2*$_{(66,8)}$
sp-en	31,9*	32,4*	32,8*	32,0*	32,8*	**32,9***	35,2*$_{(64,8)}$
en-sp	32,3*	34,2*	**34,0***	31,6*	33,5*	**34,0***	37,3*$_{(62,7)}$
sw-en	29,7	31,2	31,9	29,5	31,0	**32,4**	34,9$_{(65,1)}$
en-sw	27,9	29,2	30,1	28,3	30,1	**31,1**	33,3$_{(66,7)}$

Tableau 3.XI : Mesure d'exactitude de différents modèles d'agrégation entraînés selon différents traits caractéristiques (où A indique les traits ANA, I les traits IBM et M les traits MOS) pour la tâche de MESH. La meilleure configuration entre VP et RF est mise en gras et nous mettons une étoile pour les meilleures variantes par rapport à Moses.

Malgré le taux de silence élevé retourné par l'approche analogique Freq (66,2% en moyenne pour avec la métrique R@100), notre système analogique avec l'agrégateur RandomForest avec la troisième variante de traits (ANA+IBM+MOS) surpasse Moses dans 7 directions de traduction excluant ces trois : *en-sw*, *sw-en* et *ru-en*. La moyenne d'exactitude aussi est supérieure (31.1 > 29.3).

3.6.3 Meddra

Pour cette tâche les performances du système analogique Freq sont meilleures que celles du système de traduction statistique Moses. Le problème du silence demeure cependant élevé. Nous observons également que les systèmes analogiques, même ceux

mettant en œuvre un mécanisme simple d'agrégation (Freq), sont beaucoup plus précis que Moses : plus de 30 points en moyenne grâce à la quantité de données.

Config	Freq	R@100	Moses
sp-en	**52,2**$_{(25,1)}$	64,3$_{(34,4)}$	10,2$_{(0,0)}$
en-sp	**45,5**$_{(16,7)}$	61,8$_{(38,2)}$	11,0$_{(0,0)}$

Tableau 3.XII : Mesure d'exactitude des systèmes Freq et Moses. Le silence est indiqué entre parenthèses.

La mauvaise performance du système de traduction statistique pourrait s'expliquer par le fait que les formes dans les ensembles de données MEDDRA sont plus longues en termes de caractères, réduisant ainsi les chances d'obtenir la traduction complète. En outre, nous observons que l'utilisation d'un classificateur est préférable à la recherche de la forme la plus fréquemment générée, et encore une fois, l'algorithme RandomForest fournit la meilleure performance en moyenne. Il est toutefois notable que la performance soit bien inférieure à celle de R@100 : il y a encore place à amélioration. Pour nos configurations pour cette tâche, nous avons exclu les traits MOS car Moses obtient de mauvais résultats.

	Freq	**classification**		**réordonnancement**	
		voted perceptron		RandomForest	
Traits		ANA	ANA+IBM	ANA	ANA+IMB
sp-en	52.2	**55,1**	**56,1**	54,1	55,7
en-sp	45,5	46,8	46,9	**49,3**	**49,5**

Tableau 3.XIII : Mesure d'exactitude de différents modèles entraînés selon différents traits caractéristiques pour la tâche de MEDDRA. La meilleure configuration entre VP et RF est mise en gras.

3.6.4 Combinaison

Comme le système de référence Moses retourne toujours une décision, cela suggère que notre système analogique est en fait plus précis. Nous avons tenté de capitaliser sur cette

information en combinant les deux systèmes. Nous avons implémenté une combinaison simple où le système Freq est cru lorsqu'il propose une solution, et le système Moses sinon.

Ceci est illustré dans la colonne du tableau 3.XIV Cascade (Freq, Moses). Une autre combinaison est considéré aussi où le système voted perceptron est cru lorsqu'il propose une solution sinon Moses.

Nous observons une belle amélioration sur chaque système : près de 10 points de précision en moyenne sont gagnés par cette combinaison (38,6%) pour MESH. La mise en cascade du meilleur système analogique avec Moses donne enfin un léger accroissement de l'exactitude. En fin de compte, le meilleur système que nous avons testé traduisait correctement 41,9% des termes du test en première position en moyenne dans différentes directions de traduction. La cascade avec la variante analogique faisant usage du perceptron moyenné lors de l'agrégation (Cascade(VP, Moses)) est meilleure pour les 10 directions sur la tâche MESH et nous permet de gagner (+3.0% en moyenne) par rapport à la première cascade. Pour MEDDRA, la combinaison améliore la précision, mais l'amélioration est faible puisque Moses est beaucoup moins précis sur cette tâche.

Pour NEWS, la combinaison de Moses avec Freq n'améliore pas les performances dans les deux directions, mais la combinaison avec l'agrégateur entrainé par l'algorithme perceptron permet d'améliorer la performance par rapport à RandomForest (la version sans cascade). Nous dépassons également les performances de (Langlais, 2013) (68.9 versus 68.5) sans que l'amélioration ne soit significative.

	Config	cascade (Freq, Moses)	cascade (VP, Moses)	Moses	Freq(R@100)
MESH	*fr-en*	36,9	**38,8**	22,3	31,3$_{(68,7)}$
	en-fr	29,6	**32,8**	20,0	28,2$_{(71,8)}$
	ru-en	42,2	**45,6**	33,4	34,0$_{(66,0)}$
	en-ru	38,9	**42,5**	30,5	35,7$_{(64,3)}$
	fi-en	37,7	**40,8**	27,0	34,9$_{(65,1)}$
	en-fi	**43,3**	38,4	26,4	33,2$_{(66,8)}$
	sp-en	41,6	**43,9**	29,0	35,2$_{(64,8)}$
	en-sp	40,7	**44,9**	28,6	37,3$_{(62,7)}$
	sw-en	43,8	**47,4**	38,8	34,9$_{(65,1)}$
	en-sw	39,9	**43,7**	37,0	33,3$_{(66,7)}$
MEDDRA	*sp-en*	**53.2**	-	10,2	64,3$_{(34,4)}$
	en-sp	**46,7**	-	11,0	61,8$_{(38,2)}$
NEWS	*zh-en*	**17,5**	-	15,4	64,9$_{(32,9)}$
	en-zh	44,9	**68,9**	66,6	81,5$_{(18,5)}$

Tableau 3.XIV : Combinaison de deux variantes de système analogique (Freq et VP (ANA+IBM+MOS)) avec Moses. La meilleure configuration entre les deux cascades est mise en gras.

3.6.5 Exemples de traduction

Nous avons effectué une inspection aléatoire des sorties produites par Moses et par le système analogique qui utilise un classificateur perceptron (voted perceptron) entrainé en utilisant les traits caractéristiques ANA et IBM et excluant les caractéristiques MOS. Nous rapportons en figure 3.2 quelques exemples que nous avons trouvé représentatifs des problèmes de chaque système de traduction. L'exemple *fi-en* montre un cas où Moses ne parvient pas à produire une séquence de mots valide. L'exemple *en-sp* illustre

la faiblesse de Moses lors de la réorganisation des mots. L'exemple *zh-en* montre les translittérations incorrectes réalisées par les deux systèmes, et l'*en-zh* illustre une défaillance du moteur analogique si "ph" et "us" sont translittérés séparément.

MESH (*fi-en*)	hammasytimen sairaudet
ANALOG	**dental marrow diseases**
Moses	dental ne diseases
Reference	**dental pulp diseases**
MEDDRA (*en-sp*)	intrinsic asthma with status asthmaticus
ANALOG	**asma intrínseca con estatus asmático**
Moses	intrínseco asmático con estatus asmático
Reference	**asma intrínseca con estatus asmático**
NEWS (*en-zh*)	Adolphus
ANALOG	阿道夫厄斯
Moses	**阿道弗斯**
Reference	**阿道弗斯**
NEWS (*zh-en*)	本尼迪克特
ANALOG	Bennidickt
Moses	BenniDickert
Reference	Benedict

Figure 3.2 : Exemples des sorties de traduction de différents systèmes discutés dans cette section.

3.7 Conclusion

Dans ce chapitre, nous avons utilisé deux techniques d'apprentissage afin d'entrainer l'agrégateur embarqué par le système analogique.

Nous avons comparé les performances obtenues avec un système de référence Moses. Nous avons appliqué l'apprentissage analogique sur un ensemble des tâches impliquant différents couples de langues. Nous avons également vérifié que la cascade du système analogique avec Moses augmente la précision. Nous avons comparé un certain nombre

d'algorithmes de classification et de réordonnancement, et avons observé que globalement, le réordonnancement par `RandomForest` fournit la meilleure performance.
En dépit du problème de silence affectant nos systèmes analogiques, le système simple mettant en œuvre l'agrégateur `Freq` surpasse le système `Moses` dans plusieurs tâches (par exemple `MESH` *fr-en*, *sp-en*), ou à fortiori lorsqu'un agrégateur plus performant est embarqué (par exemple `MESH` *en-ru*, *en-fi*).

CHAPITRE 4

SOLVEUR ANALOGIQUE STRUCTURÉ

4.1 Introduction

Dans le chapitre précédent, nous avons montré qu'il était possible de distinguer à postériori les bonnes solutions produites par apprentissage analogique. Dans ce chapitre, nous nous intéressons à savoir s'il est possible de restreindre le nombre de solutions produites par le solveur analogique. Nous mettons à profit l'algorithme de perceptron structuré, un algorithme populaire de prédiction structurée afin d'apprendre à résoudre des équations analogiques. De cette façon, nous montrons qu'il est possible de contrôler le nombre et la qualité des solutions produites. Notre solveur se base sur la définition analogique de Yvon et al. (2004), bien qu'un déploiement à d'autres définitions est envisageable.
Nous comparons notre système avec des systèmes de référence sur des équations impliquant des mots et des segments (séquence de mots).

4.2 La prédiction structurée

Prédire automatiquement une structure (arbre, graphe, séquence, etc.) à partir d'une structure source est un problème sur lequel travaillent de nombreux chercheurs (Collins et Roark, 2004, Daumé III et al., 2009, Daumé III et Marcu, 2005, Huang et al., 2012).

4.2.1 De la prédiction standard vers la prédiction structurée

La prédiction est une tâche d'apprentissage d'une fonction f qui relie des entrées x dans un espace des entrées X à des sorties y dans un espace de sortie Y. Le cas de la

prédiction binaire (deux classes) a été largement étudié et a des applications multiples dont la détection de pourriel (Diale et al., 2016). Le cas de la prédiction multiclasse a également fait l'objet de nombreuses études dont (Kumar et al., 2016) pour l'analyse de sentiments. Un autre cas d'étude consiste à prédire une séquence. Les applications de cette tâche sont importantes en TALN. L'étiquetage morphosyntaxique (Bowman et al., 2016), la détection d'entités nommées (Bhargava et al., 2016) ou encore la traduction automatique (McCann et al., 2017) sont quelques exemples où la prédiction structurée a été appliquée avec succès.

Exemple L'étiquetage morphosyntaxique (*Pos tagging*) consiste à étiqueter chaque mot, comme l'illustre la figure 4.1.

Figure 4.1 : Exemple d'étiquetage de séquence (étiquetage morphosyntaxique). *DT* signifie un déterminant, *VBZ* un verbe, *JJ* un adjectif et *NN* un nom commun.

Ce problème peut être résolu en classifiant indépendamment chaque mot de la phrase (en utilisant de la classification multiclasse). Cette approche ne prend cependant pas en considération le fait empirique que les étiquettes ne sont pas produites indépendamment. En particulier, une étiquette affiche une forte dépendance conditionnelle à l'étiquette du mot précédent. Ces dépendances permettent dans certains cas de résoudre les ambiguïtés du langage naturel. Par exemple le mot "sentence" dans notre exemple peut à priori être étiqueté comme un nom ou comme un verbe, mais dans le contexte où l'étiquette précédente est un adjectif, il ne peut s'agir que d'un nom commun.

4.2.2 Formulation

La formule fondamentale de la prédiction structurée peut s'écrire comme :

$$\hat{y} \leftarrow argmax_{y \in Y} f(x,y)$$

où x est l'entrée, Y est l'ensemble des sorties possibles. La prédiction pour x est $\hat{y}$ l'élément de Y qui maximise un score défini par la fonction f. Le plus souvent, on considère une fonction qui combine linéairement les traits caractéristiques $f = w.\Phi(x,y)$ où :

- w est le vecteur de poids entrainé sur les exemples d'entrainement.
- $\Phi(x,y)$ désigne la fonction de traits caractéristiques qui prend en entrée $x \in X$ et la sortie supposée $y \in Y$. La valeur de $\Phi(x,y)$ est un vecteur de taille m, où m est le nombre de traits pour chaque exemple d'entrainement (x,y).

Cette formulation suggère que l'apprentissage structuré se ramène à un problème multi-classe où l'ensemble des classes est infini et constitué de toutes les séquences de sortie (étiquettes) que l'on peut potentiellement produire. La maximisation (*argmax*) requiert pour des raisons calculatoires des hypothèses sur la fonction f, qui typiquement revient à faire l'hypothèse de *décomposabilité*, c'est-à-dire que le score d'une séquence doit pouvoir se calculer de manière additive à l'aide du score d'une sous-séquence. Cette notion de décomposabilité est détaillée dans la section 4.6.

4.3 Choix d'algorithme

Plusieurs algorithmes ont été proposés, comme SEARN (Daumé III, 2006), CRF (Tsochantaridis et al., 2005), et l'algorithme du perceptron structuré (Collins, 2002). Nous nous concentrons dans cette thèse sur ce dernier, pour différentes raisons : il est simple à mettre en œuvre et été utilisé dans de nombreuses applications du TALN, y compris

la segmentation de mots et l'étiquetage syntaxique (Koo et Collins, 2010, Martins et al., 2013, Zhang et Clark, 2008), l'étiquetage sémantique (Zettlemoyer et Collins, 2007) ainsi que l'extraction d'information (Reichart et Barzilay, 2012). Cet algorithme a notamment montré des résultats satisfaisants dans une tâche proche de la notre, à savoir une tâche combinatoire où la taille de l'espace de recherche est factoriel (nous détaillerons cela dans la suite de ce chapitre).

4.4 Perceptron structuré

D'après Rosenblatt (1957), un perceptron est un algorithme de classification binaire simple basé sur une fonction de prédiction linéaire.

$$g(x) = \begin{cases} 1 & \text{si } w.x \geq b \\ 0 & \text{sinon.} \end{cases}$$

avec x un vecteur de nombres représentant l'entrée, y la sortie (typiquement 0 ou 1), w un vecteur de poids et b est le biais.

