Vers une meilleure utilisabilité des mémoires de traduction, fondée sur un alignement sous-phrastique

Christophe Chenon

UNIVERSITÉ JOSEPH FOURIER – GRENOBLE 1
UFR D'INFORMATIQUE ET DE MATHÉMATIQUES APPLIQUÉES

Thèse présentée
pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER
Discipline : INFORMATIQUE

Christophe Chenon

Vers une meilleure utilisabilité des mémoires de traduction, fondée sur un alignement sous-phrastique

28 octobre 2005

Jury :

Jean **Caelen**	Président
Christian **Boitet**	Directeur
Marc **Dymetman**	Rapporteur
Martin **Rajman**	Rapporteur
Hervé **Blanchon**	Examinateur
Fathi **Debili**	Examinateur
Pr. Zarin	Examinateur

Thèse préparée au sein du GETA, laboratoire CLIPS (IMAG, CNRS et UJF)

Remerciements

Je tiens à remercier en tout premier lieu le professeur Christian Boitet qui m'a accueilli au sein du GETA. Durant les années passées à Grenoble, j'ai eu le privilège d'approcher un homme généreux et clairvoyant, passionné et chaleureux. Il m'a permis de mener à bien ce travail, malgré les difficultés matérielles qui l'accompagnaient. Du fond du cœur je l'en remercie.

Je voudrais aussi dire ma gratitude à Francis Jeux et Jacques Lévy, mes manageurs à IBM pendant cette période. Ils m'ont permis d'effectuer ce travail malgré les difficultés logistiques et administratives que cela a pu occasionner. En outre, ils ont toujours cru en mon succès.

Autour de moi, mes collègues m'ont supporté, dans tous les sens du terme… En premier lieu à IBM, Marie-Thérèse et Hélène, bien sûr, mais aussi Élisabeth (avec un accent sur le E…) et Laurent. À Grenoble, je tiens à remercier Emmanuel, pour son enthousiasme communicatif, et Vincent pour sa patiente relecture. Je voudrais aussi dire ma gratitude à toute l'équipe du GETA pour sa bienveillance et sa solidarité sans faille.

Mes remerciements vont aussi à mes amis qui m'ont accompagné pendant ces années de thèse : d'abord à Karën, présente dès les première heures à Grenoble, et aussi à Pascal à Paris qui m'a donné l'ambition de l'aventure. Et puis aussi à ceux qui m'ont prêté main forte à l'occasion, ou m'ont tenu la dragée haute avec des discussions sans concession : Kai, Gwendal, Sylvie, Lionel…

Je voudrais aussi remercier ma famille, mes parents, ma sœur et mon beau-frère ainsi bien entendu que Bruno.

Table des matières

Table des figures

Table des tableaux

Introduction

Depuis une quinzaine d'années, les environnements d'aide à la traduction humaine utilisant des mémoires de traduction connaissent un succès croissant. Le rendement et la qualité du travail de traduction se sont fortement accrus grâce à l'utilisation de ces outils, qui ne nécessitent qu'un faible investissement en terme d'acquisition de ressources linguistiques, telles que grammaires ou dictionnaires informatisés.

Parallèlement, de nombreux efforts ont été consacrés à la construction de systèmes de traduction automatique à partir de connaissances extraites de corpus bilingues sans recours à d'autres ressources linguistiques. Ces systèmes, pour leur part, n'ont pas encore réussi à dépasser en qualité les systèmes classiques, élaborés depuis plus de cinquante ans, dont le fonctionnement s'appuie sur des règles et des ressources linguistiques établies par des linguistes.

Notre thèse s'articule autour de ces deux évolutions : notre but est de mieux exploiter les mémoires de traduction existantes par la mise en évidence statistique de correspondances sous-phrastiques fines dans les mémoires de traduction vues comme des corpus bilingues alignés. Nous proposons un formalisme TransTree permettant modéliser ces correspondances de manière arborescente. TransTree a été conçu pour être renseigné par voie statistique et décrire une connaissance réutilisable.

Les directions dans lesquelles nous avons envisagé d'utiliser TransTree sont multiples. Nous insisterons sur l'aspect ergonomique de la mise en correspondance de sous-segments pour assister le traducteur dans sa tâche, l'appréciation de la réutilisabilité d'une mémoire de traduction, tant dans le sens source-cible que dans le sens cible-source et éventuellement dans le sens cible-cible pour plusieurs mémoires de traduction vers différentes langues et concernant un même texte source.

Dans un premier temps, nous présenterons le contexte de ce travail, tant industriel qu'universitaire, donnerons quelques repères sur les travaux déjà réalisés dans notre domaine, et détaillerons les objectifs poursuivis.

Notre deuxième partie est consacrée au formalisme d'alignement sous-phrastique TransTree. Après l'avoir défini, nous montrerons comment on peut l'alimenter automatiquement à partir de corpus bilingues, sans autres ressources linguistiques, et discuterons quelques extensions possibles.

Enfin, nous détaillerons quelques expérimentations que nous avons menées autour de TransTree.

1 Position du problème et objectifs

Dans cette partie, nous nous concentrerons sur différents aspects de la traduction humaine assistée par ordinateur. Nous exposerons d'abord les pratiques industrielles actuelles et les directions d'étude privilégiées dans la recherche. Dans un second temps, nous nous pencherons plus particulièrement sur les environnements de traduction destinés à des traducteurs humains et utilisant des mémoires de traduction. Nous verrons alors combien ces mémoires de traduction sont finalement sous-exploitées et nous nous attacherons à dégager les principaux objectifs de cette thèse.

1.1 La TAO dans l'industrie et dans la recherche

Ce chapitre nous fera parcourir différents aspects de la traduction assistée par ordinateur, à travers le contexte dans lequel l'auteur évolue. Nous aborderons successivement l'activité d'un industriel comme IBM, les travaux de recherches menés au GETA et les différentes directions suivies par la recherche contemporaine pour l'exploitation des corpus multilingues.

1.1.1 Contexte industriel

Beaucoup d'idées et de réflexions ont été suggérées à l'auteur par son appartenance au centre de francisation d'IBM France. Une description sommaire du contexte de la traduction industrielle telle qu'elle y est pratiquée est ici proposée.

1.1.1.1 IBM

1.1.1.1.1 Historique

Un très bref historique nous permettra de resituer IBM en tant qu'acteur majeur de l'informatique contemporaine.

Le nom IBM est officiellement né en 1924, mais la société avait été créée dès 1911 par le rachat et le regroupement de sociétés, elles-mêmes fondées à la fin du 19^{e} siècle. Ces sociétés s'étaient déjà spécialisées dans le traitement mécanique de l'information par cartes perforées. Les applications comprenaient, entre autres, le dépouillement des recensements. Mais ces sociétés fabriquaient aussi des instruments de mesure ou des horloges. Thomas J. Watson, qui est considéré comme le véritable père fondateur, est nommé directeur général de la future IBM en 1914 et imprime un fort esprit d'innovation à l'entreprise. Il faut aller vers le traitement de l'information et le calcul, et s'éloigner des machines à écrire… Son fils fera de même après lui.

Si les premiers ordinateurs à relais datent de la fin des années 40, les lampes à vide arrivent vers le milieu des années 50 et les transistors avant même 1960. Le premier

ordinateur modulaire, avec plusieurs périphériques, date de 1964. Dès le milieu des années 50, IBM est présente un peu partout dans le monde et la plupart des innovations informatiques sont liées à IBM jusque dans les années 70. À cette époque, avec l'arrivée de l'informatique individuelle, IBM a dû faire face à une concurrence plus importante, tout en restant la plus grosse entreprise de son secteur.

1.1.1.1.2 IBM en France et dans le monde

Aujourd'hui, IBM compte environ 329.000 employés dans le monde pour un chiffre d'affaires mondial s'élevant à 89 milliards de dollars américains.

IBM France, qui a célébré ses 90 ans en 2004, compte environ 12 500 employés. Son chiffre d'affaires s'élevait à 5,2 milliards d'euros en 2002.

Rappelons pour l'anecdote que c'est grâce à IBM qu'a été créé le mot français « ordinateur », si l'on en croit Françoise Holtz-Bonneau dans « *L'image et l'ordinateur* ». Ce serait un néologisme proposé en 1956 par le latiniste français Jacques Perret à la demande d'IBM.

1.1.1.1.3 Positionnement croissant vers les services et l'informatique distribuée

Ces dernières années, l'activité d'IBM s'est concentrée vers les services. La création d'IBM Global Services a sorti les services de la logique d'accompagnement à la vente du matériel qui prévalait auparavant. Le conseil, l'intégration de services, l'infogérance et le financement sont au cœur des activités d'IBM Global Services, le plus gros acteur de ce marché au niveau mondial.

Il est devenu clair dans le milieu des années 1990 que le modèle client/serveur, centré autour des PC, avait fait son temps, et qu'il cédait la place aux systèmes distribués centrés sur les réseaux. Cela a eu pour conséquence que la charge s'est réorientée vers les infrastructures, tant vers les gros systèmes de stockage et de gestion de transactions que vers les éléments d'informatique diffuse. De ce fait, les notions d'interopérabilité et de standardisation sont montées en puissance.

1.1.1.2 L'activité de traduction à IBM

1.1.1.2.1 De gros volumes dans un domaine technique très spécialisé

L'activité de traduction au sein de la compagnie IBM se caractérise en particulier par son volume très important : pour la France, il s'agit d'environ vingt millions de mots anglais traduits par an.

Les textes traduits à IBM sont essentiellement techniques, et concernent principalement les produits de l'entreprise ou éventuellement d'entreprises partenaires ou clientes.On trouve aussi d'autres types de textes, par exemple juridiques et promotionnels. Les logiciels sont également traduits, interface utilisateur d'une part et aide en ligne d'autre part. La traduction de logiciels représente environ la moitié de l'effort de traduction d'IBM France.

Ces différents types de traduction sont très spécialisés. La terminologie utilisée est à la fois large et précise, en constante évolution. Elle naît le plus souvent aux Etats-Unis et doit être adaptée dans toutes les langues dans lesquelles IBM commercialise ses produits. La gestion multilingue de la terminologie est assurée par de nombreuses équipes spécialisées, coopérant à travers le monde. En France, la base terminologique bilingue contient 35000 entrées. Depuis l'été 2004, une base de données terminologique multilingue compte environs

20 000 termes anglais. La proportion traduite est variable en fonction des langues et des acteurs. Actuellement, la langue la mieux représentée dans cette base après l'anglais est le français avec la moitié des entrées traduites, suivie du hongrois, de l'allemand, de l'espagnol etc.

1.1.1.2.2 Cycle de vie des documents et redondance

L'effervescence industrielle du secteur informatique amène les produits à évoluer très rapidement. Les versions successives de ces produits se suivent dans des délais très courts, pas plus d'un an en général, éventuellement quelques semaines. Les textes liés à ces produits sont largement remaniés à chaque version, mais restent fortement redondants par rapport à la version antérieure.

Cette redondance s'observe en particulier dans les différents textes traitant des mêmes produits ou de produits proches et d'une manière générale du fait de la forte spécialisation de l'industrie de l'informatique.

De plus, cette redondance n'est pas très dispersée : pour un document donné, des passages importants peuvent être totalement repris d'une version à la suivante, et d'autres fortement transformés. On observe donc des îlots de stabilité au sein des textes, à travers les versions successives.

1.1.1.2.3 Traduction fortement multicible

Remarquons aussi que cette activité de traduction est fortement monosource et multicible : les traductions s'effectuent le plus souvent de l'anglais vers une autre langue : La liste qui suit n'est pas limitative :

Allemand, arabe, bulgare, catalan, chinois, coréen, danois, espagnol, français, grec, hébreux, hongrois, italien, japonais, néerlandais, norvégien, finnois, polonais, portugais/brésilien, roumain, russe, serbo-croate, suédois, thaï, tchèque, turc…

À cette liste, il convient d'ajouter certaines nuances : il existe deux centres de traduction pour le chinois : l'un en Chine Populaire, traduisant vers les caractères simplifiés et l'autre à Taiwan, utilisant les caractères traditionnels. Les terminologies utilisées dans ces deux contextes chinois sont également différentes, tout comme il existe des différences terminologiques qui justifient l'existence de traductions spécifiquement québécoise ou brésilienne etc.

L'activité de traduction à IBM est répartie dans différents centres de traduction, certains traitant la langue de leur pays d'appartenance, d'autres servant de pôles pour plusieurs langues, en fonction de leur implantation géographique et des volumes de marché associés.

Ce point est important dans la gestion de la terminologie par exemple : l'anglais sert le plus souvent de référence commune, non seulement dans la définition des termes, mais aussi dans la gestion de projet. Toutes les traductions s'effectuent en même temps dans le monde entier, à partir d'un même volume de texte anglais. Une conséquence est que les échanges d'information, les questions et les réponses, ne sont pas seulement axés vers l'anglais : les équipes travaillant sur les mêmes projets au même moment communiquent entre elles depuis les différent centres de traduction.

Si le nombre de langues cible s'élève à une vingtaine pour l'Europe, le volume de traduction en sens inverse varie en fonction des langues et ne représente pas plus de 5% du volume total. Les traductions n'incluant pas l'anglais sont extrêmement rares.

1.1.1.3 Outils utilisés, automatisation

1.1.1.3.1 Traduction Humaine Assistée par la Machine – THAM

Les systèmes d'aide à la traduction fondée sur la mémoire forment un ensemble d'outils informatiques visant à faciliter la réutilisation de traductions existantes. Idéalement, si ce type d'outil est utilisé, le même texte n'est jamais activement retraduit.

La forte redondance entre deux versions successives d'un document fait de ce contexte industriel le terrain privilégié de ce type d'outil. L'outil utilisé à IBM s'appelle Translation Manager, ou TM (anciennement TM/2 car il a été conçu pour la plateforme OS/2).

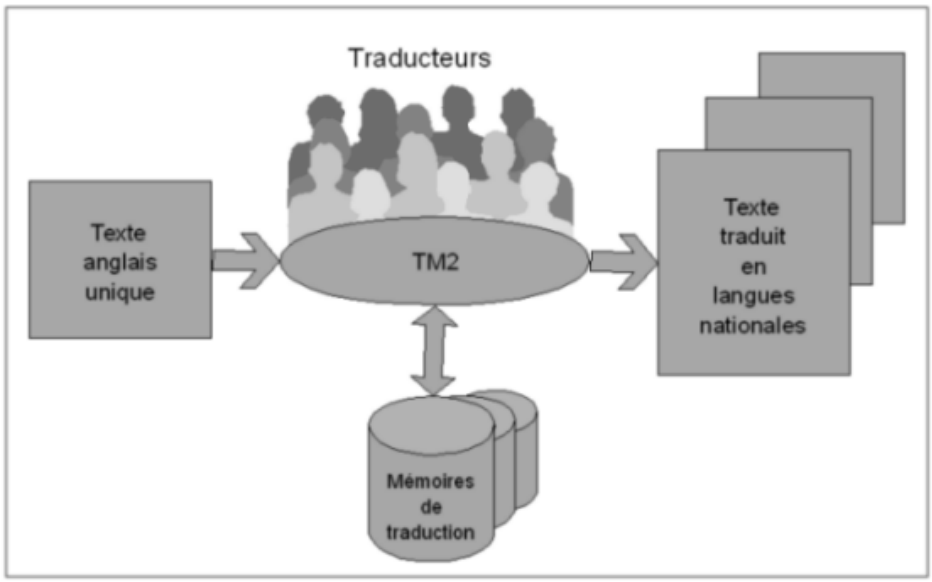

Figure 1: flot des travaux de traduction chez IBM

En outre, la possibilité d'exploiter les mémoires de traduction en réseau permet la réutilisation quasi-immédiate par un traducteur d'un segment nouvellement traduit par un de ses collègues.

La figure suivante donne un aperçu de l'environnement de travail du traducteur dans Translation Manager. Quatre sous fenêtres sont présentes :

1. le texte à traduire, dans lequel les segments sont séparés par un interligne. Le segment en court de traduction est grisé,
2. les propositions trouvées dans la mémoire de traduction soit plusieurs coïncidence floues (fuzzy matches) en général,
3. les segments sources correspondant à ces propositions,
4. les entrées du dictionnaire des termes importants du segment à traduire, repérés automatiquement par le système.

