Noyaux de réécriture de phrases munis de types lexico-sémantiques

Martin Gleize, Brigitte Grau

Noyaux de réécriture de phrases munis de types lexico-sémantiques

Martin Gleize[1,2] et Brigitte Grau[1,3]
(1) LIMSI-CNRS, Rue John von Neumann, 91405 Orsay CEDEX, France
(2) Université Paris-Sud, Orsay
(3) ENSIIE, Evry
gleize@limsi.fr, bg@limsi.fr

Résumé. De nombreux problèmes en traitement automatique des langues requièrent de déterminer si deux phrases sont des réécritures l'une de l'autre. Une solution efficace consiste à apprendre les réécritures en se fondant sur des méthodes à noyau qui mesurent la similarité entre deux réécritures de paires de phrases. Toutefois, ces méthodes ne permettent généralement pas de prendre en compte des variations sémantiques entre mots, qui permettraient de capturer un plus grand nombre de règles de réécriture. Dans cet article, nous proposons la définition et l'implémentation d'une nouvelle classe de fonction noyau, fondée sur la réécriture de phrases enrichie par un typage pour combler ce manque. Nous l'évaluons sur deux tâches, la reconnaissance de paraphrases et d'implications textuelles.

Abstract.
Enriching String Rewriting Kernels With Lexico-semantic Types
Many high level natural language processing problems can be framed as determining if two given sentences are a rewriting of each other. One way to solve this problem is to learn the way a sentence rewrites into another with kernel-based methods, relying on a kernel function to measure the similarity between two rewritings. While a wide range of rewriting kernels has been developed in the past, they often do not allow the user to provide lexico-semantic variations of words, which could help capturing a wider class of rewriting rules. In this paper, we propose and implement a new class of kernel functions, referred to as type-enriched string rewriting kernel, to address this lack. We experiment with various typing schemes on two natural sentence rewriting tasks, paraphrase identification and recognizing textual entailment.

Mots-clés : fonction noyau, variations sémantiques, réécriture de phrase, reconnaissance de paraphrases, implication textuelle.

Keywords: kernel methods, semantic variations, sentence rewriting, paraphrase identification, textual entailment.

1 Introduction

De nombreuses applications en traitement automatique des langues (TAL) reposent sur le fait de savoir reconnaître que des phrases possèdent des sens proches, que ce soit la reconnaissance d'implication textuelle (RTE) (Dagan *et al.*, 2006), de paraphrases (Dolan *et al.*, 2004) ou de similarité sémantiques (Agirre *et al.*, 2012). Ces problèmes sont généralement représentés comme des problèmes de classification résolus par des méthodes d'apprentissage supervisé reposant sur des représentations différentes des phrases et des phénomènes linguistiques à gérér. Dans (Wan *et al.*, 2006; Lintean & Rus, 2011; Jimenez *et al.*, 2013), les phrases sont représentées par des sacs de mots ou de n-grammes, et reposent essentiellement sur un appariement lexical exact. La reconnaissance de variations lexicales entre deux énoncés de sens proche est traitée par l'ajout de ressources externes, telles WordNet (Mihalcea *et al.*, 2006; Islam & Inkpen, 2009). La prise en compte d'une représentation structurée des phrases pour traiter les variations de formes de surface est fondée sur la comparaison d'arbres syntaxiques (Calvo *et al.*, 2014). (Heilman & Smith, 2010) introduisent un modèle de distance d'édition sur ces arbres. (Socher *et al.*, 2011) utilisent des auto-encodeurs récursifs opérant sur les arbres de constituants pour apprendre à identifier les paraphrases. Les meilleures systèmes combinent différentes méthodes, comme le méta-classifieur de (Madnani *et al.*, 2012), reposant sur des métriques de traduction automatique, et ayant à ce jour les meilleurs résultats sur la reconnaissance de paraphrases. Certaines méthodes détaillées dans la section suivante utilisent des fonctions noyau pour apprendre ce qui rend deux couples de phrases similaires. (Zanzotto *et al.*, 2007) proposent une fonction noyau de comparaison de couples d'arbres syntaxiques, étendue ensuite à des graphes (Zanzotto *et al.*, 2010). (Bu *et al.*, 2012) introduisent un noyau de réécriture de chaînes de caractères (*string rewriting kernel*).

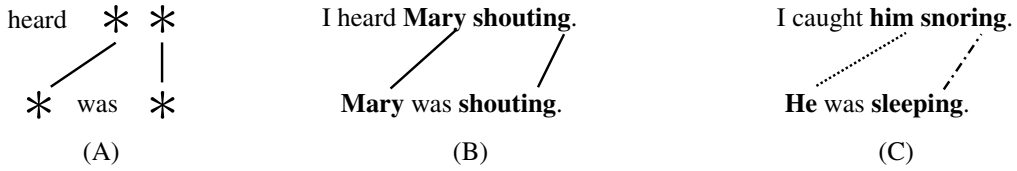

FIGURE 1 – La règle de réécriture (A) réécrit (B) mais pas (C).

Dans cet article, nous proposons d'étendre ces méthodes, en donnant la possibilité de spécifier de manière fine comment deux mots peuvent être appariés et nous définissons ainsi un nouveau noyau de réécriture de phrases enrichi par du typage. Nous détaillons comment calculer ce type de noyau efficacement et nous l'évaluons sur deux tâches de TAL. Notre méthode obtient des résultats analogues à l'état de l'art sur la reconnaissance de paraphrases et dépasse les méthodes de reconnaissance d'implication textuelle fondées sur des approches de même nature.

