Alignements monolingues avec déplacements

Julien BOURDAILLET, Jean-Gabriel GANASCIA
Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie, 104 Quai Kennedy, 75016 Paris
{julien.bourdaillet,jean-gabriel.ganascia}@lip6.fr

Résumé. Ce travail présente une application d'alignement monolingue qui répond à une problématique posée par la *critique génétique textuelle*, une école d'études littéraires qui s'intéresse à la genèse textuelle en comparant les différentes versions d'une oeuvre. Ceci nécessite l'identification des déplacements, cependant, le problème devient ainsi NP-complet. Notre algorithme heuristique est basé sur la reconnaissance des homologies entre séquences de caractères. Nous présentons une validation expérimentale et montrons que notre logiciel obtient de bons résultats ; il permet notamment l'alignement de livres entiers.

Abstract. This paper presents a monolingual alignment application that addresses a problem which occurs in *textual genetic criticism*, a humanities discipline of literary studies which compares texts' versions to understand texts' genesis. It requires the move detection, but this characteristic makes the problem NP-complete. Our heuristic algorithm is based on pattern matching in character sequences. We present an experimental validation where we show that our application obtains good results ; in particular it enables whole book alignment.

Mots-clés : alignement monolingue, distance d'édition avec déplacements, critique génétique textuelle.

Keywords: monolingual alignment, edit distance with moves, textual genetic criticism.

1 Introduction

L'alignement textuel monolingue consiste à comparer deux textes plus ou moins proches afin d'identifier leurs similitudes et leurs dissemblances ; ou plus précisément, à rechercher les parties communes à ces deux textes et les parties propres à chaque texte. Les premiers travaux d'alignements automatique peuvent être attribués à (Levenshtein, 1966) qui a introduit la distance d'édition : le nombre minimum d'opérations d'édition (insertions, suppressions et remplacements) permettant de transformer un texte en un autre. Par la suite, cette approche considérant les textes comme deux séquences de caractères a été beaucoup étudiée en informatique théorique, voir (Bergroth *et al.*, 2000) pour une synthèse récente, et appliquée où des programmes de comparaison de code source comme *Diff* ont été développés.

Ces méthodes d'alignement de code source, à savoir des langages formels et structurés, ont ensuite été naturellement adaptées pour comparer les textes en langage naturel. Dans les langages formels, on a généralement une seule instruction par ligne ; ainsi entre deux versions d'un fichier, il est relativement simple d'identifier les modifications. Par contre, dans les textes

en langage naturel, une unité entre ligne et phrase n'a pas de raison d'être, si l'on s'en tient au texte et que l'on omet les questions de mise en page liées au support. Et il se trouve en effet que les logiciels d'alignement existants présentent de mauvais résultats pour les textes en langage naturel, comme nous l'avons montré dans (Bourdaillet & Ganascia, 2006).

En Traduction Automatique, il existe une littérature importante sur l'alignement bilingue de textes qui sont généralement la traduction de l'un dans l'autre (bitexte). Ces alignements sont relatifs à des structures de haut niveau, à savoir paragraphes, phrases et plus difficilement mots (Chiao *et al*., 2006). Nous présentons ici un algorithme d'alignement monolingue au niveau des caractères, entre textes pouvant être très différents l'un de l'autre puisqu'ils peuvent comporter des insertions, suppressions, remplacements et même déplacements. Notre algorithme est plus proche de ceux utilisés en biologie moléculaire tels que (Bray *et al*., 2003), mais néanmoins il induit un alignement aux niveaux supérieurs.

C'est l'étude des processus de réécriture, dans le cadre d'un travail commun avec l'Institut des Textes et Manuscrits Modernes (ITEM), qui nous a amenée à étudier l'alignement monolingue. C'est dans ce laboratoire qu'est née la *critique génétique textuelle* (de Biasi, 2000), une école d'études littéraires étudiant la genèse des oeuvres littéraires à travers les différents états d'un texte laissés par un écrivain. Ces différentes versions, c'est-à-dire les brouillons successifs, sont annotées par l'auteur qui corrige une faute d'orthographe, affine son vocabulaire ou encore soigne son style en déplaçant un terme. D'un point de vue computationnel, les trois opérateurs classiques de la distance d'édition ne sont pas suffisants pour caractériser ces réécritures ; il est nécessaire d'introduire un opérateur de déplacement d'un bloc de caractères d'une position dans le premier texte vers une position différente dans le second. Cette modélisation correspond à la notion de *distance d'édition avec déplacements*.