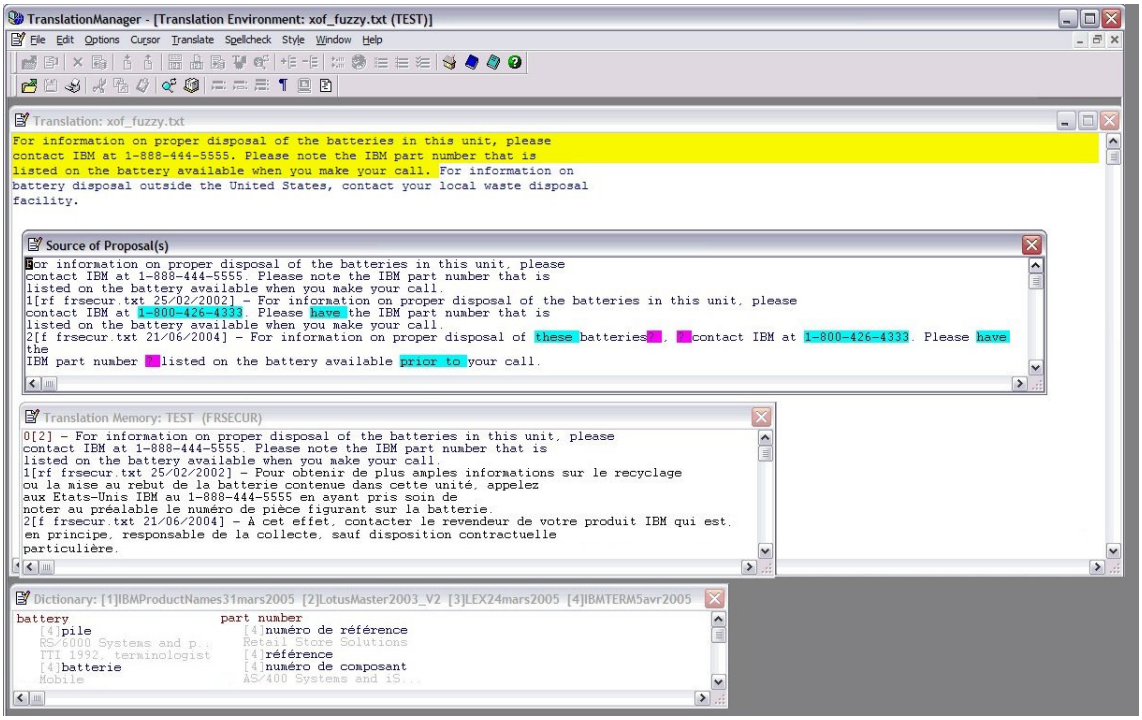

Figure 2: Translation Manager

En tant qu'environnement d'aide à la traduction humaine, TM offre en outre un système de gestion de dossiers.

1.1.1.3.2 Outils terminologiqus

La maîtrise de la terminologie employée est essentielle dans cet environnement, en particulier pour garantir la cohérence entre les traductions de l'interface utilisateur et l'aide en ligne, et aussi avec les autres types de documentation liés au logiciel. Elle est également nécessaire pour garantir la cohérence des documents entre deux versions successives, et d'une manière générale l'homogénéité globale de tous les documents produits.

TM contient un module de recherche automatique (dite « autolookup ») en dictionnaire, activable en cours de traduction. Cette terminologie immédiatement accessible au traducteur peut être mise à jour au cours de la traduction, mais elle est gérée avec un outil extérieur à TM, TransLexis.

TransLexis permet de maintenir en temps réel une base de terminologie centralisée. Un format SGML d'échange de données entre TransLexis et TM permet, d'une part, d'intégrer la terminologie découverte en cours de traduction dans la base gérée par TransLexis, et, inversement, de charger dans TM des dictionnaires extraits de la base TransLexis de façon ciblée.

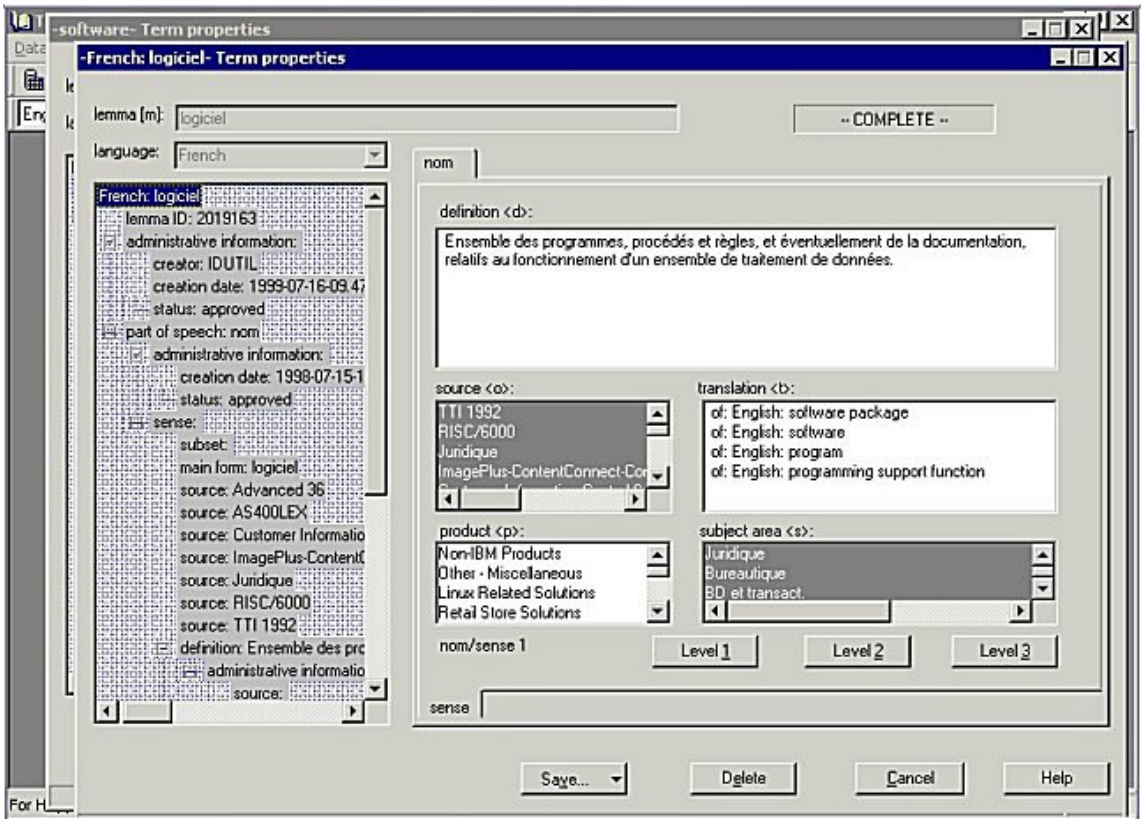

Figure 3 : l'entrée du terme « logiciel » dans TransLexis

Le modèle de données de TransLexis est beaucoup plus complet que celui de TM. Il met en œuvre, en particulier, des liens sémantiques élaborés. Il s'appuie sur le système de gestion de base de données relationnelle DB2.

1.1.1.3.3 Traduction automatique – TA – et traduction assistée par l'ordinateur – TAO

La traduction automatique n'est pas encore fortement utilisée à IBM. Des tentatives d'utilisation existent, néanmoins, notamment sur Internet pour traduire des pages Web à la volée. La traduction automatique conçue comme premier jet pour assister un traducteur dans sa tâche[1], est également utilisée, mais de manière très réduite pour le moment. À IBM, cette approche encore nouvelle est limitée à quelques langues.

On constate plutôt une sous-utilisation des systèmes de traduction automatique dans ce contexte. En revanche, il est fait une plus grande utilisation de ces outils chez des acteurs plus modestes, en particulier dans les petits bureaux de traduction.

On peut avancer diverses raisons pour expliquer cet état de choses. Premièrement, il semble acquis que dans l'état actuel de la technique, les systèmes à mémoires de traduction sont plus performants en termes de rentabilité que les systèmes d'aide utilisant la traduction automatique. Ces systèmes à mémoires de traduction, jouant sur la redondance des documents et sur la taille des mémoires utilisées, se montrent beaucoup plus pertinents dans la tâche qui leur est assignée.

[1] Il s'agit donc d'une approche de TA « du traducteur ». Voir plus bas.

Cela explique *a contrario* pourquoi les bureaux de traduction utilisent moins les mémoires : ils ne disposent pas, en général, de mémoire de traduction, dont les droits sont jalousement conservés par les propriétaires des textes traduits, et travaillent généralement sur des textes très variés, n'offrant pas la même redondance que celle que l'on trouve chez un gros acteur, propriétaire de ses textes. En outre, un bureau de traduction peut maintenir son système de traduction et notamment sa propre terminologie avec plus de souplesse. Généralement, le système de traduction automatique peut utiliser une terminologie *ad hoc* pour un texte donné, et l'utilisateur a la maîtrise directe de la configuration de son outil.

On constate cependant que les gros acteurs, pas seulement IBM, n'utilisent pas ou n'utilisent que peu les services de la traduction automatique, comme complément aux systèmes à mémoires. Il semble qu'à IBM comme ailleurs, la centralisation de la conception des logiciels de traduction automatique soit un frein à la souplesse de leur utilisation. Une grande rigidité des procédures, alliée à une communication souvent réduite pour ne pas dire inexistante entre concepteurs, techniciens et utilisateurs, explique pourquoi la traduction automatique n'est pas utilisée avec profit dans ce type de contexte.

On trouve, cependant, des situations différentes, liées à des contextes particuliers. Ainsi, la Pan American Health Organization utilise son propre système de traduction automatique de l'anglais vers l'espagnol et inversement[2], dont le résultat est rapidement relu avant diffusion[3]. Ce résultat est possible grâce au fort typage de la rédaction en langue source et à la collaboration étroite entre développeurs et utilisateurs.

1.1.2 Contexte universitaire

Notre travail de thèse s'est déroulé au sein du Groupe d'Étude pour la Traduction Automatique du laboratoire CLIPS (Communication Langagière et Interaction Personne-Système) qui appartient lui-même à la fédération IMAG (Institut d'Informatique et Mathématiques Appliquées de Grenoble).

1.1.2.1 Le GETA

1.1.2.1.1 Historique

Les origines du GETA, Groupe d'Étude pour la Traduction Automatique, remontent au début des années 1960 sous l'égide du professeur Bernard Vauquois. Pour la première fois au monde, un système de traduction automatique russe-français articulé sur un langage pivot hybride est implémenté et testé sur de gros volumes de textes (600 000 mots, 2400 pages) entre 1967 et 1970.

1.1.2.1.2 Traduction automatique et Ariane

L'un des instruments d'investigation majeur du GETA aujourd'hui est le système ARIANE-G5, un environnement de programmation destiné aux linguistes créateurs de modèles de traduction automatique et aux traducteurs-réviseurs de textes produits.

[2] EngSpan et SpanAm.

[3] Il s'agit donc bien de la traduction automatique dite « du réviseur », quasi-immédiatement exploitable, et non pas de la traduction automatique dite « du traducteur », conçue comme une aide au traducteur.

Il repose sur cinq « langages spécialisés pour la programmation linguistique » (LSPL), dans lesquels sont écrits des « linguiciels » (dictionnaires, grammaires, automates) ainsi que sur un moniteur interactif lié à un SGBD spécialisé et à un système de traitement de textes.

1.1.2.1.3 Traitement du multilinguisme

L'orientation actuelle du GETA est liée au multilinguisme et aux langages pivot, avec les projets UNL en TA de l'écrit et le projet Papillon de dictionnaire multilingue coopératif, sans oublier CSTAR (avec le projet européen Nespole! de 2000/2003) en TA de l'oral.

Les recherches en cours s'articulent autour de thèmes à dominante informatique, linguistique, et ergonomique, avec en particulier des études sur les techniques de désambiguïsation interactive, sur les bases lexicales multilingues par acceptions, sur l'intégration de possibilités d'apprentissage assisté par ordinateur (AAO), pour la découverte individuelle de connaissances linguistiques, et sur l'expérimentation de techniques modernes de génie logiciel pour (re)construire des outils informatiques (langages et environnements spécialisés pour linguistes et lexicographes) adaptés aux différents types de TAO, et en particulier à la TAO individuelle du futur.

1.1.2.2 Projets en cours au GETA

1.1.2.2.1 UNL

Le « langage UNL » est un langage informatique permettant de coder une représentation du sens d'un énoncé. On se sert de cette représentation comme d'un pivot interlingue. Le langage UNL n'est pas une langue : un hyper-graphe UNL est une structure abstraite d'un énoncé anglais, les symboles lexicaux ou « universal words » (UW) dénotant des acceptions (sens des mots) ou des ensembles d'acceptions interlingues. Il est assez « rustique » pour être utilisé directement par un non spécialiste.

La démarche d'UNL est parfaitement multilingue. En se basant sur une représentation abstraite dont on se sert comme d'un pivot interlingue, toutes les langues sont considérées sur un même pied d'égalité. Développer un outil de TA consiste à établir le lien (dans les deux sens) entre une langue et cette représentation pivot adoptée par l'ensemble des partenaires.

UNL est conçu pour être diffusé librement sur Internet.

Figure 4 : exemple de graphe UNL avec plusieurs formes déconverties

1.1.2.2.2 Nespole ! et CSTAR, TAO de parole

Le projet Nespole! vise à l'interaction multimodale, c'est-à-dire comprenant son et images, entre acteurs dans une transaction de type commercial. La participation du GETA s'articule autour de la traduction de dialogues oraux.

L'un des aspects intéressants de ce projet est qu'il intègre la modélisation du dialogue, et la notion d'acte de langage. On s'intéresse plus à modéliser l'intention du locuteur et la fonction des différents tours de parole qu'à traduire *verbatim* les propos échangés. Par exemple si le locuteur A dit : « good morning, how are you today?» la représentation dans le langage pivot pourra être simplement **A:greetings(time=morning).** Ce projet s'est terminé au printemps 2003.

1.1.2.2.3 Papillon

Le projet Papillon vise à créer un environnement coopératif permanent pour le développement et la consultation libre et personnalisable d'une base lexicale multilingue sur la Toile. Il met en œuvre un serveur de construction coopérative, où chaque contributeur a son espace propre, de sorte que ses contributions puissent être validées et intégrées dans la base par un groupe d'experts. Il s'agit notamment de fournir un outil et des méthodes génériques de fabrication de ressources lexicales riches.

Les données monolingues principales sont des « lexies » (sens de mots) au format DiCo, simplification du DEC par A. Polguère et I. Mel'cuk (Université de Montréal), très riche et systématique. Ces lexies sont reliées par des liens interlingues (axies). On ne se borne pas à générer des lexiques terminologiques multilingues, par nature symétriques, mais aussi et surtout des dictionnaires d'usage bilingues ou multicibles. Les exemples, citations, gloses, définitions, etc., relatifs à une langue donnée, constituent des lexies particulières qui ont vocation à être traduites dans les autres langues.

Des développements importants sont prévus pour la personnalisation des environnements de consultation et de contribution. Les travaux informatiques sont réalisés à Grenoble et Tokyo, le travail linguistique et lexicographique est réparti en France, au Canada, au Japon, en Australie et en Thaïlande.

Vincent berment a fait sa thèse [1] sur l'accélération de la création de ressources et d'outils pour des langues peu traitées, par réutilisation coopérative, avec application au lao.

1.1.2.3 Autres travaux menés au GETA concernant notre sujet

1.1.2.3.1 TELA, les mémoires à étages

Dans le but d'améliorer l'utilisation des mémoires de traduction, Emmanuel Planas a proposé dans sa thèse [2] un formalisme de représentation « à étages » des mémoires de traduction : TELA. La figure suivante en donne une illustration.

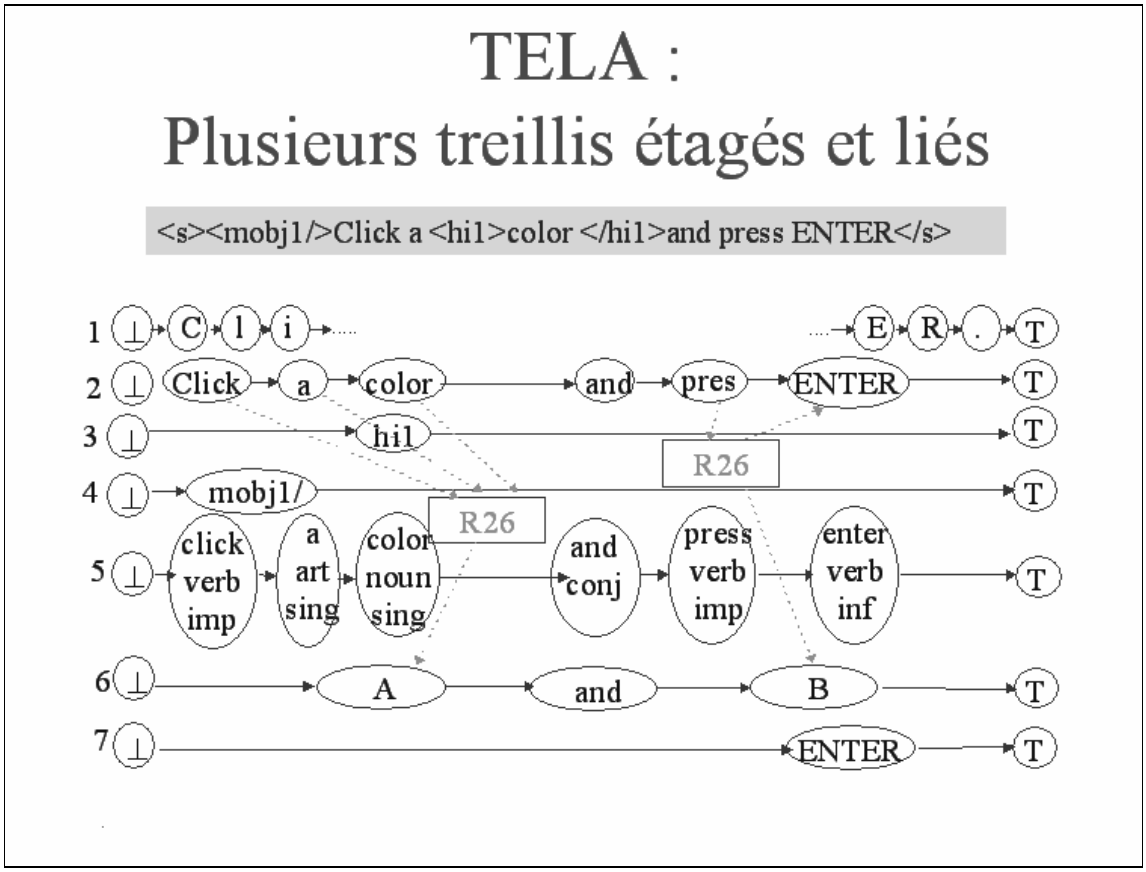

Figure 5 : vue de 7 étages TELA

Chaque étage est un treillis, dont les nœuds correspondent aux objets à représenter pour cet étage. La notion de séquentialité est donc conservée et permet de voir les nœuds d'un chemin comme autant de successeurs. Les étages sont caractérisés par la profondeur d'analyse qu'ils représentent (caractères et entités, balises, texte comme suite de formes, lemmes, termes et fragments fréquents).