En se basant sur cette version du perceptron, Collins (2002) a introduit l'algorithme du perceptron structuré que nous appelons algorithme 4 dans la suite. Ce dernier est un algorithme d'apprentissage incrémental ou en ligne, c'est-à-dire qu'il apprend à partir de données reçues au fur et à mesure. À chaque itération i, l'algorithme reçoit des données $(x_i, y_i)_{i \in [1..N]}$ de N observations où x_i est la structure d'entrée (par exemple, la phrase à traduire en anglais) et y_i la sortie correspondante (par exemple, la traduction correspondante en français). Pour chaque x_i, le perceptron calcule $\hat{y}_i$, la sortie la plus probable selon le modèle en cours caractérisé par la fonction de traits caractéristiques Φ et le vecteur de poids actuel w :

$$\hat{y}_i \leftarrow argmax_{y_i \in Y} w^T \Phi(x_i, y_i) \tag{4.1}$$

Lorsque $\hat{y}_i \neq y_i$, l'algorithme met à jour le vecteur de poids en ajoutant le vecteur paramétrique de y_i et en soustrayant le vecteur paramétrique de $\hat{y}_i$. Cette mise à jour de poids permet de rapprocher le vecteur de la sortie prédite ($\hat{y}_i$) à la sortie réelle (y_i). Intuitivement, le processus d'entrainement contraint efficacement le décodeur à produire la sortie correcte.

Algorithm 4: PERCEPTRON STRUCTURÉ $(x_{1:N}, y_{1:N}, I)$

```
//I est le nombre d'itérations
// N est le nombre d'exemples
w ←< 0....0 >
for j = 1...I do
    for i = 1...N do
        // la valeur maximale pour la dérivée ŷ_i
        ŷ_i ← argmax_{y_i ∈ Y} w^T Φ(x_i, y_i)
        // comparaison de ŷ_i avec la référence y_i
        if ŷ_i ≠ y_i then
            // mise à jour de deux vecteurs w, w_a
            w ← w + Φ(x_i, y_i) − Φ(x_i, ŷ_i)
        end
    end
end
return w
```

Selon Rosenblatt (1957), le perceptron pouvait dans certains cas classifier les données d'entrainement sans pour autant généraliser en test. Donc, en calculant la moyenne, nous pouvons obtenir des modèles qui généralisent mieux.

4.5 Perceptron structuré moyenné

En se basant sur la version du perceptron moyenné (Freund et Schapire, 1999), Collins (2002) modifie son algorithme de prédiction structurée (algorithme 4) en y ajoutant un vecteur de poids moyenné w_a (ce vecteur donne des performances meilleures en test en le comparant à w[1]) comme l'indique l'algorithme 5. Cette version de l'algorithme est

[1] On parle ici du dernier w retourné à la fin de l'apprentissage.

capable de retourner une estimation finale des poids qui est la moyenne de l'ensemble des vecteurs de poids produits en cours d'apprentissage par rapport au nombre total d'observations N multiplié par le nombre d'époques I, w_a/NI.

Collins (2002), Daumé III et Marcu (2005), Miasato et al. (2017) ont montré sur plusieurs tâches du TALN que le perceptron structuré moyenné avait un bon pouvoir de généralisation.

Algorithm 5: PERCEPTRON STRUCTURÉ MOYENNÉ $(x_{1:N}, y_{1:N}, I)$

```
//I est le nombre d'itérations
// N est le nombre d'exemples
w ←< 0....0 >
w_a ←< 0....0 >
for j = 1...I do
    for i = 1...N do
        // la valeur maximale pour la dérivée ŷ_i
        ŷ_i ← argmax_{y∈Y} w^T Φ(x_i, y_i)
        // comparaison de ŷ_i avec la référence y_i
        if ŷ_i ≠ y_i then
            // mise à jour de deux vecteurs w, w_a
            w ← w + Φ(x_i, y_i) − Φ(x_i, ŷ_i)
            w_a ← w_a + w
        end
    end
end
return w_a/NI
```

4.6 Inférence

Trouver l'*argmax* dans l'équation 4.1 par une recherche exhaustive est le plus souvent impossible en pratique à cause du nombre très grand de l'ensemble de sorties Y. Aussi, nous devons limiter f de telle sorte à pouvoir faire la maximisation sur y efficacement. L'inférence du perceptron est divisée en deux types : le premier est exact, il utilise la programmation dynamique qui consiste à résoudre un problème en le décomposant en sous-problèmes, puis à résoudre les sous-problèmes, des plus petits aux plus grands en

stockant les résultats intermédiaires ; le second est inexact, il utilise la recherche en faisceau où à chaque étape, seules les hypothèses prometteuses (selon un seuil) sont conservées et étendues, ce qui diminue la complexité. L'inférence pour la prédiction structurée repose sur la propriété de décomposabilité. Le score d'une séquence entière y pour une entrée donnée x peut se décomposer en la somme des scores des étiquettes individuelles, mais en tenant compte des étiquettes précédentes comme le montre l'équation 4.2.

$$f(x,y) = \sum_{i=k}^{|y|} f(x, y_{i-k}, ..., y_{i-1}, y_i) = \sum_{i=k}^{|y|} \Phi(x, y_{i-k}, ..., y_{i-1}, y_i).w \tag{4.2}$$

4.7 Convergence

Comme tout algorithme itératif, le perceptron structuré converge au fur et à mesure des itérations. Dans certaines circonstances cependant, cet algorithme peut diverger.

Théorème : Si l'*argmax* du perceptron structuré standard n'est pas calculé à travers une inférence exacte, il peut ne pas converger (Huang et al., 2012).

D'après Huang et al. (2012), dans une recherche exacte, la garantie de convergence est vérifiée si et seulement si les données sont séparables. Mais, dans le cas d'une recherche inexacte cette garantie n'est pas assurée. Aussi, les mises à jour doivent se faire à la suite des **violations** ; une deuxième condition que nous décrivons ci-après. Cette notion est garantie implicitement lors d'une recherche exacte, mais il faut la vérifier dans la recherche en faisceau.

Définition : Soit deux prédictions y (référence) et $\hat{y}$ (prédite) d'une entrée x et un vecteur de paramètres w, une violation est présente si et seulement si le score de la solution prédite $\hat{y}$ est supérieur au score de la solution de référence ($w \cdot \Delta\phi(x,y,\hat{y}) \leq 0$) où

$\Delta\phi(x, y, \hat{y})$ désigne la différence du score :

$$\Delta\phi(x, y, \hat{y}) = \phi(x, y) - \phi(x, \hat{y}) \tag{4.3}$$

En utilisant l'heuristique de la recherche en faisceau, la figure 4.2 montre deux cas de mise à jour : la première (à gauche) illustre une mise à jour invalide de l'algorithme de perceptron car le score de la séquence correcte est supérieur au score de la séquence incorrecte, la deuxième (à droite) illustre une mise à jour valide. La plage hachée (en vert) entre les deux séquences illustre les violations (le score de la séquence correcte est inférieur au score de l'incorrecte).

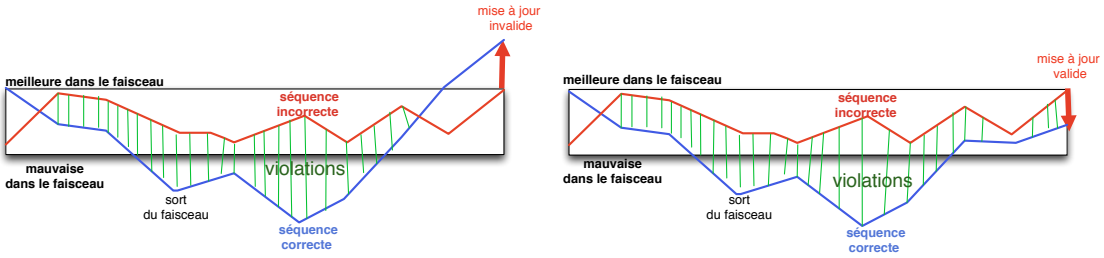

Figure 4.2 : Différentes mises à jour dans une recherche en faisceau : une valide (à droite) et une invalide (à gauche)

La figure 4.2 montre que la solution référence peut quitter le faisceau au cours du décodage. Une référence sort du faisceau lorsque son score est inférieur au score de la dernière hypothèse dans le faisceau. La figure 4.3 montre comment la référence (en bleu) peut être ratée au cours du décodage en utilisant un faisceau de taille trois, dans une simulation de traduction où les hypothèses les plus hautes sont les plus probables.

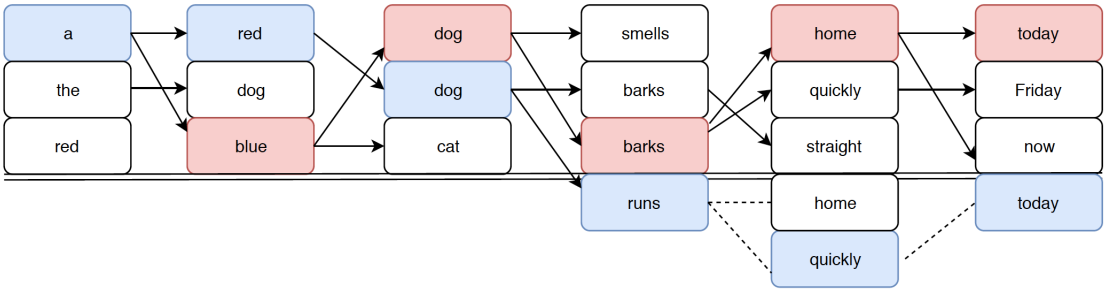

Figure 4.3 : Exemple de traduction de la phrase en français “un chien rouge court vite aujourd’hui” vers l’anglais en utilisant un faisceau de taille 3, la référence (en bleu) sort du faisceau à l’instant t=4 et si on continue le décodage en dehors du faisceau, la référence est obtenue (par les flèches en pointillé). La solution prédite et la mieux notée (en rouge) est “a blue dog barks home today".

4.8 Fixation de violation

Une solution principale nommée “fixation de violation” est proposée dans la littérature pour répondre à ce problème de divergence des modèles entraînés. Nous introduisons pour la décrire la notation suivante. Une dérivation partielle dr représente l’hypothèse pendant le décodage et cr la solution à la fin du décodage. Nous désignons dr^+, les dérivations qui mènent à la solution exacte cr^+. Nous appelons aussi dr^- les dérivations qui mènent à la solution prédite cr^-.

La première approche proposée par (Collins et Roark, 2004) est *la mise à jour anticipée* (`early-update`). Cette méthode consiste à interrompre le décodage et à réaliser une mise à jour dès que la référence sort du faisceau. La fin de la dérivation est ignorée et l’apprentissage continue avec un autre exemple. Pour plus de détails, l’algorithme 6 permet d’anticiper la mise à jour du préfixe de la solution attendue dr_m^+ dès qu’elle sort du faisceau B_i (le faisceau du taille de préfixe m). Sinon, si la solution trouvée est différente de celle attendue, une mise à jour est nécessaire. La mise à jour s’applique dans les deux derniers cas si et seulement s’il existe une violation entre soit le préfixe courant dr_m^+ de

la référence et l'hypothèse mieux notée du faisceau dr_m^- (dr^-), soit la solution cr^+ et l'hypothèse mieux notée du dernier faisceau cr^- .

Algorithm 6: Early update (Collins et Roark, 2004)

$B_0 \leftarrow [\varepsilon]$ //premier faisceau B_0 vide
for $m = 1...|x|$ **do**
  $B_m \leftarrow top_k(x, B_{m-1}, w)$ //les k meilleures hypothèses gardées dans le faisceau B_m
  if $dr_m^+ \notin B_m$ **then**
    //la séquence correcte sort du faisceau B_m
    return (x, dr_m^+, dr_m^-) //mise a jour de l'hypothèse courante
  end
end
return(x, cr^+, cr^-) //mise a jour si cr^- est différente de référence cr^+

(Daumé III et Marcu, 2005) ont proposé une autre méthode pour limiter la propagation des erreurs et pour veiller aussi à la convergence de l'algorithme. Cette méthode est LASO (Learning as Search Optimisation). LASO est similaire à la mise à jour anticipée (`early-update`) (Collins et Roark, 2004), sauf qu'après chaque mise à jour, au lieu de sauter le reste de l'exemple, LASO continue sur le même exemple avec l'hypothèse correcte (Daumé III et Marcu, 2005).

Bien que les mises à jour anticipées apprennent des modèles meilleurs que le perceptron standard pour des tâches basées sur la recherche inexacte (Collins et Roark, 2004), le temps de convergence est typiquement beaucoup plus long que l'algorithme `standard`, puisque chaque mise à jour se fait sur un préfixe (donc plus court). (Huang et al., 2012) proposent une méthode améliorée `max-violation` qui est appliquée lorsque la dérivation de la référence dr^+ sort du faisceau. Cette variante recherche la profondeur de dérivation maximisant l'écart du score entre dr^+ et celui de la meilleure hypothèse courante du faisceau dr^-. Cette méthode permet de couvrir un plus grand nombre de transitions (taille de préfixe plus grande) que la méthode `early-update`, tout en garantissant que les mises à jour sont déclenchées par des violations.

D'une manière générale, le décodage en utilisant une recherche en faisceau pour les différentes variantes du percetron structuré est réalisé avec un seul faisceau (Collins et Roark, 2004, Huang et al., 2012, Zhang et al., 2013). Mais pour des raisons de complexité de certaines tâches (comme la notre[2]), d'autres travaux (Björkelund et Kuhn, 2014, Yu et al., 2013) utilisent deux différents faisceaux : un premier B^+ avec contrainte de préconnaissance de la solution référence pour garder le maximum des hypothèses prometteuses et un autre faisceau sans aucune contrainte B^-.

En pratique la ligne 5 de l'algorithme du perceptron structuré moyenné (algorithme 7) est changée pour introduire deux lignes. La première calcule la dérivation cr^+ qui est la meilleure hypothèse dans le faisceau avec contrainte et la seconde calcule la dérivation cr^- qui est la meilleure hypothèse dans le faisceau sans contrainte.

Stratégies de mise à jour

Nous détaillons dans cette partie les différentes variantes du perceptron qu'on a implémentées et testé dans cette thèse.

Pour la version `standard`, nous appliquons l'algorithme du perceptron structuré tel qu'il est. Cette stratégie de mise à jour également connue sous le nom de *Bold update* dans (Liang et al., 2006) est appliquée dans plusieurs tâches. Elle a connu un grand succès dans de nombreux problèmes de TALN, tels que l'analyse grammaticale (McDonald et al., 2005) et l'étiquetage morphosyntaxique (Collins, 2002), mais pas pour la traduction automatique (Liang et al., 2006). Cette version est généralement appelée "échec noble" (Yu et al., 2013) car elle a des bonnes performances avec des tâches qui se basent sur la recherche exacte.

Une deuxième technique de mise à jour standard nommée `skip` est proposée. Elle fonctionne comme `standard` en évitant toutes les mises à jours invalides "non violation"

[2]Nous détaillerons cette complexité dans la section 4.10.1.