Les généticiens du texte ont redécouvert empiriquement cette notion, mais celle-ci avait été introduite auparavant en informatique par (Tichy, 1984). (Lopresti & Tomkins, 1997) ont étendu la notion en introduisant plusieurs modèles de distance d'édition par blocs. (Shapira & Storer, 2002) ont prouvé que le calcul de la distance d'édition avec déplacements entre deux textes est un problème NP-complet ; il n'existe donc pas actuellement d'algorithme le résolvant en un temps polynomial et ils ont proposé un algorithme heuristique glouton pour ce calcul.

L'automatisation de ce travail de comparaison textuelle nécessaire à la critique génétique s'avère donc être un problème difficile. Dans la section 2 nous présentons l'algorithme de notre logiciel, appelé MEDITE[1], traitant ce problème. Dans un précédent travail, nous avions montré son utilité pour la critique génétique (Ganascia & Bourdaillet, 2006). Nous montrons ici que l'algorithme glouton de (Shapira & Storer, 2002) ne permet pas de modéliser correctement ce problème et que MEDITE supporte maintenant le passage à l'échelle en permettant d'aligner différentes versions d'un livre entier (section 3).

L'alignement d'ouvrages complets est une problématique récente née de l'essor des projets de numérisation de livres à grande échelle, comme le "Million Book Project" ou celui de Google (Feng & Manmatha, 2006). De plus, la taille des textes rapproche ce problème de l'alignement des séquences d'acides nucléiques en bioinformatique (Gusfield, 1997). Néanmoins la prise en compte des déplacements n'a pas ou peu été traitée dans ces deux domaines.

Nous pouvons maintenant formuler le problème de manière plus précise. Il consiste à aligner deux textes en langage naturel A et B. Ceux-ci peuvent être vus comme des séquences de caractères de tailles respectives m et n, telles que $A = a_1, a_2, ..., a_m = [a_i]_{1 \leq i \leq m}$ et $B =$

[1] librement téléchargeable en ligne : `http://www-poleia.lip6.fr/~ganascia/medite`

$b_1, b_2, ..., b_n = [b_j]_{1 \leq j \leq n}$ et définies sur un alphabet Σ de taille finie.

Nous définissons la notion de paire de blocs (ou *bi-bloc*) par un tuplet (p, l_A, q, l_B) avec $-1 \leq p \leq |A| = m, 0 \leq l_A \leq m$ et $-1 \leq q \leq |B| = n, 0 \leq l_B \leq n$. Cela signifie qu'une sous-chaine $A[p..p + l_A - 1]$ de la première séquence est en relation avec une sous-chaine $B[q..q + l_B - 1]$ de la seconde séquence.

Finalement, nous définissons un *alignement* $\mathcal{A}(A, B)$ entre deux séquences A et B comme un tuplet tel que $\mathcal{A}(A, B) = (INV, SUP, INS, REMP, DEP)$ avec $INV, SUP, INS, REMP$ et DEP les ensembles de bi-blocs respectivement invariants, supprimés, insérés, remplacés et déplacés constituant cet alignement. Ainsi, le type de relation entre les sous-chaines constituant un bi-bloc est défini par l'ensemble auquel le bi-bloc appartient dans $\mathcal{A}(A, B)$. Les invariants, remplacements et déplacements sont des appariements de blocs effectivement présents dans A et B, alors que les suppressions et insertions sont des pseudo-appariements avec un bloc vide. Pour ce faire, un bloc ayant p ou q égal à -1 représente respectivement une insertion ou une suppression ; dans ce cas l_A ou l_B valent respectivement 0, ce qui correspond à un bloc vide.

2 Algorithme

Notre algorithme se décompose en cinq étapes. La première étape est un pré-traitement qui permet d'établir des classes d'équivalence entre caractères. La seconde étape identifie les blocs de caractères répétés entre A et B. La troisième étape aligne ces blocs répétés afin de déterminer lesquels sont invariants et lesquels sont déplacés. La quatrième étape consiste à répéter les étapes 2 et 3 sur les sous-séquences situées entre les blocs alignés lors de l'étape 3. La dernière étape est la déduction des insertions, suppressions et déplacements. La figure 1 présente cet algorithme.