La correspondance entre les nœuds de chemins différents d'un même treillis ou entre deux treillis est donnée par des échelles linéaires. Quand il s'agit de relations complexes, comme par exemple la correspondance entre deux mots discontinus et un lemme, les liaisons permettent de les spécifier. L'ensemble des liaisons définit alors une relation de « correspondance » sur l'ensemble des treillis.

La structuration des données et l'emploi d'une distance de type distance d'édition permet d'affiner les algorithmes de recherche de segment, de transférer des formats et des objets non linguistiques en général, ainsi que de produire des propositions de traduction par substitution.

1.1.2.3.2 Informatisation de langues peu dotées

Ce projet s'intéresse à la création de ressources linguistiques informatiques pour les langues qui n'ont pas encore fait l'objet d'investissement « poussé » dans ce domaine. Seul 1% des langues actuellement en usage dans le monde est convenablement informatisé, qu'il s'agisse d'aspects de surface de l'écrit (polices de caractères, gestion de la mise en page, etc.), d'aspect de moyen niveau (correction orthographique, dictée automatique, etc.) ou *a fortiori* de ressources plus complexes telles que celles nécessaires à la traduction automatique.

1.1.2.3.3 Coédition

Ce projet se focalise sur la possibilité de coédition de textes et de représentations abstraites comme les graphes UNL. Le principe est non pas d'éditer le texte, mais de l'annoter en indiquant des modifications à lui apporter (par exemple, mettre "cheval" au pluriel dans "un cheval galope"), et de répercuter cette annotation comme une modification effective sur sa forme abstraite interlingue, exprimée par un graphe UNL. En "déconvertissant" le graphe modifié dans la langue de coédition, les corrections sont réalisées (par exemple, "des chevaux galopent"). D'autre part, si on déconvertit le graphe dans d'autres langues, ces modifications se répercutent sur les versions de ce texte rédigées dans d'autres langues (de façon adaptée à chaque langue).

Ainsi, différents relecteurs/correcteurs travaillant dans différentes langues pourront-ils coopérer et éventuellement parvenir à une représentation exacte et plus fine du sens de l'énoncé. L'innovation majeure de cette approche est de permettre le "partage de la révision" entre plusieurs langues cibles, chose impossible jusqu'alors.

1.1.3 Exploitation de corpus multilingues

Dans cette section, nous parcourons différents travaux liés à l'exploitation de corpus bilingues et plus généralement multilingues. Il s'agit de travaux qui nous ont particulièrement intéressé. Ce chapitre ne prétend aucunement à l'exhaustivité mais permet de mieux apprécier une certaine facette des recherches en cours dans l'activité de traduction automatique et automatisée.

1.1.3.1 Corpus multilingues

1.1.3.1.1 Différents types de corpus

On peut considérer que le texte devient corpus quand il est vu comme un matériau, objet d'étude. Les buts et les critères qui président à la constitution d'un bon corpus sont nombreux et divers. Citons de manière non limitative :

L'exemplarité, c'est-à-dire la représentativité d'un phénomène général que l'on cherche à étudier sur des échantillons,

L'exhaustivité, i.e. la couverture de l'ensemble du sujet d'étude, par exemple l'analyse de l'œuvre d'un auteur ou l'ensemble des textes connus dans une certaine langue, etc.

L'homogénéité de surface est une caractéristique non négligeable d'un corpus quand il est destiné à être traité automatiquement.

Même si la statistique constitue l'un des outils clés dans ce type d'étude, l'exploitation de corpus pour étayer des recherches linguistiques n'est pas nouvelle. Elle précède de beaucoup l'avènement de l'informatique. Des concordanciers permettent depuis longtemps de classer et d'analyser contextes d'apparition et constructions des unités linguistiques. Young, concurrent de Champollion, établit des statistiques sur du matériau textuel recueilli en Égypte et parvint à des résultats très pertinents concernant l'interprétation des hiéroglyphes dès le début du XIX^{e} siècle [3].

De même, des études concernant le style d'un auteur, son héritage littéraire, ou tel aspect psychologique de sa personnalité, peuvent être fondées sur une étude globale de son œuvre, sondée transversalement grâce à l'outil statistique.

Des études terminologiques sur les écrits techniques reposent sur l'exploitation de corpus. C'est un des aspects approfondis dans la thèse d'Éric Gaussier [4].

On peut trouver des corpus écrits mais aussi sonores. C'est le cas, par exemple, quand il s'agit d'étudier une langue non écrite, mal connue, parlée par peu de locuteurs. C'est également le cas des études concernant le déroulement des conversations interpersonnelles, etc.

On entend par corpus multilingues les corpus constitués de textes dans plusieurs langues, ayant un étroit rapport sémantique. Les corpus multilingues sont peu nombreux par rapport aux corpus monolingues. Ils sont le plus souvent le résultat d'une traduction, encore qu'il existe des textes liés sémantiquement, mais rédigés librement dans des langues différentes. On peut penser à la production journalistique en particulier.

Cependant, on entend le plus souvent par corpus multilingue, un ensemble de textes dont l'un est considéré comme la source, rédigée librement, et le ou les autres comme les versions cible, fruits d'un travail de traduction.

1.1.3.1.2 Corpus bilingues alignés

Dans un corpus bilingue aligné, la version cible suit le même schéma rédactionnel que la version source, et les parties en correspondance sont identifiées. Ces correspondances entre les versions linguistiques du corpus sont plus ou moins fines.

En effet, on peut étudier l'alignement entre les versions linguistiques d'un corpus bilingue à différents niveaux. Il est d'abord possible que les deux versions linguistiques du corpus ne soient pas toutes de volumes identiques, et donc qu'elles présentent des lacunes l'une par rapport à l'autre et ne soient donc pas sémantiquement équivalentes. Dans le cas de versions équivalentes globalement, le schéma rédactionnel peut être différent d'une langue à l'autre : le texte peut être organisé différemment.

Plus typiquement, les deux versions sont équivalentes et sont architecturées de la même manière sur le plan rédactionnel. Elles suivent néanmoins des contraintes liées à la langue dans laquelle elles sont rédigées ou au style de traduction adopté. La finesse de l'alignement peut éventuellement aller jusqu'au niveau phrastique. Dans un corpus bilingue aligné au niveau phrastique, les phrases, ou plus généralement les segments (phrases, titres, légendes…), sont en correspondance biunivoque entre les deux versions linguistiques.

L'ambition de descendre à un alignement sous-phrastique est au cœur de notre travail. La granularité de l'alignement est donc plus fine que la phrase.

Une première possibilité est de mettre en correspondance les mots qui peuvent l'être. La notion de mot est volontairement ambiguë ici : il peut s'agir de « mots typographiques* »[4], séparés les uns des autres par des espaces ou des signes de ponctuation, ou de syntagmes plus complexes. On peut également vouloir mettre en évidence des correspondances entre des formes linguistiques plus élaborées.

Une autre possibilité consiste à mettre en correspondance des parties non linguistiquement caractérisées. La notion de « brique traductionnelle* » peut entrer dans cette catégorie. Une brique traductionnelle est constituée d'extraits de texte en traduction mutuelle, dont les frontières ne sont pas déterminées *a priori* sur des critères linguistiques. C'est à ce type d'alignement que nous nous intéresserons dans ce travail.

1.1.3.1.3 Évaluation

L'évaluation des corpus bilingues et de leur alignement a donné lieu à divers outils et protocoles. Le projet Arcade a lancé une première campagne d'évaluation d'alignement, en 1996 et 1998.[5].Une deuxième a été lancée en 2003, Arcade II ou Evalda.

Le résumé suivant se trouve à: www.technolangue.net/article 21.html#objectifs

« La proposition ACADE II vise à lancer une nouvelle campagne d'évaluation dans le domaine de l'alignement multilingue : 4 ans après ARCADE I, cette action se propose d'identifier les évolutions récentes de l'état de l'art, mais également d'approfondir l'évaluation sur des axes qui n'avaient pas été traités ou qui avaient seulement été effleurés : identification des ruptures de parallélisme, alignement de tri-textes, élargissement à des langues distantes du français, identification des cognats, appariement lexical ».

[4] Les termes entre guillemets et suivis d'une astérisque sont définis dans le glossaire.

1.1.3.2 Exploitation des corpus multilingues

1.1.3.2.1 Correspondances textuelles

L'alignement textuel consiste à aligner les corpus bilingues non alignés. Il s'agit de l'alignement de grandes unités, typiquement des phrases.

Le système SMIR (Smooth Injective Map Recogniser) proposé par Dan Melamed, a pour but l'alignement textuel de corpus bilingues [6]. Dans une première phase, dite de « génération », de nombreuses hypothèses de correspondances lexicales sont établies dans une région de texte. Lors d'une seconde phase, dite de « reconnaissance », on cherche à calculer un chemin optimal et donc à identifier les meilleures correspondances.

Olivier Kraif [7] a testé plusieurs techniques d'alignement lexical, utilisant plusieurs types d'indices statistiques, liés aux cooccurrences observées dans le corpus. Les correspondances établies sont filtrées, grâce à un jeu de seuils sur les valeurs des indices utilisés, afin d'augmenter la précision tout en conservant une bonne valeur de rappel.

Éric Gaussier [4] a décrit un modèle de réseau de flux statistique dans un graphe pour extraire de la terminologie d'un corpus bilingue.

1.1.3.2.2 Mémoires de traduction

Parmi les corpus bilingues alignés, les mémoires de traduction ont une place spécifique.

D'une part, elles n'ont pas été constituées en tant que corpus d'étude. Elles sont le fruit d'un travail de traduction et rassemblent toutes les unités (phrases, titres, légendes) rencontrées, appelées « segments* » dans ce contexte, avec leur traduction. Une mémoire de traduction sert de référence pour retrouver la traduction d'une phrase déjà traduite ou pour proposer des exemples de traduction à l'occasion de la traduction d'une phrase similaire.

En général, une mémoire de traduction ne présente pas une très bonne homogénéité de surface. En effet, elle peut avoir été constituée à partir de textes diversement formatés. Elle présente en revanche un bon alignement au niveau phrastique. Ce point est important, car de nombreux travaux ont pour objet l'alignement au niveau phrastique comme préalable à tout travail plus poussé sur les corpus. Dans le cas des mémoires de traduction, ce travail n'est, pour l'essentiel, plus à faire.

D'autre part, les mémoires de traduction existant dans une grande entité comme IBM sont constituées autour des mêmes textes source, en langue anglaise. Cela signifie qu'elles constituent en pratique des corpus multilingues alignés.

1.1.3.2.3 Autres applications

On voit fleurir de plus en plus d'ouvrages publiés en version bilingue, destinés à un lectorat d'apprenants d'une langue étrangère. En général, il s'agit de la version originale de l'auteur et de sa traduction dans la langue maternelle de l'apprenant. La comparaison des textes parallèles permet au lecteur de mieux appréhender la construction du texte dans la langue qu'il apprend.

Dans le domaine de l'enseignement des langues étrangères assisté par ordinateur, ou CALL (Computer-Assisted Language Learning), l'exploitation de corpus bilingues est largement utilisée pour proposer à l'apprenant des exemples réels illustrant tel ou tel point linguistique dans la langue étudiée. On pourra voir par exemple [8].

Enfin, une des applications importantes liées à l'exploitation des corpus multilingues est la traduction automatique (TA) "de corpus", qu'il s'agisse de TA statistique (Statistical Machine Translation : SMT), ou fondée sur l'exemple (Example Based Machine Translation : EBMT). Nous en détaillerons les différentes facettes dans la section suivante.

1.1.3.3 Une exploitation particulière : la TA

1.1.3.3.1 Méthodes statistiques

À partir de 1989, la domination de l'approche par règles a été remise en question par l'apparition de nouvelles méthodes, fondées sur l'apprentissage automatique à partir de corpus bilingues. On peut considérer qu'on est alors entré dans une nouvelle ère de la traduction automatique. Cette année-là, Brown, d'IBM, publie les résultats d'une méthode entièrement statistique. Le projet Candide se caractérisait par le rôle central du principe d'entropie maximale dans la traduction automatique [9]. : il n'y a aucune ressource linguistique autre que le corpus. C'est le corpus canadien Hansard, qui contient les débats du parlement canadien en deux langues, l'anglais et le français, qui a été utilisé.

IBM a été la première à proposer des modèles de traduction automatique statistique, fondés sur l'exploitation de corpus bilingues. Les modèles IBM 1, 2, 3 4 et 5 sont différents raffinements d'une même méthode [10]. Selon les modèles, chaque mot source est traduit par zéro, un ou plusieurs mots cible, la notion de fertilité est introduite, ainsi que la détermination de la place des mots cible en fonction ou non de celle des mots source.

Ces méthodes conduisent à des résultats décevants, inférieurs aux résultats de Systran, même sur des textes comparables. En outre, ces méthodes ne sont pas contrôlables par l'humain. À la suite de ces recherches, une nouvelle approche a consisté à utiliser des méthodes statistiques comparables, en les appliquant à des structures (comme des arbres d'analyse concrets ou abstraits, voire même des arbres sémantiques « interlingues » liés à un domaine et à une tâche particuliers). (Och [11], Roukos [12])

1.1.3.3.2 Systèmes hybrides

On entend par systèmes hybrides les systèmes qui utilisent à la fois des règles symboliques établies rationnellement par des linguistes et des règles probabilistes calculées par exploitation statistique de corpus bilingues.

C'est dans ce type de méthodes que se rangent les travaux de Richardson de Microsoft [13, 14]. À partir d'une analyse en « formes logiques » (logical forms), représentations syntaxico-sémantiques de chaque version d'un bitexte, on apprend automatiquement un dictionnaire et des règles de transfert sous forme d'un réseau appelé « MindNet » associant des sous-arbres source à des sous-arbres cible. Un MindNet est typiquement calculé sur plusieurs centaines de milliers d'arbres alignés.

Cette méthode intéressante, qui donne d'excellents résultats, nécessite toutefois l'analyse préalable par règles des textes à traduire. Autrement dit, il faut construire un module d'analyse et un module de synthèse pour toutes les langues traitées. Seul le module de transfert est acquis automatiquement par exploitation du corpus bilingue. Mais, dans le cas de Microsoft, les analyseurs et générateurs sont construits pour de multiples applications, comme la correction grammaticale, la correction stylistique, l'extraction d'entités nommées, le résumé automatique, etc.

1.1.3.3.3 Méthodes fondés sur l'exemple, analogie

L'une des difficultés essentielles liées à la traduction par l'exemple consiste à mettre en correspondance traductionnelle les différentes sous-chaînes d'un énoncé exprimé en deux langues. Mosleh Hmoud Al-Adhaileh a proposé dans sa thèse un formalisme pour ce faire [15]. Il s'agit des SSTC synchrones, dernière étape dans les modèles dérivés des « grammaires statiques » de B. Vauquois et S. Chappuy (1983, 1985) [16-18], à savoir :

1. Grammaires statiques de correspondances structurales chaîne-arbre (GSCS), ou String-Tree Correspondence Grammars (STCG),
2. Correspondances chaîne-arbre structurées, ou Structured String-Tree Correspondences (SSTC),
3. Correspondances chaîne-arbre synchronisées, ou Synchronous SSTC (S-SSTC).

Dans cette dernière approche, on construit pour chaque paire de phrases du bitexte deux arbres d'analyse (en constituants ou en dépendance, concrets ou abstraits), deux correspondances chaîne-arbre, et une correspondance entre ces deux dernières correspondances. Un éditeur spécialisé a été développé à cet effet. Une fois qu'on a quelques dizaines de milliers de telles S-SSTC, on les indexe (par les mots et les syntagmes fréquents), et on les utilise directement pour traduire de nouvelles phrases.