2 État de l'art des méthodes à noyau pour la réécriture de phrases

Les *fonctions noyau* (*kernel* en anglais) mesurent la similarité de deux éléments. Utilisées dans des méthodes d'apprentissage supervisé telles que les SVM (Vapnik, 2000), elles permettent d'apprendre des fonctions de décision complexes. L'objectif d'une fonction noyau adaptée à ces méthodes est d'avoir une valeur élevée pour deux instances de même étiquette, et une valeur faible pour deux instances d'étiquette différente (Schölkopf & Smola, 2002).
Des méthodes à noyau ont rapidement été employées en traitement automatique des langues. (Lodhi *et al.*, 2002) utilisent le noyau de chaînes (*string kernel*) pour compter le nombre de sous-séquences communes entre deux textes et l'appliquent à la classification de textes.
Classifier des réécritures de phrases revient toutefois à classifier des couples de phrases et requièrent de capturer deux formes de liens : le lien d'une phrase avec l'autre dans un même couple de réécriture, et le lien d'un couple avec un autre. (Zanzotto *et al.*, 2007) proposent une fonction noyau de comparaison de paires d'arbres syntaxiques, étendue ensuite à des graphes (Zanzotto *et al.*, 2010). Leur méthode calcule dans un premier temps le meilleur appariement des noeuds des arbres d'une même paire, pour capturer les entités sur lesquelles portent les deux phrases et former un unique arbre. Un noyau d'arbres –*tree kernel*, introduit par (Moschitti, 2006)– compte dans un second temps les sous-arbres communs des deux paires.
(Bu *et al.*, 2012) introduisent un noyau de réécriture de chaînes de caractères (*string rewriting kernel*) afin de capturer les dépendances syntaxiques sur des paires de phrases vues comme des chaînes de mots, ce qui permet d'apprendre des types de réécriture complexes. Là où l'alignement des mots des deux phrases d'un même couple est réalisé *a priori* dans (Zanzotto *et al.*, 2010) pour réduire le coût de calcul, la contribution de (Bu *et al.*, 2012) propose un algorithme efficace pour intégrer le calcul optimal de ces liens au calcul final du noyau.
Toutes ces méthodes sont toutefois incapables d'introduire des variations lexicales entre mots ou un typage sémantique de ceux-ci, limitant ainsi les types de réécritures apprises. C'est le problème que nous proposons de résoudre dans cet article, en introduisant la notion de *type* pour enrichir les noyaux de réécriture de phrases de (Bu *et al.*, 2012).

3 Noyaux de réécriture de phrases

Définis récemment, les noyaux de réécriture de phrases dénombrent les réécritures communes entre deux couples de phrases vues comme séquences de leurs mots (Bu *et al.*, 2012). La figure 1 présente un exemple de règle de réécriture (A), qui peut être vue comme une paraphrase sous-phrastique avec variables liées (Madnani & Dorr, 2010). La règle (A) réécrit la première phrase de (B) en sa seconde, mais elle ne réécrit pas les phrases de (C). Or, il pourrait être intéressant que la règle (A) se déclenche aussi sur (C), afin d'augmenter les similarités entre réécriture. C'est la motivation de notre contribution : nous présentons et implémentons des noyaux de réécriture de phrases avec types, qui prennent en compte les variations lexico-sémantiques dans les couples de mots.

Dans la suite, nous entendons par *phrase* une séquence de mots, sans y ajouter plus de contraintes linguistiques. On note $(s, t) \in (\Sigma^* \times \Sigma^*)$ une instance de réécriture de phrases, avec sa phrase source s et sa phrase cible t, toutes deux des séquences finies d'éléments de Σ un ensemble fini de mots. Supposons que l'on dispose d'instances étiquetées par

$\{+1, -1\}$ –pour paraphrase/non-paraphrase ou implication/non-implication dans les applications. Il est possible d'appliquer une méthode à noyau pour entraîner un système à classifier automatiquement les instances non étiquetées. Un noyau sur des instances de réécriture de phrases est une fonction :

$$K : (\Sigma^* \times \Sigma^*) \times (\Sigma^* \times \Sigma^*) \to \mathbb{R}$$

telle que pour tous les $(s_1, t_1), (s_2, t_2) \in \Sigma^* \times \Sigma^*$,

$$K((s_1, t_1), (s_2, t_2)) = \langle \Phi(s_1, t_1), \Phi(s_2, t_2) \rangle \quad (1)$$

où Φ projette chaque instance dans l'espace de Hilbert de grande dimension de ces caractéristiques. Les fonctions noyaux permettent d'éviter la représentation explicite potentiellement coûteuse de Φ par l'intermédiaire d'un produit scalaire. Le rôle des noyaux de réécriture de phrases est de mesurer la similarité de deux couples de phrases en comptant le nombre de règles de réécriture d'un ensemble de règles R qu'elles partagent. Φ est donc naturellement définie par $\Phi(s, t) = (\phi_r(s, t))_{r \in R}$ avec chaque caractéristique $\phi_r(s, t) = n$ le nombre de couples de sous-séquences de (s, t) que r *réécrit*. Suivant la définition de R, Φ peut être de dimension non bornée et n'est pas calculable directement, d'où l'intérêt de cette approche par noyau.

4 Noyaux de réécriture de phrases avec types

4.1 Règles de réécriture typées

Soit le domaine joker $D \subseteq \Sigma^*$ l'ensemble des phrases qui peuvent être remplacées par un joker $*$. Nous présentons maintenant le formalisme des noyaux de réécriture de phrases avec types.
Soit Γ_p l'ensemble des *types motif* et Γ_v l'ensemble des *types variable*.
On associe à un type $\gamma_p \in \Gamma_p$ la *relation de type* $\overset{\gamma_p}{\approx} \subseteq \Sigma \times \Sigma$.
On associe à un type $\gamma_v \in \Gamma_v$ la relation de type $\overset{\gamma_v}{\rightsquigarrow} \subseteq D \times D$.
Munis des relations de type associées, on désignera l'association de Γ_p et Γ_v par *schéma de types*.
Soit Σ_p défini par

$$\Sigma_p = \bigcup_{\gamma \in \Gamma} \{[a|b] \mid \exists a, b \in \Sigma, a \overset{\gamma}{\approx} b\} \quad (2)$$

Définissons enfin les règles de réécriture typées. Une *règle de réécriture typée* est un triplet $r = (\beta_s, \beta_t, \tau)$, où $\beta_s, \beta_t \in (\Sigma_p \cup \{*\})^*$ désignent les motifs typés source et cible et $\tau \subseteq ind_*(\beta_s) \times ind_*(\beta_t)$ définit les alignements entre jokers dans les deux motifs. $ind_*(\beta)$ désigne l'ensemble des indices des jokers de β.
On dit que la règle de réécriture (β_s, β_t, τ) *réécrit* un couple de phrases (s, t) si et seulement si les conditions suivantes sont vérifiées :

- Le motif β_s, resp. β_t, peut être transformé en s, resp. t, en :
  - substituant chaque élément $[a|b]$ de Σ_p dans le motif par a ou b ($\in \Sigma$)
  - substituant chaque joker dans le motif par un élément du domaine joker D
- $\forall (i, j) \in \tau$, s, resp. t, substitue les jokers à l'indice i, resp. j, par $s_* \in D$, resp. t_*, tel qu'il existe un type variable $\gamma \in \Gamma_v$ avec $s_* \overset{\gamma}{\rightsquigarrow} t_*$.