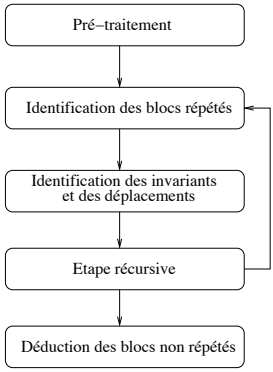

FIG. 1 – Algorithme de MEDITE

2.1 Pré-traitement

Un pré-traitement optionnel peut être appliqué aux séquences A et B. En langage naturel, il existe des caractères que l'on peut vouloir considérer comme équivalents : les caractères identiques mais avec une casse différente, par exemple "J" et "j" ; les caractères avec ou sans signe diacritique, par exemple "ç" et "c" ; ou encore les séparateurs, par exemple " ?" et " !". Pour cela les majuscules sont converties en minuscules, les caractères avec diacritique en leur équivalent sans diacritique, et tous les signes de ponctuation en un même signe, à savoir le caractère point. Ce pré-traitement peut être appliqué sur A et B en un temps linéaire. Ainsi, lors des étapes ultérieures de l'algorithme, des blocs identiques modulo les classes d'équivalence pourront être

appariés même si ces blocs sont différents dans A et B, et ceci pour un coût computationnel faible.

Pour illustrer l'algorithme, nous utilisons l'exemple suivant où nous cherchons à aligner ces deux courts textes : "Ce matin le chat observa de petits oiseaux dans les arbres." et "Le chat était en train d'observer des oiseaux dans les petits arbres ce matin. Il observa les oiseaux pendant deux heures." Après pré-traitement, les phrases deviennent : "ce.matin.le.chat.observa.de.petits. oiseaux.dans.les.arbres." et "le.chat.était.en.train.d.observer.des.oiseaux.dans.les.petits.arbres. ce.matin.il.observa.les.oiseaux.pendant.deux.heures."

2.2 Identification des blocs répétés

L'identification des blocs de caractères répétés, c'est-à-dire présents dans les deux textes, est effectuée en construisant un arbre des suffixes généralisé entre A et B (Ukkonen, 1995). Cette structure de données permet en effet d'identifier l'ensemble des blocs répétés entre A et B en un temps linéaire. Toutefois la taille de cet ensemble de blocs est exponentielle et seul un sous-ensemble est intéressant, celui des appariements exacts super-maximaux (Gusfield, 1997).

Un bi-bloc (p, l_A, q, l_B) est un appariement exact super-maximal si et seulement si :
– $A[p..p + l_A - 1] = B[q..q + l_A - 1]$ (appariement exact) ;
– $A[p - 1] \neq B[q - 1]$ et $A[p + l_A] \neq B[q + l_B]$ (maximalité) ;
– et ni $A[p..p + l_A - 1]$ ni $B[q..q + l_B - 1]$ ne sont inclus dans un autre appariement exact maximal (super-maximalité).

Cette définition n'empêche pas les chevauchements entre appariements exacts super-maximaux (bien que les inclusions le soient). Ces chevauchements peuvent être résolus heuristiquement en les scindant sur les séparateurs, en effet il est préférable d'avoir des coupures entre les mots plutôt qu'à l'intérieur dans les séquences en langage naturel. Le résultat de cette seconde étape se présente sous la forme de deux listes A' et B' de blocs (des séquences A et B) qui ont été identifiés comme faisant partie de l'ensemble des bi-blocs super-maximaux et non-chevauchants.

Dans l'exemple, avant la résolution des chevauchements, les blocs super-maximaux "s.arbres." chevauchent les blocs "oiseaux.dans.les." et "petits." ; la césure sur le séparateur permet de résoudre le conflit. Finalement, après la seconde étape, les blocs super-maximaux non-chevauchants suivants sont identifiés : "ce.matin. le.chat. observa. de. petits. oiseaux.dans.les. arbres." et "le.chat. était.en.train.d.observer.des. oiseaux.dans.les. petits. arbres. ce. matin. il. observa. les.oiseaux. pendant.deux.heures.". Le mot "oiseaux" est répété trois fois mais n'apparait pas dans la liste des blocs super-maximaux car les deux premières occurrences sont incluses dans des blocs plus longs qui eux sont super-maximaux.