Un point remarquable de cette approche est que la traduction se construit en parallèle avec l'analyse : on reconnaît la partie "source" de la S-SSTC, en commençant "par le bas", donc par les plus petites. Cela donne non seulement des sous-arbres correspondant à des fragments de la phrase à traduire, mais aussi des sous-arbres correspondant à des fragments de la phrase cible en construction, ainsi que les correspondances entre les SSTC de ces fragments.

La problématique de la mise en correspondance traductionnelle de sous-chaînes est donc au cœur de la thèse de Al Adhaileh, comme de la nôtre. Nous verrons cependant que notre démarche est sensiblement différente, en particulier parce qu'elle repose uniquement sur le traitement automatique de données acquises comme sous-produit d'une activité préexistante. Nous nous refusons en effet à faire préparer ou corriger par des humains des structures complexes comme des arbres d'analyse linguistiquement motivés, des SSTC et des S-SSTC. En cela, nous sommes beaucoup plus proche des tenants des méthodes statistiques de Brown, et Ney [19, 20] ou analogiques comme chez Yves Lepage [21].

De fait, la traduction par analogie représente également une direction intéressante. L'idée fondamentale est de calculer la traduction d'un énoncé en appliquant la méthode dite de quatrième proportionnelle à partir d'exemples de traduction. Ainsi, à partir d'un énoncé E qu'on cherche à traduire en un énoncé X, on cherche un couple d'énoncés (A,B) tel que A soit "proche" de E et que B soit une traduction de A. On résout alors (en X) l'équation analogique "A : E :: B : X", qui signifie: "A est à E ce que B est à X".

Comme on ne trouve pas toujours de telles relations analogiques dans le corpus de référence Yves Lepage a imaginé un algorithme de calcul analogique par chaînage qui permet d'établir une traduction à l'aide de plusieurs exemples et éventuellement d'un dictionnaire bilingue [21, 22].

Il y a bien d'autres recherches intéressantes dans ce domaine, qui nous ont inspiré à un titre ou à un autre, mais qu'il ne semble pas utile de présenter ici en détail. Notons surtout :

1. Simard et Langlais [23], qui se sont intéressés à mesurer la réutilisabilité de mémoire de traduction, en examinant le nombre de segments de traductions préexistantes effectivement présents dans de nouvelles traductions

2. Watanabe et alii [24] (ACL 07/2003, Sapporo)
3. Dekai WU, qui propose d'utiliser le fait qu'il s'agit de deux phrases en relation de traduction pour construire leurs deux analyses ensemble, par "analyse bilingue" (bilingual parsing) [25]

Ces études parmi d'autres cherchent à exploiter la manne que représentent les exemples de traduction de plus en plus nombreux et disponibles grâce à l'internet.

1.2 La TAO fondée sur les mémoires de traduction

Une approche efficace pour l'industrialisation de la traduction, fondée sur une rationalisation des efforts et permettant des gains de productivité effectifs, repose sur l'utilisation de mémoires de traduction. Nous verrons successivement dans quelles conditions elles sont apparues il a une dizaine d'années, comment elle sont construites et utilisées aujourd'hui, et pourquoi on peut les considérer comme objets d'étude pour l'avenir.

1.2.1 Succès des systèmes à mémoires de traduction

Les systèmes d'aide à la traduction ont connu un réel succès depuis une quinzaine d'années.Cela est en partie dû à la relative déception liée aux déboires de la traduction automatique proprement dite. Il n'est pas inutile de rappeler les étapes de ce succès, d'en préciser l'étendue et de faire un tour d'horizon des évolutions possibles et probables.

1.2.1.1 Ressource unique et incrémentale

1.2.1.1.1 Cause du succès des mémoires de traduction

Un principe essentiel des méthodes d'aide à la traduction humaine fondées sur l'utilisation de mémoires de traduction veut que ces mémoires constituent l'unique ressource utilisée. Des ressources additionnelles telles que lexiques ou dictionnaires peuvent être utilisées, mais restent optionnelles.

De plus, cette ressource s'enrichit de manière incrémentale : au fur et à mesure des traductions, de nouveaux exemples sont ajoutés à la mémoire de traduction.

Les méthodes d'aide à la traduction fondées sur les mémoires de traduction apportent un réel gain de productivité sans recourir à d'hypothétiques ressources linguistiques qui sont en générale coûteuses et incomplètes.

Le fonctionnement est relativement facile à saisir pour l'utilisateur, qui peut peu ou prou anticiper et donc maîtriser le système. On peut penser que ces différents atouts ont permis l'essor des systèmes fondés sur les mémoires de traduction.

L'acquisition de ressources linguistiques est un goulet d'étranglement, car elles sont :
1. difficiles à rassembler,
2. difficiles à harmoniser,
3. difficiles à délimiter (quoi pour faire quoi ?).

En pratique, ces difficultés se réduisent à un dénominateur commun : établir des ressources linguistiques coûte cher. Le plus souvent, les acteurs du domaine et les organismes

financeurs œuvrent à l'industrialisation de leur propre langue seulement. C'est pourquoi la plupart des langues sont considérées aujourd'hui comme « peu dotées », sans parler des couples de langues.

1.2.1.1.2 Indépendances vis-à-vis de la langue

Une autre raison qui explique le succès des mémoires de traduction tient à l'indépendance fonctionnelle vis-à-vis des langues utilisées.

En effet, ces méthodes sont applicables à toute langue dès l'instant qu'on en connaît l'encodage et que l'on sait la segmenter. Ce dernier point peut être délicat. En pratique, la segmentation automatique mise en œuvre est largement relayée par des traducteurs humains.

1.2.1.1.3 Réponse naturelle à la situation du domaine

Comme nous l'avons vu (1.1.1.3), le fondement même de l'utilisation des mémoires de traduction tient à la redondance du texte source. Cette redondance existe à l'intérieur d'un même projet de traduction, entre différents projets de traduction sur un même sujet, dans un même domaine ainsi que dans la vie d'un document, dont les différentes versions seront traduites au fur et à mesure de son évolution.

L'utilisation des mémoires de traduction permet donc d'exploiter partiellement cette redondance. Le rappel par le système de segments déjà traduits, en coïncidence floue avec un nouveau segment à traduire, permet de réutiliser une partie significative de l'effort déjà fourni.

En outre, les mémoires de traduction permettent une meilleure maîtrise du style et de la terminologie. Autrement dit, elles favorisent implicitement une coopération entre les traducteurs et conduit naturellement à maintenir l'homogénéité terminologique et stylistique des traductions produites.

1.2.1.2 Évolution

L'évolution des systèmes à mémoire de traduction semble s'orienter vers l'utilisation de modules complémentaires, d'une plus large mutualisation des données à travers les réseaux (publics comme Internet, ou privés), ainsi que vers une représentation plus fine des données, comme c'est l'objet de cette thèse.

1.2.1.2.1 Formats XML spécifiques orientés traduction

Le besoin d'intégrer des données propres à la traduction dans le document à traduire se fait de plus en plus sentir. Il peut s'agir de décomptes de mots, nécessaires à la facturation, comme de commentaires ou d'informations linguistiques.

En outre, le fait de formater les documents destinés à être traduits selon un format unique permet de simplifier le traitement amont d'identification des textes à traduire (par opposition au formatage et autres éléments non textuels).

Des outils différents peuvent s'appuyer sur ces formats, permettant l'échange des données tout en favorisant la diversité de l'offre. Différents standards pour stocker ou échanger des mémoires de traduction ont vu le jour ces dernières années. Il y a eu OpenTag qui n'est plus maintenu aujourd'hui mais dont sont inspirés les deux standards suivant :

TMX : Translation Memory eXchange développé par le groupe OSCAR de LISA (Localisation Industry Standards Association), qui vise à l'échange de mémoires de traduction (http://www.lisa.org/tmx/)

XLIFF Maintenu par OASIS (Organization for the Advancement of Structured Information Standards) dont l'objectif est de conserver une information structurée des mémoires de traduction. (http://www.oasis-open.org/committees/xliff/charter.php)