Algorithm 7: PERCEPTRON STRUCTURÉ MOYENNÉ pour une recherche inexacte $(x_{1:N}, y_{1:N}, I)$

```
//I est le nombre d'itérations
// N est le nombre d'exemples
w ←< 0....0 >
w_a ←< 0....0 >
for j = 1...I do
    for i = 1...N do
        B_0^+ ← [ε] //premier faisceau B_0^+ vide
        B_0^- ← [ε] //premier faisceau B_0^- vide
        B_|y|^+ ← top_k(x, B_|y|-1^+, w)  //les k meilleures hypothèses gardées
        dans le dernier faisceau B_|y|^+
        B_|y|^- ← top_k(x, B_|y|-1^-, w)  //les k meilleures hypothèses gardées
        dans le dernier faisceau B_|y|^-
        cr^+ = argmax B_|y|^+
        cr^- = argmax B_|y|^-
        if cr^- ≠ cr^+ then
            // comparaison de cr^- avec la référence cr^+
            w ← w + Φ(x_i, cr^+) − Φ(x_i, cr^-)
            //mise a jour si cr^- est différente de référence cr^+
            w_a ← w_a + w
        end
    end
end
return w_a/NI
```

($w \cdot \Delta\phi(x, cr^+, cr^-) > 0$) pour des raisons de convergence du modèle. Cette technique est utilisée par (Zhang et McDonald, 2012) dans une tâche d'analyse de dépendance et (Zhang et al., 2013) pour un apprentissage en ligne pour la recherche inexacte d'hypergraphe.

Une autre technique est discutée dans cette thèse nommée *dernière violation* (`latest`). Contrairement à la mise à jour anticipée, avec la version (`latest`), nous pouvons également choisir le dernier point où la mise à jour reste valide (Huang et al., 2012). `latest` est appliqué lorsque la mise à jour standard est invalide : au lieu de la sauvegarder vers une mise à jour anticipée, elle utilise la mise à jour impliquant le préfixe le plus long possible.

En résumé, nous présentons les différentes méthodes que nous allons exploiter pour concevoir les différentes variantes de notre solveur.

	si $w \cdot \Delta\phi(x, cr^+, cr^-) \leq 0$	**sinon**
`early`	MAJ	première violation
`standard`	MAJ	MAJ
`skip`	MAJ	ignorer cet exemple
`latest`	MAJ	dernière violation

Figure 4.4 : La différence entre les quatre variantes du perceptron structuré. MAJ désigne la mise à jour.

La figure 4.5 montre l'ordre chronologique des différentes stratégies de mise à jour du perceptron. La zone rectangulaire représente le faisceau qui réalise la maximisation sans contrainte, et la courbe bleu symbolise le décodage forcé (maximisation où la séquence est connue).

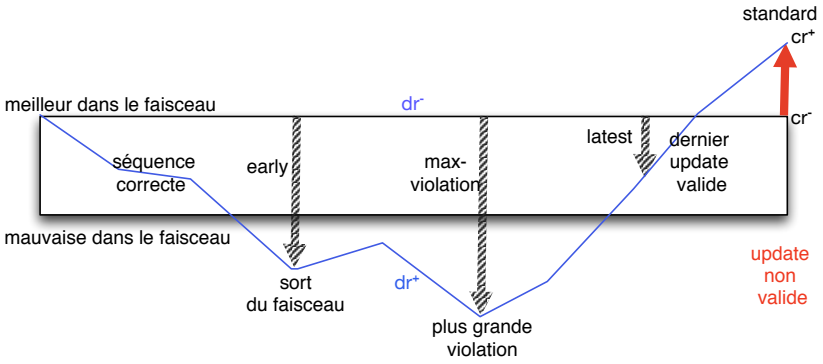

Figure 4.5 : Les différentes méthodes de mise à jour du perceptron structuré selon (Huang et al., 2012)

La figure 4.6 présente en ordre chronologique les différentes variantes de perceptron structuré utilisées en TALN. Elle identifie les principales techniques de mise à jour adoptées et l'évolution de cet algorithme au moyen des principaux articles.

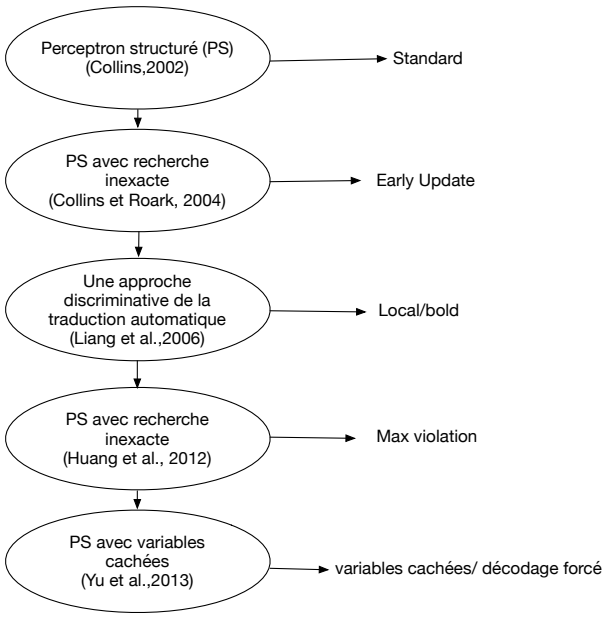

Figure 4.6 : Relation avec les travaux antérieurs.

4.9 Systèmes de référence

4.9.1 Mikolov et al. (`word2vec`)

Mikolov et Dean (2013) ont introduit `word2vec`, une boîte à outils populaire pour le calcul des plongements (*embeddings*) de mots. Cette technique permet de représenter les mots d'un corpus par des vecteurs afin de faciliter leur analyse sémantique et syntaxique. Ainsi, chaque mot est représenté par un vecteur de réels et les mots apparaissant dans des contextes similaires auront des vecteurs plus proches que d'autres apparaissant dans des contextes différents. Les auteurs ont découvert que l'espace vectoriel induit par `word2vec` a la propriété intéressante de préserver les analogies, c'est-à-dire que la différence de la représentation vectorielle de deux mots a et b dans une analogie $[a : b :: c : d]$ est un vecteur qui est près de la différence des vecteurs associés aux deux autres mots c et d. Ils proposent de résoudre une équation $[a : b :: c : ?]$ en calculant :

$$\hat{d} = argmax_{d \in V} \cos(d, c - a + b) \tag{4.4}$$

où cos est le cosinus.

Alors que ce solveur peut gérer des analogies sémantiques et formelles, il ne peut produire que des solutions qui ont été vues au moment de l'entrainement, ce qui est peu intéressant dans notre cas. Il est également très lent à l'exécution puisqu'il faut parcourir le vocabulaire complet de l'application (V). Néanmoins, la capacité d'incorporer des méthodes pour capturer des analogies a récemment pris une certaine traction, ce qui a conduit à des performances que nous pouvons reproduire et comparer. Nous avons mis en œuvre l'approche de Mikolov et Dean (2013) (équation 4.4), en utilisant des *embeddings* entrainés par les auteurs. Ils ont entrainé des embeddings de dimension 300 pour 3 millions de mots vus au moins 5 fois dans le corpus `google`, entrainé avec le modèle *skip-gram*.

La figure 4.7 (ligne 1) montre les cinq meilleures solutions produites par le solveur `word2vec` à l'équation analogique [*reader* : *unreadable* :: *doer* : ?]. Nous observons des solutions sémantiquement acceptables comme *illegible* et d'autres simplement reliées à la bonne solution ici *undoable*.

`word2vec` ($d = 300$)	*unreadable* *0.574*, *illegible* *0.496*, *scrawled* *0.496*, *scribbled* *0.496* *executor* *0.475*
`alea` ($\rho = 10$)	*undabloe* *4*, *undableo* *3*, *unabldoe* *2*, *undoeabl* *2*, *unodable* *2*
`alea` ($\rho = 50$)	**undoable** *63* *undabloe* *45*, *undaoble* *27*, *dunoable* *27*, *unadoble* *22*
`early` ($beam = 100$)	***undoable*** *510.9*, *undaoble* *488.9*, *undabole* *488.9*, *undabloe* *488.9*, *unadbloe* *488.94*

Figure 4.7 : 5 premières solutions produites par différents solveurs à l'équation [*reader* : *unreadable* :: *doer* : ?]. Voir le texte pour les détails sur les configurations. Notez que `word2vec` classe les mots dans le vocabulaire, tandis que d'autres solveurs génèrent leurs solutions.

4.9.2 Langlais et al. (`alea`)

Langlais et al. (2009) ont proposé un algorithme[3] pour résoudre une équation en prenant aléatoirement un nombre fini de shuffles[4] ρ puis en calculant l'opération complémentaire. De cette façon, l'automate n'est jamais construit, ce qui conduit à un algorithme très efficace. La figure 4.7 présente les résultats avec cet algorithme (`alea`) pour deux valeurs de ρ. Le solveur peut manquer la solution correcte si le nombre de shuffles considérés est trop faible ($\rho = 10$).

Ainsi à l'équation [$therefore\_this$: *so* :: $therefore\_if\_this$: ?][5], différentes configurations du solveur `alea` produisent des solutions où celle attendue n'est pas en tête, tel que l'illustre la figure 4.8 .

[3] Déjà décrit dans le chapitre 2.

[4] Nous appelons le nombre de shuffles pris aléatoirement, le taux d'échantillonnage.

[5] Pour des raisons d'implémentation nous désignons l'espace par un caractère spécial qui est _

ρ	**nb de solutions**	**top-5**
1	2	_ifso *1*, if_so *1*
5	5	soif_ *3* , _ifso *2* , if_so *2*, **so_if** *2*, _soif *1*
10	7	**so_if** *8*, soif_ *8*, _ifso *2*, if_so *1*, ifs_o *1*
20	12	soif_ *13*, **so_if** *12*, s_ifo *4*, _ifso *3*, if_so *3*
50	15	soif_ *37*, **so_if** *35*, s_ifo *11*, sif_o *11*, _ifso *11*
100	15	soif_ *71*, **so_if** *67*, s_ifo *17*, sif_o *12*, _ifso *11*
300	22	soif_ *223*, **so_if** *209*, _ifso *52*, s_ifo *44*, if_so *43*
500	24	soif_ *320*, **so_if** *293*, _ifso *140*, if_so *121*, s_ifo *64*

Figure 4.8 : les cinq premières solutions générées par le solveur `alea` pour différents taux d'échantillonnage (ρ). La bonne solution est en gras.

Comme la solution la plus fréquente n'est pas toujours la bonne, l'idée est ainsi d'apprendre à un solveur à prédire l'alignement le plus probable de façon à produire moins de solutions lors de la résolution.

4.10 Notre proposition : solveur analogique structuré (STRUCTANALOG)

Après avoir décrit les systèmes de référence, nous présentons maintenant notre système basé sur l'algorithme perceptron structuré.

Pour chaque tâche de prédiction structurée, il faut définir les deux structures d'entrée et de sortie. Pour l'analogie, son entrée particulière est $x = (a,b,c) \in X$, nous cherchons à produire un ensemble de solutions les plus pertinentes $y = (b \bullet c \backslash a) \in Y$. Un exemple de cette structure est donné par la figure 4.9. La pertinence d'une solution est donnée par une fonction de score f paramétrée par un vecteur de poids (un par trait caractéristique) que nous cherchons à apprendre à l'aide d'un corpus d'entrainement. Pour une analogie donnée $[\mathit{unread} : \mathit{undo} :: \mathit{unreadable} : \mathit{undoable}]$, nous pouvons la représenter comme suit :

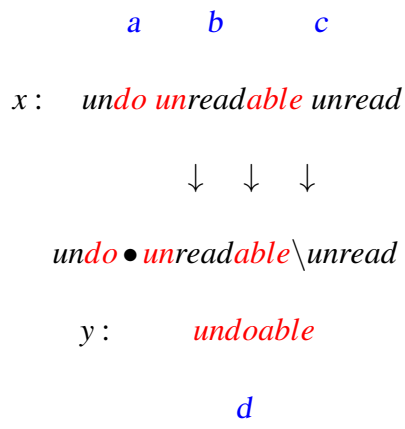

Figure 4.9 : Exemple d'une analogie sous forme d'une prédiction structurée.

En pratique, l'espace de recherche est trop important pour être visité au complet, donc nous avons mis en œuvre une heuristique pour restreindre cet espace. Cette heuristique élimine les solutions les moins prometteuses de l'espace de recherche, mais sa mise au point est délicate puisqu'elle peut potentiellement interagir avec la fonction de score. Nous décrivons ceci après l'organisation de notre espace de recherche.

4.10.1 Espace de recherche d'une équation analogique

Pour résoudre une équation $[a : b :: c : ?]$, notre solveur explore un espace de recherche dans lequel un état est représenté par un quintuplet $< \texttt{sh}, i_a, i_b, i_c, \texttt{p} >$, qui signifie que :

- i_a est la position courante de la tête de lecture dans la chaîne a, $a[i_a :]$ désigne le suffixe de a commençant en position i_a avec : désigne la fin de la chaîne (ici a).
- idem pour i_b et i_c.

  Ces trois indices de lecture doivent encore être visités.
- `p` est le préfixe actuel d'une solution.
- `sh` est la séquence du mélange considéré.

L'état initial de l'espace de recherche est $< \varepsilon, 0, 0, 0, \varepsilon >$ où ε désigne la chaîne vide. Les états finaux sont de la forme $< \varepsilon, |a|, |b|, |c|, sol >$, où *sol* est une solution à l'équation. Ils respectent les conditions suivantes :

1. taille de la solution $|sol| = |b| + |c| - |a|$.
2. $\texttt{sh} = \varepsilon$.

Il existe trois actions **X**, **Y** et **Z** qui peuvent être appliquées à un état donné et qui sont décrites dans la figure 4.10.

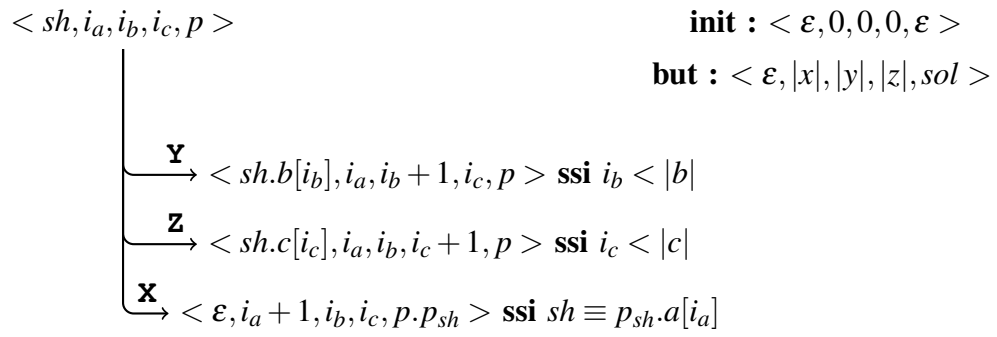

Figure 4.10 : Espace de recherche des solveurs d'apprentissage structurés.

