Alignement monolingue avec recherche de déplacements pour la critique génétique

Julien Bourdaillet

Laboratoire RALI
Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
C.P. 6128, succursale Centre-Ville
H3C 3J7, Montréal, Québec, Canada
julien.bourdaillet@umontreal.ca

RÉSUMÉ. *Cet article présente la problématique de l'alignement monolingue avec recherche de déplacements. Celle-ci est posée par la critique génétique, une discipline d'études littéraires. Les solutions informatiques existantes ne sont pas satisfaisantes pour répondre à ce problème NP-difficile. Nous proposons d'emprunter à la bioinformatique et l'algorithmique textuelle une famille d'algorithmes appelée alignement par fragments. Une adaptation de ce type d'algorithmes pour le TAL est décrite. Notre méthode permet d'aligner deux textes en recherchant les déplacements, au caractère près, en passant à l'échelle, et pour n'importe quelle langue alphabétique. Une évaluation expérimentale présente les bons résultats obtenus face à d'autres méthodes.*

ABSTRACT. *This paper describes the problem of monolingual alignment with search for displaced segments (hereafter called move search). It occurs in genetic criticism, a subfield of literary studies. Existing alignment applications fare poorly on this NP-hard problem. We propose to borrow a family of algorithms from bioinformatics and text algorithmics called fragment chaining alignment. An adaptation of this type of algorithms to NLP is described. Our method enables us to align a pair of texts with move search at the character level. It works for any alphabetic language and achieves scalability. An experimental evaluation presents the good results we obtained compared to other approaches.*

MOTS-CLÉS : *alignement monolingue, recherche de déplacements, algorithmique textuelle, critique génétique.*

KEYWORDS: *monolingual alignment, move search, text algorithmics, genetic criticism.*

1. Introduction

L'alignement monolingue consiste à rechercher les invariants et les différences existants entre deux textes écrits dans une même langue. Très tôt dans l'histoire de l'informatique cette question s'est révélée importante. Pouvoir comparer deux versions du code source d'un programme, afin d'en visualiser facilement les différences, est à l'origine des premiers aligneurs, tel le programme `diff` (Hunt et McIlroy, 1976). La sauvegarde incrémentale des fichiers texte, à l'époque où l'espace disque était rare, fut également une des principales motivations liées au développement de ces aligneurs. Avec la généralisation de la diffusion des ordinateurs et l'apparition des applications de traitement de texte, l'alignement s'est naturellement imposé comme un outil des plus utiles pour l'écriture sur machine. On trouve par exemple dans Microsoft Word un outil intégré d'alignement textuel.

De nombreux autres aligneurs commerciaux sont également disponibles. Une première étude qualitative, portant sur neuf aligneurs commerciaux dont Microsoft Word, nous a montré qu'ils obtenaient de mauvais résultats dès lors que les textes à comparer devenaient trop différents (Bourdaillet et Ganascia, 2006). Les aligneurs ont alors beaucoup de difficulté à repérer les différences avec précision, soit au caractère près ; celles-ci sont largement exagérées et l'alignement résultant devient très vite peu lisible.

Classiquement, les différences identifiées entre deux textes sont de trois types : suppressions, insertions et remplacements. On notera qu'un remplacement n'est rien d'autre qu'une suppression suivie d'une insertion, à la même position dans les deux textes. Un autre type de différence peut également être envisagé : les déplacements, soit la suppression d'un bloc de texte suivie de l'insertion de ce même bloc à une position différente. Aucun des aligneurs évalués durant l'étude n'était en mesure de rechercher des déplacements correctement : huit sur les neuf ne proposaient pas cette fonctionnalité, et WinMerge, qui la proposait, se révélait finalement très mauvais. Or il nous semble que cette fonctionnalité puisse être de première importance pour étudier l'évolution d'un corpus textuel.

En effet, le travail présenté dans ces pages est motivé par un besoin propre à la *critique génétique*. Cette discipline littéraire s'intéresse aux brouillons produits par les écrivains lors de la rédaction d'une œuvre (de Biasi, 2000). Ce genre de corpus est constitué d'un ensemble de versions d'un texte liées par une relation de parenté. En comparant ces brouillons, les généticiens du texte cherchent à reconstituer la genèse de l'œuvre sous ses différents aspects, principalement littéraires, linguistiques et cognitifs. Outre les trois opérations classiques de comparaison textuelle, les généticiens du texte recherchent également avec intérêt les déplacements effectués par l'auteur. Ceux-ci mettent en évidence le travail de mise en forme stylistique ou d'agencement des idées.

Effectuer de telles comparaisons manuellement, comme le font habituellement les généticiens du texte, comporte d'importantes difficultés. Il convient de déceler les modifications au caractère près ; celles-ci peuvent révéler la correction de fautes d'or-

thographe par l'auteur, le changement de catégorie morphosyntaxique d'un mot, ou encore la substitution d'un radical par un autre en vue de parfaire le style du texte. C'est là une besogne extrêmement fastidieuse nécessitant de constants allers-retours entre les deux textes avec une attention des plus soutenues. Si en plus on recherche les déplacements, la difficulté s'accroît de façon exponentielle avec la taille de l'empan sur lequel on s'autorise à les rechercher. Finalement, la tâche devient vite insurmontable dès que les textes font plus de quelques pages, et le cas où l'on souhaiterait aligner deux versions d'un livre reste de l'ordre de l'utopie.

Le but de ce travail est de fournir un outil d'alignement monolingue avec déplacements précis et efficace. L'outil que nous avons développé, nommé MÉDITE, était destiné initialement aux généticiens du texte. Il répond au projet de recherche de Jean-Gabriel Ganascia et Jean-Louis Lebrave d'introduction des outils informatiques dans la critique génétique (Ganascia et Lebrave, 2007). Une première version du logiciel fut présentée dans (Ganascia *et al.*, 2004) ; nous présentons ici ses derniers développements. De manière plus générale, ce travail s'inscrit dans la perspective décrite ci-dessus qui montre que l'alignement monolingue est une question importante en informatique et tout particulièrement en TAL.

Nous n'abordons pas ici les aspects linguistiques liés à cette problématique ; quelques références sont disponibles dans la section 5 en conclusion. Nous nous concentrons en effet sur l'aspect informatique. Dans le section 2, nous présentons les travaux apparentés et montrons en quoi la plupart ne répondent pas au besoin exprimé ci-dessus. Nous verrons ainsi pourquoi notre algorithme d'alignement est largement inspiré de ceux utilisés en bioinformatique. La section 3 présente les détails de notre algorithme d'alignement. L'outillage algorithmique déployé, et emprunté à l'algorithmique textuelle, est peu commun pour du TAL, mais c'est là que réside la force de notre méthode et c'est ce qui permet la précision et le passage à l'échelle. Enfin, celle-ci est évaluée grâce à un protocole objectif sur des données artificielles dans la section 4.

2. État de l'art

La croissance exponentielle des données textuelles disponibles sur Internet a considérablement stimulé le domaine de la recherche d'information. En particulier, la recherche de plagiat ou de réutilisation textuelle est devenue une question de première importance. Les techniques d'indexation, de mesure de similarité par cosinus ou par recouvrement de n-grammes, de *fingerprinting* sont communément utilisées (Clough *et al.*, 2002 ; Aizawa, 2003 ; Stein *et al.*, 2007). Elles permettent de détecter des duplications au niveau du document en fournissant une mesure de similarité globale entre deux textes. Ces méthodes ne se concentrent pas sur la recension exhaustive des similarités et différences locales entre deux textes. Elles se révèlent donc incapables d'établir un alignement.

En traduction automatique, l'alignement est largement utilisé. Des aligneurs bilingues, fondés sur des modèles statistiques génératifs, sont entraînés à partir de corpus parallèles ou bi-textes (Brown *et al.*, 1993). Ces derniers sont alignés au niveau des phrases, ce qui permet de collecter des statistiques de co-occurrence lexicale. Entre une phrase et sa traduction, il peut exister des éléments supprimés ou insérés sciemment par le traducteur, sans pour autant nuire à la qualité de la traduction. De même une phrase peut être scindée en deux lors de la traduction. Ainsi, il existe des opérations d'édition locales à une paire de phrases, mais pas d'opérations globales entre deux paires de phrases, comme des déplacements, puisqu'un bi-texte est seulement une liste de paires de phrases et non un texte au sens habituel.

La comparaison de versions avec les trois opérations classiques se résout optimalement par programmation dynamique en calculant la *distance d'édition* : le nombre d'opérations d'édition permettant de transformer un texte en l'autre (Levenshtein, 1966). Ceci requiert un temps quadratique en fonction de la taille des textes, ce qui se révèle très rapidement trop long dès que les textes à aligner dépassent quelques centaines de caractères. De nombreux algorithmes plus rapides que cette borne supérieure, optimaux ou non, ont été proposés (Crochemore *et al.*, 2001) ; `diff` est un de ceux-là.

L'introduction d'une quatrième opération d'édition, le déplacement d'un bloc de caractères, dans le calcul de la distance d'édition a été formalisée dans (Shapira et Storer, 2002) sous le nom de *distance d'édition avec déplacements*. Les auteurs ont démontré que son calcul est NP-difficile. Ils ont également proposé un algorithme d'approximation glouton très simple de ce calcul. Dans la section 4, notre algorithme d'alignement est comparé à celui-ci.