Ces deux standards sont complémentaires.Nous présentons ci-dessous un exemple de mémoire de traduction formatée avec XLIFF.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE xliff PUBLIC "-//XML-INTL XLIFF-XML 1.0//EN"
"xliff.dtd">
<xliff version="1.0">
<file datatype="xml" source-language="en-USA" target-
language="es-ESP">
<header>
<count-group name="Totals">
<count count-type="TextUnits" unit="transUnits">40</count>
<count count-type="TotalWordCount" unit="words">416</count>
</count-group>
</header>
<body>
<trans-unit id="t1">
<source> Xml:tm</source>
<target> Xml:tm </target>
</trans-unit>
<trans-unit id="t2">
<source> Xml:tm is a revolutionary technique for dealing with
the problems of translation memory for XML documents by using
XML techniques and embedding memory directly into the XML
documents themselves. </source>
<target> Xml:tm es una técnica revolucionaria que trata los
problemas de memoria de traducción en documentos XML usando
técnicas XML e incluyendo la memoria en el documento mismo.
</target>
</trans-unit>
<trans-unit id="t3">
<source> It makes extensive use of XML namespace. </source>
<target>Esta técnica hace extenso uso de XML namespace.
</target>
</trans-unit>
. . .
```

Figure 6 : exemple de mémoire formatée avec XLIFF

1.2.1.2.2 Substitution, calcul de distance (TELA)

Le travail d'Emmanuel Planas [26] explorait les possibilités d'enrichir les correspondances entre segments avec toutes sortes d'informations, linguistiques, sémantiques ou même typographiques afin de permettre une meilleure réutilisabilité des mémoires de traduction.

Depuis, Emmanuel Planas a développé un produit complet, Similis, muni d'un aligneur très performant, dans le cadre de la société Lingua&Machina qu'il a créée en 2001 après un

postdoc de 2 ans à NTT où il avait construit un prototype opérationnel et démontré les avantages de son approche sur celles des produits commerciaux alors disponibles.

1.2.1.2.3 Grandes bases de mémoires

Une autre orientation consiste à rassembler le plus grand nombre de mémoires de traduction possible afin d'augmenter les chances de trouver des segments utiles au traducteur. On rassemble alors des mémoires développées dans des contextes différents.

Le nombre de propositions augmente effectivement, mais on se heurte à deux types de difficultés.

D'une part, la cohérence de la langue utilisée n'est plus aussi bien garantie. Notamment, la terminologie risque d'être complètement désorganisée. Il est fréquent, par exemple, que la terminologie élaborée par un éditeur de logiciels pour un de ses produits diffère complètement de la terminologie utilisée ailleurs. Les aléas du marché entraînant fusions et scissions des acteurs industriels, différentes politiques terminologiques, éventuellement contradictoires, se retrouvent en opposition frontale. Cela peut également être le cas des logiciels conçus pour être utilisés sur plusieurs plates-formes et dont il est difficile d'harmoniser les différents choix terminologiques.

Corrélativement, il peut arriver qu'un même segment ait été traduit de plusieurs manières différentes dans différents projets de traduction. On assiste alors à une profusion de propositions différentes, requérant une validation manuelle fort coûteuse.

Il ne faut donc pas négliger l'avantage de n'utiliser que des mémoires de traduction construites sur un même projet, afin de tirer le meilleur parti leur spécificité.

1.2.1.3 Quelques produits

1.2.1.3.1 IBM Translation Manager

Apparu dès la fin des années 80, IBM Translation Manager est certainement le précurseur des environnements à mémoires de traduction. Il a été conçu et développé pour répondre aux besoins de traduction interne d'IBM et a été commercialisé à partir du milieu des années 90. Il ne l'est plus aujourd'hui. Il a évolué moins vite que ses concurrents et c'est sans doute la raison pour laquelle il existe aujourd'hui une offre assez large de systèmes commerciaux d'aide à la traduction fondés sur la mémoire, qui sont tous plus ou moins directement inspirés de lui.

Parmi les nombreux atouts de Translation Manager, il faut noter sa rapidité et l'intégration assez efficace de dictionnaires terminologiques. Il offre une bonne interface de gestion de documents et de projets. La possibilité de scinder ou au contraire de regrouper des segments a été introduite avec lui et n'est pas toujours présente sur les systèmes plus récents. Le travail en réseau sur une mémoire partagée, permettant à plusieurs traducteurs de bénéficier chacun du travail des autres est aussi un atout majeur de Translation Manager. De plus, Translation Manager reste l'un des rares produits, avec Trados, à même de gérer des systèmes d'écriture non latins.

Cependant, on reproche fréquemment à Translation Manager son interface peu ergonomique et sa relative difficulté d'apprentissage. Le système utilise son propre éditeur, et ne s'intègre pas dans un logiciel de traitement de texte comme la plupart de ses concurrents actuels. Au contraire, au lieu d'être interprété, le balisage de mise en page apparaît directement dans les segments traités, ce qui peut dérouter l'utilisateur. En outre, la recherche

terminologique dans les mémoires directement n'est pas prévue, contrairement à Déjà Vu et Transit, par exemple.

Translation Manager fait maintenant figure d'ancêtre vénérable. Il n'est plus commercialisé mais continue d'être utilisé par IBM et ses nombreux partenaires à travers le monde.

1.2.1.3.2 Les autres produits

Il existe aujourd'hui plusieurs systèmes de traduction fondés sur les mémoires de traduction. Ceux qui fonctionnent en réseau (Translation Manager, Trados…) sont préférés par les grosses entreprises de traduction, et leur prix est également plus important.

Le tableau suivant est extrait de l'excellente étude du LISA (disponible à http://www.lisa.org/products/survey/2004/tmsurvey.html) sur le sujet en 2004. Il donne les pourcentages d'utilisateurs parmi les personnes ayant participé à l'étude en fonction des produits les plus courants.

TM tool	Percentage of users
TRADOS	71%
SDLX	28%
Déjà Vu	24%
Alchemy Catalyst	20%
STAR Transit	19%
WordFast	18%
Internal	12%
Passolo	5%
Foreign Desk	5%
OmegaT	4%
IBM Translation Manager	3%

Tableau 1 : comparaison de l'utilisation des produits sur le marché

Comme on le voit, beaucoup de traducteurs utilisent plusieurs produits dans leur travail, puisque la somme des pourcentages est de 209. Ces différents systèmes sont tous fondés sur les mémoires de traduction ce qui et offrent donc des fonctionnalités base comparables. Nous donnons ici quelques points de repère, glanés parmi les quantités d'informations sur le sujet disponibles sur le web.

Le produit de TRADOS, Translation Workbench, arrive en tête principalement parce qu'il est facile à utiliser et qu'il s'intègre dans plusieurs traitements de texte, Word notamment. L'interface graphique permet l'utilisation de menus et de boutons, qui s'intègrent dans le menu du traitement de texte. Bien entendu, les raccourcis claviers sont utilisables pour les utilisateurs avertis. Parmi les fonctionnalités originales, il faut souligner le concordancier dynamique, activable à partir de n'importe quel segment de texte surligné. Le système d'alignement de Trados est actuellement un des plus performants du marché. L'un des défauts important de Trados est lié à sa richesse : les données cachées introduites par le logiciel dans le document sont nombreuses et peuvent interférer avec d'autres appartenant à d'autres produits supportés par le traitement de texte. Il en résulte alors une corruption des données en question. Il faut donc sauvegarder son travail le plus souvent possible…

SDLX a été introduit récemment par SDL, qui est une grosse entreprise de traduction anglaise. Il s'agit d'une suite d'outils déconnectés dont l'interface de traduction fondée sur la mémoire, qui s'appelle tout simplement SDL. Ses fonctionnalités sont très comparables à celles de Translation Manager. Toutefois l'environnement d'édition interprète les balises de formatage mais il ne permet pas de retailler les segments.

Déjà Vu d'Atril est très apprécié des traducteurs individuels. Sa capacité à supporter pratiquement tous les formats connus le rend très accessible, il est très facile à utiliser et possède une grande souplesse d'utilisation de la mémoire : rétropropagation des correctionx etc. Déjà Vu s'illustre en outre pour sa fonction « assemble » qui permet de construire des propositions de traduction par assemblage de sous-segments issus de la mémoire de traduction. Enfin, il faut souligner son prix très compétitif ainsi que la grande proximité entre l'équipe de développement et les utilisateurs.

Alchemy Catalyst s'intègre dans tous les logiciels de Microsoft et supporte bien les fichers XML ou propres à l'Internet. Il offre la possibilité de qualifier les traductions avec des drapeaux. On peut ainsi assigner une priorité de réutilisation des segments. En outre, il est contient un kit de développement qui permet d'étendre ses fonctionnalités.

Transit de Star fait figure d'ousider parmi les environnements de traduction. En effet, le système n'utilise pas de mémoire de traduction à proprement parler. En lieu et place, le système construit dynamiquement un réseau associatif (Associative Network), un index en fait, à partir des dossiers et fichiers préalablement traduits (avec l'outil) qui lui sont indiqués par l'utilisateur lors de la préparation de son travail. En outre, cette sélection est révisable pendant la traduction proprement dite. Les dossiers en question restent stockés là ou l'utilisateur a choisi de les mettre sur son disque dur. On peut en outre corriger les fichiers qui servent de référence directement pendant la traduction. Transit propose aussi un concordancier très efficace et un langage de macros.

Citons encore Similis de Lingua et Machina, société fondée en 2001 par Emmanuel Planas, qui se présente comme un système de deuxième génération. En effet, l'exploitation des mémoires de traduction est faite par morceaux, grâce à des méthodes analogiques.

Il faudrait également mentionner Tr Aid développé par l'équipe du professeur Piperidis de l'Institute for Language and Speech Processing (ILSP) en Grèce, Eurolang Optimizer de SITE, MetaTexis, Multitrans, Sprint, Internal, Passolo, Foreign Desk, OmegaT, Trans Suite 2000, Wordfast et tant d'autres.

1.2.1.3.3 Quelques captures d'écran