Il convient de noter que l'action **X**, l'action qui implémente l'opération de complémentarité, est la seule qui contribue à ajouter des symboles à la solution générée. Cette action est applicable si et seulement si le dernier caractère dans la séquence `sh` est identique au caractère dans la chaîne *a* à l'indice i_a. Les deux actions **Y** et **Z** permettent, en plus d'incrémenter les deux index i_b et i_c, d'augmenter la taille de `sh` d'un symbole (caractère). La figure 4.11 donne un exemple concret d'un état intermédiaire $(uu, 0, 1, 1, \varepsilon)$ de l'équation analogique $[unread : \underline{u}ndo :: \underline{u}nreadable : ?]$ où les caractères soulignés indiquent les positions actuelles des têtes de lecture. Ce nœud est produit après deux actions de consommation (**Y** et **Z**) du nœud initial ($< \varepsilon, 0, 0, 0, \varepsilon >$) comme illustré en figure 4.12. Les trois actions applicables pour cet état $(uu, 0, 1, 1, \varepsilon)$ sont **X**, **Y** et **Z**. Trois nouveaux

nœuds sont donc générés (figure 4.11). Il est important de souligner que notre espace de recherche est différent des espaces typiques où la plupart des actions ont une incidence immédiate sur la solution, comme par exemple en traduction automatique. La seule action qui augmente la taille de préfixe est **X** où la taille de préfixe passe de 0 à 1 dans l'exemple. Les deux autres actions (**Y** et **Z**) augment seulement la taille du shuffle.

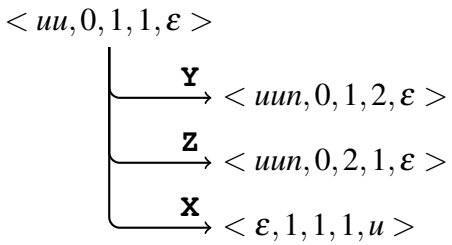

Figure 4.11 : Exemple d'un nœud intermédiaire ($< uu, 0, 1, 1, \varepsilon >$) de l'équation analogique [*unread* : *undo* :: *unreadable* : ?] avec les nœuds résultants après les trois actions **Y**, **Z** et **X**.

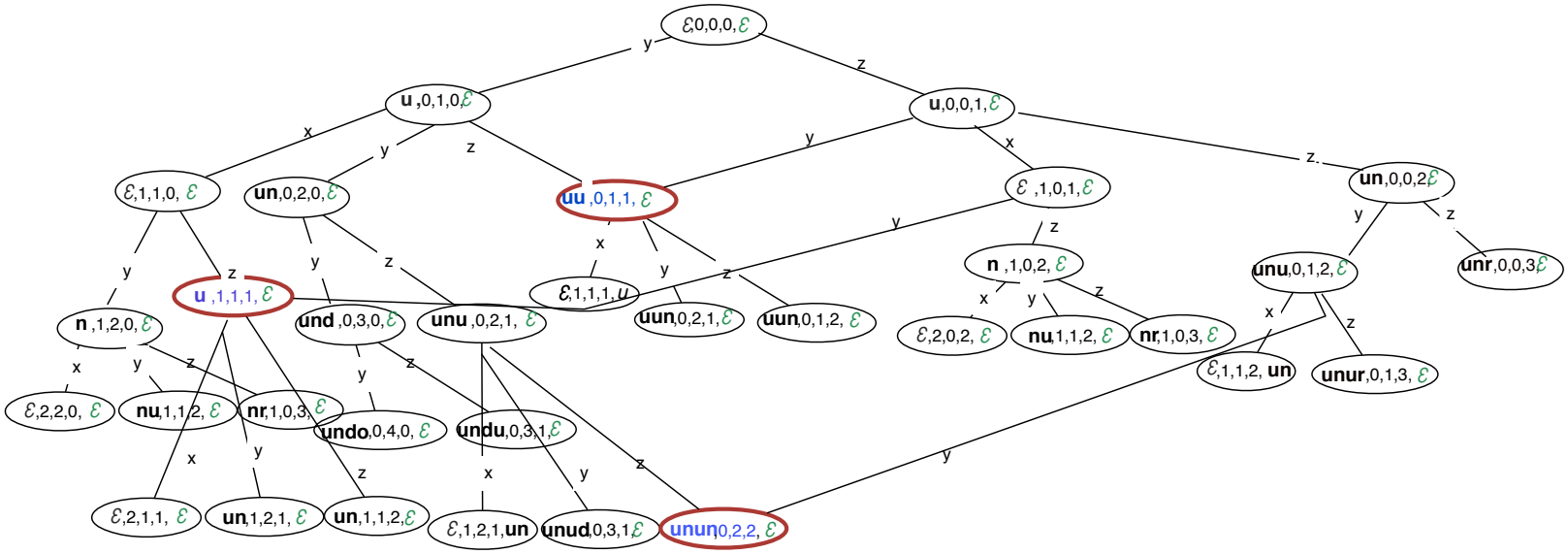

Figure 4.12 : Sous ensemble de l'espace de recherche pour l'équation $[unread : \underline{u}ndo :: \underline{u}nreadable : ?]$.

Spécification et méta-paramètres

Notre espace de recherche peut être organisé comme un arbre ternaire dont la profondeur p de l'arbre est la somme des longueurs des trois chaînes a, b et c ($p = |a| + |b| + |c|$). Le nombre de noeuds à parcourir est égal à la somme d'une série mathématique. Nous commençons du sommet (premier nœud) jusqu'à arriver à la profondeur de l'arbre (les feuilles).

$$nb_{noe} = \sum_{i=0}^{p} 3^i + 1$$

En pratique, certains états de l'espace de recherche (dont l'état initial) ne peuvent être suivis des trois actions (**`X`**, **`Y`** et **`Z`**), aussi le nombre de nœuds indiqué nb_{noe} est une borne supérieure. Il en reste pas moins que pour une analogie le nombre de nœuds crée approximativement est $nb_{noe} = (\sum_{i=0}^{p} 2^i + 1)$, ce qui reste trop important en pratique. C'est pour cela que nous appliquons les heuristiques suivantes.

Lorsque deux nœuds encodent le même état, ils sont réunis, ce qui évite d'avoir à exploiter les deux. Cela revient à organiser l'espace sous forme de graphe. La figure 4.12 montre quelques nœuds (en rouge) qui ont été fusionnés. Cela réduit en toute sécurité l'espace de recherche, sans sacrifier l'optimisation. La réduction opérée par cette stratégie varie selon les trois chaînes qui constituent une équation analogique.

Par exemple pour l'équation [*international_investigation* : *international_democracy* :: *investigation_in* : *democracy_in*], le nombre de nœuds parcouru est égal à 3 727 332. En utilisant la stratégie du merge, le nombre de nœuds devient 1 676 602, une réduction de 55%. Cela n'est cependant pas suffisant en pratique.

Une autre vérification que nous pouvons faire est la possibilité d'une consommation **`X`** dans les actions futures. Nous vérifions si le caractère actuel $a_{[i_a]}$ existe dans le suffixe

restant de deux chaines b et c ($b_{[i_b:]}.c_{[i_c:]}$).

$$a_{[i_a]} \in b_{[i_b:]}.c_{[i_c:]} \tag{4.5}$$

En vérifiant, la possibilité de **x**, le nombre de nœuds crées pour résoudre l'analogie précédente devient 1,434170. Une diminution de 15%.

Comme autre stratégie, nous déployons une heuristique de recherche en faisceau pour éliminer les hypothèses moins prometteuses. En raison de la spécificité de l'espace de recherche, la comparaison des hypothèses qui mènent au même préfixe `p` est difficile, et nous avons dû recourir à une politique de faisceau sophistiquée (contrôlée par des métaparamètres).

Nous choisissons un algorithme de recherche en faisceau par pile à l'instar par exemple de `Moses` (Koehn et al., 2007). Nous gardons à chaque niveau de profondeur une pile guidée par la taille du préfixe. Nous gardons dans chaque pile les n meilleures hypothèses, et nous étendons l'hypothèse du meilleur score parmi toutes les piles, comme l'indique l'algorithme 8. La spécificité de notre problème réside dans le grand nombre des nœuds sans préfixe générés au début de la recherche et dans la difficulté de les comparer. Par exemple dans la figure 4.12, la majorité des nœuds générés au début dans le graphe sont sans aucun préfixe généré (ε en couleur verte). Donc les hypothèses gardées surtout dans ce niveau peut-être aléatoire, d'où l'idée d'organiser dans chaque pile des hypothèses comparables (c'est à dire par la taille de préfixe de la solution et aussi la même taille de shuffle |b+c|). Nous utilisons à cet effet une matrice rangée par la taille de préfixe $|b|+|c|-|a|$ et par la taille du shuffle $|b|+|c|$.

Algorithm 8: Recherche en faisceau pour analogie

p_{ij} // pile d'indices i (la taille du préfixe) et j (la taille du shuffle)

$p_{00} \rightarrow (\varepsilon, 0, 0, 0, \varepsilon)$

ajouter les deux nœuds résultants de **Y** et **Z** à p_{01}.

while $\exists$ *hypothèses dans les piles* **do**

rechercher h dans toutes les piles //hypothèse mieux notée

Y de $h \rightarrow$ h'

Z de $h \rightarrow$h"

X de $h \rightarrow$ h'"

ajouter chaque hypothèse h', h" et h"' dans la pile appropriée.

end

Retourner l'ensemble des solutions $p_{|b+c-a||b+c|}$ s'en existe

La figure 4.13 montre la distribution du nombre d'hypothèses pour résoudre l'équation [*international_investigation* : *international_democracy* : :

investigation_in : *?*].

Les cases sont colorées en fonction du nombre des nœuds dans chacune des piles. Les cases en blanc sont vides tout au long du processus de recherche. La grande concentration est marquée par les trois couleurs jaune, orange et rouge. Cette figure illustre bien la spécificité de notre espace de recherche : un excès de nœuds avec des préfixes vides (la première ligne rouge). La partie en rouge au milieu indique dès que la taille du shuffle est à 30%, il y a une explosion du nombre de nœuds générés. Comme la taille de préfixe augmente quand la taille du shuffle augmente (le préfixe est généré à partir du shuffle), nous remarquons une concentration sur la diagonale.

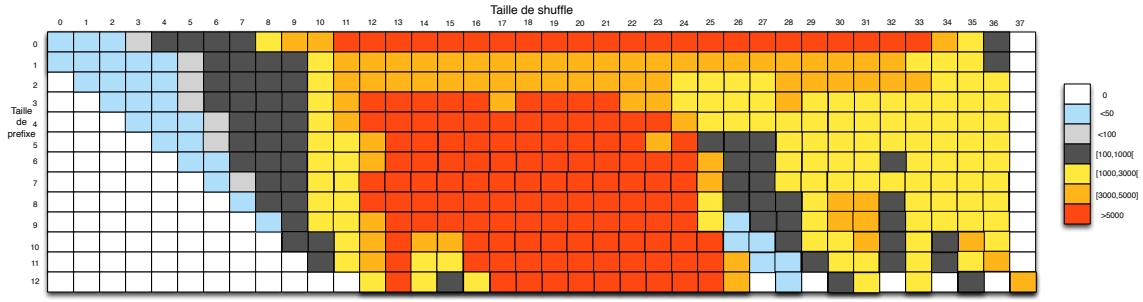

Figure 4.13 : Pourcentage de remplissage des noeuds d'une matrice lors de la résolution de l'équation [*international_investigation* : *international_democracy* :: *investigation_in* : *?*]

Une autre stratégie consiste à guider la recherche en forçant le processus de recherche à un certain moment d'exécuter l'action de complémentarité **X** pour augmenter la taille du préfixe courant ou éliminer certains nœuds après un certain nombre de consommations. Un méta-paramètre contrôle le nombre d'hypothèses qui peuvent se développer sans générer un symbole de la solution (sans consommer **X**). En utilisant ce méta-paramètre avec la valeur 7 (η=7), nous constatons une chute du nombre des hypothèses gardées pour les premiers niveaux (où le préfixe est encore vide ou de petite taille). Mais à partir de la ligne 5 de la figure 4.14 (préfixe =4), nous nous retrouvons avec les mêmes configurations qu'avant, ce qui confirme la complexité de la recherche. Nous expérimentons les valeurs de ce méta-paramètre η dans la section 4.13.2.

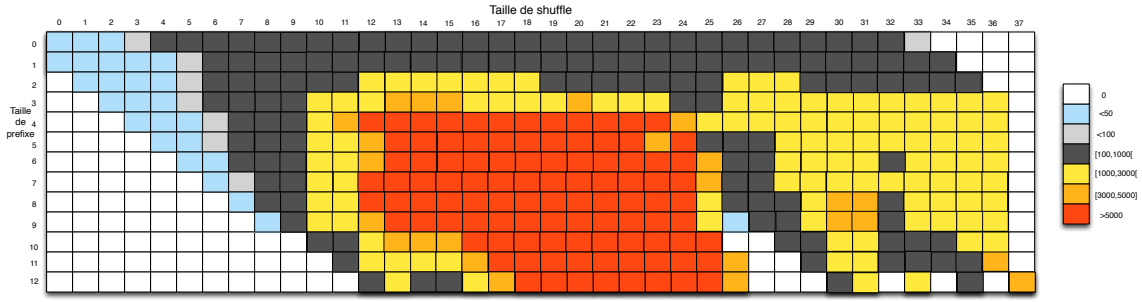

Figure 4.14 : Pourcentage de remplissage des noeuds d'une matrice lors de la résolution de l'équation [*international_investigation* : *international_democracy* :: *investigation_in* : *?*] avec η=7.

4.10.2 Les différentes variantes du STRUCTANALOG

Nous allons utiliser les descriptions des quatre différentes du perceptron structuré pour construire quatre versions. Notre solveur structuré STRUCTANALOG dispose donc de quatre variantes qui sont décrites ci-après.

Comme notre espace de recherche ne peut pas générer des préfixes de la solution dès les premiers pas de recherche (premiers niveaux du graphe), nous avons choisi d'utiliser deux décodages différents en parallèle. Le premier décodage garde seulement les dérivations correctes (dr^+) de cr^+ (la solution), menant à la sortie de référence. Le deuxième calcule l' *argmax* cr^- (la meilleure dans le faisceau) sans aucune contrainte.

4.10.2.1 Standard

Pour cette première variante, nous appliquons l'algorithme qui est décrit en section 4.5 et qui est résumé en figure 4.15. Nous calculons les deux *argmax* cr^+ et cr^- et l'une des trois actions est attendue :

1. Si la sortie cr^- est différente de cr^+, le vecteur de poids w est mis à jour en ajoutant le vecteur de traits caractéristiques de l'exemple d'entrainement et en

soustrayant le vecteur de traits caractéristiques de la sortie du décodeur comme l'indique l'équation suivante :

$$\mathbf{w} \leftarrow \mathbf{w} + \Phi(x, cr^{+}) - \Phi(x, cr^{-})$$

2. Si la sortie cr^{-} est correcte (identique à cr^{+}), aucune mise à jour n'est effectuée.

3. Si l'un des deux *argmax* (cr^{-} ou cr^{+}) échoue, nous éliminons cet exemple et nous passons au suivant.

Nous appelons cette version `standard` plus tard dans la section résultats. Comme nous utilisons un faisceau lors de la recherche, le nœud qui mène à la solution peut être éliminé à tort du faisceau. Donc, dans certains cas, `standard` n'arrive pas à effectuer l'un ou l'autre des *argmax*.