En bioinformatique, les séquences d'ADN ou de protéines sont modélisées par des séquences de caractères définies sur un alphabet de 4 ou 20 caractères respectivement. Aligner de telles séquences permet de découvrir des régularités servant à identifier des gènes sur un génome ou des propriétés d'une protéine. La taille des génomes de mammifères est très importante : par exemple celle du génome humain est de l'ordre de 3 milliards de caractères. Calculer un alignement optimal avec la distance d'édition sur ces séquences est inenvisageable. Des techniques d'alignement heuristique ont ainsi été développées afin d'aligner en un temps raisonnable, soit sous-quadratique, les séquences de grande taille (Gusfield, 1997).

Ces techniques, regroupées sous le nom d'*alignement par fragments* sont toutes fondées sur un stratégie d'alignement à base d'*ancres*. Le principe en est le suivant :

1) recherche d'un ensemble de blocs similaires relativement à un critère de similarité défini ; ces blocs sont appelés *fragments* ;

2) recherche d'un sous-ensemble de ces fragments qui soient non chevauchants deux à deux et alignement de ceux-ci ; ces fragments sont appelés *ancres* ;

3) alignement des sous-séquences comprises entres les ancres en appliquant récursivement les deux premières étapes, ou en utilisant la programmation dynamique, ou encore l'alignement local (Smith et Waterman, 1981) par exemple.

Des structures de données comme les arbres des suffixes permettent d'effectuer la première étape en un temps linéaire en fonction de la taille des séquences (Ukkonen, 1995). Lors de la seconde étape, l'alignement s'effectue alors sur un ensemble d'ancres qui est très inférieur à la taille des séquences. La dernière étape permet de compléter l'alignement.

C'est là, de façon très schématique, le principe sur lequel sont fondés tous les aligneurs qui opèrent sur des génomes de mammifères (Delcher *et al.*, 1999 ; Bray *et al.*, 2003 ; Brudno *et al.*, 2003 ; Darling *et al.*, 2004 ; Abouelhoda, 2005). Tous ces algorithmes ont bien sûr chacun leurs propres raffinements et spécificités. Nous proposons une adaptation de ce principe pour le TAL dans la section 3.

3. Algorithme d'alignement avec recherche des déplacements

Les deux textes que nous souhaitons aligner sont considérés comme deux séquences de caractères A et B définies sur un même alphabet Σ de taille finie. On note les longueurs des deux séquences $m = |A|$ et $n = |B|$.

3.1. *Pré et post-traitements*

Une étape de pré-traitement des séquences A et B peut-être effectuée avant l'alignement proprement dit. Ce pré-traitement établit des classes d'équivalence entre certains caractères afin de permettre des appariements approximatifs lors de l'alignement.

Nous cherchons à aligner des séquences de textes en langue naturelle, et il existe trois classes de caractères que l'on peut vouloir considérer comme équivalents :

1) les majuscules et les minuscules ;

2) les caractères accentués et leur équivalent sans accent ;

3) tous les signes de ponctuation ou séparateurs.

Pour établir ces classes d'équivalence, on convertit : les majuscules en minuscules, les caractères accentués en leur équivalent sans accent, et tous les séparateurs ou ponctuations en un seul signe, à savoir le point. Ces conversions sont effectuées hors-contexte. Pour cela, A et B sont parcourues et chaque caractère est remplacé par le représentant de sa classe d'équivalence. De plus, les séparateurs adjacents sont fusionnés en un seul séparateur. Lors des étapes ultérieures, l'alignement sera alors effectué sur ces séquences modifiées. Finalement, le post-traitement permettra de réindexer l'alignement sur les séquences originales A et B.

Dans les étapes ultérieures, seules des répétitions exactes sont recherchées. Si dans ces répétitions, des caractères avaient en fait été remplacés par le représentant de leur classe d'équivalence, une fois le post-traitement appliqué, on obtient alors des répétitions approximatives. L'intérêt des pré et post-traitements est de permettre des appariements approximatifs pour un coût linéaire en $O(m + n)$.

Exemple 1. Afin d'illustrer notre algorithme d'alignement, l'exemple récurrent suivant est utilisé. Nous cherchons à aligner les deux séquences $A = [Ce\ matin\ le\ chat\ observa\ de\ petits\ oiseaux\ dans\ les\ arbres.]$ et $B = [Le\ chat\ était\ en\ train\ d'observer\ des\ oiseaux\ dans\ les\ petits\ arbres\ ce\ matin.\ Il\ observa\ les\ oiseaux\ pendant\ deux\ heures.]$

Après le pré-traitement avec les trois paramètres d'équivalence entre classes de caractères à vrai, on obtient : $A = [ce.matin.le.chat.observa.de.petits.oiseaux.dans.les.arbres.]$ et $B = [le.chat.etait.en.train.d.observer.des.oiseaux.dans.les.petits.arbres.ce.matin.il.observa.les.oiseaux.pendant.deux.heures.]$. L'alignement sera alors effectué sur ces deux séquences modifiées.

3.2. *Étape 1 : recherche des répétitions*

Entre deux séquences A et B, le nombre de répétitions peut croître de façon exponentielle, si l'on ne prend pas soin de définir précisément la notion de répétition. Or nous avons évoqué dans la section 2 que nous cherchions à aligner un ensemble de répétitions dont la cardinalité soit très inférieure à m et n. Il est donc crucial de contenir cette explosion combinatoire. Pour cela, nous nous basons sur la méthode de recherche des *répétitions exactes super-maximales* exposée dans (Gusfield, 1997), pages 143-148.

Commençons par introduire la notion de répétition exacte maximale.

Définition 1. *Étant donné deux séquences A et B et une répétition exacte entre A et B. Celle-ci est notée $(l, [occ_A, occ_B])$, où l est la longueur de la répétition, et occ_A et occ_B les positions de ses occurrences dans A et B respectivement. Cette répétition est* maximale à gauche *si $A[occ_A - 1] \neq B[occ_B - 1]$. Elle est* maximale à droite *si $A[occ_A + l + 1] \neq B[occ_B + l + 1]$. Finalement, une répétition est une* répétition exacte maximale *si et seulement si elle est à la fois maximale à gauche et à droite.*

Ceci nous permet alors de définir la notion de répétition exacte super-maximale.

Définition 2. *Une répétition exacte maximale est une* répétition exacte supermaximale *si aucune des ses occurrences n'est incluse entièrement dans une autre répétition exacte maximale.*

On utilisera l'acronyme *SMER* pour *Supermaximal Exact Repeat*. Un SMER est représenté par une paire $(longueur, listeOccurrences)$, où $longueur$ est la longueur de la chaîne répétée, et $listeOccurrences$, la liste de ses occurrences indexées sur la concaténation AB des deux séquences A et B.

Le principe de l'étape 1 est de rechercher l'ensemble des SMER existants entre A et B, dont la taille est supérieure à une taille minimale. Pour cela, un arbre des suffixes de A et B, noté $\mathcal{AS}(A,B)$, est construit. L'ensemble des SMER est ensuite identifié en parcourant cet arbre.

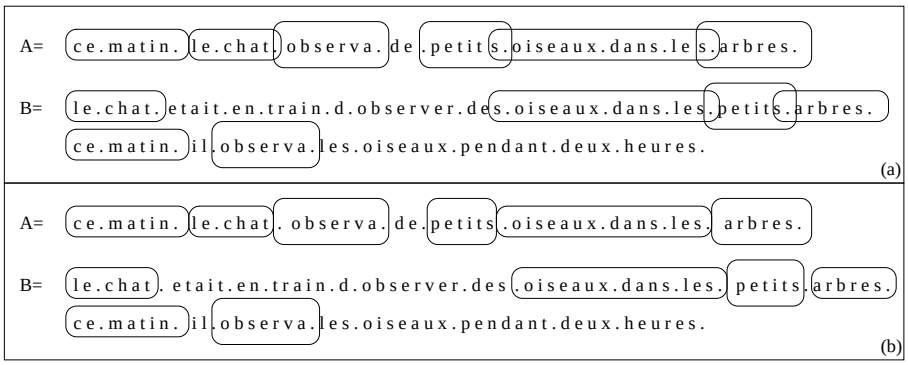

Figure 1. *(a) Ensemble des SMER extraits de A et B après l'étape 1, où l'on constate qu'il existe des recouvrements entre SMER ; (b) Ensemble des NOSMER après l'étape 2.1 d'élimination des recouvrements*

Les arbres des suffixes sont une structure de données recensant l'ensemble des suffixes d'une séquence en espace linéaire et avec un temps de construction linéaire (Weiner, 1973 ; McCreight, 1976 ; Ukkonen, 1995). De plus, ils recensent l'ensemble des répétitions présentes dans une séquence. Pour cela, une fois $\mathcal{AS}(A,B)$ construit, deux parcours de l'arbre sont nécessaires. Lors du premier parcours les répétitions maximales à gauche, c'est-à-dire celles qui ne peuvent être étendues à gauche sans perdre la propriété de répétition entre les deux chaînes, sont identifiées par marquage des nœuds correspondants de l'arbre. Lors du second parcours, le sous-ensemble de ces répétitions qui sont également maximales à droite est identifié. D'après la définition 2, les répétitions respectant ces deux propriétés sont des SMER (Gusfield, 1997).

La taille de $\mathcal{AS}(A,B)$ étant, par définition, linéaire par rapport à la somme de la taille des deux séquences, les deux parcours s'effectuent en $O(m+n)$, d'où une complexité temporelle identique à l'étape 1. De même, la complexité spatiale est celle de $\mathcal{AS}(A,B)$, soit également en $O(m+n)$.