Nous avons rassemblé ici quelques captures d'écran des principaux logiciels cités plus hauts, afin de donner un aperçu de leur interface.

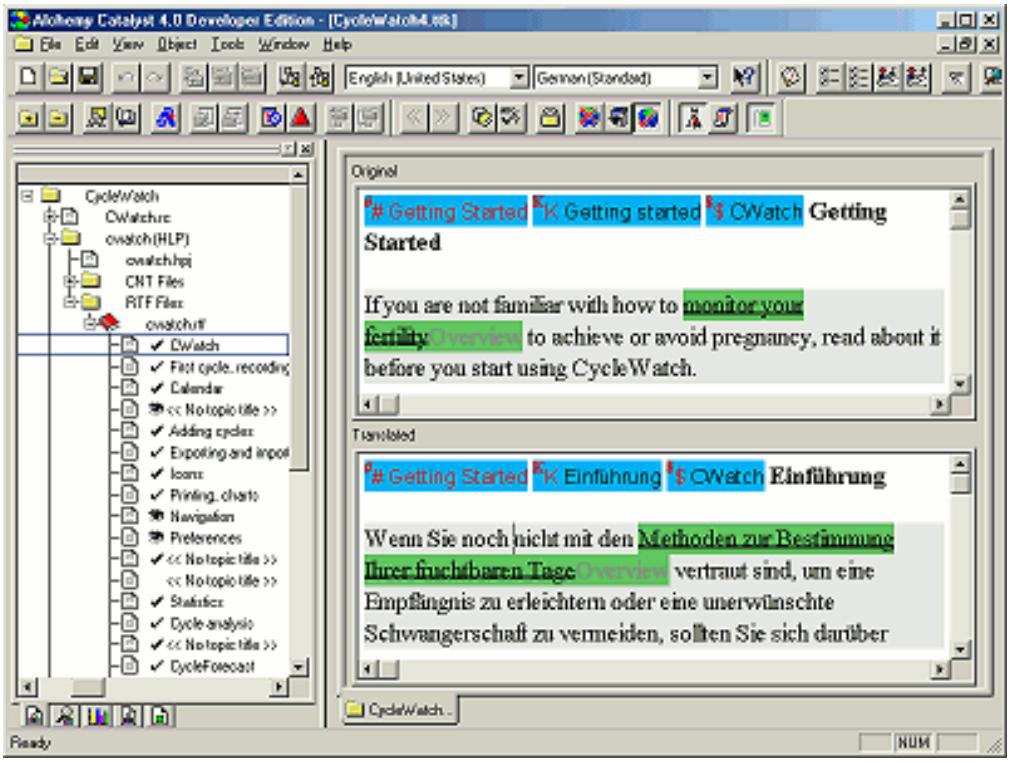

Figure 7 : Alchemy Catalyst

Alchemy Catalyst présente le segment original et le segment traduit l'un au dessus de l'autre. La hierarchie des documents est visible sur la gauche.

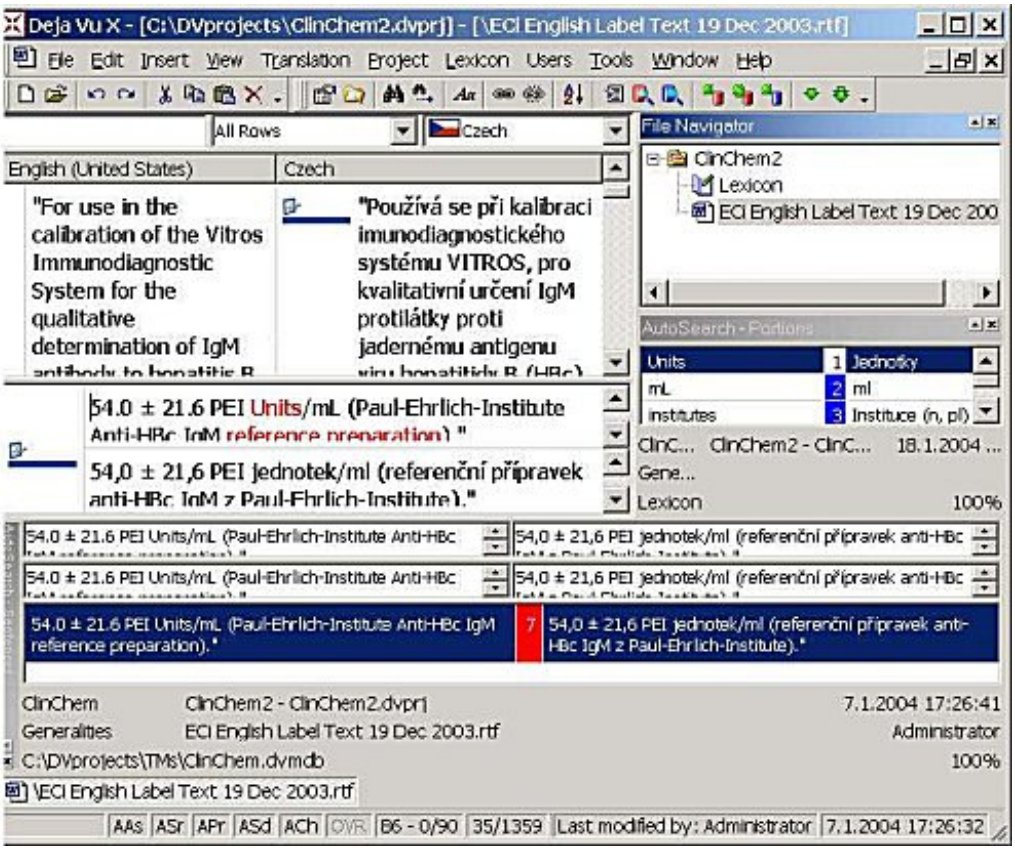

Figure 8 : DéjàVu

Dans Déjà Vu, le texte original et la traduction sont disposés l'un à côté de l'autre. On voit à droite la recherche automatique de termes dans le dictionnaire.

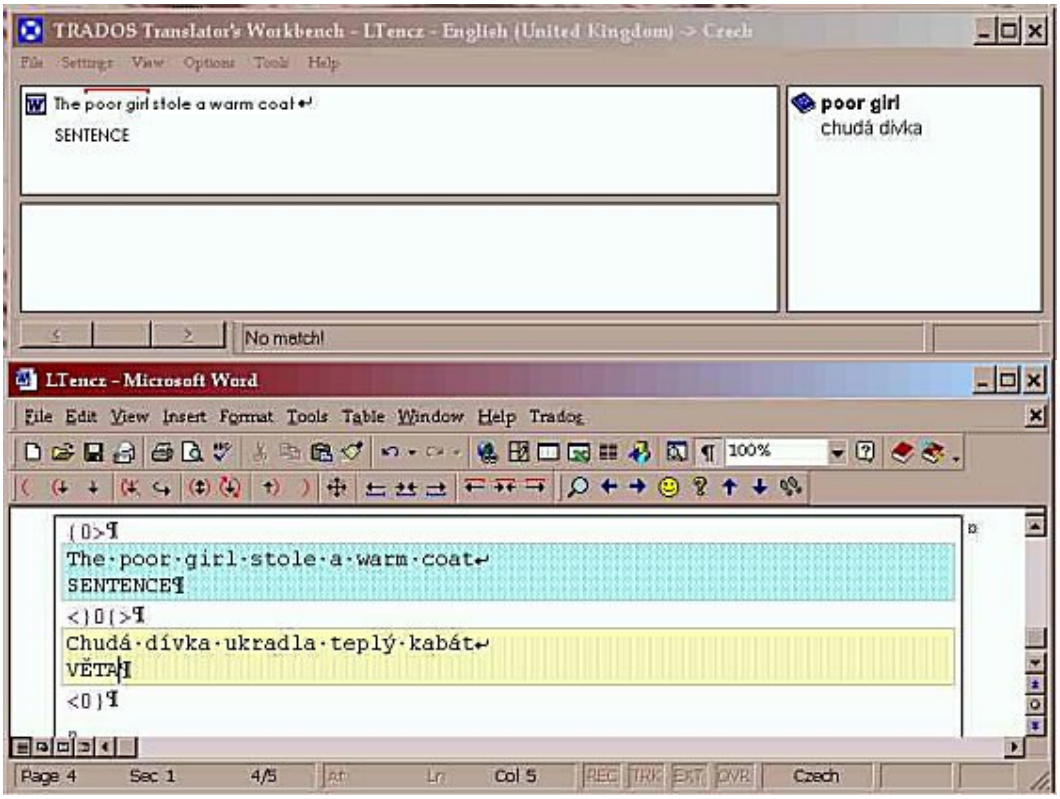

Figure 9 : Trados

Trados peut être intégré dans un logiciel de traitement de texte : ici Word.

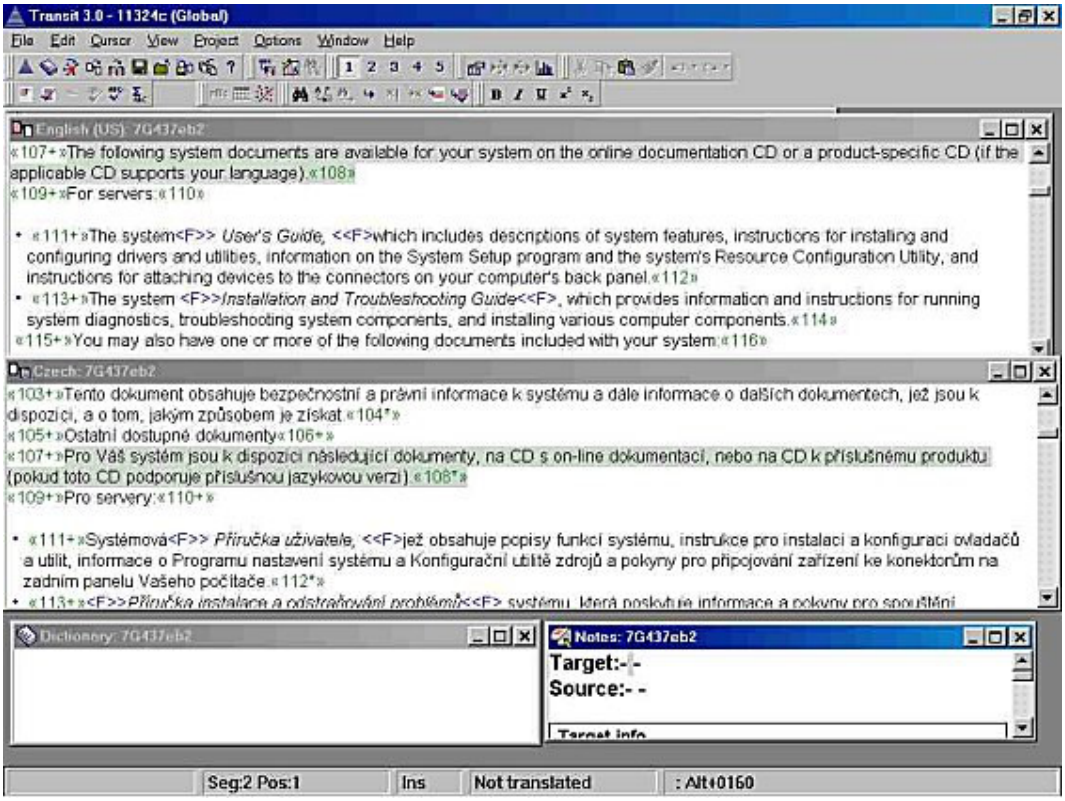

Figure 10 : Transit

Transit présentent les traduction l'un au dessus de l'autre. Des notes de traduction peuvent être enregistrées dans fenêtre en bas à droite.

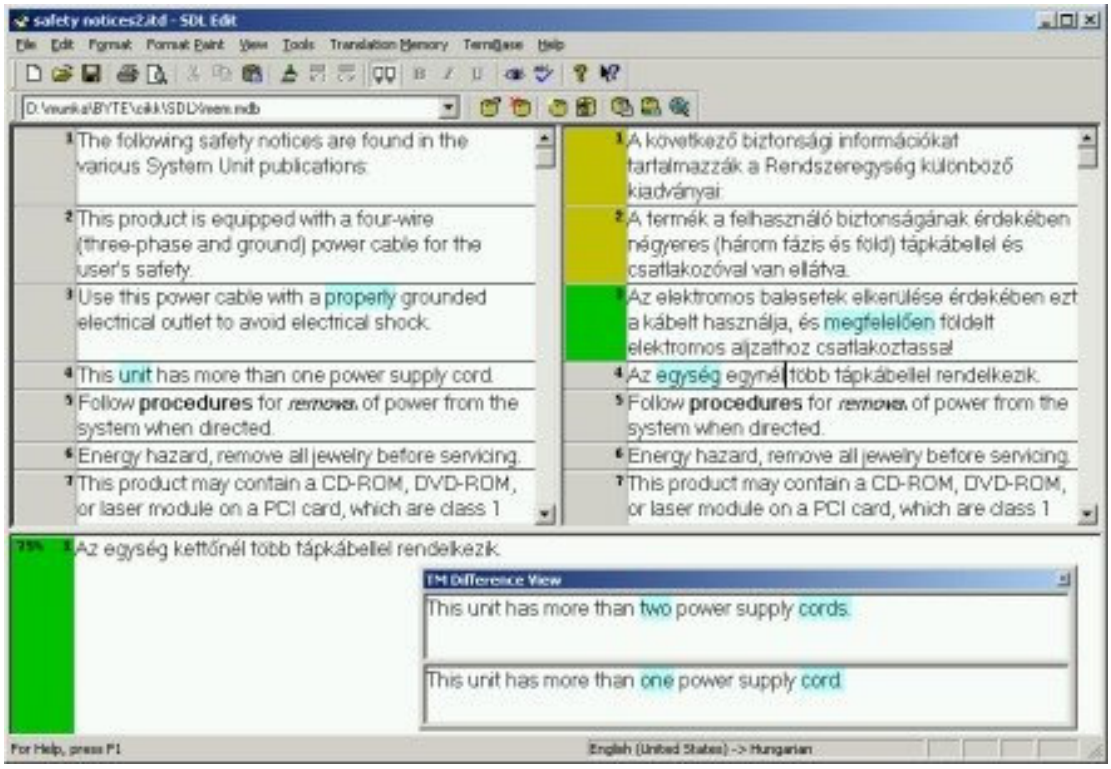

Figure 11 : SDLX

SDLX surligne les différences entre les différents segments proposés au traducteur. Les versions source et cible sont disposées latéralement.

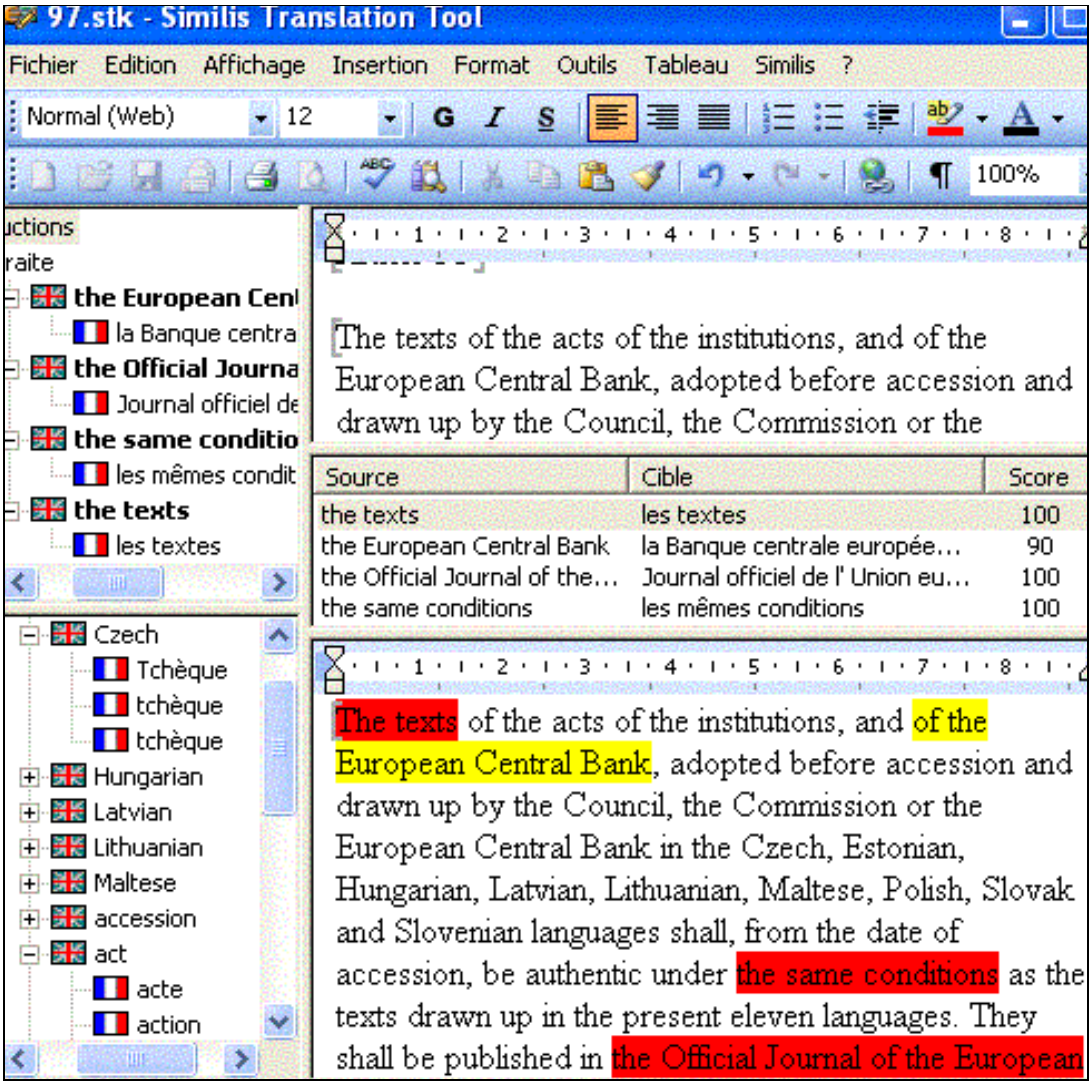

Figure 12 : Similis

Similis propose des briques traductionnelles, qui sont surlignées dans le texte, en clair pour les propositions approchées et en foncé pour les propositions exactes.

1.2.2 Principes de fonctionnement

La traduction humaine fondée sur la mémoire part d'un principe simple : il s'agit de collecter des traductions de phrases au moment où elles sont produites par les traducteurs afin de les réutiliser. Cependant, il existe maintes déclinaisons possibles de ce principe fondamental.

1.2.2.1 Fonctionnement et aspect dynamique

1.2.2.1.1 Historique des versions

Les systèmes d'aide à la traduction fondés sur la mémoire représentent un ensemble d'outils informatiques visant à faciliter la réutilisation de traductions existantes. On y trouve un environnement de gestion de dossiers grâce auquel l'utilisateur peut lier un ensemble de

fichiers à un ensemble de mémoires de traduction, de tables de marquage (pour l'analyse en segments), de dictionnaires etc.

Ils s'inscrivent dans l'approche de la TAO du traducteur, par opposition à la TAO du rédacteur et à la TAO du réviseur.

L'utilité des mémoires de traduction est de disposer d'un fonds d'énoncés réutilisables lors d'un travail de traduction. L'équivalence sémantique des constituants d'un bisegment est garantie par l'expertise humaine des traducteurs.

Le principe mis en œuvre consiste à enregistrer systématiquement le travail du traducteur sous forme de paires de segments en langues source et cible. Lorsque, au cours de la traduction d'un nouveau texte, un segment se trouve correspondre à l'un de ceux préalablement enregistrés dans le système, sa traduction peut être récupérée et réutilisée par le traducteur. Celui-ci peut alors l'éditer pour l'adapter éventuellement.

1.2.2.1.2 Un seul traducteur

Lorsque le traducteur travaille seul, il bénéficie de l'historique de son travail, qui vient s'ajouter à la base de connaissances contenue dans la mémoire de traduction.

Ainsi, il est assuré non seulement d'optimiser son travail en évitant de retraduire les mêmes choses, mais également de maintenir une réelle cohérence aussi bien stylistique que terminologique.

1.2.2.1.3 Réseau

Dans la situation où plusieurs traducteurs travaillent en réseau, le système permet un partage contrôlé des tâches. En effet, la mémoire de traduction est partagée entre tous, et les nouvelles traductions produites par eux sont mises en commun. À tout moment, de nouvelles traductions viennent enrichir la base de connaissances, dont chacun peut profiter.

Le système permet de garantir la cohérence stylistique et terminologique entre les différents traducteurs et contribue efficacement à la convergence progressive de la terminologie utilisée.

1.2.2.2 Enregistrement de bisegments

1.2.2.2.1 Présentation

Pendant la traduction, les bisegments sont constitués. Le système analyse le texte source avec une ou plusieurs mémoires de traduction. Les différentes propositions de traduction admissibles pour chaque segment source sont recueillies et classées en fonction des options qu'a choisies l'utilisateur. Au moment de la traduction proprement dite, c'est le traducteur humain qui tranche pour chaque segment. En effet, il peut :

décider de ne pas traduire, ce qui revient à valider le segment original comme traduction.

valider une proposition de traduction exacte parmi plusieurs traductions retrouvées par le système.

éditer et retravailler la traduction d'un des segments trouvés en correspondance floue (approchée) par le système.

écrire une traduction entièrement nouvelle.

Dans ces deux derniers cas, la version du traducteur est enregistrée dans la mémoire de traduction, associée au segment source.

Il est également possible que le système effectue une substitution automatique, s'il n'y a aucune ambiguïté sur la correspondance exacte trouvée dans la mémoire. Cette éventualité dépend des options choisies au moment de l'analyse. Dans ce cas, le segment source à traduire n'est même pas présenté au traducteur, puisque les critères (liés aux options) utilisés par le système sont supposés suffisants pour ne pas nécessiter de validation humaine.

1.2.2.2.2 Enregistrement d'un segment en contexte

Translation Manager enregistre quelques métadonnées avec le bisegment, notamment la date de traduction, ainsi que les références du fichier dans lequel il figure et son emplacement dans ce fichier.

Deux segments source formellement identiques peuvent recevoir deux traductions différentes : ils sont enregistrés séparément dans la mémoire. Les mémoires de traduction de Translation Manager contiennent donc l'ensemble des informations nécessaires pour reconstituer l'ensemble du texte.

1.2.2.2.3 Enregistrement d'un segment sans contexte

D'autres systèmes factorisent les segments source. L'emplacement de chaque occurrence du segment n'est pas conservé. Eventuellement, il est possible d'enregistrer un certain nombre de traductions, considérées comme exceptionnelles, d'un segment source donné.

Le système gérera alors les éventuelles traductions concurrentes en tenant compte des différentes occurrences du texte source.

1.2.2.3 Segment à traduire et segment enregistrés

1.2.2.3.1 Coïncidence exacte

On parle de « coïncidence exacte » (en anglais : « exact match ») lorsque le segment à traduire est déjà présent dans la mémoire de traduction. Selon les options de fonctionnement, ou bien le segment ainsi retrouvé est directement substitué et n'est jamais soumis au traducteur, ou bien il est soumis au traducteur pour relecture.

C'est en particulier le cas lorsque plusieurs traductions concurrentes existent au sein de la mémoire de traduction, sans que le système dispose de critères suffisants pour déterminer la bonne traduction. On parle alors de « coïncidence exacte double » (en anglais : « double exact match »). Les critères éventuellement utilisables pour permettre au système de faire un choix comprennent entre autres l'emplacement du segment et la date de traduction.

1.2.2.3.2 Coïncidence floue

On dit qu'il y a « coïncidence floue » (en anglais : « fuzzy match ») quand la mémoire de traduction contient un ou plusieurs segments dont la distance au segment à traduire est inférieure à un certain seuil, dans la langue source. Ces segments sont présentés au traducteur qui est libre de les utiliser pour formuler sa traduction. Typiquement, le système ne présente pas plus de trois segments en coïncidence floue.

C'est cette situation qui justifie en pratique l'utilisation des systèmes d'aide à la traduction fondés sur la mémoire.

1.2.2.3.3 Autres types de coïncidence

En fonction des options du système et des modules utilisés, différents calculs peuvent être effectués sur les segments avant de les présenter au traducteur.

Il peut y avoir des substitutions simples, telles que des chiffres ou des dates, par similitude entre le segment à traduire et les coïncidences floues trouvées.

De plus en plus, un module de traduction automatique peut être utilisé comme dernier recours, quand aucune coïncidence floue n'est présente dans la mémoire. Le résultat de la traduction automatique est présenté au traducteur qui peut, là encore, l'utiliser ou non.

De fait, le couplage des systèmes de traduction fondés sur la mémoire avec les systèmes de traduction automatique connaît un certain succès. La compétence de relecteur de traduction automatique est assez différente de celle qui consiste à produire une traduction nouvelle. C'est donc une évolution du métier des traducteurs habitués aux systèmes fondés sur la mémoire qui est nécessaire. La qualité attendue d'un système de traduction automatique destiné au traducteur (TA du traducteur) est *a priori* très différente de celle qu'on attend pour une simple relecture (TA du réviseur). Les méthodes actuellement utilisée pour apprécier la qualité d'une traduction automatique sont actuellement les mêmes dans les deux cas, ce qui très peu satisfaisant. Nous discutons ce point en 2.3.3.3.2. Par ailleurs, on se trouve ici aussi confronté au problème de la cohérence terminologique qui, en pratique, n'est généralement respectée par les systèmes de traduction automatique.

1.2.3 Mémoires de traduction et corpus bilingues

Les mémoires de traduction ont été créées pour un fonctionnement bien précis. De même que la pierre de Rosette ou le corpus Hansard, qui reprend les débats du parlement canadien, les mémoires de traduction constituent, de manière annexe, des corpus bilingues relativement homogènes.

1.2.3.1 Alignement quasi parfait

1.2.3.1.1 Contenu

Une mémoire de traduction, fondamentalement, est une collection de bisegments. On entend par bisegments des couples de segments, c'est à dire d'énoncés complets sémantiquement équivalents et exprimés dans deux langues différentes et identifiées. Typiquement, les segments sont des phrases, mais cela n'est pas nécessaire : il peut s'agir d'unités disparates, comme des titres, des éléments de menus d'interface utilisateur, des messages d'avertissement, etc. Un segment peut être réduit à un mot simple ou à un syntagme nominal, ou contenir un énoncé en style abrégé etc. En outre, les mémoires de traduction peuvent contenir des indications de mise en forme du texte. Les trois exemples qui suivent donnent un aperçu du contenu d'une mémoire de traduction.