4.10.2.2 Skip

Comme déjà vu, pour garantir la convergence de notre solveur STRUCTANALOG et sa performance, il nous faut garantir que les mises à jour sont faites si et seulement s'il existe des violations. Ceci donne lieu à la version `skip` décrite en figure 4.15. Cette version a été proposée par Zhang et McDonald (2012).

4.10.2.3 Early

La fixation de violation est une autre stratégie utilisée pour garantir la convergence et surtout la non-propagation des erreurs. Nous appliquons une méthode de fixation de violation qui est la mise à jour anticipée pour notre solveur STRUCTANALOG. Nous appelons cette version `early` tout au long de nos nos expériences. Collins (2002) a montré la nécessité de prévenir la mise à jour dès que la référence sort du faisceau.

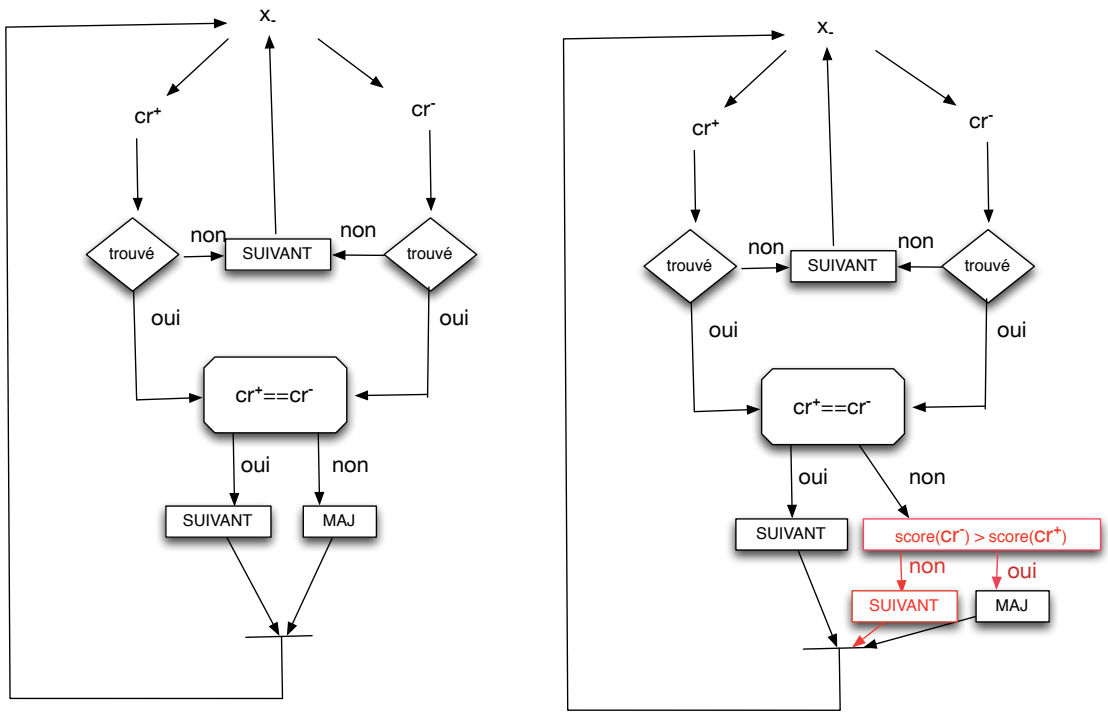

Figure 4.15 : La première figure (à gauche) décrit le processus de la version `standard` et la deuxième (à droite) décrit le processus de la version `skip` .

Comme l'indique son nom, la mise à jour anticipée cherche à anticiper la propagation des erreurs en mettant à jour le préfixe de la solution cr^+ dès que la dérivation correcte sort du faisceau lors de décodage. Si toutes les dérivations de cr^+ sortent du faisceau, le décodage est arrêté, ensuite nous comparons la dernière dérivation dr^+ dans le faisceau avec la dérivation de même taille de préfixe dr^- du cr^+. S'il y a une violation, une mise à jour est appliquée, sinon l'exemple est rejeté et nous passons au suivant. Si une dérivation correcte reste dans le faisceau tout au long du décodage, le même processus que la version `skip` est appliqué. La figure 4.16 détaille ce processus. La différence par rapport à la variante `skip` est marquée en bleu.

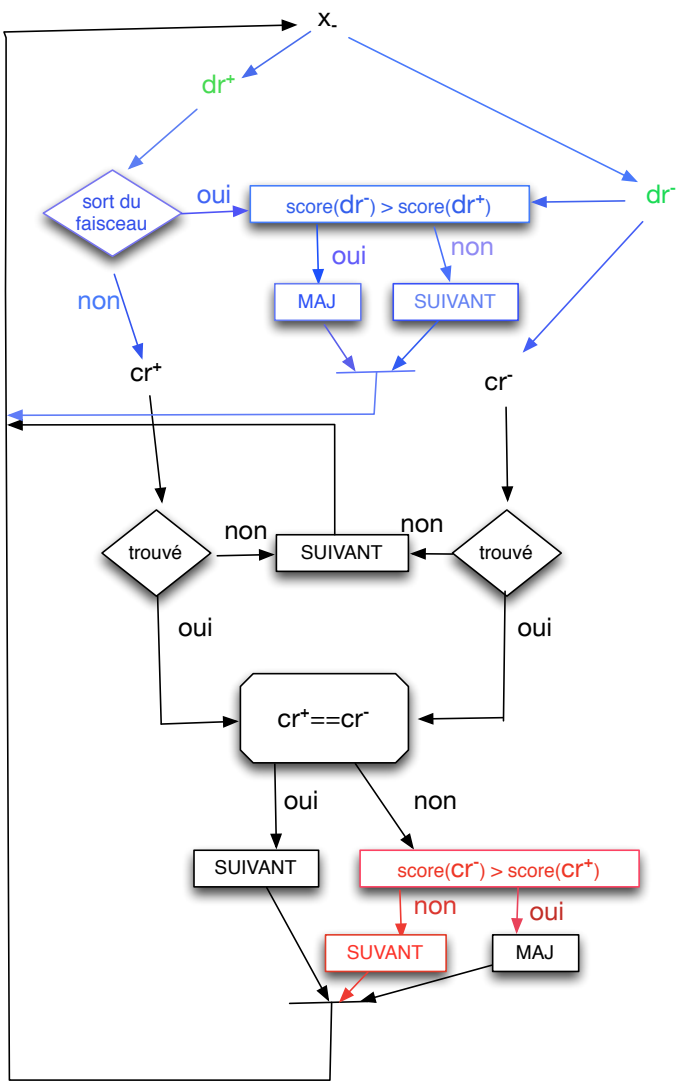

Figure 4.16 : Le processus de la version `early` .

Dans nos expériences, le problème majeur de cette méthode est sa convergence lente. Pour notre tâche, la mise à jour dans `early` se fait en moyenne avec un préfixe de taille de 40% de la taille de la solution finale.

4.10.2.4 La dernière violation (Latest)

Comme l'indique son nom, cette version consiste à prendre la dernière violation possible dans le faisceau si et seulement si le score de la sortie du décodeur cr^- est supérieure à la sortie de référence cr^+.

Cette version `latest` converge plus rapidement que la précédente méthode `early`, car la mise à jour se fait avec une moyenne de longueur de 70% par rapport à la taille de la solution (du moins dans nos expériences). En pratique, cette variante est l'inverse de la version `early`. Nous continuons le décodage même quand le cr^+ sort du faisceau(c-à-d quand le score de cr^+ est inférieur au dernière hypothèse dans le faisceau sans contrainte). Nous mettons à jour la dernière violation possible avec le préfixe le plus long.

4.10.3 Les traits caractéristiques

Les quatre variantes décrites partagent toutes les mêmes traits qui sont des traits binaires. Nous encodons tous les traits sous forme d'intervalles. Les fonctions binaires vérifient la probabilité d'appartenance à une plage spécifique. Nous présentons les principales familles de fonctions que nous avons programmées.

1. **Modèle de langue (14 traits) :** puisque notre solveur produit un sous-ensemble de permutations de la même forme comme une solution, le classement des solutions avec un modèle de langue favorise les hypothèses avec un préfixe `p` qui est probable dans une langue donnée. Nous calculons la probabilité d'un préfixe `p` selon un modèle de langage n-gram entrainé sur un ensemble large de données externes. Nous avons également des traits caractéristiques de la forme

$$prob(p_i|p_{i-1}\dots p_{i-n+1}) \leq \delta \tag{4.6}$$

qui s'arment quand un caractère d'un préfixe donné `p` est prédit avec une probabilité inférieure à δ. Par exemple pour les deux chaînes *dd* ou *rrr*, la probabilité donnée par le modèle est trop faible, car ces séquences sont non vues dans l'entraînement.

Nous avons également déployé des traits similaires pour les shuffles `sh` de taille supérieure à trois, car nous excluons toujours le dernier caractère du shuffle qui peut potentiellement être consommé avec la fonction de complémentarité de notre solveur.

2. **Distance d'édition (20 traits) :** la distance d'édition entre deux chaînes calcule le nombre minimal d'opérations d'édition (insertion, suppression et substitution) nécessaires pour transformer l'une des chaînes en l'autre. La solution d'une équation analogique [a : b :: c :?] partage des séquences avec les deux chaînes b et c. Par exemple, dans [*reader* : *unreadable* :: *doer* :?], la solution *undoable* partage avec *doer* le préfixe *do* et aussi la solution partage avec *unreadable* le préfixe *un* et le suffixe *able*. L'idée derrière l'utilisation de ces traits est de capturer les séquences communes entre les deux chaînes b et c et les solutions possibles avec l'intuition que les solutions qui ressemblent le plus à l'une de ces chaînes sont plus susceptibles d'être les bonnes. Les distances d'édition sont ensuite transformées en fonctions binaires. Intuitivement, cette famille des traits préfère les solutions de distance faible.

3. **Guide de recherche (20k traits) :** notre espace de recherche est spécifique, car la génération des préfixes des solutions n'est pas immédiate habituellement. L'idée est d'ajouter des traits qui capturent l'espace de recherche visité. Donc, cette famille des traits est implémentée pour guider la recherche de l'état initial vers un état terminal en bénéficiant de l'historique de recherche pendant l'entraînement.

- Nous mesurons le pourcentage de consommation dans chaque forme a, b et c.
- La deuxième famille de traits capte la dernière opération effectuée, fournissant ainsi une information markovienne de premier ordre : soit **`X`**, **`Y`** ou **`Z`**.
- Nous calculons également le nombre total d'opérations consécutives (**`Y`** et **`Z`**) prises depuis la dernière consommation en **`X`**. Cette famille pourrait aider le mécanisme d'apprentissage à favoriser une opération de complémentarité si trop d'opérations de produit de mélange (**`Y`** et **`Z`**) ont eu lieu.
- Nous disposons également de fonctionnalités qui enregistrent la valeur de chaque indice. Nous avons une fonction binaire pour chaque valeur possible. En outre, nous calculons également un certain nombre de fonctions binaires pour détecter si des configurations spéculatives de symboles ont été observées dans l'espace de recherche visité lors de l'application de la production de la solution de référence à une équation analogique. Pour chaque équation d'entrainement, nous exécutons le solveur sous la contrainte cr^+ pour produire sa solution de référence et pour enregistrer toutes les configurations de traits caractéristiques que nous rencontrons (comme référence). Ces traits sont conçus pour capturer la présence des caractères dans l'entraînement. Cette fonction s'allume si et seulement si les caractères indexés par les têtes de lecture ont déjà été vus auparavant :
  - un trait caractéristique binaire pour chaque triplet possible$(a[i_a], b[i_b], c[i_c])$,
  - une caractéristique binaire pour chaque bigramme à la fin du shuffle `sh`,
  - une caractéristique binaire pour chaque bigramme à la fin du préfixe `p`.

4.11 Données

Nous appliquons notre solveur STRUCTANALOG à deux tâches qui diffèrent quant à la longueur des formes impliquées dans les équations : la première concerne deux corpus de mots (`google` et `msr`), la deuxième est sur deux corpus de segments `simple` et `hard`.

4.11.1 Analogies sur les mots (Google et Microsoft)

Les données sont des analogies sur les mots. Les deux jeux de données sont `google` et `msr`. La taille moyenne des mots dans les deux corpus est de six caractères.

1. `google` : Mikolov et Dean (2013) ont conçu une tâche complète pour évaluer la propension des *embeddings* de mots à préserver les relations analogiques. Ce corpus contient 19544 analogies distribuées sur 14 catégories comme l'indique le tableau 4.I. Environ 55% de ces analogies sont réellement syntaxiques et capturent différents phénomènes morphologiques en anglais. Par exemple, la catégorie *gram4-superlative* regroupe des analogies impliquant des alternances superlatives (par exemple, [*warm* : *warmest* :: *strange* : *strangest*]). Beaucoup de ces analogies syntaxiques, en fait, ne sont pas formelles (par exemple, [*rare* : *rarely* :: *happy* : *happily*]). Par conséquent, nous avons supprimé les analogies non formelles pour construire un corpus de 4977 analogies nommées `google-form` par la suite. Ce corpus représente un quart du corpus initial. Les deux corpus formels et non formels partagent les 9 types d'analogies (*gram1-adjective-to adverb*, ..., *gram7-past-tense*) comme l'indique le tableau 4.I.

2. `msr` : Nous avons également utilisé l'ensemble de données `msr` de 8k analogies syntaxiques[6], dont 3664 sont formelles et nommées `msr-form` par la suite (envi-

[6] `http://www.marekrei.com/blog/linguistic-regularities-word-`

ron 45,8% du corpus `msr`). Ces analogies sont distribuées en 8 catégories de 1000 analogies chacune comme l'indique le tableau 4.I (total 8000).