Un noyau de réécriture de phrases avec types (TESRK) est simplement un noyau de réécriture de phrases comme défini à l'équation 1 mais avec R un ensemble de règles de réécriture typées. Cette classe de fonctions noyau dépend du domaine joker D et de l'ensemble R, qui peut être choisi de façon à permettre plus de flexibilité dans l'appariement de couples de mots dans les réécritures.
Suivant ce formalisme, le noyau de réécriture de phrases bijectif sur k-grammes (kb-SRK) est défini par le domaine joker $D = \Sigma$ et les règles

$$R = \{(\beta_s, \beta_t, \tau) \mid \beta_s, \beta_t \in (\Sigma_p \cup \{*\})^k, \tau \text{ bijective}\}$$

sous le schéma de types $\Gamma_p = \Gamma_v = \{id\}$ avec $a \overset{id}{\approx} b \Leftrightarrow a \overset{id}{\rightsquigarrow} b \Leftrightarrow a = b$.

4.2 Exemple

Dans cette section, nous présentons un exemple d'application de kb-SRK à un couple réel de phrases, en mettant en avant les limites du noyau et comment il est possible de les dépasser en changeant de schéma de types. Reprenons la figure 1 :

(A) est une règle de réécriture avec β_s = (*heard*, $*$, $*$), β_t = ($*$, *was*, $*$), $\tau = \{(2,1);(3,3)\}$. Chaque motif a la même longueur, et les couples de jokers dans les deux motifs sont alignés de manière bijective. Elle est donc une règle valide de kb-SRK. Elle réécrit le couple de phrases (B) : chaque couple de jokers peut en effet être substitué dans les phrases source et cible par le même mot et les motifs de (A) peuvent ainsi être transformés en couple de sous-phrases de (B).
Cependant, (A) ne peut pas réécrire (C) avec la définition originale de kb-SRK. Redéfinissons alors Γ_p en {hypernym, id} où $a \overset{hypernym}{\approx} b$ si et seulement si a et b ont un hypernyme commun dans WordNet. Changeons aussi Γ_v en Γ_v = {same_pronoun, entailment, id} où $a \overset{same\_pronoun}{\rightsquigarrow} b$ si et seulement si a et b sont un pronom de même personne et même nombre, et où $a \overset{entailment}{\rightsquigarrow} b$ si et seulement si le verbe a a une relation sémantique *entailment* avec b dans WordNet.
En rédéfinissant ainsi le schéma de types, la règle (A) peut maintenant réécrire (C).

5 Calcul de TESRK

5.1 Formulation du problème

La fonction noyau kb-SRK peut être calculée efficacement (Bu *et al.*, 2012). Cette section montre qu'il est possible d'en calculer efficacement une version enrichie de types. S'il est impossible de conserver les mêmes bornes théoriques de complexité en temps, les expériences menées montrent que le temps de calcul est du même ordre de grandeur.
Le kb-SRK avec types est paramétré par k la longueur des k-grammes, et par son schéma de types constitué des ensembles Γ_p and Γ_v et des relations associées. Nous omettrons dans la suite d'annoter K_k et $\bar{K}_k$ de Γ_p and Γ_v par souci de clarté et parce que ces paramètres resteront d'ordinaire constants pour des valeurs de k variables dans nos expériences.
Réécrivons le produit scalaire de l'équation 1 pour mieux refléter les contraintes imposées par les k-grammes :

$$K_k((s_1,t_1),(s_2,t_2)) = \sum_{\substack{\alpha_{s_1}\in k-grams(s_1) \\ \alpha_{t_1}\in k-grams(t_1)}} \sum_{\substack{\alpha_{s_2}\in k-grams(s_2) \\ \alpha_{t_2}\in k-grams(t_2)}} \bar{K}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2})) \tag{3}$$

où $\bar{K}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2}))$ est le nombre de règles de réécriture différentes qui réécrivent à la fois le couple de k-grammes $(\alpha_{s_1},\alpha_{t_1})$, et le couple de k-gramme $(\alpha_{s_2},\alpha_{t_2})$ (la même règle ne peut en effet pas se déclencher deux fois dans des paires de k-grammes) :

$$\bar{K}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2})) = \sum_{r\in R} \mathbb{1}_r(\alpha_{s_1},\alpha_{t_1})\mathbb{1}_r(\alpha_{s_2},\alpha_{t_2}) \tag{4}$$

avec $\mathbb{1}_r$ la fonction indicatrice de la règle r : 1 si r réécrit le couple de k-grammes considéré, 0 sinon.
K_k n'est évidemment pas calculable efficacement en suivant sa définition à l'équation 3. La somme contient $\mathcal{O}((n-k+1)^4)$ termes, avec n la longueur de la phrase la plus longue, et chaque terme implique d'énumérer toutes les règles de réécriture de R.

5.2 Calcul de $\bar{K}_k$ pour kb-SRK avec types

L'énumeration elle-même des règles de réécriture, à l'équation 4, n'est pas calculable en temps raisonnable : il y a $|\Sigma|^{2k}$ règles sans jokers et sans autre relation de type que l'identité, et $|\Sigma|$ sera sans doute la taille d'un lexique d'une langue dans toute application pertinente. En réalité, il suffit de générer de manière constructiviste les règles dont les motifs sont simultanément correctement substitués par $(\alpha_{s_1},\alpha_{t_1})$ et $(\alpha_{s_2},\alpha_{t_2})$.

Soit l'opérateur $\otimes$ tel que $\alpha_1 \otimes \alpha_2 = ((\alpha_1[1],\alpha_2[1]),...,(\alpha_1[k],\alpha_2[k]))$. Cette opération est en général appelée *zip* en programmation fonctionnelle. Il est possible grâce à la fonction *CompterCouplagesParfaits* calculée par l'algorithme 1 de dénombrer récursivement les règles de réécriture réécrivant simultanément $(\alpha_{s_1},\alpha_{t_1})$ et $(\alpha_{s_2},\alpha_{t_2})$. Nous présentons la formule que nous utilisons pour calculer $\bar{K}_k$, et nous expliquons sa correction par le détail de l'algorithme :

$$\bar{K}_k((\alpha_{s_1},\alpha_{t_1}),(\alpha_{s_2},\alpha_{t_2})) = \text{CompterCouplagesParfaits}(\alpha_{s_1}\otimes\alpha_{s_2},\alpha_{t_1}\otimes\alpha_{t_2}) \tag{5}$$

L'algorithme 1 prend en entrée le reste des couples de mots de $\alpha_{s_1}\otimes\alpha_{s_2}$ et $\alpha_{t_1}\otimes\alpha_{t_2}$ et produit en sortie le nombre de façons communes qu'ils ont de se réécrire.