Exemple 2. Poursuivons l'exemple récurrent. Après avoir appliqué l'étape 1 sur les deux séquences A et B on obtient l'ensemble des SMER illustré par la figure 1(a).

3.3. *Étape 2 : extraction des ancres et alignement*

La seconde étape de notre algorithme d'alignement consiste à extraire de l'ensemble des fragments issu de l'étape 1 un sous-ensemble dont les éléments soient non chevauchants, les ancres, et à les aligner. Logiquement cette étape se décompose en deux sous-étapes. La section 3.3.1 détaille l'étape 2.1 d'extraction d'un sous-ensemble de fragments non chevauchants et la section 3.3.2 détaille l'étape 2.2 d'extraction et alignement des ancres.

3.3.1. *Étape 2.1 : Extraction d'un sous-ensemble de fragments non chevauchants*

Afin d'illustrer la nécessité et les difficultés de l'étape 2.1, il convient d'examiner la figure 1(a) qui représente les SMER issus de l'étape 1.

On constate qu'il existe un certain nombre de recouvrements entre eux. Par exemple, $[s.oiseaux.dans.les]$ et $[.petits.]$ se chevauchent. De plus, leurs occurrences ne sont pas colinéaires (voir définition 4) ; ainsi l'un sera considéré comme un invariant et l'autre comme un déplacement lors de l'étape 2.2 (voir section 3.3.2). Or la sous-séquence $[s.]$ commune aux deux SMER ne peut être à la fois invariante et déplacée. Deux solutions sont alors possibles. La première consiste à ne conserver qu'un seul des deux SMER dans l'alignement, par exemple le plus long : on perd alors une information importante pour l'alignement (l'autre SMER), et ainsi sa qualité décroît en fonction du nombre de répétitions. Une seconde solution, celle que nous avons retenue, consiste à éliminer le recouvrement en attribuant celui-ci à l'un des deux SMER et en raccourcissant l'autre. Deux critères d'élimination nous semblent intéressants : favoriser le bloc le plus long afin d'obtenir un alignement plus lisible, et favoriser les raccourcissements sur les séparateurs afin de respecter les mots des textes en langue naturelle.

Résoudre l'ensemble des chevauchements en optimisant de façon globale suivant ces deux critères comporte deux difficultés. Une telle méthode d'optimisation globale est inconnue et il est fort probable que le problème soit NP-difficile. De plus, c'est un problème d'optimisation bi-critère, ce qui introduit une difficulté supplémentaire quant à la question de l'agrégation des deux critères en une fonction de décision (Ehrgott, 2005).

Au vu de ces difficultés, nous proposons un algorithme de résolution des recouvrements à l'aide d'une série de choix locaux. Ceci est une méthode de résolution classique des problèmes difficiles. Quant au choix lié à l'élimination du recouvrement, nous agrégeons les deux critères en favorisant le bloc le plus long, puis en résolvant sur le séparateur le plus proche de la frontière de ce bloc. Ceci nous permet d'obtenir une fonction d'agrégation heuristique simple.

On remarquera qu'en appliquant une telle fonction de décision pour raccourcir les SMER, ceux-ci vont perdre leur propriété de super-maximalité. Ceci permet de gagner la propriété de non-chevauchement entre blocs et c'est ce qui permet également d'aligner les blocs afin d'obtenir des invariants et des déplacements dans l'étape 2.2. Néanmoins la propriété de super-maximalité a déjà été exploitée lors de l'étape 1, sa perte durant l'étape 2.1 est donc peu dommageable.

Définition 3. *Appelons $\mathcal{SMER}$ l'ensemble des SMER entre deux séquences A et B. Nous définissons $\mathcal{NOSMER}$ comme étant l'ensemble des* répétitions exactes sous maximales non chevauchantes, *pour* Non Overlapping Submaximal Exact Repeat, *issu de l'étape 2.1 à partir de $\mathcal{SMER}$.*

Le principe de l'algorithme d'élimination des recouvrements de l'étape 2.1 est alors le suivant. C'est un algorithme glouton de résolution des recouvrements qui par-

court $\mathcal{SMER}$ de façon décroissante, du plus long bloc au plus court, grâce à une file de priorité indexée sur la taille des blocs. Pour chaque occurrence d'un SMER, on vérifie s'il existe un recouvrement avec une occurrence d'un NOSMER existant. Si c'est le cas, une position de résolution du recouvrement sur un séparateur est recherchée. Chaque occurrence du SMER peut avoir un recouvrement différent, il est donc nécessaire de coordonner le raccourcissement de toutes les occurrences du SMER de la même façon. Pour cela, on applique le raccourcissement maximal à gauche et à droite à toutes les occurrences d'un SMER, afin de conserver la propriété de répétition exacte entre toutes les occurrences. Une fois le SMER raccourci, il est placé à nouveau dans la file de priorité, afin de traiter des blocs plus longs auparavant s'il en existe. Si un SMER a au moins deux occurrences sans recouvrement, alors ces deux occurrences forment un NOSMER qui est ajouté à $\mathcal{NOSMER}$; les autres occurrences étant traitées de la façon ci-dessus.

Les détails de l'algorithme sont disponibles dans (Bourdaillet, 2007). Sa complexité temporelle est en $O(m+n+(s+\alpha)\log(s+\alpha))$, où s est le nombre d'occurrences de SMER, et α le nombre de recouvrements entre SMER. Comme on s'attend à ce que $s \ll m,n$ (grâce à la super-maximalité des SMER), cette complexité est proche de $O(m+n)$ et croît avec le nombre de SMER et le nombre de recouvrements. La complexité spatiale est linéaire en $O(m+n+s+\alpha)$.

Exemple 3. Poursuivons l'exemple récurrent. La figure 1(b), page 67, représente l'ensemble des NOSMER entre les séquences A et B à l'issue de l'étape 2.1. Par rapport à l'étape précédente, on constate que tous les recouvrements ont été éliminés.

3.3.2. *Étape 2.2 : extraction et alignement des ancres*

À l'issue de l'étape 2.1, nous disposons d'un ensemble de répétitions qui sont non chevauchantes deux à deux. Ainsi, chacune de ces répétitions peut être incluse dans l'alignement que nous recherchons. De plus, chacune de ces répétitions peut être considérée soit comme un invariant soit comme un déplacement. C'est le rôle de l'étape 2.2 de décider quel type assigner à chaque répétition. Dans la terminologie de l'alignement par fragments, les ancres sont alors équivalentes à nos invariants.

Au niveau des caractères auquel nous avons choisi de nous placer, aucune propriété ne permet de distinguer un invariant d'un déplacement. Ce sont chacun des répétitions exactes. En revanche, la différence prend sens dès lors qu'on les considère les uns en relation avec les autres. Pour ce, introduisons la notion de colinéarité :

Définition 4. *Deux répétitions sont* colinéaires *si le début de l'une est strictement inférieur au début de l'autre dans les deux séquences A et B à aligner.*

Étant donné un ensemble d'invariants, ceux-ci sont nécessairement colinéaires. En effet, par définition, chacune de ces répétitions apparaît dans le même ordre dans les deux séquences. En revanche, si on introduit un déplacement, alors celui-ci est nécessairement non colinéaire à au moins un invariant. En conséquence, si l'on arrive à identifier un ensemble d'invariants à partir d'un ensemble de répétitions non chevauchantes, on identifie également par complémentarité un ensemble de déplacements.

L'étape 2.2 consiste donc à trouver un ensemble de répétitions colinéaires. Reste à définir un critère suivant lequel choisir cet ensemble. Nous proposons de maximiser la somme de la longueur des blocs invariants. En effet, bien que la contribution du présent article réside dans la recherche des déplacements, lorsque l'on souhaite aligner deux textes, le plus important demeure la recherche des invariants.

Pour trouver cet ensemble maximal de blocs invariants, nous proposons d'utiliser une variante de l'algorithme de la recherche d'une plus longue sous-séquence commune (Bergroth *et al.*, 2000). Appelons A' et B' les deux séquences de répétitions issues de l'étape 2.1 et appartenant respectivement à A et B. Une séquence de répétitions colinéaires correspond à une sous-séquence commune de A' et B'. La problématique de cette étape devient donc la recherche d'une plus longue sous-séquence commune à A' et B'. Il existe un grand nombre de variantes de cet algorithme. Nous proposons d'utiliser l'algorithme de calcul de la plus lourde sous-séquence croissante (HIS, pour *Heaviest Increasing Subsequence*) de (Jacobson et Vo, 1992). En effet chaque bloc de A' et B' peut être transformé en un entier *via* une fonction de hachage. De plus, chaque bloc peut être valué par sa longueur. On obtient ainsi deux séquences de blocs identifiés par un entier et un poids, soient les entrées nécessaires au calcul de la HIS.

Le calcul de la HIS s'effectue en $O(r \log r)$ où r représente le nombre d'appariements possibles entre répétitions. Un pré-traitement en $O(s \log s)$ est nécessaire pour extraire A' et B' à partir de $\mathcal{NOSMER}$. La complexité temporelle de l'étape 2.2 est donc en $O(r \log r + s \log s)$. La complexité spatiale est en $O(r + s)$.