```
<Segment>0000000019
<Control>(non imprimable)</Control>
<Source>Conferencing</Source>
<Target>Systèmes de conférence</Target>
</Segment>

<Segment>0000000015
<Control>(non imprimable)</Control>
<Source>They will provide you the details on what type of
education vehicle is available to you as part of this
offering.</Source>
<Target>On vous indiquera le mode de paiement disponible dans
le cadre de cette offre.</Target>
</Segment>

<Segment>0000000024
<Control>(non imprimable)</Control>
<Source>:h5.Planned Availability Date</Source>
<Target>:h5.Date de commercialisation prévue</Target>
</Segment>
```

Figure 13 : extrait d'une mémoire de traduction de Translation Manager

Dans cet extrait, les informations relatives aux langues source et cible, à l'emplacement du segment dans le document ainsi que la date et de l'heure de la traduction sont stockées sous forme non imprimable dans la balise `<Control>`.

1.2.3.1.2 Analyse en segments

L'analyse d'un texte en segments se fait grâce à l'utilisation d'une « table de marquage ». Celle-ci contient les critères qui permettent d'établir le découpage. Par exemple, s'il s'agit d'un fichier texte non formaté, la table de marquage contiendra des règles permettant d'utiliser les majuscules et les points pour distinguer des phrases. S'il s'agit d'un document formaté, les balises de mise en forme seront également utilisées.

Il existe de nombreuses tables de marquage à IBM, chacune liée à un format particulier. Ces tables de marquage évoluent avec le temps, et il est souvent nécessaire de les mettre à jour.

Lors de l'analyse du texte, la table de marquage à employer est précisée au système par l'utilisateur.

1.2.3.1.3 Fabrication d'une mémoire à partir d'un texte bilingue

En général, une mémoire de traduction est constituée à l'occasion d'une première traduction. Cette construction est gérée par le système d'aide à la traduction.

Mais il arrive que des textes aient été traduits en dehors du système d'aide à la traduction. On construit alors une « mémoire de traduction initiale » (en anglais « initial translation memory », ITM) à l'aide d'un module particulier du logiciel.

En règle générale, la mémoire de traduction ainsi constituée n'est pas utilisable en l'état car elle contient de nombreux défauts d'alignement. L'utilisateur doit alors revoir la segmentation et l'alignement du bitexte. Il peut fusionner des segments ou les scinder, et modifier les correspondances établies automatiquement entre les segments source et cible.

Une fois ce travail terminé, la mémoire peut être utilisée pour une traduction nouvelle.

1.2.3.2 Rappel : à IBM : une langue source, plusieurs langues cible

1.2.3.2.1 Rôle central de l'anglais

La langue utilisée par les laboratoires d'IBM, où les produits sont conçus et développés, est l'anglais. La terminologie de référence utilisée à IBM est en anglais américain. L'orthographe de référence est l'orthographe américaine.

Il arrive, mais c'est rare, qu'un produit soit développé en deux langues simultanément : l'anglais et une langue plus familière aux concepteurs et développeurs. Ces situations sont liées à une concentration historique des compétences dans certains centres de production, comme par exemple le produit Pac Base développé au laboratoire IBM de Paris.

Ainsi, l'anglais sert de langue pivot non seulement dans les rapports humains dans les contextes internationaux mais également pour l'essentiel de la production écrite de la compagnie.

Les mémoires de traduction sont, sans exception, systématiquement orientées de l'anglais comme langue source vers une autre langue, le reste de la traduction n'étant pas suffisamment redondant et se faisant hors de Translation Mananger.

1.2.3.2.2 Corpus multilingue

Les produits d'IBM sont traduits dans une trentaine de langues (voir 1.1.1.2.3). Tous les produits ne sont pas traduits dans toutes ces langues, mais certains le sont. En général, les produits et les documents les accompagnant ont été rédigés en anglais. Autrement dit, le même texte anglais est traduit vers de nombreuses langues et donne lieu à autant de mémoires de traductions portant sur les mêmes textes.

La segmentation du texte source est généralement la même dans toutes ces mémoires. Même s'il peut exister des différences locales, introduites par les traducteurs, on peut considérer qu'au moins une sous-partie de l'ensemble de ces mémoires forme un corpus multilingue aligné.

L'intérêt d'un tel corpus, c'est qu'il fournit des passerelles entre les différentes versions linguistiques d'un même texte. Autrement dit, il représente tous les couples de langues présentes et non pas seulement les couples dont la langue source est l'anglais. Ce point de vue est à nuancer, comme nous allons le voir à présent.

1.2.3.2.3 Polarité de la traduction et foisonnement

Dans une mémoire de traduction, le segment source a été rédigé librement, le segment cible a été traduit par un traducteur humain. Or la traduction est toujours sujette à une certaine asymétrie, à la fois distorsion et augmentation ou "foisonnement", que nous appellerons *polarité*. L'adage *traduttore, traditore* le dit bien !

Quelle est l'importance de cette polarité ? La différence de taille entre le texte source et le texte cible, de l'ordre de 15%, est un indice du foisonnement dû à la traduction, et donc de cette polarité. Il est cependant tentant d'envisager de réutiliser les mémoires de traduction dans le sens cible source, voire entre plusieurs langues cible.

Dans quelle mesure cela est-il possible ? Peut-on utiliser avec profit la connaissance contenue dans des mémoires de traduction pour faciliter la tâche du traducteur quand il ne

travaille pas à partir de la langue source (le plus souvent l'anglais, mais aussi le japonais etc.) de cette mémoire ? Un de nos objectifs est de nous doter d'un outil pour mesurer la réutilisabilité des mémoires de traduction.

1.2.3.3 Les mémoires sont spécialisées

1.2.3.3.1 Les mémoires sont liées à des projets

Les mémoires de traduction ont une origine. Elles ont été constituées, en général, lors d'un projet de traduction particulier. Une conséquence importante est qu'elles ne sont pas anonymes. On connaît leur histoire qui permet de se faire une idée de leur contenu. La décision d'utiliser une certaine mémoire de traduction pour un certain projet de traduction est fortement liée à cette connaissance extérieure.

Quand un document suit des évolutions, les mémoires de traduction qui ont été établies à l'occasion d'une traduction antérieure de ce même document sont considérées comme les plus pertinentes pour une nouvelle traduction de ce document.

1.2.3.3.2 Champ sémantique, domaine

Mais cette utilisation récurrente des mémoires de traduction n'est pas la seule possible. On peut décider d'utiliser une mémoire de traduction liée à un autre projet parce qu'on sait que le champ sémantique ou le domaine couvert par cette mémoire est proche de ceux du document à traduire.

Une certaine expertise est donc nécessaire, lors de la préparation du projet de traduction, pour rassembler les mémoires de traduction pertinentes, au-delà de la simple réutilisation d'une mémoire antérieure liée au même document. Cette même expertise est indispensable lors d'une première traduction.

1.2.3.3.3 Aspects terminologique et stylistique

Enfin, il faut souligner l'importance du bon choix des mémoires de traduction en fonction des impératifs terminologiques et stylistiques. La seule couverture du champ sémantique peut ne pas se révéler suffisante pour déterminer l'adéquation d'une mémoire particulière si les aspects terminologiques ou stylistiques ne sont pas pris en compte.

Il arrive que les choix terminologiques effectués lors de la traduction de documents différents divergent fortement, même si le domaine traité est semblable. C'est notamment le cas de produits différents mais similaires, éventuellement concurrents, pour lesquels la spécificité de ces aspects est primordiale.

Le choix d'une mauvaise mémoire de traduction peut ne pas être détecté assez tôt et provoquer néanmoins de grandes disparités dans le texte final, au détriment de la cohérence générale du texte.

1.2.3.4 Hétérogénéité des mémoires : un phénomène général

1.2.3.4.1 Différents types d'écrits

Les mémoires de traduction sont utilisées dans des contextes très variés. Les fichiers traités peuvent être de natures très différentes.

Par exemple, il peut s'agir de textes rédigés, la documentation d'un logiciel par exemple, ou l'aide proposée par le logiciel à l'utilisateur. Mais il peut aussi s'agir de l'interface utilisateur elle-même, à savoir les barres de menus, les fenêtres contextuelles, les libellés des champs de saisie, etc.

Dans ce cas, les fichiers traités sont, dans le meilleur des cas, constitués de listes de variables, structurées de façon diverses, voire de programmes en code source, dans lesquels se trouvent les segments à traduire.

1.2.3.4.2 Différents formats

Il existe donc différents formats de texte, pour lesquels ont été créées des tables de marquage adaptées.

Cependant, les mémoires de traduction contiennent aussi des segments non linguistiques. Les outils de segmentation, en dehors d'une supervision humaine, peuvent isoler de nombreux segments sans aucun contenu linguistique. Il peut s'agir de balisage pur, de balises XML par exemple.

En outre, le texte peut contenir des variables : un terme, une quantité, un lien hypertextuel etc. Ces situations conduisent éventuellement à une adaptation dynamique du texte source plus ou moins complexe en fonction de l'ingéniosité du rédacteur/programmeur. Ces automates sont rarement transposables tels quels en cours de traduction et leur expression peut être la source de segments inutilisables dans les mémoires de traduction.

1.2.3.4.3 Différents types de segments

Typiquement, les segments véritablement linguistiques sont des phrases. Mais on peut trouver aussi des termes simples, par exemple dans un index.

On peut aussi trouver des paragraphes entiers si la segmentation a été artificiellement revue dans ce sens. C'est par exemple le cas de textes juridiques, peu variables, délicats à rédiger comme à traduire, dont la formulation est identique à travers les produits et leurs versions successives. Lors de la traduction, ces segments donneront lieu à des « coïncidences exactes » ne nécessitant pas l'intervention du traducteur.

Il existe aussi des segments très fortement localisés, traduits de manières non standard. On trouvera par exemple une adresse à New York traduite par une adresse à Paris. Bien entendu, ces correspondances ne sont valables que dans leur contexte très particulier et ne peuvent être utilisés en dehors de lui.

1.3 Ambition : exploitation approfondie des mémoires de traduction

Ce chapitre expose les ambitions et les objectifs qui ont conduit à la création du formalisme proposé dans la deuxième partie. L'importance de l'indépendance des mémoires de traduction vis-à-vis des langues naturelles qu'elles supportent sera mise en évidence, la possibilité d'affiner la segmentation des unités sera ensuite détaillée. L'idée principale qui sous-tend notre démarche est d'arriver à établir des alignements plus fins que le segment entier.

1.3.1 Buts et hypothèses

Nous allons nous intéresser aux mémoires de traduction en tant que corpus bilingues alignés. Les mémoires de traduction représentent un ensemble de connaissances très riche qui semble incomplètement exploité. Les systèmes d'aide à la traduction humaine fondés sur les mémoires de traduction connaissent un réél succès ce qui a comme conséquence que ce matériau est maintenant abondant. L'objectif de cette thèse est de mieux exploiter les informations que contiennent les mémoires de traduction, sans avoir recours à d'autres ressources.

1.3.1.1 Restructurer les mémoires de traduction

1.3.1.1.1 Structurer les segments en sous-segments

Notre premier objectif consiste à repérer des sous-segments à l'intérieur des segments enregistrés dans les mémoires de traduction, dans les deux langues. Le critère de découpage de ces sous-segments visera à préserver une certaine autonomie. Une des hypothèses fondamentales du fonctionnement des mémoires de traduction est justement l'autonomie des segments.

Cette autonomie n'est pas parfaite : les segments dépendent les uns des autres ; ils sont liés à leur contexte d'apparition dans le document d'origine. Dans le cas de sous-segments, cette dépendance est vraisemblablement plus forte encore : les groupes de mots à l'intérieur du segment sont fortement dépendants des autres. En particulier, dans les langues flexionnelles, ils s'accordent morphologiquement les uns avec les autres. Cependant, certains mots, certains groupes de mots concentrent le sens plus que d'autres et forment comme des foyers de sens.

Nous voudrions mettre en évidence une structure de ces foyers au sein du segment et expliciter, dans une certaine mesure, les dépendances qu'ils entretiennent entre eux.

1.3.1.1.2 Structurer les correspondances sous-phrastiques

Dans une mémoire de traduction, dans un corpus bilingue en général, nous bénéficions d'une propriété importante : le sens des mots, des groupes de mots ou sous-segments, du segment en général, est reflété d'une langue à l'autre. C'est ce que nous appelons des correspondances. Une certaine structuration du sens dans chacun des deux segments est indissolublement liée à une structuration des correspondances elles-mêmes, au sein du couple formé par ces segments, le bisegment. Nous voudrions exploiter cette propriété.

L'avantage de ces structurations est de donner à un système d'aide à la traduction, et par delà, à son utilisateur, des points de repère pour formaliser différents éléments dans les exemples de traduction proposés. Ces exemples sont, dans les systèmes existants, assez peu documentés : tel segment se traduit par tel autre, au traducteur de comprendre pourquoi et ce qui est réutilisable, modifiable, etc.

En structurant des correspondances sous-phrastiques à l'intérieur du bisegment, nous ouvrons la porte à une compréhension beaucoup plus fine de ce qui s'y passe.

1.3.1.1.3 Dépolariser la traduction

Les systèmes actuels d'aide à la traduction fondés sur les mémoires de traduction distinguent systématiquement une langue source et une langue cible. Cela est lié à la granularité retenue, celle des segments entiers, associés en bisegments. En effet, à cette échelle, les effets stylistiques sont très importants, et sont contraints, dans le segment de la langue cible, par la formulation du segment de la langue source.

Une structuration plus fine des bisegments et des correspondances entre sous-segments permet d'envisager une forte diminution de cette polarité. À l'échelle de la terminologie, les correspondances sont largement dépolarisées. La polarisation, qui permet à l'humain de distinguer facilement le texte original et sa traduction, va croissant avec la taille des sous-segments envisagés.

On pourrait donc résumer notre approche en disant qu'il s'agit de montrer que :

on peut calculer des correspondances sous-phrastiques fines sans constituer manuellement de ressources linguistiques spécifiques,

à cette échelle, la polarité est assez faible pour qu'on puisse utiliser ces correspondances de façon inverse et même transverse, c'est-à-dire de langue cible à langue cible, le cas échéant.

Bien entendu, la polarité dépend aussi du génie propre de chaque langue, voire de chaque culture, à exprimer plus ou moins efficacement une même réalité. Certains termes, certaines tournures particulièrement efficaces dans une langue ne se rencontrent jamais dans une autre qui s'intéressera à un autre aspect de la même réalité ou exprimera un autre point de vue avec son propre génie.

1.3.1.2 Transformer les données pour les réutiliser

1.3.1.2.1 Constituer une base de connaissance

Travailler sur la restructuration des mémoires de traduction nous mènera à une analyse de leurs éléments constitutifs. Ces éléments, initialement assez subjectifs, seront « objectivés » de façon à les transformer en données que nous pourrons consigner à part pour les réutiliser, d'une part pour l'analyse de nouveaux segments ou bisegments, et d'autre part pour d'autres tâches.

Notre analyse des mémoires sera très largement statistique. Elle s'appuiera donc sur un grand nombre de données que nous consignerons dans une base de connaissances pour usage quasi-immédiat. Cette base de connaissances pourra également servir pour analyser d'autres mémoires de traduction, voire pour synthétiser des propositions de bisegments à partir d'un seul segment.

Nous nous attacherons donc à spécifier une formalisation homogène de cette base de connaissances.

1.3.1.2.2 Reconnaître des briques traductionnelles

Lors de l'analyse d'une mémoire de traduction, nous chercherons donc à structurer les correspondances entre « sous-segments ». Les sous-segments ainsi mis en correspondance seront choisis de façon à pouvoir constituer des « briques traductionnelles ». Une brique traductionnelle est suffisamment autonome par rapport à son contexte pour que les sous-segments qui y participent puissent être considérés comme étant en traduction mutuelle, à l'instar du bisegment entier.

L'intérêt des briques traductionnelles est d'être réutilisables dans des traductions autres que celles dont elles sont extraites. Elles peuvent être présentées au traducteur pour qu'il les insère dans son travail. Les briques traductionnelles peuvent donc être enregistrées pour elles-mêmes, et réutilisées indépendamment de leur bisegment d'origine.

1.3.1.2.3 Apprendre des patrons de traduction

À partir des briques traductionnelles, nous voudrions construire (par « abstraction ») des « patrons de traduction », ou schémas traductionnels. Par leur généricité même, ces patrons seront réutilisables non pas tellement par des traducteurs humains, mais bien par des traducteurs automatiques. Ils s'apparentent à des règles de grammaire établissant des ponts entre les deux textes en traduction mutuelle.

Ces patrons de traduction seront eux aussi consignés dans notre base de connaissances.

1.3.1.3 S'affranchir de la dépendance par rapport aux ressources linguistiques « manuelles »

1.3.1.3.1 Absence de connaissance extérieure au corpus

Une hypothèse forte qu'il convient de rappeler dès le début de cette présentation est que nous ne voulons pas, dans ce travail, dépendre de données autres que celles qui nous sont fournies par les mémoires de traduction vues comme un corpus. En particulier, nous n'avons pas cherché à utiliser de tagueur, de lemmatiseur ou d'analyseur morphosyntaxique.

Une petite restriction doit cependant être faite : nous connaissons l'ensemble des signes, alphabétiques, numériques et de ponctuation. En outre, nous filtrons les mémoires de traduction afin d'éliminer des segments non pertinents pour notre propos, par exemple, des segments contenant du code. Lors de ce filtrage, nous avons un certain présupposé sur ce qu'est un segment linguistiquement intéressant.

Ces restrictions ne nous semblent pas assez importantes pour remettre en cause notre idéal d'indépendance vis-à-vis de la langue traitée, une des clés du succès des environnements d'aide à la traduction fondés sur les mémoires de traduction.

1.3.1.3.2 Contenant : les bisegments

Nous considérons les bisegments comme l'enveloppe de ce que nous cherchons à modéliser. Autrement dit, nous ne nous intéressons pas à l'interaction des bisegments entre eux, et les supposons indépendants. Nous ne cherchons pas non plus à modéliser les mémoires en tant que telles, ni à étudier leur éventuelle structuration ou autre propriété suprasegmentale.

En outre, nous ne remettrons pas en cause l'alignement phrastique que représente un bisegment. C'est l'expertise humaine concrétisée par cet alignement que nous cherchons à modéliser. De même, nous ne chercherons pas à intégrer les balises de mise en forme dans nos données. Ces aspects sont le plus souvent traités par d'autres modules, qui savent tenir compte de l'éventuelle incomplétude au niveau d'un segment (balise non fermée, etc.).

En revanche, nous filtrons le corpus pour ne garder que les bisegments qui ressemblent à des phrases, dans les deux langues. Comme nous l'avons vu, les mémoires sont très hétérogènes et l'on y trouve de tout : les segments permettant une modélisation de la traduction du langage ne sont pas toujours les plus nombreux dans une mémoire de traduction.

Enfin, nous avons limité notre démarche au cas de deux langues, même si, comme nous le verrons, TransTree s'étend immédiatement à la description de correspondances entre plus de deux langues.

1.3.1.3.3 Contenu : les mots typographiques

La construction que nous avons imaginée ne s'applique qu'aux langues dont le système d'écriture contient des séparateurs permettant de délimiter des mots. Corrélativement, la granularité que nous avons adoptée dans notre démarche est le « mot typographique* ». Nous avons choisi le terme de mot typographique pour éviter toute ambiguïté sur la notion de « mot ».

Un mot typographique est une suite finie de signes appartenant à un ensemble fini connu, et délimitée par des séparateurs appartenant eux-mêmes à un ensemble fini connu. Dans la pratique, les mots typographiques sont délimités par des blancs, des signes de ponctuation, ou les extrémités gauche et droite des segments. Les caractères de retour chariot sont considérés comme des blancs, et deux blancs de suite sont considérés comme un seul.

1.3.2 Point de vue épistémologique