Premièrement (lignes 2 et 3), le cas de base où les deux entrées sont vides est géré. Il y a exactement 1 façon de réécrire l'ensemble vide en lui-même : c'est de ne rien faire.
Puis, aux lignes 4 à 9, il ne reste plus de couples de mots source, donc l'algorithme continue d'épuiser les couples cible tant qu'ils ont un type motif commun. Si un couple cible n'a pas de type motif commun, c'est que ces deux mots dépariés devraient substituer un joker dans une règle valide, mais comme la source est vide, ce joker ne peut pas être aligné et l'algorithme retourne 0.
Dans le cas général (lignes 11 à 19), on considère le premier couple de mots (a_1, a_2) dans le reste de $\alpha_{s_1} \otimes \alpha_{s_2}$ à la ligne 12. Le reste du calcul dépend de ses types. Tout couple de mots dans $\alpha_{t_1} \otimes \alpha_{t_2}$ qui peut s'associer par ses types variable avec (a_1, a_2) (lignes 15 à 19) est un nouvel alignement commun de jokers potentiel donc l'algorithme teste tous les alignements possibles et continue récursivement le calcul après avoir retiré les deux couples alignés. Et si (a_1, a_2) sont des mots avec un type motif commun, ils ne sont pas forcés de substituer un joker (lignes 13 et 14) et nous pouvons donc choisir de ne pas créer de nouvel alignement à cette étape, mais juste de continuer la récursion en "oubliant" le couple motif.
Cet algorithme énumère essentiellement toutes les configurations telles que chaque couple de mots est assuré d'avoir un type motif en commun ou d'avoir un alignement exclusif avec un couple de mots dont il partage les types variable (condition de la ligne 15), ce qui est exactement la définition d'une réécriture de règle réécrivant avec succès dans TESRK.

Algorithm 1: Dénombrement naïf de couplages parfaits

```
1  Function CompterCouplagesParfaits(remS, remT)
       Data:
       remS : couples de mots restants dans la source
       remT : couples de mots restants dans la cible
       graph : α_{s1} ⊗ α_{s2} et α_{t1} ⊗ α_{t2} comme graphebiparti, omis dans les arguments pour éviter d'alourdir les appels
       récursifs
       ruleSet : Γ_p et Γ_v
       Result: Nombre de règles de réécriture réécrivant (α_{s1}, α_{t1}) et (α_{s2}, α_{t2})
2      if remS == ∅ and remT == ∅ then
3          return 1 ;
4      else if remS == ∅ then
5          (b1, b2) = remT.first() ;
6          if ∃γ ∈ Γ_p | b1 ≈^γ b2 then
7              return CompterCouplagesParfaits(∅, remT - {(b1, b2)}) ;
8          else
9              return 0 ;
10     else
11         result = 0 ;
12         (a1, a2) = remS.first() ;
13         if ∃γ ∈ Γ_p | a1 ≈^γ a2 then
14             res += CompterCouplagesParfaits(remS - {(a1, a2)}, remT) ;
15         for (b1, b2) ∈ remT | ∃γ ∈ Γ_v | a1 ⇝^γ b1 and a2 ⇝^γ b2 do
16             res += CompterCouplagesParfaits(
17                 remS - {(a1, a2)},
18                 remT - {(b1, b2)}
19             ) ;
```

Le nom de la fonction CompterCouplagesParfaits est justifié car ce problème est en fait équivalent au dénombrement des couplages parfaits dans le graphe biparti des jokers potentiels, i.e. le graphe ayant pour sommets les jokers potentiels et avec une arête entre deux couples de mots si ils vérifient la condition de la ligne 15 sur leurs types variable. (Valiant, 1979) démontre que ce problème n'est pas calculable efficacement. Notre implémentation représente le graphe par sa matrice de biadjacence, et si on suppose nos relations de type calculables en temps constant par rapport à k, la fonction a une complexité temporelle de $\mathcal{O}(k)$ en ignorant les appels récursifs. Le nombre d'appels récursifs peut dépasser $k!^2$ qui est le nombre de couplages parfaits d'un graphe biparti complet à $2k$ sommets. A la section 6 nous montrons toutefois que sur des données linguistiques, l'algorithme effectue un nombre linéaire d'appels récursifs pour des faibles valeurs de k,

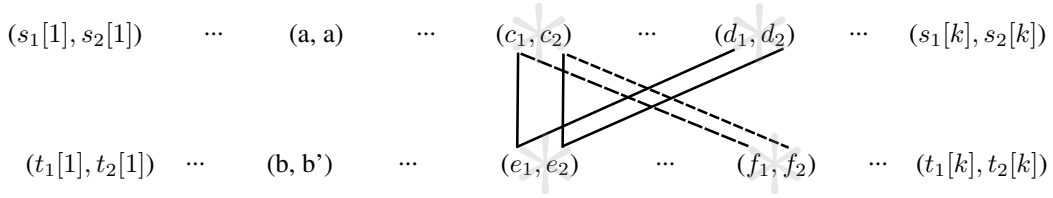

FIGURE 2 – Graphe biparti des couples de mots, avec des arêtes entre les jokers potentiels

et jusqu'à quadratique pour des valeurs de k supérieures à 10 – valeurs pour lesquelles le noyau devient de toute façon inefficace.