Le fait que tous les NOSMER ne faisant pas partie de la séquence de répétitions colinéaires puissent quand même faire partie de l'alignement, est possible uniquement grâce à l'étape 2.1 de résolution des recouvrements, qui assure le non-recouvrement entre NOSMER. Sans cette phase de résolution des recouvrements, deux blocs se recouvrant ne pourraient être inclus dans un même alignement lors de l'étape 2.2 et seul un des deux blocs serait conservé. On retrouve là la première des deux solutions évoquée au début de la section 3.3.1. C'est en fait la solution retenue par les aligneurs par fragments en bioinformatique qui ne passent pas par une telle étape, comme AVID (Bray *et al.*, 2003), Shuffle-LAGAN (Brudno *et al.*, 2003), Mauve (Darling *et al.*, 2004), ou CHAINER (Abouelhoda, 2005). Dans la section 4, ces deux approches sont comparées expérimentalement.

Exemple 4. Poursuivons l'exemple récurrent. À l'issue de l'étape 2.2, on obtient l'alignement partiel illustré par la figure 2.

3.4. *Étape 3 : complétion de l'alignement*

Au sortir de l'étape 2, un ensemble d'invariants et de déplacements existants entre A et B a été identifié. Dans la section 3.4.1, nous montrons que l'alignement peut être amélioré grâce à l'étape 3.1 qui applique récursivement les étapes 1 et 2. Dans la

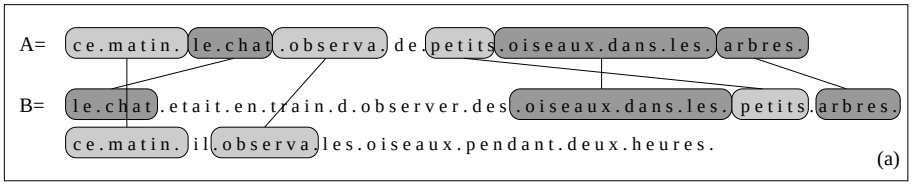

Figure 2. *Alignement des blocs invariants (cadres arrondis sur fond gris foncé) et déplacés (cadres arrondis sur fond gris clair) de A et B après l'étape 2.2*

section 3.4.2 est présentée l'étape 3.2 qui termine la construction de l'alignement en identifiant les suppressions, insertions et remplacements.

3.4.1. *Étape 3.1 : alignement récursif*

Afin d'illustrer l'intérêt d'une étape récursive d'alignement, examinons la figure 3(a) qui présente l'alignement partiel identifié au sortir de l'étape 2. On constate que la répétition $[.observa.]$ a été identifiée comme un déplacement. En examinant les deux sous-séquences comprises entre les blocs invariants $[le.chat]$ et $[.oiseaux.dans.les.]$, on constate qu'il existe une répétition $[.observ]$ dont une occurrence est incluse dans le déplacement $[.observa.]$.

Dans un premier temps, remarquons que comme une occurrence du bloc $[.observ]$ est incluse dans une occurrence de $[.observa.]$, ce bloc ne respecte pas la propriété de super-maximalité requise lors de l'étape 1 pour être sélectionné comme un SMER. C'est la raison pour laquelle $[.observ]$ n'a pas été inclus dans $\mathcal{SMER}$. Dans un second temps, comme nous l'avons déjà mentionné dans la section 3.3.2, rappelons que la qualité essentielle d'un aligneur, fut-il avec déplacements, est d'identifier les invariants. Au vu de cette remarque, il semble plus judicieux de conserver le bloc $[.observ]$ comme un invariant que le bloc $[.observa.]$ comme un déplacement.

C'est là le principe de l'étape 3.1 : en alignant récursivement les paires de sous-séquences de A et B comprises entre deux mêmes paires de blocs invariants, nous sommes en mesure d'identifier de nouveaux invariants. C'est là bien sûr une solution de compromis où ce gain s'effectue au prix de la perte de certains déplacements.

Ainsi les étapes 1 et 2 sont appliquées sur un ensemble de paires de sous-séquences, dont la somme de leur taille est nécessairement inférieure à $m + n$, du fait que les blocs invariants ne sont pas alignés à nouveau récursivement. C'est une approche *diviser pour régner* où la taille des sous-problèmes de chaque niveau de récursion est bornée par $m + n$. Ceci accroît la complexité de notre méthode d'un facteur logarithmique. Ainsi la complexité temporelle devient en $O(T \log T)$ où $T = m + n + (s + \alpha) \log(s + \alpha) + r \log r$, et la complexité spatiale en $O(U \log U)$ où $U = m + n + s + \alpha + r$.

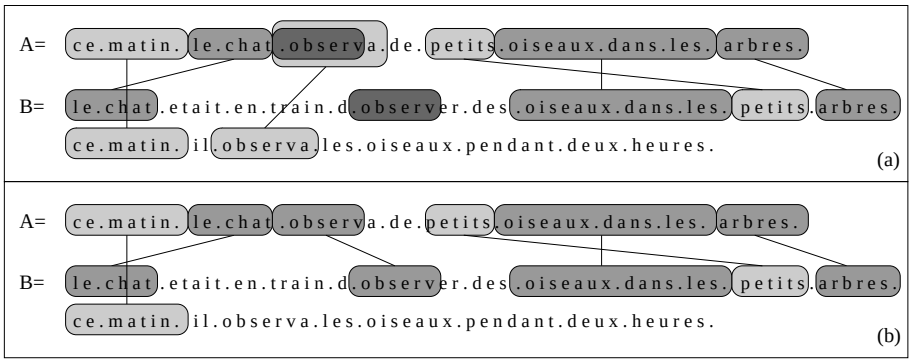

Figure 3. *(a) : alignement avant l'étape récursive 3.1 ; (b) : alignement après l'étape récursive 3.1. Blocs invariants (cadres arrondis sur fond gris foncé) et déplacés (cadres arrondis sur fond gris clair), ainsi que la répétition* $[.observ]$ *(cadres arrondis sur fond gris très foncé)*

Exemple 5. Dans notre exemple récurrent, une fois l'étape 3.1 appliquée l'alignement correspond à celui présenté dans la figure 3(b).

3.4.2. *Étape 3.2 : identification des suppressions, insertions et remplacements*

Finalement, l'étape 3.2 identifie les suppressions, insertions et remplacements existants entre les séquences A et B, ce qui complète l'alignement des deux séquences.

Dans un premier temps, les suppressions et insertions entre A et B sont identifiées. Ceci s'effectue suivant un processus purement déductif. Toutes les sous-séquences de A et B non incluses dans un invariant ou un déplacement ne sont pas des répétitions. Elles sont donc présentes uniquement dans A ou uniquement dans B, et peuvent être considérées comme des suppressions ou des insertions respectivement.

Dans un second temps, on examine, les suppressions et insertions situées entre deux paires de blocs invariants consécutifs. Si le ratio de leur taille est supérieur à un certain seuil (par défaut 0,5), alors on considère que l'on est en présence d'un remplacement et on apparie la suppression et l'insertion pour former un remplacement.

Il n'existe pas de critère *a priori* et simple pour identifier précisément un remplacement qui soit valide au niveau sémantique. Ce manque de critère discriminant nous a amenés à utiliser ce critère heuristique très simple. On peut rapprocher cette heuristique de celle utilisée par Gale et Church en alignement multilingue de phrases (Gale et Church, 1993). Ceux-ci apparient en priorité les phrases ayant une longueur proche.

Exemple 6. Poursuivons l'exemple récurrent. La figure 4 illustre l'étape 6. Après la déduction des insertions et suppressions, on obtient l'alignement illustré en 4(a).

Lors de l'identification des remplacements, les blocs situés entre $[.observ]$ et $[.oiseaux.dans.les.]$, soit la suppression $[a.de.]$ et l'insertion $[er.des]$ sont appariés en un remplacement. En effet, le ratio de leur taille $\frac{5}{6}$ est supérieur au ratio par défaut de 0,5. Après l'identification des remplacements, on obtient l'alignement illustré en 4(b), ce qui correspond à la fin de l'étape 3.2.

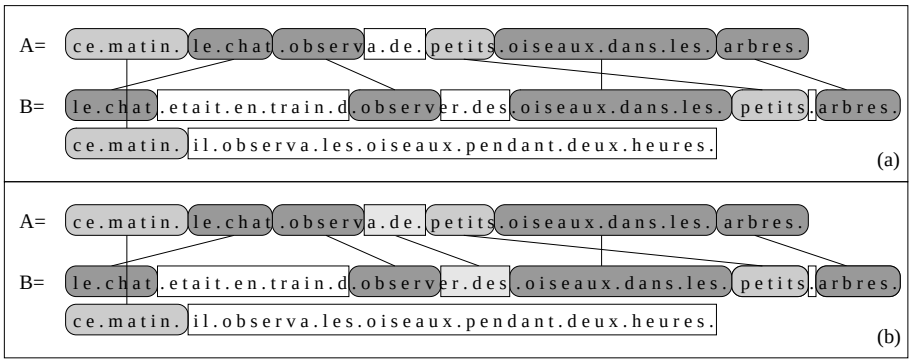

Figure 4. *(a) : alignement après la déduction des suppressions (encadrées sur fond blanc dans A) et des insertions (encadrées sur fond blanc dans B) ; (b) : alignement après l'identification des remplacements (encadrés sur fond gris clair), soit à la fin de l'étape 3.2*

3.5. *Synthèse*

Dans la section 3, nous avons présenté notre méthode d'alignement textuel monolingue avec recherche de déplacements. Cette méthode utilise les techniques issues de l'algorithmique textuelle et de la bioinformatique pour aligner des séquences de textes en langue naturelle. En effet, elle est fondée sur le principe de l'alignement par fragments.