Type	**Taille**	**% dans le corpus**	**Sous-type**	**Exemples**
`google`				
sémantique	8869	45	capitales-pays	[*Tokyo* : *Japan* :: *Madrid* : *Spain*]
			capitale-monde	[*Dublin* : *Ireland* :: *Jakarta* : *Indonesia*]
			devise	[*USA* : *dollar* :: *India* : *rupee*]
			ville-état	[*Modesto* : *California* :: *Miami* : *Florida*]
			famille	[*brother* : *sister* :: *husband* : *wife*]
`google-syntax`				
`google-form`	4977	25,5	gram1-adjectif-à-adverbe (ADJ-ADV)	[*amazing* : *amazingly* :: *serious* : *seriously*]
			gram3-comparative (COMP)	[*fast* : *faster* :: *bright* : *brighter*]
			gram4-superlative (SUP)	[*warm* : *warmest* :: *strange* : *strangest*]
			gram8-pluriel (PLUR)	[*eye* : *eyes* :: *donkey* : *donkeys*]
			gram9-pluriel-verbs (PL-VB)	[*listen* : *listens* :: *eat* : *eats*]
syntaxique	5698	29	gram2-contraire (OPP)	[*comfortable* : *uncomfortable* :: *rational* : *irrational*]
non			gram5-présent-participe (PP)	[*sit* : *sitting* :: *shuffle* : *shuffling*]
formelle			gram6-nationalité-adjectif (NAT)	[*Ireland* : *Irish* :: *Brazil* : *Brazilian*]
			gram7-passé (PAST)	[*feeding* : *fed* :: *looking* : *looked*]
Total	19544			
`msr`				
`msr-form`	3664	45,8	adj.base/comparatif (JJ-JJR)	[*high* : *higher* :: *wild* : *wilder*]
			adj.compar/superlatif (JJR-JJ)	[*greater* : *greatest* :: *earlier* : *earliest*]
			adj.base/superlatif (JJS-JJ)	[*low* : *lowest* :: *short* : *shortest*]
			nom singulier/pluriel (NN-NNPOS)	[*problem* : *problems* :: *program* : *programs*]
			verbes base / troisième personne (VB-VBD)	[*take* : *takes* :: *run* : *runs*]
syntaxique	4336	54,2	verbes base/passé (VBD-VB)	[*leave* : *left* :: *require* : *required*]
non			verbes passé/troisième personne (VBZ-VB)	[*did* : *does* :: *sought* : *seeks*]
formelle			nom singulier / nom pos (NNPOS-NN)	[city : city's : : film : film's]
Total	8000			

Tableau 4.I : Distribution des types d'analogies Google et Microsoft

Le degré des analogies qui forment les deux corpus `google-form` et `msr-form` est

`representations/`

typiquement inférieur à trois. Donc, ces analogies sont à priori faciles à résoudre. Les caractéristiques principales de deux corpus que nous avons extraits de ces ressources sont décrites dans le tableau 4.II (première partie).

Corpus	nb	long
	mots	
msr-form	3664	6
		[*high* : *higher* :: *wild* : *wilder*]
google-form	4977	7
		[$fast$: $faster$ :: *bright* : *brighter*]
	segments	
segments	2000	16
simples		[*international investigation* : *international democracy* :: *an international investigation* : *an international democracy*]
segments	2000	17
difficiles		[*their governments to* : *their governments are* :: *their governments and to* : *and their governments are*]

Tableau 4.II : Nombre d'analogies dans chaque corpus (nb), longueur moyenne des mots dans chaque corpus (long) ainsi qu'un exemple de chaque corpus (nous détaillons les deux corpus de segments dans la section suivante).

4.11.2 Analogies sur les séquences de mots

4.11.2.1 Collecte de données

Identifier des analogies formelles sur les segments n'est en réalité pas le genre de tâche qu'un humain serait disposé à faire de manière exhaustive et systématique. On pourrait facilement produire des analogies telles que [*She loves Paul* : *He loves Paul* : : *She likes Mark* : *He like Mark*], mais cela deviendrait rapidement une tâche ardue de collecter des analogies représentatives, c'est-à-dire des analogies qui capturent un ensemble riche de phénomènes linguistiques (comme le fait qu'un verbe à la 3^{eme} personne au singulier au présent prend un **s** en anglais). Par conséquent, nous avons eu recours à une procédure automatique pour acquérir des analogies formelles entre des segments. Pour ce faire, nous avons d'abord formé un système de traduction automatique basé sur des segments à partir du bitexte EUROPARL[7] anglais-espagnol, en utilisant la boîte à outils Moses (Koehn et al., 2007). Nous avons ainsi collecté des millions de couples de segments tels que ceux de la figure 4.17. Nous retenons ceux ayant un bon score d'association ($prob \geq 0.1$). Un sous-ensemble de couples de segments a été élu comme référence $\mathscr{R}$, les couples restants étant conservés en tant que mémoire de traduction $\mathscr{M}$. Ensuite, nous avons appliqué un processus de traduction par apprentissage analogique pour traduire les segments espagnols. Pour une forme donnée à traduire, u, ce système identifie (a,a'),(b,b'),(c,c')$\in \mathscr{M}$ tel que [$a : b :: c : u$], et résout les équations [a' : b' : : c' : ?]. Chaque fois qu'une solution à une telle équation produit u' , nous considérons [a' : b' : : c' : u'] comme une analogie utile. Nous aurions pu identifier directement les analogies dans la partie anglais, mais nous aurions fini avec de nombreuses analogies parasites, c'est-à-dire de vraies analogies formelles qui sont tout simplement fortuites comme [*croyons* : *creons* :: *montroyal* : *montreal*]. Ce même problème de vérification des vraies analogies a été abordé par (Lepage, 2004). L'hypothèse ici est que, si l'on

[7] `http://statmt.org`

peut identifier une analogie de source parasite, il est très peu probable que sa projection dans la langue cible (anglais) aboutisse à une équation pour laquelle la traduction de référence est une solution.

```
a actualizar los acuerdos ||| to update the agreements ||| 1 0.004748
a cambiar la base ||| to change the basis ||| 0.5 0.003554
basado en el trabajo de ||| based on the efforts of ||| 1 2.02579e-05
```

Figure 4.17 : Couples de segments collectés par un système de traduction statistique sur le corpus EUROPARL espagnol-anglais. Le format montre le segment source (espagnol), le segment cible (anglais) et les deux premiers scores estimant leur probabilité d'être en relation de traduction.

4.11.2.2 Filtrage des segments

Avec ces analogies, nous avons formé deux types de corpus analogique :

1. un corpus nommé `simple` pris aléatoirement parmi ces analogies. Ce corpus est de taille 10k analogies, nous prenons 1k analogies pour l'entraînement et 9k analogies pour le test divisé sur 9 corpus.

2. le deuxième corpus `hard` est constitué de la manière suivante : nous excluons les 10k analogies du premier corpus, ensuite nous appliquons notre solveur aléatoire `alea` sur ce corpus avec un taux d'échantillonnage de 10. Nous avons ensuite divisé les équations selon le rang de la traduction de référence parmi la liste de solutions proposées. Nous avons collecté plus précisément 400 analogies classées dans chacun des intervalles suivants : [1-5], [6-10], [11-20], [21-50] et [51-100], conduisant à un total de 2000 analogies (1000 pour le corpus d'entrainement et 1000 pour le test), dont certaines sont rapportées à la figure 4.18 (deuxième partie).

La taille moyenne des segments retenus dans les deux corpus est de 17 caractères. Le corpus `simple` contient 80% d'analogies de degré deux ou trois. Le deuxième corpus `hard` est un peu plus difficile puisque nous gardons seulement les analogies plus

difficiles à résoudre et qui présentent généralement un degré supérieur ou égal à trois. Seulement 20% des analogies dans ce corpus sont faciles (degré ≤ 3).

simple

▷ [*international investigation* : *international democracy*
::
an international investigation : *an international democracy*]
▷ [*young girls* : *training of young girls*
::
young girls and : *training of young girls and*]
▷ [*political situation is viable* : *political situation is still*
::
the political situation is viable : *the political situation is still*]

hard

▷ [*adopted recently by* : *recently adopted by*
::
study published recently by : *study recently published by*]
▷ [*competition and the* : *competition and against*
::
competition and of the : *of competition and against*]
▷ [*their governments to* : *their governments are*
::
their governments and to : *and their governments are*]

Figure 4.18 : Exemples d'analogies sur les séquences de mots identifiées automatiquement.

4.12 Mesure d'exactitude

La mesure que nous avons utilisée est la justesse ou exactitude qui calcule le pourcentage des solutions correctes par rapport au nombre total des exemples visités.

$$exactitude = \frac{nombre\_de\_solutions\_correctes}{nombre\_total\_d\_exemples} \tag{4.7}$$

4.13 Résultats

Dans cette section, nous présentons les résultats par rapport `alea` comme point de comparaison à l'état de l'art. En effet, pour cette tche, le solveur de Mikolov et Dean (2013) est inopérationnel, car il n'est pas possible d'obtenir un embedding pour toute séquence de mots (ou du moins pas avec des résultats probantes).

4.13.1 Équations de mots

Nous commençons avec des faisceaux de petites tailles pour nos quatre modèles structurés (`standard`, `skip`, `latest` et `early`). Pour que les résultats de notre solveur STRUCTANALOG et le solveur de référence `alea` soient comparables, nous procédons comme suit : nous commençons par prendre des taux d'échantillonnage ρ de 1 jusqu'à 20 pour le nombre de mélanges. De la même façon, nous élargissons le faisceau de 1 jusqu'au 20 pour nos quatre modèles structurés. Dès que la taille du faisceau atteint 20, tous les modèles obtiennent un rappel de 100% comme l'indique la figure 4.19. Les différentes variantes du STRUCTANALOG dépassent toutes les configurations du solveur `alea` ou presque. Les quatre variantes ont presque la même performance avec un léger avantage à la version `early`, cela confirme bien les résultats et l'intuition donnée par Collins et Roark (2004).

Comme les analogies utilisées sont de longueur moyenne de six caractères (de petite taille), les trois autres versions `standard`, `skip` et `late` sont moins bonnes. Mais ça n'empêche pas ces derniers modèles de surpasser la performance du système `alea`.

Nous voulons nous comparer avec d'autres travaux, mais le problème est que la majorité applique leur modèle sur toutes les analogies syntaxiques `msr` et `google-syntax`. Nous avons donc réimplémenté la méthode `word2vec` (*skip-gram*) de Mikolov et Dean (2013).

Pour reproduire exactement les mêmes performances données par l'auteur, nous avons

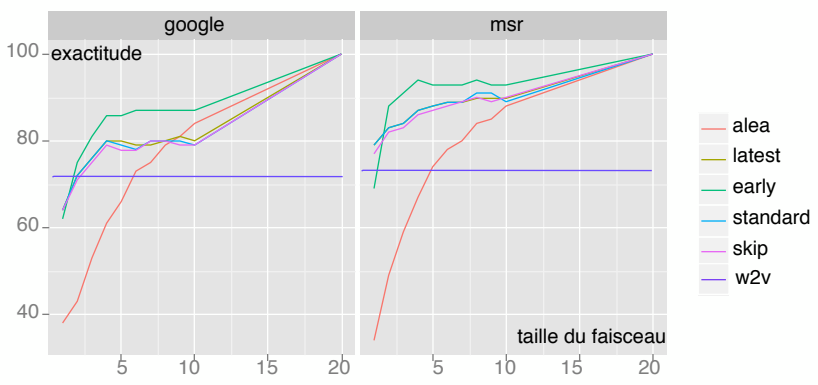

Figure 4.19 : Performance de nos configurations analogiques structurées face aux modèles de référence `alea` et `word2vec`.

réutilisé les modèles `word2vec` déjà entraînés par Mikolov disponibles sur le web[8]. Nous calculons toutes les sorties des équations analogiques en calculant la distance cosinus en appliquant sa définition d'analogie déjà décrite en section 4.9.1. Nous avons réussi à reproduire les solutions données par Mikolov et Dean (2013) pour trouver la solution d'une équation analogique avec les mêmes performances que celles décrites dans leur article. La performance de `word2vec` sur `msr-form` est de 78% et de 71% sur `google-form`. Nous observons que tous les solveurs analogique (incluant `alea`) atteignent 100% d'exactitude, soit un gain de plus de 20% par rapport à `word2vec`. Ces résultats sont dans le tableau 4.III.

Configuration	`word2vec`	`early` / `alea`
`msr-form`	78%	100%
`google-form`	71%	100%

Tableau 4.III : Comparaison de la méthode `word2vec` *skip-gram* de Mikolov et Dean (2013) avec notre modèle structuré `early` et le solveur aléatoire `alea` pour les corpus de `msr-form` et `google-form`.

Malgré le fait que toutes les analogies formelles sont de degré deux et donc à priori

[8] `https://code.google.com/archive/p/word2vec`

simple à résoudre, toutes les configurations du solveur analogique sont largement supérieures à `word2vec`. Cette basse performance de `word2vec` est due à la difficulté de résolution les équations analogiques de quelques catégories. La figure 4.20 détaille ces performances et compare l'exactitude des solveurs `early` et `word2vec` sur chaque catégorie d'équations pour les deux ensembles de données `google-form` et `msr-form`. Le solveur `early` surpasse systématiquement `word2vec`, en particulier pour quelques catégories telles que NN-NNPOS dans le le jeu de données `msr-form` ou ADJ-ADV dans le corpus `google-form`. Il y a principalement deux raisons à cela. Premièrement, la plupart des noms en anglais peuvent aussi être des verbes (ainsi des couples comme *walk : walks*), cela vient du fait qu'il y a un problème d'homonymie, mais chaque mot de vocabulaire reçoit un seul embedding, ce qui conduit à quelques erreurs. Deuxièmement, `word2vec` sort très souvent des termes sémantiquement liés à la solution attendue. Par exemple, il produit la forme *fantastic* à l'équation [*cold* : *colder* :: *great* : *?*].

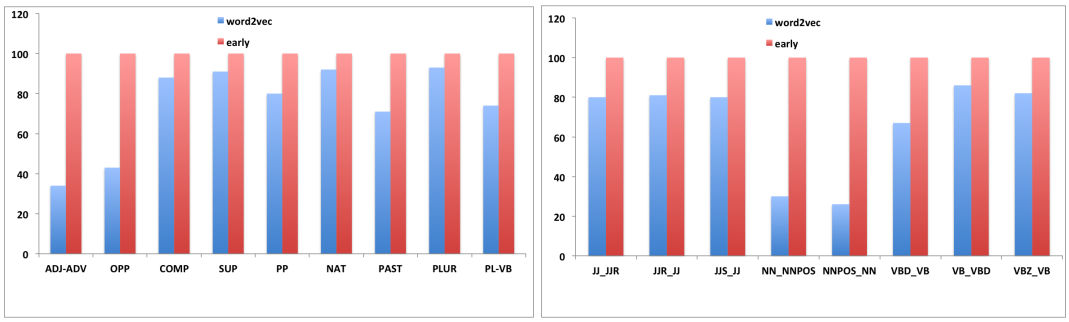

Figure 4.20 : Mesure d'exactitude entre les deux versions `word2vec` et `early` pour les deux jeux de données `google-form` (partie gauche) et `msr-form` (partie droite) détaillés par catégories d'équations.

Afin de produire des résultats sur les corpus `msr` et `google-syntax`, nous combinons notre solveur `early` que nous appliquons aux seules analogies formelles, au système `word2vec` *skip-gram* de Mikolov et Dean (2013) que nous appliquons sur les autres équations. Les performances de cette hybridation sont rapportées dans le tableau 4.IV.