La figure 2 montre un exemple de k-grammes source et cible zippés vus comme graphe biparti et avec des arêtes reliant les potentiels jokers. En supposant que les sommets (a, a) et (b, b') ont un type motif commun, ils peuvent être ignorés dans le calcul comme aux lignes 7 et 14 de l'algorithme 1. En revanche, les couples de mots (c_1, c_2) à (f_1, f_2) doivent substituer des jokers dans une règle réécrivant les deux instances. Dans le cas où la ligne 16 aligne (c_1, c_2) avec (e_1, e_2), l'appel récursif retourne 0 car les deux autres couples ne peuvent pas être alignés. Une règle valide est générée seulement si les c sont liés aux f et les d aux e. Le kb-SRK n'avait pas à tester toutes ces possibilités grâce à la transitivité de son seul type (l'*identité*)(Bu *et al.*, 2012). Pour kb-SRK enrichis par des types, il y a moins de contraintes sur les appariements de jokers, ce qui donne une meilleure expressivité mais entraîne aussi l'explosion combinatoire du calcul.

5.3 Calcul de K_k

Même avec une méthode efficace pour calculer $\bar{K}_k$, l'implémentation de K_k en appliquant directement l'équation 3 reste coûteuse. L'idée principale de notre algorithme est de déterminer efficacement un ensemble de taille raisonnable $\mathbb{C}$ d'éléments $((\alpha_{s_1}, \alpha_{t_1}), (\alpha_{s_2}, \alpha_{t_2}))$ ayant la propriété fondamentale d'inclure tous les éléments tels que

$$\bar{K}_k((\alpha_{s_1}, \alpha_{t_1}), (\alpha_{s_2}, \alpha_{t_2})) \neq 0$$

Par définition de $\mathbb{C}$, il suit qu'il est possible de calculer efficacement :

$$K_k((s_1, t_1), (s_2, t_2)) = \sum_{((\alpha_{s_1}, \alpha_{s_2}), (\alpha_{t_1}, \alpha_{t_2})) \in \mathbb{C}} \bar{K}_k((\alpha_{s_1}, \alpha_{t_1}), (\alpha_{s_2}, \alpha_{t_2})) \tag{6}$$

Il existe de multiples façons de réaliser cette réduction de domaine, avec un équilibre à trouver entre temps de calcul et taille du domaine $\mathbb{C}$ conservé. Notre méthode suit la propriété suivante : $\bar{K}_k((\alpha_{s_1}, \alpha_{t_1}), (\alpha_{s_2}, \alpha_{t_2})) = 0$ si il existe un couple de mots $(a_1, a_2) \in \alpha_{s_1} \otimes \alpha_{s_2}$ sans type motif en commun tel qu'on ne trouve pas un couple $(b_1, b_2) \in \alpha_{t_1} \otimes \alpha_{t_2}$ lui étant compatible pour un type variable, c'est-à-dire avec $a_1 \overset{\gamma}{\rightsquigarrow} b_1$ et $a_2 \overset{\gamma}{\rightsquigarrow} b_2$ pour un $\gamma \in \Gamma_v$. C'est évidemment aussi appliquable pour un couple de $\alpha_{t_1} \otimes \alpha_{t_2}$. Plus simplement, les mots qui ne relèvent pas d'un même type motif sont dépariés et doivent nécessairement substituer un joker dans une règle réécrivant à la fois $(\alpha_{s_1}, \alpha_{t_1})$ et $(\alpha_{s_2}, \alpha_{t_2})$. Si il advient que l'on ne peut pas trouver d'alignement de jokers pour un tel couple de mots, aucune règle ne réécrit les deux paires de k-grammes et il est possible d'ignorer ces entrées dans le calcul de K_k. Ce filtrage peut être effectué en temps raisonnable et l'ensemble $\mathbb{C}$ produit ne contient uniquement qu'un nombre linéaire d'entrées en fonction de n d'après nos expériences.

L'algorithme 2 calcule un ensemble $\mathbb{C}$ à utiliser dans l'équation 6 pour obtenir la valeur finale de la fonction noyau K_k. Toutes les *maps* de l'algorithme désignent une implémentation à base de tables de hachage et sont des maps vers des multiensembles (*multiset* en anglais). Les multiensembles sont utilisés tout au long du calcul : ce sont des extensions des ensembles où les éléments peuvent apparaître en plusieurs exemplaires, ce nombre d'exemplaires étant appelé la *multiplicité*. Classiquement implémenté par des tables de hachage vers des entiers, ils permettent de récupérer en temps constant le nombre d'un élément donné. Les opérations d'union et d'intersection ont des définitions spéciales pour les multiensembles, qu'il est bon de rappeler puisque ces opérations sont utilisées dans l'algorithme. Si $\mathbb{1}_A(x)$ désigne la multiplicité de x dans A, on a $\mathbb{1}_{A \cup B}(x) = max(\mathbb{1}_A(x), \mathbb{1}_B(x))$ et $\mathbb{1}_{A \cap B}(x) = min(\mathbb{1}_A(x), \mathbb{1}_B(x))$.

Commentons le déroulement de l'algorithme. Aux lignes 1 à 4, il indexe les mots des phrases source par les mots des phrases cible qui ont des types variable communs, et vice versa. Cela permet aux lignes 15 à 19 d'associer efficacement un

Algorithm 2: Calcul d'un ensemble incluant toutes les entrées telles que $\bar{K}_k \neq 0$

Data: s_1, t_1, s_2, t_2 phrases, et k un entier
Result: Ensemble $\mathbb{C}$ qui inclut toutes les entrées sur lesquelles $\bar{K}_k \neq 0$

Initialize maps $e^i_{s\to t}$ and maps $e^i_{t\to s}$, for $i \in \{1,2\}$;
for $i \in \{1,2\}$ **do**
    for $a \in s_i, b \in t_i \mid a \overset{\gamma}{\rightsquigarrow} b, \gamma \in \Gamma_v$ **do**
        $e^i_{s\to t}[a]$ += (b,γ) ; $e^i_{t\to s}[b]$ += (a,γ);
$w_{s\to t}, aP_t$ = InclusionJoker($s_1, s_2, t_1, t_2, e^1_{s\to t}, e^2_{s\to t}$);
$w_{t\to s}, aP_s$ = InclusionJoker($t_1, t_2, s_1, s_2, e^1_{t\to s}, e^2_{t\to s}$);
Initialize multiset res;
for $(\alpha_{s_1}, \alpha_{s_2}) \in aP_s$ **do**
    for $(\alpha_{t_1}, \alpha_{t_2}) \in aP_t$ **do**
        res += $((\alpha_{s_1}, \alpha_{s_2}), (\alpha_{t_1}, \alpha_{t_2}))$;
res = res $\cup w_{s\to t} \cup w_{t\to s}.map(swap)$;
return res;