La méthode a une complexité temporelle en $O(T \log T)$ avec $T = m + n + (s + \alpha) \log(s + \alpha) + r \log r$, et une complexité spatiale en $O(U \log U)$ avec $U = m+n+s+\alpha+r$, où m et n sont les tailles des deux séquences à aligner, s le nombre total d'occurrences de répétitions, α le nombre de recouvrements entre répétitions, et r le nombre d'appariements possibles entre répétitions. Comme on s'attend à ce que s,$r \ll m$,n, la complexité s'approche de $O((m + n) \log(m + n))$.

Ainsi, cette complexité autorise le passage à l'échelle, soit l'alignement de séquences de plusieurs centaines de milliers de caractères comme des livres. Cet alignement s'effectue au niveau des caractères, ce qui permet de déceler la moindre modification survenue entre deux versions d'un ouvrage.

De plus, nous avons présenté là, une méthode permettant de rechercher les déplacements à grande échelle pour le TAL. Or, à notre connaissance, il n'existe pas d'aligneur permettant de rechercher les déplacements dans les textes en langue naturelle de façon précise et efficace.

Finalement, on notera que cette méthode est naturellement multilingue. Elle peut s'appliquer à l'alignement dans n'importe quelle langue alphabétique, après quelques adaptations simples liées aux classes d'équivalence définies pour les pré et post-traitements.

4. Évaluation expérimentale

La méthode d'alignement présentée dans la section 3 a été implémentée dans le logiciel MÉDITE. Dans cette section, nous évaluons celui-ci en testant certains choix algorithmiques et en le comparant à d'autres algorithmes d'alignement.

La difficulté majeure liée à l'évaluation de ce travail réside dans le manque de corpus monolingue de référence correspondant à la tâche que nous proposons de résoudre. Un tel corpus devrait contenir un ensemble de versions d'un ou plusieurs textes. Ces textes devraient être d'une taille assez conséquente. Les alignements devraient être étiquetés avec les quatre opérations d'édition, en particulier les déplacements. La précision des alignements devrait être au caractère près. Malheureusement, on ne dispose pas d'un tel corpus monolingue qui aurait été étiqueté par des annotateurs humains.

Pour contourner cette difficulté, nous proposons une évaluation sur des données artificielles telles que présentées dans la section 4.1. Ces données artificielles peuvent ensuite être utilisées pour mener des séries d'expériences visant à l'évaluation du logiciel. Ces expériences sont présentées dans les sections 4.2 à 4.4.

4.1. *Protocole expérimental*

4.1.1. *Données artificielles*

La création des données artificielles repose sur l'utilisation d'un générateur de bruit. Étant donné un premier texte en langue naturelle et un générateur de bruit, l'application du générateur sur ce texte produit un second texte artificiel. Si pendant le processus de génération du bruit, on enregistre les modifications effectuées sur le premier texte, on obtient finalement un alignement de référence entre les deux textes.

On dispose de quatre opérations de modification du premier texte : suppression d'un bloc de texte, insertion d'un bloc, remplacement d'un bloc par un autre, et déplacement d'un bloc d'une position vers une autre. On définit, pour chacune de ces opérations, un taux de modification et on les applique à tour de rôle jusqu'à ce que le taux de chacune soit atteint.

Pour cela, pour chaque opération effectuée sur le premier texte, on choisit, au hasard, pour les insertions, suppressions et remplacements, la position de l'opération dans le texte, et la longueur du bloc concerné (pour les remplacements, la taille des deux blocs reste identique). Pour les déplacements, on choisit la position et la longueur du bloc déplacé, ainsi que la position à laquelle il doit être déplacé. Pour les insertions et les remplacements, on choisit une séquence de mots, de la longueur spécifiée par le tirage aléatoire, dans un dictionnaire du français.

Finalement, on appelle *jeu de données artificielles* le triplet composé du premier texte, du second texte bruité et de leur alignement de référence. En répétant les tirages aléatoires, on obtient les jeux de données artificielles nécessaires.

Nous utilisons deux textes, sur lesquels appliquer le générateur de bruit, de tailles différentes ; ce qui donne deux types de jeux de données artificielles. Le premier est un texte d'environ 40 000 caractères, pour 9 pages. Le second est « *Le Mystère de la Chambre Jaune* » de Gaston Leroux ; il comporte environ 500 000 caractères, pour 350 pages en édition de poche. Pour les séries d'expériences avec le premier jeu de données, on lance 40 expériences afin de calculer les moyennes des différentes mesures, et pour le second on lance 20 expériences. Nous nommerons dorénavant ces jeux de données par leur taille, soit les jeux de données de 40 K et 500 K.

Dans les expériences suivantes, lorsque nous ne présentons pas les résultats avec le jeu de données de 40 K, c'est parce que les résultats sont du même ordre que ceux du jeu de données de 500 K, et n'apportent pas d'information supplémentaire.

Toutes les expériences sont exécutées sur un Pentium 4 Core 2 cadencé à 2,66 GHz et doté de 3 Go de RAM.

4.1.2. *Mesures d'évaluation*

Pour évaluer MÉDITE, des séries d'expériences sont lancées afin de tester les paramètres de la méthode et de comparer celle-ci à d'autres algorithmes.

Pour chaque expérience, un jeu de données artificielles est généré aléatoirement et donné en entrée de l'algorithme à évaluer. Celui-ci aligne les deux textes, et grâce à l'alignement de référence, les mesures de précision et rappel sont calculées au niveau des caractères. On moyenne ensuite sur la série d'expériences.

Pour cela, on considère le problème d'alignement comme un problème de classification, où les invariants, déplacements, remplacements, suppressions et insertions sont cinq classes, avec lesquelles l'algorithme à évaluer doit étiqueter chaque caractère des deux textes à aligner. Ainsi, l'alignement de référence correspond à un ensemble d'étiquettes de référence, nommé $étiq_{ref}$. De même, l'algorithme à évaluer produit un ensemble d'étiquettes, nommé $étiq_{sys}$. Une fois l'alignement produit, et grâce à l'alignement de référence, on peut vérifier si la classe de chaque caractère est correcte, ce qui donne, pour chaque classe, les quantités $étiq_{sys} \cap étiq_{ref}$. On normalise ensuite suivant l'alignement produit par l'algorithme ou suivant l'alignement de

référence, pour obtenir respectivement les mesures de précision et rappel, propres à chaque classe, soit :

$$\text{Précision} = \frac{\acute{e}tiq_{sys} \cap \acute{e}tiq_{ref}}{\acute{e}tiq_{sys}} = \frac{\text{Nombre d'étiquettes correctes}}{\text{Nombre d'étiquettes produites}}$$

$$\text{Rappel} = \frac{\acute{e}tiq_{sys} \cap \acute{e}tiq_{ref}}{\acute{e}tiq_{ref}} = \frac{\text{Nombre d'étiquettes correctes}}{\text{Nombre d'étiquettes correctes existantes}}$$

Finalement, on calcule les scores globaux de précision et rappel d'un alignement en combinant les mesures des cinq classes suivant leur poids respectif, défini *a priori* par le taux de modification du jeu de données artificielles. Par exemple, pour une expérience où 10 % de bruit de chaque type est inséré, on obtient : $Précision = 0{,}6 \times Préc_{INV} + 0{,}1 \times Préc_{DEP} + 0{,}1 \times Préc_{SUP} + 0{,}1 \times Préc_{INS} + 0{,}1 \times Préc_{REMP}$.

4.1.3. *Plan d'expérience*

Les expériences menées sont de deux types : d'une part nous souhaitons évaluer les paramètres de notre méthode, et d'autre part la comparer à d'autres aligneurs.

Par paramètres de notre méthode, nous entendons les choix algorithmiques que nous avons faits dans notre méthode. Nous allons ainsi les comparer à d'autres choix possibles et les évaluer grâce au protocole décrit ci-dessus. Ces choix sont essentiellement concentrés sur les étapes 1 de recherche de répétitions, 2.1 de résolution des recouvrements, 2.2 d'alignement des répétitions, et 3.1 l'étape récursive. Dans les sections 4.2 et 4.3, nous présentons les expériences relatives aux étapes 1 et 2.1 respectivement. Faute de place, nous ne présentons pas les expériences relatives aux étapes 2.2 et 3.1. Celles-ci sont présentées dans (Bourdaillet, 2007).

Les étapes de pré et post-traitement ne présentent pas vraiment de choix. Reste l'étape 3.2 de complétion de l'alignement ; la recherche des insertions et suppressions semble naturelle. La recherche des remplacements est faite suivant l'heuristique décrite précédemment (voir section 3.4.2) ; une recherche plus précise nécessiterait des critères plus linguistiques non intégrés à notre méthode.

Le second type d'expérience consiste à comparer MÉDITE à trois autres algorithmes. Le premier est l'algorithme glouton de calcul de la distance d'édition avec déplacements de (Shapira et Storer, 2002). Nous avons évoqué cet algorithme dans la section 2. Le deuxième est l'algorithme de (Feng et Manmatha, 2006). Nous avons implémenté une version modifiée de cet algorithme prenant en compte les déplacements. Le troisième algorithme est une version modifiée de MÉDITE qui reproduit l'approche la plus utilisée par les aligneurs par fragments, c'est-à-dire sans l'étape 2.1 d'élimination des recouvrements.