Configuration	word2vec	word2vec+early
msr	67%	72 %
google-syntax	63%	71%

Tableau 4.IV : Combinaison de résultats de early avec word2vec.

Notre combinaison dépasse largement la performance de word2vec. Nous gagnons +5% en exactitude pour le corpus msr et plus de +8% pour le corpus google-syntax. La configuration word2vec+early obtient donc des performances compétitives. Il n'en reste pas moins que les résultats sont purement illustratifs dans la mesure où notre hybridation s'appuie sur l'information qu'une équation a une solution formelle, information qui n'est bien sûr pas disponible en pratique.

Plusieurs travaux se sont intéressés à cette tâche d'analogies en utilisant la méthode word2vec décrite par Mikolov et Dean (2013). Les meilleures performances ont été données par Levy et al. (2015). Les auteurs ont réussi à dépasser les performances de Mikolov et Dean (2013). Ils utilisent plusieurs méthodes, mais celle qui a les meilleurs performances est SGNS-LS (*skip-gram negative sampling large scale*) avec une *accuraccy* de 72,9% sur msr et 75,8% sur google-syntax. Cette méthode nous dépasse de +0,9 (72,9% contre 72%) sur msr et de +4,8% (75,8% contre 71%) sur google-syntax. Ceci nous permet cependant de conclure que les solveurs alea et early sont très compétitifs.

4.13.2 Équations de segments

Nous comparons les résultats de quatre modèles de STRUCTANALOG par rapport au système de référence alea. Le tableau 4.V détaille nos résultats sur les deux corpus test-simple et test-hard en utilisant des modèles déjà entrainés sur les mêmes types de données train-simple et train-hard. L'écart en pourcentage par rapport au solveur alea est mentionné en deux couleurs : en bleu si un modèle gagne

en exactitude et en rouge si l'un des modèles perd toujours par rapport à `alea`. Nous avons choisi deux valeurs pour le méta-paramètre η : 7 et 20, qui définit la limite du nombre d'opérations **`Y`** ou **`Z`** sans consommer dans **`X`**. La première valeur est considéré comme stricte, car elle permet de consommer au maximum 20% de shuffle $|b|+|c|$ (la longueur moyenne de chaque symbole est de 17, ce qui donne 34 pour les deux symboles, 7/34=20%). Par contre, la deuxième valeur de méta-paramètre 20 est considérée comme large, car elle laisse consommer environ 60% de $|b|+|c|$.

Pour le `test-simple`, la plupart des résultats sont inférieurs aux performances d'`alea` (33%). Seul `early` avec le méta-paramètre $\eta = 7$ dépasse le solveur `alea` avec un gain de 5,4%. Comme la plupart des analogies sont faciles, `alea` a les meilleurs performances avec un taux d'échantillonnage $\rho = 100$ (mêmes performances aussi avec $\rho = 1000$), la variante rapportée en tableau 4.V. Il convient cependant de noter que ce jeu d'analogie est bien plus difficile que les jeux d'analogies sur les mots, puisqu'on mieux, nous obtenons 38% de justesse. Ce qui montre le fort impact de la longueur des formes. Avec une longueur moyenne égale à 17 caractères (entre 8 et 26), plusieurs solveurs n'arrivent pas à résoudre ces équations analogiques. Ainsi, (Kaveeta et Lepage, 2016) ont signalé une défaillance de leur modèle pour résoudre des équations impliquant des formes longues.

Pour le corpus `test-simple`, nos quatre modèles dépassent `alea` pour toutes les configurations. Nous concluons que notre solveur STRUCTANALOG a des meilleurs performances que `alea` pour des analogies difficiles. La variante `early` demeure la meilleure.

TRAIN	train-simple		train-hard	
TEST	test-simple		test-hard	
Config	η= 7	η=20	η= 7	η=20
standard	30.6 -2.4 (7)	22.7 -10.3 (24)	**24** **+6** (56)	**25.5** **+7.5** (50)
early	**38.4** **+5.4** (8)	23,3-9.7 (26)	**22.9** **+4.9** (7)	**29.5**+11.5 (13)
latest	26.,9 -6.1 (9)	26.1 -6.9 (28)	**20.1** **+2,1** (6.7)	**20,1** **+2,1** (20)
skip	25,4 -7.6 (8)	25.9-7.1 (30)	**21** **+3** (56)	**21.8**+3.8 (51)
alea ($\rho = 100$)	33.0 (0)		18.1 (0)	

Tableau 4.V : Mesure d'exactitude de quatre modèles de prédiction structurée (standard, early, latest et skip) et du solveur alea en fonction de la valeur η. Le silence de chaque solveur est indiqué entre parenthèses. L'écart de performance avec alea est indiqué en bleu (gain) ou en rouge (perte).

Pour le corpus test-simple, nous constatons que le taux de silence augmente pour $\eta = 20$ par rapport à $\eta = 7$ ce qui signifie que des hypothèses prometteuses sont exclues du faisceau au début du processus de recherche. Le taux de silence aussi pour le corpus test-hard est beaucoup plus élevé car les équations formant ce test sont plus difficiles à résoudre. Par contre, le taux de silence pour la configuration de alea avec ρ=100 est nul : elle produit des solutions même si elles sont fausses.

Nos résultats sont en accord avec les résultats de (Huang et al., 2012) pour la tâche d'analyse morphologique où la meilleure performance est obtenue par early par rapport aux variantes latest, max-violation et standard.

Apprentissage des modèles

Nous avons premièrement vérifié si nos modèles apprennent lors du processus d'apprentissage ou non. Nous prenons le corpus `train-hard` et nous vérifions la performance des quatre modèles pour la première et la dixième itération. Les performances sont nettement meilleures après dix itérations comme l'indique la figure 4.21. Même aussi après une seule itération, les performances d'`early` et `standard` avec $\eta = 20$ sont comparables à `alea` avec un faisceau de 100. Les quatre modèles apprennent avec les deux valeurs $\eta = 7$ et $\eta = 20$.

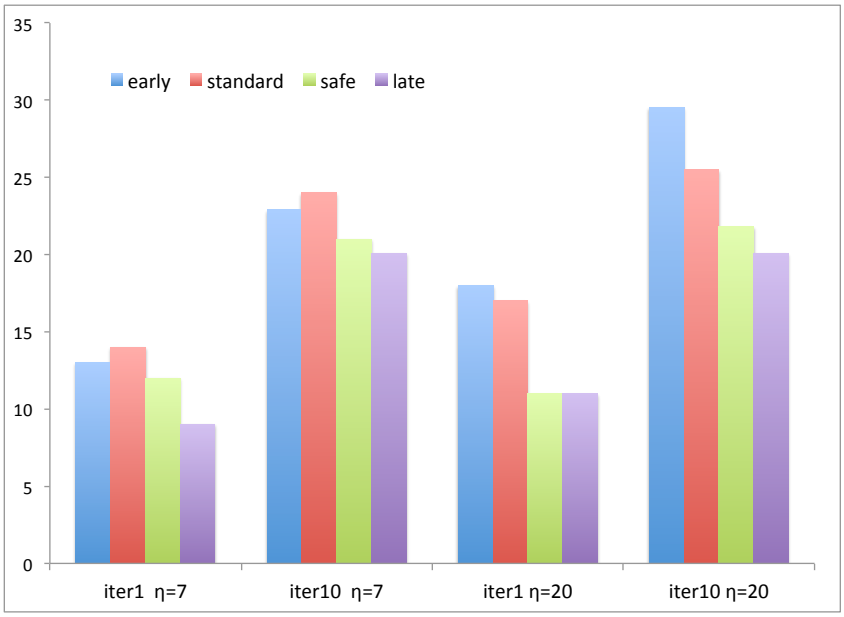

Figure 4.21 : Mesure d'exactitude des différentes variantes de STRUCTANALOG après 1 et 10 itérations, et pour deux valeurs du méta-paramètre η.

Résultats détaillés

Ces performances que nous avons présentées jusqu'à maintenant sont des moyennes sur dix tranches dans le cas de test `test-simple` et sur cinq tranches dans le cas de `test-hard`.

Le tableau 4.VI montre que `early` avec η=7 est la seule configuration qui dépasse `alea` pour tous les tests (t1,t2,t3,..t10).

Config	η/ρ	**t1**	**t2**	**t3**	**t4**	**t5**	**t6**	**t7**	**t8**	**t9**	**t10**	**moyenne**
`alea`	100	36	31	33	31	35	33	35	32	31	33	33.0
`standard`	7	27	27	26	24	23	25	28	25	24	25	25.4
	20	25	27	27	24	25	26	28	26	23	28	25.9
`skip`	7	33	**32**	31	29	28	31	33	30	28	31	30.6
	20	23	25	21	22	22	23	25	22	20	24	22.7
`early`	7	**41**	**38**	**41**	**37**	**38**	**38**	**40**	**38**	**36**	**37**	**38.4**
	20	25	23	21	22	22	23	27	22	21	27	23.3
`late`	7	28	28	28	26	24	26	30	26	26	27	26.9
	20	25	27	27	24	25	27	29	26	22	29	26.1

Tableau 4.VI : Performances sur les dix tranches de `test-simple` avec les quatre modèles de STRUCTANALOG (`standard`, `skip`, `late` et `early`) et `alea`.

Le tableau 4.VII montre que notre solveur STRUCTANALOG avec les différentes variantes résout les équations difficiles contrairement à `alea` qui a une exactitude presque nulle pour les tranches difficiles (tranche 2, tranche 3, tranche 4 et tranche 5). Mais ce n'est pas le cas pour la seule tranche facile (tranche 1) où `alea` dépasse toutes les configuration de STRUCTANALOG. Apprendre à résoudre est donc préférable à priori, et ce d'autant plus que les analogies sont susceptibles d'être de degré élevé.

Config	η/ρ	**tranche1**	**tranche2**	**tranche3**	**tranche4**	**tranche5**	**moy**
`alea`	100	18.04	0.04	0.04	0.0	0.0	18.1
`standard`	7	12.0	3.2	2.9	1.3	1.6	**21.0**
	20	10.9	4.0	3.1	1.4	2.4	**21.8**
`skip`	7	13.6	3.4	3.1	1.7	2.2	**24.0**
	20	13.2	3.8	3.5	2.0	3.0	**25.5**
`early`	7	13.1	3.4	2.3	2.1	2.0	**22.9**
	20	14.9	5.1	3.8	2.9	2.8	**29.5**
`late`	7	10.2	3.4	3.3	1.2	2.0	**20.1**
	20	10.2	3.4	3.3	1.2	2.,0	**20.1**

Tableau 4.VII : Les performances des cinq tranches de `test-hard` avec les quatre modèles de STRUCTANALOG (`standard`, `skip`, `late` et `early`) par rapport à `alea`

L'entrainement des modèles de notre solveur STRUCTANALOG pour 10 itérations conduit déjà à une exactitude beaucoup plus grande que le solveur `alea` avec un taux d'échantillonnage $\rho = 1000$ dont l'exactitude est égale à 18.

Permutation des modèles

Jusqu'à maintenant, les modèles ont été entrainés et ont été testés sur les mêmes types de corpus (`simple` ou `hard`). Nous permutons le test lorsqu' entrainé sur `simple` et réciproquement.

En ce qui concerne `test-simple`, les performances des différentes configurations sont au-dessous de `alea`, même la version `early`. Par contre, la performance de différentes configurations pour `test-hard` dépasse `alea`, excluant `skip` avec $\rho = 7$. Même si nous utilisons des modèles entrainés sur des corpus moins difficiles que le test, nos modèles arrivent à battre le système de référence `alea`. Le problème cependant est que le niveau de performance des solveurs résultants est plus faible globalement.

Nous remarquons aussi que si nous utilisons un corpus d'entrainement `train-simple`, `early` avec $\rho = 7$ est nettement meilleur pour les deux tests. En effet, l'exactitude de `test-simple` est de 38.4% (la meilleure) et l'exactitude de `test-hard` est de 26%, ce qui est supérieur de (+3.0 %) à la même configuration en utilisant des modèles entrainés sur `test-hard`.

Ces résultats décevants mettent en relief l'importance du corpus d'entrainement utilisé pour apprendre les solveurs. Trouver des équations utiles à la tâche en test est un problème qui mérite plus d'étude. Nous pouvons éventuellement penser à classer les analogies comme esquisse avec `simple` et `hard` et apprendre des modèles discrètes par classe de difficulté. Certes, il faudrait alors savoir classer les équations analogiques par difficulté, c'est à dire par degré. Il me semble que ce problème est central dans cette explication de l'analogie par mélange, et qu'elle est au cœur de l'utilisation de la fonction

mélange. Dans le meilleur des cas, il faut appliquer récursivement cette fonction $|a|$fois, c'est à dire la longueur de la première composante de l'équation analogique.

TRAIN	train-hard		train-simple		train-simple		train-hard	
TEST	test-simple				test-hard			
Config	η= 7	η=20	η= 7	η=20	η= 7	η=20	η= 7	η=20
standard	30.1 -2.9 (10)	27.1 -5.9 (26)	30,6 -2,4 (7)	22,7 -10,3 (24)	**19.6 +1.6** (17)	**23.4 +5.4** (16)	**24 +6** (56)	**25,5 +7,5** (50)
early	27.6 -5.4 (10)	31-2 (26)	**38,4 +5.4** (8)	23,3-9,7 (26)	**26 +8** (6)	**23,3+5.3** (15.6)	**22,9 +4.9** (7)	**29,5+11,5** (13)
latest	20.3 -12.7 (11)	18.9 -14.1 (19)	26,9 -6,1 (9)	26,1 -6,9 (28)	**18.9 +0.9** (9)	**22.5 +4.5** (20)	**20,1 +2,1** (6,7)	**20,1 +2,1** (20)
skip	25.6 -7.4 (13)	21.2-11.8 (27)	25,4 -7,6 (8)	25,9-7,1 (30)	14.6 -3.4 (18,7)	**22.1+4.1** (22)	**21 +3** (56)	**21,8+3,8** (51)
alea	33 (0)				18 (0)			

Tableau 4.VIII : Comparaison de l'exactitude des quatre modèles de prédiction structurés `standard`, `early`, `latest` et `skip` par rapport à `alea`. Nous précisons le taux de silence entre parenthèses au-dessous de chaque configuration.

4.14 Discussion

Le problème tiré de la stratégie `word2vec` est l'impossibilité de résoudre des équations analogiques sur des mots qui n'ont pas été vus auparavant. Ainsi, si nous n'avons pas entraîné un vecteur sur un mot donné ou que le mot n'est pas dans le vocabulaire, nous ne pouvons pas résoudre l'équation analogique. Par contre, ce problème ne se pose pas avec notre solveur STRUCTANALOG, qui gère la présence de mots inconnus.