2.3 Identification des blocs invariants et déplacés

Les blocs invariants sont ceux qui apparaissent à la même position dans A et B, et les déplacés ceux dont la position a changé. Or chacun des blocs super-maximaux identifiés lors de l'étape précédente peut être soit un bloc invariant, soit un bloc déplacé. En effet, lorsque deux blocs sont permutés, on peut établir que l'un est invariant et l'autre déplacé ou vice-versa, et il n'existe pas de variable permettant de prendre la bonne décision de façon certaine. Néanmoins nous pouvons utiliser le critère heuristique suivant : entre une telle paire de blocs, le plus long sera considéré

comme invariant et l'autre comme déplacé.

La méthode exhaustive permettant de prendre l'ensemble de ces décisions pour tous les blocs consiste à parcourir l'espace des alignements possibles afin de trouver l'optimum suivant une certaine fonction de coût. Or la taille de cet espace est combinatoire, ce qui rend cette recherche peu opérationnelle. C'est pourquoi nous utilisons l'algorithme de type A* suivant.

Les alignements possibles sont évalués à l'aide d'une fonction de coût c ; l'objectif est de trouver un alignement de coût minimal. Ceci est équivalent à un problème de plus court chemin dans un graphe où l'état final correspond à l'alignement de coût minimal et l'état initial à l'état où aucune décision n'a encore été prise. A chaque étape de l'algorithme, une décision est prise en choisissant de désigner l'appariement d'un bloc A'_i avec un bloc B'_j comme étant un bi-bloc invariant. Ce choix est conduit grâce à la fonction de coût c qui estime le coût de l'alignement final induit par ce choix. Lorsque l'état final est atteint, c'est-à-dire lorsque l'on ne peut plus choisir de bloc invariant, tous les blocs qui n'ont pas été choisis durant le parcours sont considérés comme des blocs déplacés. Afin d'atteindre l'état final, c doit être admissible, c'est-à-dire ne jamais surestimer le coût de l'alignement ; nous détaillons ci-dessous pourquoi c est admissible.

L'évaluation du coût de l'alignement induit par le choix de l'appariement de A'_i et B'_j est calculé par la fonction $c(i,j)$. Celle-ci décompose ce coût en un coût $g(i,j)$ de l'alignement effectué lors des étapes précédentes, et une estimation heuristique $h(i,j)$ du coût de l'alignement qu'il reste à effectuer durant les étapes ultérieures, tel que $c(i,j) = g(i,j) + h(i,j)$. Ces coûts sont calculés de la façon suivante :

- $NA(i,j) = non-appariés(A'[1..i-1], B'[1..j-1])$ est l'ensemble des blocs non appariés durant les étapes précédentes, ceux qui n'ont pas été choisis comme invariants et seront considérés comme déplacés.
- $g(i,j) = \sum_{b \in NA(i,j)} |b|$ est la somme de la taille des blocs précédemment non choisis comme étant invariants, c'est-à-dire que seuls les déplacements vont pénaliser le coût d'un alignement.
- $DS(i,j) = A'[i+1..|A'|] \ominus B'[j+1..|B'|]$ est la différence symétrique des deux ensembles de blocs à aligner durant les étapes suivantes. Ces blocs sont présents soit uniquement dans $A'[i+1..|A'|]$ soit uniquement dans $B'[j+1..|B'|]$, il ne sera donc pas possible de les apparier ultérieurement.
- $h(i,j) = \sum_{b \in DS(i,j)} |b|$ est la somme de la taille des blocs de $DS(i,j)$. $h(i,j)$ est la borne inférieure du coût des blocs restant à aligner. h ne surestime jamais le coût de l'alignement, c'est pourquoi c est admissible et A* trouve l'alignement optimal au sens de notre critère.

Ce calcul permet de rechercher un alignement optimal au sens de la maximisation de la taille des blocs invariants et de la minimisation de la taille des blocs déplacés.