Ce dernier algorithme compense le fait qu'il ne nous a pas été possible de nous comparer aux aligneurs par fragments développés en bioinformatique. En effet, ceux-ci ont été conçus afin de comparer des séquences d'ADN ou de protéines, et non des séquences en langue naturelle. Nous en avons testé quatre (MAVID, MUMmer, Shuffle-LAGAN et Mauve), et constaté deux causes d'erreurs : soit les applications

refusent de lire un fichier avec un alphabet non reconnu, soit il se produit une erreur ultérieurement qui est également due à un problème relatif à l'alphabet.

Nous n'avons pas de résultats quantitatifs avec d'autres aligneurs commerciaux, du type de celui intégré dans Microsoft Word. Dans ces applications, l'alignement produit par l'application est disponible uniquement grâce à la visualisation à l'écran. Il serait très fastidieux d'extraire les données d'alignement sous un format traitable automatiquement. Or, cela est nécessaire pour mener des expériences d'évaluation quantitatives suivant le protocole proposé. Toutefois, comme nous l'avons mentionné dans la section 1, nous avons pu constater qualitativement que ce genre d'aligneurs obtenait de mauvais résultats (Bourdaillet et Ganascia, 2006).

4.2. *Évaluation de l'étape 1 de recherche des répétitions*

Dans cette expérience, nous remplaçons l'algorithme de l'étape 1 de recherche des SMER par un algorithme de recherche des n-grammes répétés et présents dans les deux textes. Rappelons qu'un n-gramme est une séquence de n mots consécutifs. L'algorithme recherche les n-grammes répétés au moins une fois dans les deux textes ; ceci s'effectuant en un temps linéaire par rapport à la taille des textes. Nous faisons varier la taille des n-grammes répétés recherchés de 1 à 4, c'est-à-dire que l'on recherche des séquences répétées de 1 à 4 mots.

La figure 5 présente les résultats pour le jeu d'essai de 40 K. On constate qu'en recherchant les SMER plutôt que les n-grammes, les taux de précision sont entre 2 et 10 points supérieurs, et les taux de rappel entre 3 et 42 points supérieurs.

Les courbes de rappel sont très contre-intuitives : les SMER ont de bons résultats, puis les 1-grammes moins, jusqu'aux 4-grammes qui ont les moins bons taux de rappel. En examinant, dans l'interface graphique de MÉDITE, un alignement obtenu avec 60 % de bruit avec la recherche de 4-grammes, on constate immédiatement que l'alignement est très mauvais. Ceci est dû au fait que les textes sont tellement modifiés qu'il existe peu de 4-grammes répétés entre les deux textes ; or ce sont les blocs invariants et déplacés trouvés par l'algorithme. En examinant, les matrices de confusion, on constate qu'en effet, les invariants et déplacements sont très mal classés, ce qui confirme le mauvais alignement. En revanche, en recherchant des 1-grammes, ce problème disparaît puisqu'il reste toujours un taux suffisant de 1-grammes répétés.

La recherche des SMER permet en fait de trouver les répétitions de taille adéquate, ce qui fait chuter le nombre de répétitions. Au contraire, en recherchant des 1-grammes, on trouve un nombre très important de répétitions ; ceci fait augmenter très fortement les paramètres s, le nombre d'occurrences de répétitions, et r, le nombre d'appariements possibles entre répétitions, d'où le temps de calcul beaucoup plus long.

La figure 6 présente les résultats pour le jeu d'essai de 500 K. On constate exactement les mêmes phénomènes, et ce pour les mêmes raisons.

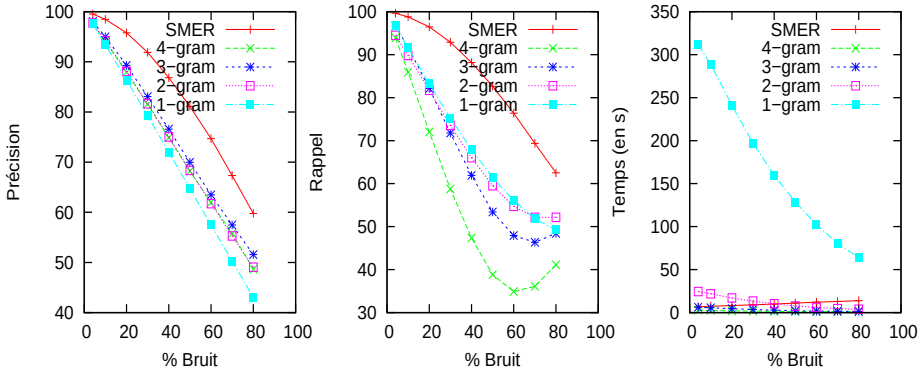

Figure 5. *Test des algorithmes de recherche des répétitions avec le jeu d'essai de 40 K*

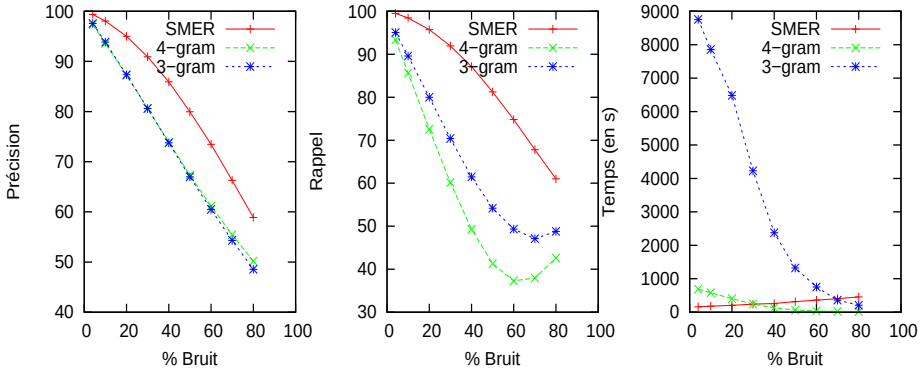

Figure 6. *Test des algorithmes de recherche des répétitions avec le jeu d'essai de 500 K*

Avec les SMER le temps d'exécution croît en fonction du bruit. Quand il y a peu de bruit, la taille des SMER est longue, ce qui réduit leur nombre total et le temps d'alignement ultérieur. Quand le bruit augmente, la taille moyenne des SMER diminue, ce qui fait qu'ils sont plus susceptibles d'être répétés, provoquant ainsi l'augmentation des paramètres s et r.

En revanche, le nombre de 4-grammes diminue en fonction du bruit pour la raison évoquée précédemment ; ainsi, lorsqu'il existe trop de bruit, il n'y a presque plus de blocs à aligner, d'où la rapidité d'exécution, et les mauvais résultats.

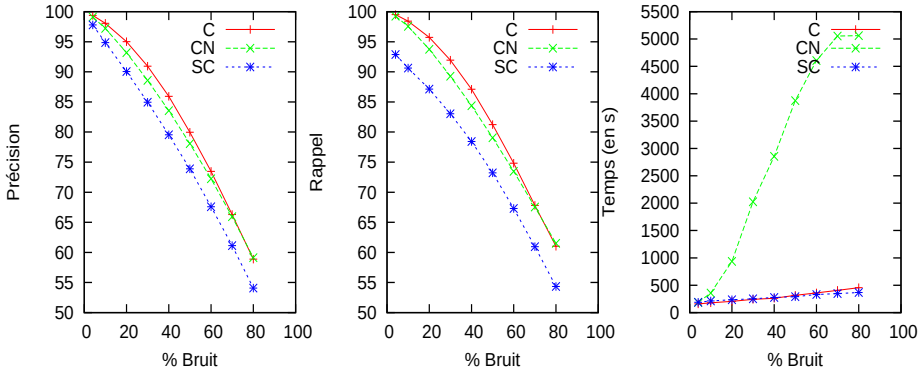

Figure 7. *Test des algorithmes d'élimination des recouvrements avec le jeu d'essai de 500 K*

Pour les 2-grammes, nous avons arrêté l'expérience après environ 8 heures de calcul pour 4 % de bruit.

Cette expérience montre clairement que la recherche des SMER donne de meilleurs taux de précision et rappel, dans tous les cas. D'autre part, les temps d'exécution augmentent en fonction du bruit, du fait de l'augmentation du nombre de SMER. Pour les n-grammes, les résultats sont moins bons, et les temps de calcul prohibitifs.

4.3. *Évaluation de l'étape 2.1 de résolution des recouvrements*

Dans cette expérience, nous comparons notre algorithme d'élimination des recouvrements présenté dans la section 3.3.1, et nommé C dans la figure 7, à deux algorithmes :

– un algorithme d'élimination des recouvrements avec coordination naïve, nommé CN : les recouvrements entre SMER sont tous comparés deux à deux et éliminés s'ils existent ; ce processus est itéré jusqu'à élimination totale des recouvrements ;

– un algorithme d'élimination des recouvrements sans coordination, nommé SC : les recouvrements entre SMER sont examinés suivant leur ordre dans AB ; chaque recouvrement est résolu de façon locale sans coordonner sa résolution avec les autres occurrences du SMER ; ce processus est itéré jusqu'à élimination totale des recouvrements.