Function `InclusionJoker` $(s_1, s_2, t_1, t_2, e^1, e^2)$
    Initialize map d multisets resWildcards, resAllPatterns;
    for $(\alpha_{s_1}, \alpha_{s_2}) \in kgrams(s_1) \times kgrams(s_2)$ **do**
        for $(b_1, b_2) \mid \exists \gamma \in \Gamma_v, (a_1, a_2) \in \alpha_{s_1} \otimes \alpha_{s_2}, (b_i, \gamma) \in e^i[a_i]\ \forall i \in \{1,2\}$ **do**
            $d[(b_1, b_2)]$ += $(\alpha_{s_1}, \alpha_{s_2})$;
    for $(\alpha_{t_1}, \alpha_{t_2}) \in kgrams(t_1) \times kgrams(t_2)$ **do**
        for $(b_1, b_2) \in \alpha_{t_1} \otimes \alpha_{t_2} \mid b_1 \overset{\gamma}{\neq} b_2 \forall \gamma \in \Gamma_p$ **do**
            if *compatWKgrams not initialized* **then**
                Initialize multiset compatWKgrams = $d[(b_1, b_2)]$;
            compatWKgrams = compatWKgrams $\cap\ d[(b_1, b_2)]$;
        if *compatWKgrams not initialized* **then**
            resAllPatterns += $(\alpha_{t_1}, \alpha_{t_2})$;
        for $(\alpha_{s_1}, \alpha_{s_2}) \in$ *compatWKgrams* **do**
            resWildcards += $((\alpha_{s_1}, \alpha_{s_2}), (\alpha_{t_1}, \alpha_{t_2}))$;
    return (resWildcards, resAllPatterns);

Type	Relation de type entre les mots (a, b)	Outils/ressources
id	mots ayant la même forme de surface et même POS tag	OpenNLP tagger
idMinusTag	mots ayant la même forme de surface	OpenNLP tokenizer
lemma	mots ayant le même lemme	WordNetStemmer
stem	mots ayant la même racine	Porter stemmer
synonym, antonym	mots ayant la relation [type]	WordNet
hypernym, hyponym entailment, holonym	b est [type] de a	WordNet
lvhsn	mots ayant une distance d'édition de 1	Levenshtein distance

TABLE 1 – Types définis pour les expérimentations

couple de mots avec l'ensemble des paires de k-grammes opposées contenant un couple avec des types variable communs, c'est-à-dire les couples de k-grammes avec lesquels il pourrait substituer un joker aligné. Aux lignes 20 à 28, seuls les quadruplets de k-grammes dont les couples de mots sans type motif commun d'un côté ont chacun un couple associé pour les types variable de l'autre côté. A la ligne 26, il n'y a pas de couple de mots sans type motif commun ; nous sauvergardons donc ce pur motif dans "all-Patterns". Les k-grammes purs motifs sont appareillés deux à deux aux lignes 8 à 10. Enfin, à la ligne 11, l'algorithme calcule l'union multiensemble des entrées satisfaisant le test d'inclusion de jokers ; l'appel de *swap* dans un cas est nécessaire pour toujours avoir les sources du côté gauche et les cibles du côté droit.

6 Expériences

6.1 Présentation des systèmes

Nos expérimentations portent sur deux tâches : la reconnaissance de paraphrases et la reconnaissance d'implications textuelles. Nous avons utilisé la même configuration dans les différents tests, en faisant varier quelques paramètres que nous avons voulu étudier : le nombre d'exemples d'apprentissage, k, le schéma de types. Nous avons implémentés deux fonctions noyau, le noyau initial kb-SRK de (Bu *et al.*, 2012), dénommé par la suite ***SRK***, et notre noyau enrichi, dénommé ***TESRK***. L'étiquetage morpho-syntaxique est réalisée avec OpenNLP (Baldrige & G., 2010) et les mots sont racinisés par l'algorithme de Porter (Porter, 2001) dans le cas de SRK. Différents prétraitements sont réalisés pour TESRK, pour définir les types. Ils sont détaillés Table 1. Nous avons utilisé LIBSVM (Chang & Lin, 2011) pour entraîner un classifieur SVM binaire avec chacun des deux noyaux. L'algorithme de LIBSVM par défaut utilise un paramètre C, qui peut être grossièrement vu comme un paramètre de régularisation. Nous optimisons ce paramètre pour le f-score par validation croisée sur les données d'entraînement. Tous les noyaux ont été normalisés par $\tilde{K}(x, y) = \frac{K(x,y)}{\sqrt{K(x,x)}\sqrt{K(y,y)}}$. Nous notons "$+$" une somme de noyaux, avec normalisation avant et après la somme. Nous avons suivi le protocole expérimental de Bu et al. (Bu *et al.*, 2012), et avons introduit un noyau de vecteurs supplémentaire, dénommé ***PR***, composé de deux traits représentant la *précision sur des uni-grammes* et le *rappel*, définis dans (Wan *et al.*, 2006). Dans nos tests, le noyau linéaire fournit de meilleurs résultats.

6.2 Reconnaissance de paraphrase

La reconnaissance de paraphrases consiste à déterminer si deux phrases ont le même sens. Le jeu de données que nous avons utilisé pour évaluer nos systèmes est le MSR Paraphrase Corpus (Dolan & Brockett, 2005), qui contient 4076 couples de phrases d'entraînement et 1725 couples de test en langue anglaise. Par exemple, les phrases *"An injured woman co-worker also was hospitalized and was listed in good condition."* et *"A woman was listed in good condition at Memorial's HealthPark campus, he said."* sont des paraphrases dans ce corpus. En revanche, *"There are a number of locations in our community, which are essentially vulnerable," Mr Ruddock said.* et *"There are a range of risks which are being seriously examined by competent authorities," Mr Ruddock said.* ne sont pas des paraphrases.