Dans notre exemple, après cette étape, les blocs encadrés en gras désignent les invariants et les autres blocs encadrés les déplacements : "ce.matin. **le.chat.** observa. de. petits. **oiseaux.dans.les.** **arbres.**" et "**le.chat.** était.en.train.d.observer.des. **oiseaux.dans.les.** petits. **arbres.** ce. matin. il. observa. les.oiseaux. pendant.deux.heures.".

2.4 Recherche récursive d'appariements

Lors de cette quatrième étape, on considère chaque sous-chaine de A et B située entre deux bi-blocs invariants. Ces sous-chaines sont examinées à nouveau par les étapes 2 et 3 afin de découvrir d'éventuels nouveaux bi-blocs invariants, auquel cas ceux-ci sont ensuite inclus dans

l'alignement principal. Cette étape récursive permet de répondre aux *effets de masquage* qui se produisent lorsque les séquences A et B comportent un nombre important de sous-chaines répétées : dans ces cas là, les algorithmes classiques d'alignement omettent des appariements importants qui sont masqués par des appariements moins importants (Ganascia & Bourdaillet, 2006). De tels phénomènes ont également été identifiés dans les séquences d'acides nucléiques en biologie moléculaire (Arslan *et al.*, 2001).

Dans notre exemple, entre les bi-blocs invariants "le.chat." et "oiseaux.dans.les." se trouvent les sous-chaines "observa.de.petits." et "était.en.train.d.observer.des.". L'étape récursive va permettre d'identifier le bi-bloc "observ" comme invariant, ce qui donne l'alignement final suivant : "ce.matin. **le.chat.** **observ**a.de. petits. **oiseaux.dans.les.** **arbres.**" et "**le.chat.** était.en.train.d.**observ**er.des. **oiseaux.dans.les.** petits. **arbres.** ce. matin. il.observa.les. oiseaux.pendant.deux.heures.". Ainsi, un bi-bloc déplacé a été perdu ("observa.") mais un bi-bloc invariant a été gagné ("observ") ; ceci permet de favoriser les appariements locaux au détriment des appariements longue-distance et les invariants plutôt que les déplacements.

2.5 Déduction des autres types de blocs

Les insertions, suppressions et remplacements peuvent finalement être déduits des étapes précédentes. En effet, les suppressions sont les blocs non répétés et présents uniquement dans A, et les insertions ceux présents uniquement dans B.

L'identification des remplacements se fait de manière heuristique : lorsqu'entre deux bi-blocs invariants se trouve un bloc supprimé s dans A et un bloc inséré i dans B et que le ratio entre leur taille $|s|/|i|$ atteint un certain seuil t, alors ces blocs sont retirés des ensembles SUP et INS (cf. section 1), et appariés en un bi-bloc r placé dans $REMP$, signifiant ainsi que le bloc dans A a été remplacé par le bloc dans B. Le seuil t est fixé par défaut à $0,5$.

Dans l'exemple, le bi-bloc constitué des chaines "a.de." et "er.des." sera considéré comme un remplacement et les deux autres blocs non encadrés de la seconde séquence comme des insertions.

Finalement, un post-traitement permet de retrouver les positions des blocs dans les séquences originales (c'est-à-dire sans les classes d'équivalence).

3 Validation expérimentale

3.1 Application à la critique génétique textuelle

Cette expérience consiste à aligner deux versions d'un même texte et évaluer l'alignement résultant. Pour ce faire, nous allons comparer les résultats de MEDITE à ceux de GREEDY qui est un algorithme glouton de calcul de la distance d'édition avec déplacements (Shapira & Storer, 2002). Ce dernier sélectionne à chaque itération le plus grand appariement qu'il considère comme un déplacement et finalement calcule une distance d'édition classique par programmation dynamique.