La figure 7 présente les résultats pour le jeu d'essai de 500 K. Notre algorithme améliore légèrement les taux de précision et rappel, d'au maximum 2 points, par rapport à la version avec coordination naïve (CN). En revanche, la complexité de

celui-ci ne permet pas le passage à l'échelle, car elle est en $O(m + n + s^2)$ contre $O(m + n + (s + \alpha) \log(s + \alpha))$ pour C.

Par rapport à la méthode sans coordination (SC), notre algorithme améliore la précision de 1 à 5 points et le rappel de 8 à 10 points. Ceci montre l'intérêt de coordonner l'élimination des recouvrements d'un SMER : sans cela, un grand nombre de répétitions sont perdues et la qualité de l'alignement décroît fortement.

Cette expérience montre clairement le gain de l'élimination coordonnée des recouvrements entre SMER par rapport à la méthode sans coordination. De plus, elle justifie l'usage de l'artillerie algorithmique déployée par rapport à une stratégie plus naïve.

4.4. *Comparaison à d'autres aligneurs*

Dans cette section, notre méthode est comparée à trois autres algorithmes.

Le premier est l'algorithme glouton de calcul de la distance d'édition avec déplacements de (Shapira et Storer, 2002). Le principe de l'algorithme est de rechercher la plus longue sous-séquence commune de A et B, de la remplacer par un caractère c n'appartenant pas à Σ, et d'itérer ce processus jusqu'à ce qu'il n'y ait plus de sous-séquence commune entre A et B. La distance d'édition (simple) est alors calculée. Puis, grâce aux appariements de sous-séquences communes identifiés lors de la première étape, les déplacements de blocs sont identifiés, ce qui permet de trouver un alignement de A et B avec déplacements de blocs. Les auteurs démontrent que l'algorithme a une complexité quadratique en $O(mn)$ et qu'il approxime le calcul de la distance d'édition avec déplacements à un facteur en $O(\log n)$ près.

Le deuxième est l'algorithme de (Feng et Manmatha, 2006) qui évalue la performance d'un OCR en comparant les sorties de l'OCR à une version texte existante du livre numérisé par l'OCR. C'est un problème d'alignement monolingue proche du nôtre car les données sont fortement bruitées (même s'il n'y a pas de déplacements) et la problématique de passage à l'échelle est prise en compte. Nous avons implémenté une version modifiée de cet algorithme prenant en compte les déplacements.

Le principe de cet algorithme est proche du principe de l'alignement par fragments. Dans un premier temps, les hapax présents dans les deux séquences sont recherchés. Ils constituent les fragments de l'alignement et correspondent à notre recherche de SMER. Ces hapax sont alignés en utilisant un Modèle de Markov Caché (HMM). Dans un second temps, une étape récursive est appliquée : les séquences de mots entre une même paire d'hapax sont alignées avec un HMM. Enfin, les séquences de caractères situées entre une même paire de mots alignés dans l'étape précédente, sont alignées avec un HMM. Nous avons ajouté une fonction permettant de rechercher les déplacements. Une fois l'alignement effectué, elle vérifie s'il existe des répétitions non colinéaires à la séquence de blocs invariants, auquel cas on les considère comme des déplacements. De plus, comme dans notre algorithme, les insertions et suppressions

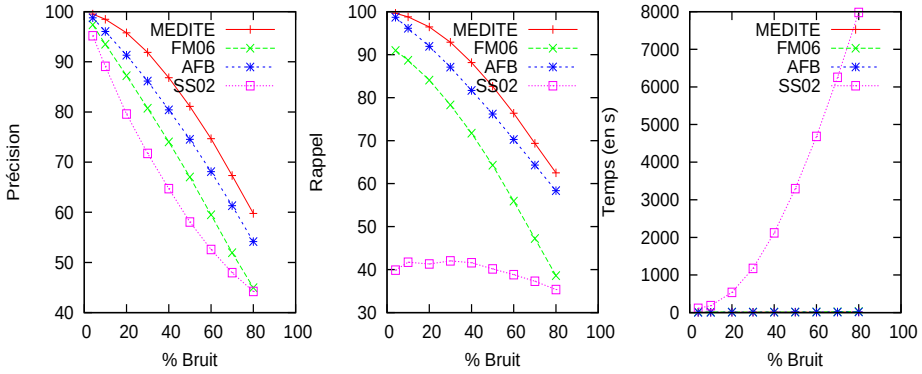

Figure 8. *Comparaison avec d'autres algorithmes avec le jeu d'essai de 40 K*

qui présentent un ratio de leur taille supérieur à un certain seuil sont appariées et deviennent des remplacements.

Le troisième algorithme est désigné par l'acronyme AFB, pour alignement par fragments de base. C'est une version modifiée de notre algorithme qui reproduit l'approche de base de l'alignement par fragments telle qu'utilisée en bioinformatique. Dans cette version, les recouvrements entre SMER ne sont pas résolus par l'étape 2.1 de notre méthode. En conséquence, lors de l'étape 2.2 d'alignement des répétitions, deux répétitions se chevauchant ne peuvent être sélectionnées, ni pour faire partie de la séquence des blocs colinéaires, les invariants, ni de celle des blocs non colinéaires, les déplacements. Cette approche correspond à la sélection de fragments utilisée dans AVID, Shuffle-LAGAN, Mauve, ou CHAINER. AFB a une complexité temporelle en $O(T' \log T')$ avec $T' = m + n + s \log s + r \log r$, et une complexité spatiale en $O(U' \log U')$ avec $U' = m + n + s + r$, ce qui est inférieur à la complexité de MÉDITE.

4.4.1. *Jeu d'essai de 40 K*

La figure 8 présente les résultats de l'expérience pour le jeu d'essai de 40 K.

MÉDITE a une précision entre 2 et 16 points supérieure à AFB, (Feng et Manmatha, 2006) et (Shapira et Storer, 2002). La rappel est entre 1 et 6 points supérieur à AFB, et entre 10 et 24 points supérieur à (Feng et Manmatha, 2006); (Shapira et Storer, 2002) a un mauvais rappel autour de 40.

(Shapira et Storer, 2002) a une complexité en $O(mn)$, d'où ses temps d'exécution très longs. L'algorithme est incapable de passer à l'échelle. Son intérêt est sa simplicité et son approximation de l'erreur du calcul de la distance d'édition avec déplacements.

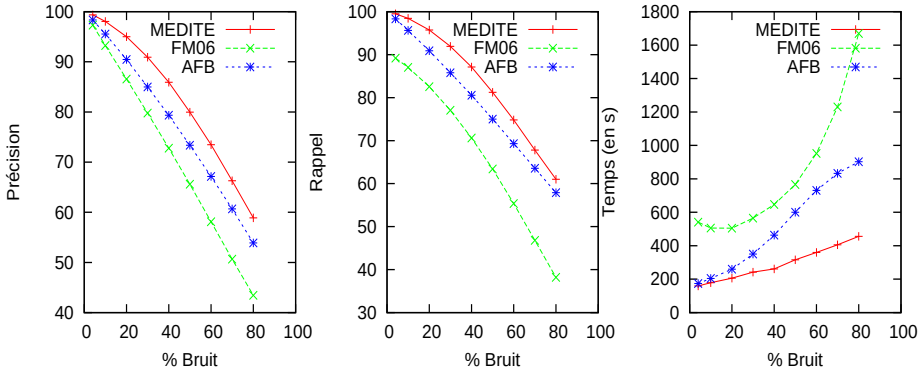

Figure 9. *Comparaison avec d'autres algorithmes avec le jeu d'essai de 500 K*

4.4.2. *Jeu d'essai de 500 K*

La figure 9 présente les résultats de l'expérience pour le jeu d'essai de 500 K.

MÉDITE a une précision entre 2 et 16 points supérieure à AFB (alignement par fragments de base) et entre 2 et 25 points supérieure à (Feng et Manmatha, 2006). Le rappel est entre 2 et 16 points supérieur à AFB et entre 10 et 23 points supérieur à (Feng et Manmatha, 2006). Cela signifie que les alignements produits par MÉDITE sont meilleurs que ceux d'AFB et (Feng et Manmatha, 2006).

La comparaison entre MÉDITE et AFB montre clairement l'intérêt d'inclure l'étape 2.1 de résolution des recouvrements et de ne pas s'être contenté d'une adaptation à l'identique de l'alignement par fragments utilisé en bioinformatique pour le TAL. Bien que la complexité temporelle d'AFB soit meilleure que celle de MÉDITE, les temps d'exécution d'AFB sont moins bons. Comme moins de fragments colinéaires peuvent être sélectionnés comme invariants par AFB, lors de l'étape récursive, les sous-séquences entre deux paires de blocs invariants consécutifs sont plus longues. Ceci a deux conséquences : d'une part les sous-séquences comparées récursivement sont plus longues, et d'autre part, la profondeur des appels récursifs augmente. Ces deux facteurs expliquent l'accroissement des temps d'exécution, malgré la meilleure complexité théorique.

On constate que (Feng et Manmatha, 2006) a un temps d'exécution qui croît de façon non linéaire en fonction du bruit. En effet, quand le bruit augmente, l'alignement au niveau des caractères par le HMM est effectué plus souvent ; ce qui fait croître le temps d'exécution de cet algorithme.

4.4.3. *Discussion*

Cette expérience montre que MÉDITE a de meilleurs résultats que les algorithmes de (Shapira et Storer, 2002), (Feng et Manmatha, 2006) et AFB.