La table 2 présente nos meilleurs résultats, avec le système ***TESRK + PR***, défini par la somme de PR et de kb-SRK de k allant de 1 à 4, munis des types $\Gamma_p = \Gamma_v = \{stem, synonym\}$. Nous constatons que nos résultats sont comparables à l'état de l'art sur cette tâche et en particulier, ils sont meilleurs que ceux de kb-SRK original. Nous avons aussi essayé

Système Paraphrase	Accuracy	F-score
All paraphrase	66.5	79.9
Wan et al. (2006)	75.6	83.0
Bu et al. (2012)	76.3	N/A
Socher et al. (2011)	76.8	83.6
Madnani et al. (2012)	**77.4**	**84.1**
PR	73.5	82.1
SRK + PR	76.2	83.6
TESRK	76.6	83.7
TESRK + PR	**77.2**	**84.0**

TABLE 2 – Résultats d'évaluation sur MSR Paraphrase Corpus

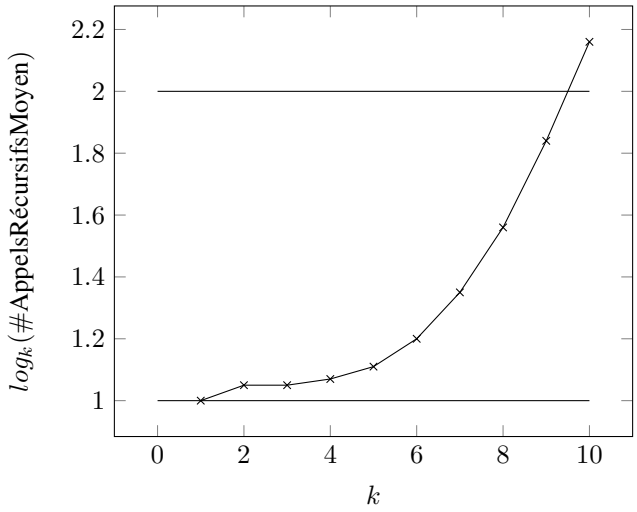

FIGURE 3 – Evolution du nombre d'appels récursifs à CompterCouplagesParfaits en fonction de k

d'autres combinaisons de types mais elles ne donnaient pas mieux : on peut probablement attribuer cela à la nature du corpus de MSR, qui ne semble pas contenir des variations lexico-sémantiques des mots très avancées. Nous avons fait notre possible pour reproduire les perfomances du kb-SRK original (Bu *et al.*, 2012) : notre implémentation et la leur devraient théoriquement être équivalente.

La figure 3 représente le nombre d'appels récursifs moyens à CompterCouplagesParfaits pendant l'exécution de TESRK en fonction de k. Comme nous avons composé ce nombre avec log_k, il est très facile de voir si la complexité observée est plus proche de $\mathcal{O}(k)$ ou de $\mathcal{O}(k^2)$. Nous observons ainsi que la complexité est linéaire pour les faibles valeurs de k mais semblent exploser rapidement quand k dépasse 7. Heureusement, compter le nombre de règles de réécriture communes sur des paires de (7 à 10)-grammes donnent rarement des résultats non nuls, donc il n'est pas judicieux d'utiliser ces valeurs de k.
La figure 4 représente la taille moyenne de l'ensemble $\mathbb{C}$ produit par l'algorithme 2 en fonction de k, divisée par la somme des tailles des 4 phrases impliquées dans le calcul du noyau. On peut voir que cette quantité est linéaire par rapport à la taille des entrées, avec un pic pour les petites valeurs, ce qui n'est pas un problème, puisque le calcul de $\bar{K}_k$ est très rapide pour ces valeurs de k.

6.3 Reconnaître l'implication textuelle

Reconnaître l'implication textuelle consiste à déterminer si une phrase *hypothèse* peut être raisonnablement déduite en lisant une phrase *texte*. Le jeu de données que nous avons utilisé pour évaluer nos systèmes est RTE-3 (Dagan *et al.*, 2006), en langue anglaise. Comme les travaux similaires (Heilman & Smith, 2010; Bu *et al.*, 2012), nous avons gardé en entraînement l'intégralité des couples texte-hypothèse de RTE-1 et 2, combinés à l'ensemble d'entraînement de RTE-3,

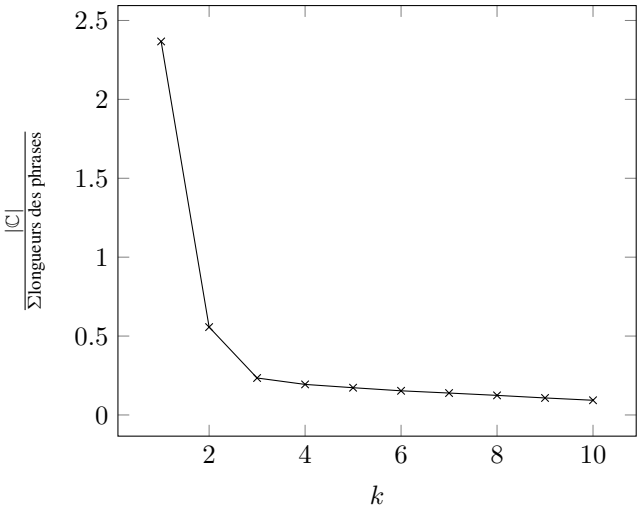

FIGURE 4 – Evolution du nombre d'éléments de $\mathbb{C}$ produits par l'algorithme 2 en fonction de k

Système RTE	Accuracy
All entailments	51.2
Heilman and Smith (2010)	62.8
Bu et al. (2012)	65.1
Zanzotto et al. (2007)	65.8
Hickl et al. (2006)	**80.0**
PR	61.8
SRK + PR	63.8
TESRK (All)	62.1
TESRK (Syn) + PR	64.1
TESRK (All) + PR	**66.1**

TABLE 3 – Résultats d'évaluation sur RTE-3

ce qui donne 3767 couples de phrases. Pour tester, nous avons simplement pris l'ensemble de test de RTE-3 contenant 800 couples de phrases.
Un exemple d'implication textuelle valide trouvé dans ce jeu de données est le couple de phrases *"In a move widely viewed as surprising, the Bank of England raised UK interest rates from 5% to 5.25%, the highest in five years."* et *"UK interest rates went up from 5% to 5.25%."* : la première implique la seconde. En revanche, les phrases *"Former French president General Charles de Gaulle died in November. More than 6,000 people attended a requiem mass for him at Notre Dame cathedral in Paris."* et *"Charles de Gaulle died in 1970."* ne constituent pas d'implication textuelle.