L'évaluation des alignements résultants se fait en calculant les fonctions de score suivantes à partir d'un alignement $\mathcal{A}(A, B)$:

– Nous définissons au préalable une fonction $somme(S) = \sum_{(p,l_A,q,l_B)\in S} l_A + l_B$ qui somme la taille de tous les bi-blocs d'un ensemble de bi-blocs S.
– On cherche à maximiser la somme de la taille des blocs invariants et à minimiser la somme de la taille des autres types de bloc, d'où la fonction :

$$x = \left(1 + \frac{somme(INV) - \sum_{s\in S} somme(s)}{|A| + |B|}\right)/2 \quad \text{avec } S = \{SUP, INS, REMP, DEP\} \tag{1}$$

– On cherche à maximiser la taille moyenne des blocs afin d'éviter la fragmentation de l'alignement :

$$y = \left(\sum_{s\in S}\left(\frac{somme(s)}{|s|}\right) / \max(s)\right)/5 \quad \text{avec } S = \{INV, SUP, INS, REMP, DEP\}, \; |s| \text{ le nombre de blocs dans s et } \max(s) \text{ la taille du plus grand bloc de s} \tag{2}$$

– On cherche à maximiser le ratio des déplacements par rapport aux autres blocs non-invariants et le ratio des remplacements par rapport aux autres blocs non-invariants (sauf les déplacements) :

$$z = \left(\frac{somme(MOV)}{somme(S_1)} + \frac{somme(REMP)}{somme(S_2)}\right)/2 \quad \text{avec } S_1 = \{SUP, INS, REMP, DEP\} \text{ et } S_2 = \{SUP, INS, REMP\} \tag{3}$$

– Finalement, ceci nous permet de définir une fonction de similarité globale combinant les équations précédentes ; les pondérations sont fixées arbitrairement mais reflètent les priorités accordées aux différentes fonctions :

$$sim = 0.5x + 0.35y + 0.15z \tag{4}$$

Les termes de normalisation rendent ces équations un peu chargées, mais les idées sous-jacentes sont très simples.

Les textes à aligner sont les suivants : deux versions d'un poème d'Andrée Chedid "La Robe Noire" de 2 Ko, nommé A ci-dessous ; un cahier d'expérience de Claude Bernard et une synthèse académique de ce cahier (7.5 Ko, B) ; un sous-ensemble de la partie française du Hansard et la traduction en français de la partie correspondante anglaise[2] (20 Ko, C) ; et deux versions d'un texte de Louis Althusser "Freud et Lacan" (50 Ko, D). Le tableau 1 présente les résultats de ces alignements.

On peut constater que MEDITE obtient de meilleurs résultats pour tous les textes et critères (sauf pour z sur B et C). Le critère x signifie que MEDITE trouve plus de blocs invariants que GREEDY ; y signifie que les blocs alignés sont plus longs ; et z qu'on favorise les déplacements au détriment des autres types de blocs non-invariants et les remplacements au détriment des insertions et suppressions, en effet ces blocs apportent plus d'informations. On remarquera aussi les différences considérables en temps de calcul.

[2] corpus pré-traité et mis à disposition par le RALI

Algorithme	GREEDY				MEDITE			
Texte	A	B	C	D	A	B	C	D
x	0.3654	0.2657	0.4106	0.7835	0.4934	0.2697	0.4936	0.9223
y	0.1161	0.0793	0.0784	0.1397	0.3331	0.2488	0.1951	0.2318
z	0.1971	0.2340	0.4096	0.1653	0.2003	0.1676	0.2937	0.2587
sim	0.2529	0.1957	0.2942	0.4655	0.3933	0.2471	0.3591	0.5811
Temps	0mn 18s	12mn 5s	1h 1mn	29mn 3s	0mn 1s	0mn 2s	0mn 6s	0mn 2s

TAB. 1 – Alignements avec GREEDY et MEDITE

3.2 Alignement de données synthétiques

Le but de cette seconde expérience est d'évaluer la qualité des alignements de MEDITE sur des données synthétiques où il existe un alignement de référence. Etant donné un texte et un générateur de bruit, un second texte est généré en altérant le premier. L'alignement entre les deux textes est enregistré durant le processus d'altération ; il est alors possible d'évaluer la qualité d'un aligneur en comparant ses résultats avec l'alignement de référence.

Premier générateur de bruit Le générateur de bruit permet de générer un second texte à partir de l'original de la façon suivante. Des ratios d'insertions, suppressions et remplacements sont fixés avant de commencer. Des blocs de caractères sont alors insérés dans le second texte, supprimés dans l'original et remplacés entre les deux textes, de façon répétée jusqu'à ce que les ratios soient atteints. Les positions des modifications sont choisies aléatoirement sur toute la longueur des textes (le chevauchement d'opérations n'est pas permis). La taille des blocs est choisie aléatoirement entre 1 et 25 caractères. Durant ce processus, les positions des modifications sont enregistrées, ce qui permet d'obtenir un alignement de référence.