L'algorithme de (Shapira et Storer, 2002) est un algorithme glouton très simple de calcul de la distance d'édition, donc ce résultat est logique au vu de la technique largement plus sophistiquée d'alignement par fragments que nous utilisons. En revanche, c'est un algorithme d'approximation de la distance d'édition avec déplacements à un facteur en $O(\log n)$ près. Au vu des résultats en précision et rappel de cette expérience, et bien que notre algorithme ne calcule pas la distance d'édition avec déplacements, on peut émettre l'hypothèse que notre algorithme obtient une meilleure approximation de la distance d'édition avec déplacements. En revanche, cela semble très difficile à démontrer étant donné le caractère heuristique de notre méthode.

Les résultats meilleurs qu'AFB, la méthode d'alignement par fragments de base, sont plus significatifs. En effet, cela valide notre choix d'inclure l'étape 2.1 de résolution des recouvrements entre SMER. Cette étape améliore non seulement les alignements en termes de précision et rappel, mais accélère également les temps d'exécution.

Enfin, les résultats meilleurs que (Feng et Manmatha, 2006) sont aussi intéressants. D'une part, la technique de (Feng et Manmatha, 2006) est une technique proche de l'alignement par fragments, différente de la nôtre mais avec des objectifs communs : aligner des livres entiers, au niveau des caractères et en un temps raisonnable. En revanche, ils ne recherchent pas les déplacements, et nous avons modifié cet algorithme pour ajouter cette fonctionnalité. Néanmoins, le principe de l'algorithme reste le même. Cet algorithme est donc un algorithme qui traite une problématique proche de la nôtre, et de plus avec une technique relativement proche (excepté l'utilisation des HMM).

D'autre part, Feng et Manmatha ont conçu cet algorithme en tant que chercheurs invités chez Google. On peut supposer qu'il est maintenant utilisé dans le cadre du projet de bibliothèque numérique Google Book. En effet, dans (Feng et Manmatha, 2006), le but des auteurs est d'aligner la sortie de l'OCR d'un livre numérisé et une version électronique de référence du même texte. Ceci nous semble être un gage de la qualité de cet algorithme, et le fait que nous ayons de meilleurs résultats nous semble très intéressant.

5. Conclusion

Dans cet article est présentée la problématique de l'alignement textuel monolingue avec recherche de déplacements. Afin de répondre aux généticiens du texte et face au manque d'applications d'alignement avec déplacements, nous avons proposé l'algorithme d'alignement détaillé ci-dessus et implémenté dans le logiciel MÉDITE. Celui-ci est fondé sur l'approche de l'alignement par fragments développée en bioinformatique. Nous l'avons adaptée pour traiter les textes en langue naturelle. L'évaluation

expérimentale démontre nos meilleurs résultats face à trois autres approches. Elle autorise la recherche de déplacements à grande échelle dans des textes en langue naturelle. Or, à notre connaissance, il n'existe pas d'aligneur permettant de rechercher les déplacements dans les textes en langue naturelle de façon précise et efficace.

Le parti pris très algorithmique de ce travail diffère considérablement des deux grandes approches utilisées généralement en TAL : linguistique et à base d'apprentissage machine. Nous avons cherché à démontrer que cette approche pouvait se révéler fertile. Les travaux d'algorithmique textuelle sont très riches et leurs applications se trouvent majoritairement en bioinformatique. Nous pensons que c'est là un gisement d'idées fort intéressantes pour le TAL.

Le logiciel MÉDITE est maintenant utilisé par les généticiens du texte pour comparer les brouillons d'écrivains. Des études génétiques on été conduites avec son aide (Fenoglio, 2007 ; Fenoglio et Ganascia, 2007 ; Mahrer, 2007). De plus, l'alignement de versions de livres entiers est maintenant possible automatiquement et sans effort. Ceci a permis le développement d'un projet en cours d'édition des œuvres complètes de Charles Ferdinand Ramuz, le grand écrivain Suisse Romand du début du XX^{e} siècle. MÉDITE permet de comparer les différentes versions de chacun de ses romans ; en effet Ramuz a réécrit en grande partie ses ouvrages à chaque réédition. Ce travail automatisé d'établissement des variantes pour des œuvres complètes est une réalisation importante en littérature et dans le monde de l'édition scientifique.

6. Bibliographie

Abouelhoda M. I., Algorithms and a Software System for Comparative Genome Analysis, PhD thesis, Université d'Ulm, Allemagne, 2005.

Aizawa A. N., « Analysis of Source Identified Text Corpora : Exploring the Statistics of the Reused Text and Authorship », *Proceedings of ACL*, p. 383-390, 2003.

Bergroth L., Hakonen H., Raita T., « A Survey of Longest Common Subsequence Algorithms », *Proceedings of SPIRE*, 2000.

Bourdaillet J., Alignement textuel monolingue avec recherche de déplacements : algorithmique pour la critique génétique, PhD thesis, Université Pierre et Marie Curie, Décembre, 2007.

Bourdaillet J., Ganascia J.-G., « MEDITE : A Unilingual Textual Aligner », *Proceedings of FinTAL*, vol. 4139 of *Lecture Notes in Artificial Intelligence*, Springer, p. 458-469, 2006.

Bray N., Dubchak I., Pachter L., « AVID : A Global Alignment Program », *Genome Res.*, vol. 13, n° 1, p. 97-102, 2003.

Brown P. E., Della Pietra V. J., Della Pietra S. A., Mercer R. L., « The Mathematics of Statistical Machine Translation : Parameter Estimation », *Computational Linguistics*, vol. 19, p. 263-311, 1993.

Brudno M., Malde S., Poliakov A., Do C. B., Couronne O., Dubchak I., Batzoglou S., « Glocal Alignment : Finding Rearrangements during Alignment », *Bioinformatics*, vol. 19, p. i54-i62, 2003.

Clough P., Gaizauskas R., Piao S., Wilks Y., « METER : MEasuring TExt Reuse. », *Proceedings of ACL*, p. 152-159, 2002.

Crochemore M., Hancart C., Lecroq T., *Algorithmique du texte*, Vuibert, 2001.

Darling A. C., Mau B., Blattner F. R., Perna N. T., « Mauve : Multiple Alignment of Conserved Genomic Sequence With Rearrangements », *Genome Res.*, vol. 14, n° 7, p. 1394-1403, 2004.

de Biasi P.-M., *La Génétique des Textes*, Nathan Université, 2000.

Delcher A. L., Kasif S., Fleischmann R. D., Peterson J., White O., Salzberg S. L., « Alignment of Whole Genomes », *Nucl. Acids. Res.*, vol. 27, n° 11, p. 2369-2376, 1999.

Ehrgott M., *Multicriteria Optimization. Second edition.*, Springer, Berlin, 2005.

Feng S., Manmatha R., « A Hierarchical, HMM-Based Automatic Evaluation of OCR Accuracy for a Digital Library of Books », *Proceedings of JCDL*, ACM Press, p. 109-118, 2006.

Fenoglio I., « Fête des Chants du Marais, un conte inédit de Pascal Quignard. Genèse in vivo et « traitement de texte » », *Genesis*, vol. 27, p. 73-93, 2007.

Fenoglio I., Ganascia J.-G., « EDITE, un programme pour l'approche comparative de documents de genèse », *Genesis*, vol. 27, p. 166-168, 2007.

Gale W. A., Church K. W., « A Program for Aligning Sentences in Bilingual Corpora », *Computational Linguistics*, vol. 19, n° 1, p. 75-102, 1993.

Ganascia J.-G., Fenoglio I., Lebrave J.-L., « EDITE MEDITE : un logiciel de comparaison de versions », *Actes des JADT*, 2004.

Ganascia J.-G., Lebrave J.-L., « Trente ans de traitements informatiques des manuscrits de genèse », *ACCEDIT*, 2007.

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

Hunt J. W., McIlroy M. D., An Algorithm for Differential File Comparison, Technical Report n° CSTR 41, Bell Laboratories, Murray Hill, NJ, 1976.

Jacobson G., Vo K.-P., « Heaviest Increasing/Common Subsequence Problems », *Proceedings of CPM*, Springer, p. 52-66, 1992.

Levenshtein V. I., « Binary Codes Capable of Correcting Deletions, Insertions and Reversal », *Cybernetics and Control Theory*, vol. 10, n° 8, p. 707-710, 1966.

Mahrer R., « La Génétique Assistée par Ordinateur : MEDITE au banc d'essai ou Du tout neuf pour le Tout-Vieux », *Genesis*, vol. 27, p. 168-172, 2007.

McCreight E. M., « A Space-Economical Suffix Tree Construction Algorithm », *J. ACM*, vol. 23, n° 1, p. 262-272, 1976.

Shapira D., Storer J. A., « Edit Distance with Move Operations. », *Proceedings of CPM*, vol. 2373 of *Lecture Notes in Computer Science*, Springer, p. 85-98, 2002.

Smith T. F., Waterman M. S., « Identification of Common Molecular Subsequences », *Journal of Molecular Biology*, vol. 147, p. 195-197, 1981.

Stein B., zu Eissen S. M., Potthast M., « Strategies for Retrieving Plagiarized Documents », *Proceedings of SIGIR*, p. 825-826, 2007.

Ukkonen E., « On-Line Construction of Suffix Trees », *Algorithmica*, vol. 14, n° 3, p. 249-260, 1995.

Weiner P., « Linear Pattern Matching Algorithms », *Proceedings of the 14th IEEE Symposium on Switching and Automata Theory*, p. 1-11, 1973.