La table 3 présente nos meilleurs résultats, avec le système *TESRK (All) + PR*, défini comme la somme de PR, 1b-SRK (the original kb-SRK for $k = 1$) et des kb-SRK avec types pour k de 2 à 4. Les types utilisés sont $\Gamma_p = \{\text{stem, synonym}\}$ and $\Gamma_v = \{\text{stem, synonym, hypernym, hyponym, entailment, holonym}\}$. Il semble intéressant de comparer nos résultats uniquement avec les systèmes utilisant des techniques et ressources de même nature, mais nous incluons tout de même le meilleur système à RTE-3 pour référence (Hickl *et al.*, 2006). Cette fois nous n'avons pas réussi à reproduire fidèlement les performances de (Bu *et al.*, 2012), mais nous observons tout de même que kb-SRK avec types améliore significativement les performances du kb-SRK de base, allant même jusqu'à faire mieux que l'implémentation originale. Nous avons aussi expérimenté avec des types moins riches pour le système *TESRK (Syn) + PR* en enlevant tous les types WordNet sauf les synonymes, ce qui aboutit cependant à des performances moins élevées. Cela semble vouloir indiquer qu'un système de types riche aide effectivement à capturer des réécritures plus complexes.
Notons le besoin pour $k = 1$ de remplacer TESRK par SRK, sans quoi nos performances chutaient considérablement. Notre hypothèse est qu'inclure des types riches n'aide vraiment que si ils sont capturés au sein d'un contexte d'au moins quelques mots.

7 Conclusion

Nous avons développé une extension de types pour une classe déjà expressive de noyaux de réécriture de phrases. Les types fournissent une plus grande flexibilité dans le décompte des règles de réécritures communes et peuvent aussi ajouter une couche sémantique aux couples de phrases. Nous avons détaillé une méthode efficace pour calculer le noyau de réécriture de phrases bijectif sur les k-grammes muni de types. Un classifieur SVM utilisant ces noyaux enrichis de relations de type provenant de ressources lexico-sémantiques obtient des performances similaires à ou meilleures que l'état de l'art en reconnaissance de paraphrases et implications textuelles.
Nous aimerions nous pencher dans l'avenir sur les applications de ce noyau à d'autres tâches, comme la réponse automatique à des questions.

Références

AGIRRE E., DIAB M., CER D. & GONZALEZ-AGIRRE A. (2012). Semeval-2012 task 6 : A pilot on semantic textual similarity. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1 : Proceedings of the main conference and the shared task, and Volume 2 : Proceedings of the Sixth International Workshop on Semantic Evaluation*, p. 385–393 : Association for Computational Linguistics.

BALDRIGE, J. M. T. & G. B. (2010). Opennlp.

BU F., LI H. & ZHU X. (2012). String re-writing kernel. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics : Long Papers-Volume 1*, p. 449–458 : Association for Computational Linguistics.

CALVO H., SEGURA-OLIVARES A. & GARCÍA A. (2014). Dependency vs. constituent based syntactic n-grams in text similarity measures for paraphrase recognition. *Computación y Sistemas*, **18**(3), 517–554.

CHANG C.-C. & LIN C.-J. (2011). Libsvm : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, **2**(3), 27.

DAGAN I., GLICKMAN O. & MAGNINI B. (2006). The pascal recognising textual entailment challenge. In *Machine learning challenges. evaluating predictive uncertainty, visual object classification, and recognising tectual entailment*, p. 177–190. Springer.

DOLAN B., QUIRK C. & BROCKETT C. (2004). Unsupervised construction of large paraphrase corpora : Exploiting massively parallel news sources. In *Proceedings of the 20th international conference on Computational Linguistics*, p. 350 : Association for Computational Linguistics.

DOLAN W. B. & BROCKETT C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proc. of IWP*.

HEILMAN M. & SMITH N. A. (2010). Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies : The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, p. 1011–1019 : Association for Computational Linguistics.

HICKL A., WILLIAMS J., BENSLEY J., ROBERTS K., RINK B. & SHI Y. (2006). Recognizing textual entailment with lcc's groundhog system. In *Proceedings of the Second PASCAL Challenges Workshop*.

ISLAM A. & INKPEN D. (2009). Semantic similarity of short texts. *Recent Advances in Natural Language Processing V*, **309**, 227–236.

JIMENEZ S., BECERRA C., GELBUKH A., BÁTIZ A. J. D. & MENDIZÁBAL A. (2013). Softcardinality : hierarchical text overlap for student response analysis. In *Proceedings of the 2nd joint conference on lexical and computational semantics*, volume 2, p. 280–284.

LINTEAN M. C. & RUS V. (2011). Dissimilarity kernels for paraphrase identification. In *FLAIRS Conference*.

LODHI H., SAUNDERS C., SHAWE-TAYLOR J., CRISTIANINI N. & WATKINS C. (2002). Text classification using string kernels. *The Journal of Machine Learning Research*, **2**, 419–444.

MADNANI N. & DORR B. J. (2010). Generating phrasal and sentential paraphrases : A survey of data-driven methods. *Computational Linguistics*, **36**(3), 341–387.

MADNANI N., TETREAULT J. & CHODOROW M. (2012). Re-examining machine translation metrics for paraphrase identification. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, p. 182–190 : Association for Computational Linguistics.

MIHALCEA R., CORLEY C. & STRAPPARAVA C. (2006). Corpus-based and knowledge-based measures of text semantic similarity. In *AAAI*, volume 6, p. 775–780.

MOSCHITTI A. (2006). Efficient convolution kernels for dependency and constituent syntactic trees. In *Machine Learning : ECML 2006*, p. 318–329. Springer.

PORTER M. F. (2001). Snowball : A language for stemming algorithms.

SCHÖLKOPF B. & SMOLA A. J. (2002). *Learning with kernels : Support vector machines, regularization, optimization, and beyond*. MIT press.

SOCHER R., HUANG E. H., PENNIN J., MANNING C. D. & NG A. Y. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems*, p. 801–809.

VALIANT L. G. (1979). The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, **8**(3), 410–421.

VAPNIK V. (2000). *The nature of statistical learning theory*. Springer Science & Business Media.

WAN S., DRAS M., DALE R. & PARIS C. (2006). Using dependency-based features to take the "para-farce" out of paraphrase. In *Proceedings of the Australasian Language Technology Workshop*, volume 2006.

ZANZOTTO F. M., DELL'ARCIPRETE L. & MOSCHITTI A. (2010). Efficient graph kernels for textual entailment recognition. *Fundamenta Informaticae*.

ZANZOTTO F. M., PENNACCHIOTTI M. & MOSCHITTI A. (2007). Shallow semantics in fast textual entailment rule learners. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, p. 72–77 : Association for Computational Linguistics.