Pour cette expérience, nous avons choisi un texte de 520 Ko comme texte original, soit la taille d'un livre d'environ 350 pages. Cinq textes synthétiques différents ont été générés et alignés chacun avec l'original via MEDITE, puis les scores de précision calculés. Deux séries de tests avec différents ratios de modifications ont été menées : dans la première il y a 5% d'insertions, 5 % de suppressions et 5 % de remplacements, ce qui signifie que les textes altérés présentent 15 % de différences avec l'original ; dans la seconde série, les ratios sont portés à 10 %, ce qui signifie qu'il y 30 % de différences entre les textes.

Pour chacun des quatre types de caractères (invariants, insertions, suppressions et remplacements) le taux de précision est défini comme le nombre de caractères correctement alignés / le nombre total de ces caractères. Les précisions moyennes sur les cinq alignements sont alors calculées. Pour la précision pondérée, les précisions de chaque type sont pondérées par leurs poids respectifs dans les textes ; par exemple pour la première série de tests on aura $Prec.pondérée = 0.85 * Prec.INV + 0.05 * Prec.INS + 0.05 * Prec.SUP + 0.05 * Prec.REMP$. Les deux premières colonnes du tableau 2 présentent les résultats de cette expérience.

On peut constater que les précisions moyennes sont bonnes, en particulier les précisions pondérées, et que les temps de calcul sont raisonnables ; il serait inenvisageable de traiter ces textes avec GREEDY. L'expérience a été réalisée sur un Pentium 4, 2.4 GHz avec 1 Go de RAM. MEDITE est implémenté en Python, un langage de haut niveau, bon pour le prototypage mais

Générateur de bruit	sans déplacements		avec déplacements	
Ratio de modifications	5 %	10 %	5 %	10 %
Précision moyenne	94.48 %	89.27 %	86.56 %	78.36 %
Précision pondérée moyenne	98.16 %	94.0 %	95.19 %	86.18 %
Temps moyen	11 mn 5 s	27 mn 53 s	27 mn 8 s	77 mn 17 s

TAB. 2 – Alignement de données synthétiques avec MEDITE

lent. Une implémentation en C permettrait de gagner considérablement en vitesse d'exécution. Néanmoins, le goulot d'étranglement de notre algorithme reste le calcul des différences symétriques entre listes de blocs (voir section 2.4), qui est quadratique par rapport à la taille de ces listes.

Générateur de bruit avec déplacements Ce second générateur de bruit est similaire au premier mais en plus des déplacements seront générés entre texte original et texte altéré. Ainsi, des blocs de caractères sont déplacés d'une position dans le texte original vers une seconde dans le texte altéré. Les ratios de modifications sont toujours fixés à 5 et 10 % par opérations, ce qui donne des textes avec 20 et 40 % de différences respectivement. Deux séries de tests sont à nouveau conduites et les moyennes des résultats présentées dans les deux dernières colonnes du tableau 2.

On peut constater que les précisions moyennes décroissent significativement mais que les précisions pondérées conservent de meilleurs scores. Il faut toutefois garder à l'esprit que les ratios de différences sont de 20 et 40 % contre 15 et 30 % avec le premier générateur de bruit. De plus, la différence entre les précisions pondérées et non-pondérées indiquent que les blocs invariants ont un meilleur taux de classification. Ceci est confirmé dans le tableau 3 qui présente la moyenne des matrices de confusion : les blocs de référence sont en lignes et ceux trouvés par MEDITE en colonnes. Les erreurs les plus importantes proviennent des déplacements qui sont identifiés comme insertions et suppressions ; or un déplacement peut être considéré comme une suppression suivie d'une insertion. De même les insertions et suppressions sont confondues avec des remplacements ; or les remplacements peuvent aussi être considérés comme une suppression suivie d'une insertion. Ceci implique un problème de décision : notre modèle de décision est très simple et pourrait être amélioré ; néanmoins les résultats présentés ont le mérite d'être consistants.