(Kaveeta et Lepage, 2016) présentent un réseau de neurones pour résoudre des équations analogiques sur des chaînes de caractères. Malheureusement, nous ne pouvons pas nous comparer avec leurs résultats parce que les auteurs combinent 3370 analogies formelles

du corpus Google avec 2423 analogies formelles venant aussi de différentes langues (japonais, malais, chinois et anglais). Parmi ces analogies, ils prennent aléatoirement 10% des analogies pour le test. Donc, ces résultats ne sont pas comparables aux nôtres.

4.15 Conclusion

Nous avons montré qu'il est possible d'apprendre un solveur analogique structuré et cela nous donne de meilleurs résultats que le système `alea` actuellement utilisé. Nous montrons qu'il est important d'entrainer nos solveurs sur des corpus adaptés. Les meilleures performances ont été obtenues à l'aide d'un corpus réunissant des analogies de degré différent. Les solveurs résultants produisent beaucoup moins de solutions tout en obtenant des exactitudes au moins égales à celle du solveur `alea` actuellement déployé.

CHAPITRE 5

CONCLUSION ET TRAVAUX FUTURS

Dans cette thèse nous avons traité un problème de réordonnancement en deux phases. Nous cherchons tout d'abord à déterminer la bonne solution parmi celles produites par un système natif et ensuite à l'amener au premier rang. Nous avons pou cela exploité des algorithmes de réordonnancement et de classification. Nous avons réussi à atteindre presque la performance du `R@100` dans plusieurs tâches et à battre les performances des systèmes de référence comme `Moses` dans plusieurs configurations.

Nous avons montré l'efficacité de l'analogie dans le cas de la translittération des noms propres de l'anglais au chinois.

Dans une seconde phase, nous avons formalisé la résolution d'une équation analogique comme une tâche de prédiction structurée. Nous avons utilisé le perceptron structuré comme méthode discriminative et nous avons étudié plusieurs variantes de mise à jour comme `early-update`, `latest`. Nous avons appliqué la prédiction structurée (le perceptron structuré) pour résoudre et générer moins des solutions. Nous avons comparé notre solveur avec d'autres pour une tâche d'analogie sur les mots (`google` et `msr`) et sur une tâche d'analogie sur les segments (`hard` et `simple`). Nos performances sont meilleures que des systèmes de référence sur les mots et aussi sur une tâche plus complexe de segments. Pour la tâche de segments, nous sommes encore loin en termes d'exactitude. Bien que nos résultats soient prometteuse, apprendre à résoudre une équation n'est pas chose facile.

En résumé, cette thèse a réussit d'améliorer les performances de système d'apprentissage analogique en intégrant deux composants d'apprentissage statistiques. Ces deux composants permettent d'améliorer les performances de notre système et dépassent celles des

systèmes existants. Nous pouvons considérer ces résultats comme excellents sur les mots et encourageants sur les segments.

5.1 Travaux futurs

Au début de cette thèse, nous avions plusieurs défis à relever et plusieurs problèmes à résoudre.

La meilleure variante de notre solveur est la variante `early`. Nous souhaitons étudier différentes variantes de fixation de violation. Aussi pour améliorer les performances de nos modèles entrainés, il nous faut bien choisir les corpus d'entrainement. Ce qui doit être mieux étudié.

Faire l'étude de l'importance de certains traits caractéristiques, où l'ajout de nouveaux se doit d'être étudié de manière systématique. Nous pensons notamment qu'une meilleure caractérisation des hypothèses au début de la recherche (où le préfixe peut être vide) est importante.

Dans ce travail, nous nous sommes intéressés aux deux dernières étapes de l'apprentissage analogique : le solveur et le sélecteur. A l'avenir, nous pourrions nous intéresser à la première étape d'apprentissage analogique qui est la recherche.

Avec les performances enregistrées, nous avons constaté que l'apprentissage par analogie peut aller plus loin, surtout si on arrive à insérer des modules d'apprentissage automatique dans la première phase, la phase de recherche des équations analogiques en terme probabiliste en retournant la meilleure sortie o^* d'une entrée i parmi toutes les sorties o tels que :

$$o^* = \arg\max_o P_{ana}(o|i),$$

en ajoutant, comme nous l'avons fait dans la phase de sélection et le solveur, un module d'apprentissage pour sélectionner l'ensemble des candidats pour chaque solution possible afin d'éliminer ceux qui apparaissent inutiles.

Nous pouvons revisiter des tâches concrètes comme la traduction des termes médicaux et la translittération des noms propres par exemple en utilisant STRUCTANALOG au lieu du solveur `alea`. En plus, nous pouvons utiliser notre solveur STRUCTANALOG pour d'autres applications.

BIBLIOGRAPHIE

Académie-Française. *Dictionnaire de l'Académie française*, volume 1. Adphlphe Wahlen, 1838.

Shivani Agarwal et Partha Niyogi. Stability and generalization of bipartite ranking algorithms. Dans *COLT*, volume 3559, pages 32–47. Springer, 2005.

John Robert Anderson, Ryszard Spencer Michalski, Ryszard Stanisław Michalski, Thomas Michael Mitchell et al. *Machine learning : An artificial intelligence approach*, volume 2. Morgan Kaufmann, 1986.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan et Uri Shaft. When is "nearest neighbor" meaningful ? Dans *International conference on database theory*, pages 217–235. Springer, 1999.

Rupal Bhargava, Bapiraju Vamsi et Yashvardhan Sharma. Named entity recognition for code mixing in indian languages using hybrid approach. *Facilities*, 23:10, 2016.

Anders Björkelund et Jonas Kuhn. Learning structured perceptrons for coreference resolution with latent antecedents and non-local features. Dans *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, volume 1, pages 47–57, 2014.

Samuel R Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D Manning et Christopher Potts. A fast unified model for parsing and sentence understanding. *arXiv preprint arXiv :1603.06021*, 2016.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra et Robert L Mercer. The mathematics of statistical machine translation : Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.

Chih-Chung Chang et Chih-Jen Lin. Libsvm : a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27, 2011.

Michael Collins. Discriminative training methods for hidden markov models : Theory and experiments with perceptron algorithms. Dans *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8. Association for Computational Linguistics, 2002.

Michael Collins et Brian Roark. Incremental parsing with the perceptron algorithm. Dans *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 111. Association for Computational Linguistics, 2004.

W Bruce Croft, Donald Metzler et Trevor Strohman. *Search engines : Information retrieval in practice*, volume 283. Addison-Wesley Reading, 2010.

Sandipan Dandapat, Sara Morrissey, Sudip Kumar Naskar et Harold Somers. Mitigating problems in analogy-based ebmt with smt and vice versa : a case study with named entity transliteration. 2010.

Hal Daumé III. *Practical structured learning techniques for natural language processing*. ProQuest, 2006.

Hal Daumé III, John Langford et Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Hal Daumé III et Daniel Marcu. Learning as search optimization : Approximate large

margin methods for structured prediction. Dans *Proceedings of the 22nd international conference on Machine learning*, pages 169–176. ACM, 2005.

Arnaud Delhay et Laurent Miclet. Analogical equations in sequences : Definition and resolution. Dans *Grammatical Inference : Algorithms and Applications*, pages 127–138. Springer, 2004.

Etienne Denoual. Analogical translation of unknown words in a statistical machine translation framework. *Proceedings of Machine Translation Summit XI, Copenhagen*, 2007.

Melvin Diale, Christiaan Van Der Walt, Turgay Celik et Abiodun Modupe. Feature selection and support vector machine hyper-parameter optimisation for spam detection. Dans *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech), 2016*, pages 1–7. IEEE, 2016.

Thomas G Evans. A program for the solution of geometric-analogy test questions. *Semantic Information Processing, M. Minsky, Ed., MIT Press, Cambridge, Mass*, 1968.

Yoav Freund et Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

Nabil Hathout. From wordnet to celex : acquiring morphological links from dictionaries of synonyms. Dans *LREC*, 2002.

Tin Kam Ho. Random decision forests. Dans *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

Douglas R Hofstadter. The copycat project : An experiment in nondeterminism and creative analogies. Rapport technique, DTIC Document, 1984.

Liang Huang, Suphan Fayong et Yang Guo. Structured perceptron with inexact search. Dans *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 142–151. Association for Computational Linguistics, 2012.

Thorsten Joachims. Optimizing search engines using clickthrough data. Dans *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

Vivatchai Kaveeta et Yves Lepage. Solving analogical equations between strings of symbols using neural networks. Dans *ICCBR Workshops*, pages 67–76, 2016.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens et al. Moses : Open source toolkit for statistical machine translation. Dans *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

Terry Koo et Michael Collins. Efficient third-order dependency parsers. Dans *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics, 2010.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus et Richard Socher. Ask me anything : Dynamic memory networks for natural language processing. Dans *International Conference on Machine Learning*, pages 1378–1387, 2016.

Philippe Langlais. Étude quantitative de liens entre l'analogie formelle et la morphologie constructionnelle. Dans *16è Conférence sur le Traitement Automatique des Langues Naturelles (TALN'09)*, Senlis, France, June 2009.

Philippe Langlais. Mapping source to target strings without alignment by analogical learning : A case study with transliteration. Dans *Proceedings of the 51 st annual meetings of the Association of Computationel Linguistics*, pages 684–689, 2013.

Philippe Langlais et Alexandre Patry. Enrichissement d'un lexique bilingue par analogie. *Benarmara et al*, pages 101–110, 2007a.

Philippe Langlais et Alexandre Patry. Translating unknown words by analogical learning. Dans *EMNLP-CoNLL*, pages 877–886, 2007b.

Philippe Langlais et François Yvon. Scaling up analogical learning. Dans *COLING (Posters)*, pages 51–54, 2008.

Philippe Langlais, François Yvon et Pierre Zweigenbaum. Improvements in analogical learning : application to translating multi-terms of the medical domain. Dans *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 487–495. Association for Computational Linguistics, 2009.

Yves Lepage. Un algorithme de resolution d'analogies entre mots (version française de (lepage 98)).

Yves Lepage. Solving analogies on words : an algorithm. Dans *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, pages 728–734. Association for Computational Linguistics, 1998.

Yves Lepage. *De l'analogie rendant compte de la commutation en linguistique*. Thèse de doctorat, Université Joseph-Fourier-Grenoble I, 2003.

Yves Lepage. Lower and higher estimates of the number of true analogies between sentences contained in a large multilingual corpus. Dans *Proceedings of the 20th in-*

ternational conference on Computational Linguistics, page 736. Association for Computational Linguistics, 2004.

Yves Lepage et Etienne Denoual. Purest ever example-based machine translation : Detailed presentation and assessment. *Machine Translation*, 19(3-4):251–282, 2005.

Vincent Letard. *Apprentissage incrémental de modèles de domaines par interaction dialogique*. Thèse de doctorat, Paris Saclay, 2017.

Vincent Letard, Gabriel Illouz et Sophie Rosset. Evaluation de l'apprentissage incrémental par analogie. Dans *TALN, JEP-TALN-RECITAL*, 2016.

Omer Levy, Yoav Goldberg et Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

Haizhou Li, A Kumaran, Vladimir Pervouchine et Min Zhang. Report of news 2009 machine transliteration shared task. Dans *Proceedings of the 2009 Named Entities Workshop : Shared Task on Transliteration*, pages 1–18. Association for Computational Linguistics, 2009.

Percy Liang, Alexandre Bouchard-Côté, Dan Klein et Ben Taskar. An end-to-end discriminative approach to machine translation. Dans *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 761–768. Association for Computational Linguistics, 2006.

Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

André FT Martins, Miguel B Almeida et Noah A Smith. Turning on the turbo : Fast third-order non-projective turbo parsers. 2013.

Bryan McCann, James Bradbury, Caiming Xiong et Richard Socher. Learned in translation : Contextualized word vectors. *arXiv preprint arXiv :1708.00107*, 2017.

Ryan McDonald, Koby Crammer et Fernando Pereira. Online large-margin training of dependency parsers. Dans *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 91–98. Association for Computational Linguistics, 2005.

Valter Akira Miasato, Bruno Gonçalves, Bruno Ribeiro Costa et José Eduardo De Carvalho Silva. Distributed averaged perceptron for brazilian portuguese part-of-speech tagging. Dans *Cognitive Technologies*, pages 27–36. Springer, 2017.

T Mikolov et J Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013.

Fabienne Moreau, Vincent Claveau et Pascale Sébillot. Automatic morphological query expansion using analogy-based machine learning. Dans *European Conference on Information Retrieval*, pages 222–233. Springer, 2007.

Jean Marie Pierrel. Atilf, j.(2004). *Trésor de la langue française informatisé. Dictionnaire de la langue française du XIXe et du XXe siècle*, 2004.

V Pirelli et S Federici. On the pronunciation of unknown words by analogy in text-to-speech systems. 1994.

Andreas Podelski et Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. Dans *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 239–251. Springer, 2004.

Henri Prade et Gilles Richard. *Computational Approaches to Analogical Reasoning : Current Trends*, volume 548. Springer, 2014.

Roi Reichart et Regina Barzilay. Multi event extraction guided by global constraints. Dans *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 70–79. Association for Computational Linguistics, 2012.

Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

Christian Rutten. L'analogie chez aristote. *Revue de philosophie ancienne*, 1(1):31–48, 1983.

Manabu Sassano. An experimental comparison of the voted perceptron and support vector machines in japanese analysis tasks. Dans *IJCNLP*, volume 8, pages 829–834, 2008.

Bhawani Selvaretnam et Mohammed Belkhatir. Natural language technology and query expansion : issues, state-of-the-art and perspectives. *Journal of Intelligent Information Systems*, 38(3):709–740, 2012.

Nicolas Stroppa et François Yvon. An analogical learner for morphological analysis. Dans *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 120–127. Association for Computational Linguistics, 2005.

Andrew Trotman. Learning to rank. *Information Retrieval*, 8(3):359–381, 2005.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann et Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(Sep):1453–1484, 2005.

Peter D Turney et Michael L Littman. Corpus-based learning of analogies and semantic relations. *Machine Learning*, 60(1-3):251–278, 2005.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

Heng Yu, Liang Huang, Haitao Mi et Kai Zhao. Max-violation perceptron and forced decoding for scalable mt training. Dans *EMNLP*, pages 1112–1123, 2013.

François Yvon, Nicolas Stroppa, Arnaud Delhay et Laurent Miclet. Solving analogical equations on words. *Rapport interne D*, 5, 2004.

Luke Zettlemoyer et Michael Collins. Online learning of relaxed ccg grammars for parsing to logical form. Dans *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

Hao Zhang, Liang Huang, Kai Zhao et Ryan McDonald. Online learning for inexact hypergraph search. Dans *Proceedings of EMNLP*, 2013.

Hao Zhang et Ryan McDonald. Generalized higher-order dependency parsing with cube pruning. Dans *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331. Association for Computational Linguistics, 2012.

Yue Zhang et Stephen Clark. A tale of two parsers : investigating and combining graph-based and transition-based dependency parsing using beam-search. Dans *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics, 2008.