Ratio de modifications	5 %					10 %				
Type de bloc	INV	INS	SUP	REMP	DEP	INV	INS	SUP	REMP	DEP
Invariants	**98.07**	0.56	0.52	0.76	0.08	**94.0**	1.75	1.6	2.32	0.32
Insertions	0.21	**92.16**	0	7.55	0.07	0.21	**85.1**	0	14.52	0.17
Suppressions	0.14	0	**87.76**	9.25	1.76	1.46	0	**76.77**	17.78	3.99
Remplacements	0.70	5.40	4.65	**88.40**	0.84	0.72	11.32	9.43	**76.34**	2.18
Déplacements	1.43	13.70	14.05	4.38	**66.43**	1.47	15.16	15.55	8.25	**59.56**

TAB. 3 – Moyenne des matrices de confusion (en %) pour le générateur de bruit avec déplacements

4 Conclusion

Nous avons présenté MEDITE, un aligneur monolingue détectant les déplacements entre deux textes. Nous traitons ce problème d'alignement difficile par un algorithme heuristique de recherche d'homologies dans les séquences. Notre validation expérimentale montre que MEDITE présente de bons résultats et qu'il est capable d'aligner des livres entiers en un temps raisonnable, tout en identifiant les déplacements.

MEDITE est maintenant utilisé par les généticiens du texte pour aligner différentes versions de livres entiers. Ce travail fastidieux nécessiterait plusieurs mois, voire plusieurs années de travail sans l'usage de la machine. Nous projetons maintenant de l'utiliser pour établir des éditions électroniques d'ouvrages en intégrant directement le logiciel dans le support électronique.

Références

ARSLAN A. N., EGECIOGLU O. & PEVZNER P. A. (2001). A new approach to sequence comparison : normalized sequence alignment. *Bioinformatics*, **17**(4), 327–337.

BERGROTH L., HAKONEN H. & RAITA T. (2000). A Survey of Longest Common Subsequence Algorithms. In *SPIRE '00 : Proceedings of the Seventh International Symposium on String Processing Information Retrieval*.

BOURDAILLET J. & GANASCIA J.-G. (2006). MEDITE : A unilingual textual aligner. *In Proceedings of FinTAL, 5th International Conference on Natural Language Processing, Lecture Notes in Artificial Intelligence*, **4139**, 458–469.

BRAY N., DUBCHAK I. & PACHTER L. (2003). AVID : A Global Alignment Program. *Genome Res.*, **13**(1), 97–102.

CHIAO Y.-C., KRAIF O., LAURENT D., NGUYEN T. M. H., SEMMAR N., STUCK F., VÉRONIS J. & ZAGHOUANI W. (2006). Evaluation of multilingual text alignment systems : the ARCADE II project. *Proceedings of the LREC 2006 Conference*.

DE BIASI P.-M. (2000). *La Génétique des Textes*. Nathan Université.

FENG S. & MANMATHA R. (2006). A hierarchical, HMM-based automatic evaluation of OCR accuracy for a digital library of books. In *JCDL'06*, p. 109–118 : ACM Press.

GANASCIA J.-G. & BOURDAILLET J. (2006). Alignements unilingues avec MEDITE. In *8^{émes} Journées Internationales d'Analyse Statistique des Données Textuelles (JADT 2006)*.

GUSFIELD D. (1997). *Algorithms on Strings, Trees and Sequences : Computer Science and Computer Biology*. Cambridge University Press.

LEVENSHTEIN V. (1966). Binary codes capable of correcting deletions, insertions and reversal. *Cybernetics and Control Theory*, **10**(8), 707–710.

LOPRESTI D. P. & TOMKINS A. (1997). Block Edit Models for Approximate String Matching. *Theoretical Computer Science*, **181**(1), 159–179.

SHAPIRA D. & STORER J. A. (2002). Edit Distance with Move Operations. In *CPM*, volume 2373 of *Lecture Notes in Computer Science*, p. 85–98 : Springer.

TICHY W. F. (1984). The String-to-String Correction Problem with Block Moves. *ACM Trans. Comput. Syst.*, **2**(4), 309–321.

UKKONEN E. (1995). On-Line Construction of Suffix Trees. *Algorithmica*, **14**(3), 249–260.