Les travaux de modélisation et d'exploitation des mémoires de traduction proposés ici sont sous-tendus par une approche volontairement empirique. Une certaine conception de la nature des gisements potentiels d'informations nouvelles et de ces informations elles-mêmes constitue le substrat de cette démarche. Après avoir posé l'existence d'un lien ontologique exploitable entre compression de données et cognition, nous donnerons trois axes d'acquisition de connaissance, inspirés de ceux établis par la linguistique traditionnelle. Enfin, nous examinerons l'intérêt des correspondances sous-phrastiques établies statistiquement au sein des bisegments, même si elles sont peu corroborées par l'intuition linguistique.

1.3.2.1 Compression et cognition

1.3.2.1.1 Rechercher des régularités, factorisation

Notre démarche adhère au principe d'économie suivant : une modélisation efficace de tout phénomène physique doit minimiser (c'est à dire rendre le plus petit possible) le nombre de données nécessaires pour décrire le phénomène en question.

En informatique, nous pouvons assimiler ce point de vue à la compression de données. Dans les deux cas, il s'agit de réduire la redondance. Nous chercherons donc à factoriser le plus possible les éléments d'information accumulés au cours de l'analyse du corpus.

Notre travail consiste alors à analyser le corpus dont nous disposons de telle sorte que les observations collectées au cours de cette analyse soient utilisables dans le cadre de l'analyse elle-même d'une part, et pour des applications différentes d'autre part.

1.3.2.1.2 Cognition

Compression et cognition sont-elles deux aspects d'un même processus ? C'est ce que laisse penser Gerry Wolff dans : http://www.cognitionresearch.org.uk/cognition.html. Nous pensons avec lui qu'une partie au moins de la faculté cognitive humaine réside dans la capacité à factoriser des informations. Cette factorisation permanente d'informations nouvelles établit peu à peu le système de référence permettant d'intégrer de nouvelles informations.

Nous aurons ainsi recours à la classification systématique des éléments dégagés au cours de l'apprentissage statistique mené sur le corpus bilingue. Cette classification nous sera utile dans la constitution du modèle de langage. Nous aurons également recours au regroupement de ces éléments dans l'ordre linéaire où ils se présentent, grâce à des arbres binaires statistiquement motivés.

1.3.2.1.3 Occurrences et prototypes

Nous distinguerons dans tout ce qui suit les « occurrences* » et les « prototypes* ». Cette notion est délicate parce qu'on identifie une occurrence de quelque chose quand on a déjà son prototype en tête, et qu'inversement on définit le prototype par abstraction de toutes les occurrences.

Pour fixer les idées, prenons, par exemple, l'ensemble de tous les caractères du corpus. Le cardinal de cet ensemble est quasi-proportionnel à la taille du corpus. Il s'agit d'un ensemble d'occurrences. L'ensemble des caractères différents utilisés dans le corpus constitue, lui, un ensemble de prototypes. Son cardinal est possiblement lié à la taille du corpus, mais pas linéairement, en général. Le 12983 ème caractère du corpus est une occurrence de caractère. Il s'agit peut-être d'une occurrence du caractère 'C', considéré cette fois comme prototype.

Bien entendu, cette distinction ne s'arrête pas aux seuls caractères. Nous la retrouvons pour pratiquement pour tous les objets rencontrés dans un corpus. Ainsi, le mot « ordinateur » est un prototype quand il est évoqué en général. Il en existe une ou plusieurs occurrences dans un texte particulier. Lorsqu'il n'existe qu'une seule occurrence d'un prototype, on parle d'« hapax ». Dans un corpus qui ne serait constitué que d'hapax, la distinction entre prototypes et occurrences n'aurait plus lieu d'être.

Considérer des prototypes constitue en soi une factorisation implicite des données, qui s'appuie sur la forme des occurrences : toutes les occurrences d'un prototype sont identiques par leur forme. Nous allons au cours de notre démarche fabriquer des objets nouveaux, pour lesquels cette distinction entre occurrences et prototype s'appliquera. La compression des données consiste, de ce point de vue, à satisfaire une double contrainte : augmenter le nombre d'occurrences et diminuer le nombre de prototypes.

1.3.2.2 Démarche méta-linguistique

1.3.2.2.1 Axe interlingue

Nous cherchons à exploiter les phénomènes co-occurrents entre les deux langues. Il s'agit des ponts interlingues qu'on peut établir en tirant parti de la comparaison systématique des segments en traduction réciproque. Ce faisant, nous chercherons à distinguer des éléments apparemment identiques en surface dans une langue, mais dont on constate qu'ils donnent lieu à des traductions différentes. On pourra considérer qu'ils sont donc homographes, si l'on s'en tient à leur forme écrite, et hétéronymes, si l'on considère que la traduction nous informe sur leur sens profond.

Le but de toute la démarche consiste à mettre en évidence des *briques traductionnelles* et, par abstraction, des *schémas de traduction*. Ce type d'information nous permettra non seulement d'identifier des équivalences de surface, mais également d'isoler des structures plus profondes.

Un autre aspect important de l'axe interlingue est qu'il permet de renforcer certaines observations faites dans une langue par d'autres observations établies dans une autre langue, qui viennent se conforter mutuellement. Notre démarche utilise ce principe à plusieurs reprises.

1.3.2.2.2 Axe syntagmatique

Notre deuxième axe d'information est de nature syntagmatique. Nous voulons reconnaître l'agencement des unités sur l'axe linéaire de l'énoncé, c'est à dire l'ordre linéaire des objets, et la structuration des segments. Nous cherchons à mettre en évidence des sous-segments, ayant une certaine autonomie dans le mécanisme de traduction. Ces sous-segments s'apparentent à des syntagmes et permettent d'ancrer des dépendances à distance dans l'énoncé.

Bien entendu, rien dans un texte n'est complètement indépendant du reste, ni un mot, ni un syntagme, ni même une phrase. Corrélativement, chaque partie d'un texte source est plus ou moins liée à toutes les parties d'un texte cible.

Pour rendre compte statistiquement de ces dépendances, nous structurerons chacun des segments d'un bisegment en un arbre binaire, de façon coordonnée.

1.3.2.2.3 Axe paradigmatique

Enfin, nous chercherons le plus possible à factoriser les informations apprises dans le corpus en regroupant les unités rencontrées. Notre but est d'une part de contrer la rareté intrinsèque des données dans un corpus linguistique et d'autre part de dégager des règles par généralisation des instances rencontrées dans le corpus.

On entend classiquement par paradigme, en linguistique, un ensemble de formes différentes obtenues par dérivation morphologique d'un même lemme et porteuses d'un ensemble de traits morpho-syntaxiques prédéfinis.

Nous ne mettrons pas en évidence dans ce travail des paradigmes au sens strict comme on vient de l'évoquer. Nous chercherons simplement à regrouper les formes observées et collectées dans le corpus en différents ensembles et à caractériser ces ensembles par leur contexte syntaxique dans les bisegments.

1.3.2.3 Granularité et compositionnalité

1.3.2.3.1 Principe

La « granularité » désigne la taille et le type des éléments terminaux. Elle est à l'analyse de texte ce que la définition est à l'image. Pour nous, ce sera, au départ, le mot typographique, c'est à dire la graphie séparée par des espaces ou des signes de ponctuation.

La compositionnalité est généralement entendue au sens de la calculabilité du sens. En l'occurrence, nous nous intéresserons à la calculabilité de la traduction.

Quel est le degré de granularité souhaitable pour modéliser un phénomène ? C'est une question que l'on retrouve souvent en linguistique, où la définition des entités fondamentales constitue un problème récurrent.

Nous essaierons de nous tenir à l'idée selon laquelle la granularité optimale s'évalue en fonction de la compositionnalité des éléments.

Nous voulons éviter d'enregistrer des observations calculables, et donc redondantes, tout comme nous ne voulons pas conserver d'observation ne permettant aucun calcul. Pour déterminer la granularité que nous voulons atteindre à une étape quelconque du calcul, nous chercherons donc à préserver la compositionnalité des éléments manipulés.

1.3.2.3.2 Mots typographiques et signes

L'identification et la nature des éléments manipulés ne sont donc pas données *a priori*, comme fruit d'une connaissance monolingue. Au contraire, nous chercherons le plus possible à repérer les éléments fondamentaux par confrontation des deux textes.

Pour initier le processus d'apprentissage, nous nous fonderons sur les mots typographiques quand le système d'écriture s'articulera sur des mots typographiques.

Nous avons également exploré la possibilité de partir des signes eux-mêmes, qu'il s'agisse de signes alphabétiques ou idéographiques, mais n'avons pas encore de résultats probants dans cette situation.

1.3.2.3.3 Dépolarisation

Un autre critère important pour déterminer la granularité tient dans la dépolarisation du bitexte traité. En effet, la polarisation est moindre pour les éléments lexicaux que pour les segments complets.

Cela veut dire que les éventuelles particularités stylistiques permettant au lecteur humain de déterminer le texte original et sa traduction sont plus importantes à l'échelle des segments entiers qu'à celle des mots ou des syntagmes.

Pour utiliser dans les deux sens des informations traductionnelles extraites d'un bitexte, il nous faudra donc utiliser une granularité beaucoup plus fine que les segments entiers.

1.3.3 Bénéfices attendus

Ces principes étant posés, il est important de préciser les bénéfices concrets recherchés. Nous en avons poursuivi plusieurs. Nous voudrions d'abord fixer des orientations de recherche dans l'exploitation de corpus bilingues. Notre ambition est également d'enrichir des propositions de

traduction à partir du matériau disponible, voire d'en synthétiser de nouvelles afin d'assister plus efficacement le traducteur dans son travail de traduction. Enfin, nous voudrions nous doter de métriques pour évaluer la richesse d'une mémoire de traduction, tant intrinsèquement que relativement à une tâche de traduction donnée.

1.3.3.1 Mise en place d'un paradigme de recherche

1.3.3.1.1 Modélisation des correspondances sous-phrastiques

Comment représenter et formaliser les correspondances bilingues sous-phrastiques observées dans un corpus ? Peut-on pérenniser ces correspondances dans un document ? Peut-on partager cette représentation entre plusieurs acteurs, plusieurs programmes ? Un objectif majeur de ce travail est de construire un formalisme de représentation des correspondances sous-phrastiques.

Une des premières nécessités est donc d'identifier un langage de représentation de données largement répandu, standardisé et suffisamment stabilisé. XML répond à ces critères et nous l'avons utilisé. Nous avons d'autant plus volontiers adopté XML que, dans ce langage, les spécifications d'un modèle de données se font avec souplesse et sans ambiguïté. En outre, une structuration arborescente et récursive est particulièrement aisée en XML, ce qui convient à la représentation des résultats de l'analyse que nous cherchons à mener. Enfin, XML permet de mélanger librement des contenus textuels et structurés. Nous avons largement fait usage de cette propriété.

Le résultat de notre modélisation est TransTree, qui propose une représentation arborescente de correspondances gigognes dans un bisegment. La présentation détaillée de TransTree est faite au chapitre 2.1.

1.3.3.1.2 Outils algorithmiques et méthodes

Le deuxième objectif consiste à se doter d'outils pour identifier et structurer des correspondances sous-phrastiques conformes à la modélisation TransTree. Nous avons abordé plusieurs directions, que nous avons qualifiées de métalinguistiques (cf. 1.3.2.2).

En effet, il nous a semblé utile de reprendre les méthodes de travail liées à la modélisation de la langue par les linguistes. C'est ainsi que nous avons repris les notions d'axe syntagmatique et d'axe paradigmatique. Nous leur avons adjoint un axe 'interlingue', qui exploite et modélise la confrontation de deux énoncés de même sens en deux langues.

Ces trois axes suggèrent trois types de méthodes :

recherche de correspondances élémentaires dans le segment (axe interlingue)

structuration de l'agencement des unités dans un bisegment (axe syntagmatique)

factorisation des connaissances et recherche de patrons (axe paradigmatique)

L'utilisation de ces trois axes est en pratique plus diffuse que cette présentation schématique pourrait le laisser croire. En particulier, l'axe interlingue intervient largement dans la structuration des bisegments comme dans la factorisation des connaissances. Les détails de ces méthodes et les algorithmes mis en œuvre sont décrits au chapitre 2.2.

1.3.3.1.3 Stratégie de mise en œuvre

Après nous être doté d'outils d'analyse des données, il nous restera à imaginer une stratégie d'utilisation. Un des problèmes difficiles rencontrés est de trouver une stratégie convergente, c'est-à-dire qui permette un raffinement progressif des connaissances acquises. Les différents modules liés aux axes que nous avons décrits précédemment sont fortement interdépendants. Autrement dit, la difficulté tient au « bootstrapping » : comment et par où commencer ?

Cet aspect nous a longuement préoccupé et nous avons finalement élaboré plusieurs stratégies pour combiner les différents axes d'acquisition de connaissances. Une méthode complètement réentrante, qui s'enrichirait continûment au cours du temps, reste encore à mettre au point.

Les détails des stratégies mises en place sont donnés en 2.3**.**

1.3.3.2 Méthodes d'alignement sous-phrastiques

1.3.3.2.1 Proposition d'alignements sous-phrastique

Tous les bénéfices concrets liés à notre travail reposent sur la notion d'alignement sous-phrastique. C'est à partir de cet objectif fondamental que nous avons élaboré diverses directions d'investigation.

Mais l'alignement sous-phrastique est aussi un but en soi, directement exploitable par un utilisateur humain. En effet, le traducteur qui travaille dans un environnement d'aide à la traduction fondé sur les mémoires de traduction se voit proposer des segments entiers dont il sait qu'« ils ne sont pas tout à fait sans rapport » avec la tâche de traduction immédiate qu'il a à remplir. Il va donc lire ces propositions, dans la langue cible, et s'efforcer de comprendre en quoi ce matériau est intéressant pour lui. Essentiellement, cela consiste pour lui à repérer les sous-segments communs à la phrase qu'il a à traduire et la traduction qu'on lui propose. Cette identification peut être simple, mais pas toujours.

En lui présentant d'emblée des propositions sous-segmentées, dans lesquelles les parties en correspondances sont identifiées, nous améliorons l'ergonomie de ces systèmes. Cet aspect de notre travail est détaillé dans le chapitre 2.1.2.

1.3.3.2.2 Exploitations inverse et transverse

Comme nous l'avons précisé, l'effort de traduction mené à IBM, par exemple, est largement multicible. Cela signifie que le même texte en langue source est traduit dans plusieurs langues.

Nous appelons exploitation « inverse » des mémoires de traduction la possibilité d'utiliser des mémoires de traduction dans le sens de la langue cible vers la langue source et exploitation « transverse », la possibilité de les utiliser d'une langue cible vers une autre langue cible.

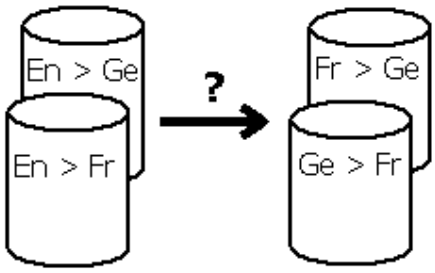

Figure 14 : problème de l'exploitation en sens « transverse »

Exploiter des mémoires de traduction en sens inverse, de la cible vers la source, par segments entiers, est très largement voué à l'échec, étant donné que la langue utilisée en traduction est toujours tant soit peu artificielle. Peu de segments cible, fruits d'une traduction humaine, sont susceptibles d'apparaître tels quels dans un document rédigé librement dans la langue cible. Par contre, travailler au niveau des sous-segments semble possible.

En outre, une mémoire de traduction est fortement liée au document, voire à l'ensemble de documents, qui lui a donné naissance. Il n'est pas vraisemblable, dans un contexte industriel, que ces documents soient traduits en sens inverse pendant leur cycle de vie. Il faudrait que les originaux en langue source aient disparu.

1.3.3.2.3 Analyse ou synthèse de bisegments quelconques

Par bisegment quelconque, nous entendons un bisegment qui n'appartient pas à notre corpus de travail, les mémoires de traduction. Il peut s'agir d'un bisegment complet mais non structuré en correspondances sous-phrastiques. Il s'agit donc de l'analyser. Plus ambitieusement, il peut s'agir d'un segment unique pour lequel nous voudrions calculer un segment en correspondance. Ce deuxième point s'apparente à de la traduction automatique.

Dans les deux cas, nous allons mettre à profit la base de connaissances établie lors de l'analyse de notre corpus de référence. Autrement dit, les observations faite dans le corpus doivent être suffisamment génériques pour s'appliquer à des segments non rencontrés dans le corpus.

Dans ce travail, nous donnerons les directions et méthodes que nous avons imaginées et mises en œuvre dans la perspective de cette vaste ambition.

1.3.3.3 Autres bénéfices possibles

1.3.3.3.1 Amélioration des calculs de distance

La détermination des coïncidences floues, les « fuzzy matches » est fondamentalement liée à la performance des environnements d'aide à la traduction fondés sur les mémoires de traduction. Les propositions faites au traducteur doivent être suffisamment pertinentes pour l'aider effectivement. Le but n'est pas de lui faire perdre du temps.

La notion de distance entre deux segments est primordiale dans ce calcul. Il s'agit, de fait, d'un secret industriel jalousement gardé par les éditeurs. Cette distance est plus ou moins inspirée de la distance d'édition, qui est le nombre minimal d'insertions et de suppressions de caractères (ou de mots entiers) nécessaires pour passer d'un segment à l'autre.

En structurant les bisegments, nous pourrons apporter une information supplémentaire, pertinente dans ce calcul. En effet, au-delà des mots ou des caractères, nous pouvons envisager de décompter les éléments de notre structure sous-segmentale à substituer pour passer d'un segment à l'autre. Cette approche est semblable à celle d'Emmanuel Planas dans [27] nous pensons, comme lui, que cette approche permettrait d'améliorer la pertinence des propositions faites au traducteur et de préparer le terrain pour d'éventuelles substitutions automatiques.

1.3.3.3.2 Mesure de la richesse d'une mémoire de traduction

Une mémoire de traduction est plus ou moins volumineuse, plus ou moins redondante, plus ou moins cohérente etc. Certains aspects, comme le volume sont directement accessibles. D'autres nécessiteraient une analyse plus poussée pour être évalués.

En outre, le gain de temps escomptable au moment d'utiliser une mémoire est mal connu. La proximité des coïncidences floues, comme nous venons de le voir, est essentiellement fondée sur la notion de distance d'édition en surface, ce qui donne une indication floue elle aussi sur l'éventuel gain de productivité.

Une analyse sous-phrastique, parce qu'elle permettrait d'évaluer plus finement le contenu des mémoires de traduction procurerait des éléments de réponse pour explorer ces dimensions.

1.3.3.3.3 Utilisation de plusieurs langues

Les langues naturelles sont toujours ambiguës. L'échange langagier dépend largement d'une préconception du monde (et de la langue) de la part des interlocuteurs. Sous leur forme écrite, souvent plus riche que le signal oral, ses ambiguïtés demeurent néanmoins. Elles constituent un obstacle majeur dans le domaine de la traduction automatique.

Parmi les ambiguïtés classiquement étudiées par les linguistes, rappelons la résolution des anaphores, de la coordination, de la structure de complémentation etc. À ces difficultés s'ajoutent l'ambiguïté lexicale, voire l'ambiguïté syntaxique ou même, dans le cas de l'oral, l'analyse du signal acoustique.

Cependant, lors de la traduction d'un énoncé par un traducteur humain, une grande partie de ces ambiguïtés est spontanément levée par ce dernier, grâce à sa connaissance du monde, à son bon sens. Traduire un texte exprimé dans deux langues différentes vers une langue tierce semble être une bonne direction pour lever une grande partie de ces ambiguïtés et améliorer du même coup la qualité de la traduction obtenue. L'étude de l'exploitation de cette information complémentaire est encore balbutiante. On trouvera néanmoins des informations chez Och et Ney : [28].

Une structuration automatique des correspondances entre trois segments « synonymes hétéroglottes » permettrait d'envisager l'exploitation de la richesse cognitive introduite par le traducteur humain.

2 Une solution : le modèle TransTree d'alignement sous-phrastique

Cette deuxième partie est entièrement consacrée à la mise en place des propositions que nous faisons pour mieux exploiter les mémoires de traduction. Dans un premier temps, nous détaillerons TransTree, un formalisme d'alignement sous-phrastique, multilingue et récursif, que nous proposons pour structurer les données apprises automatiquement à partir des mémoires de traduction. Dans un deuxième temps, nous exposerons les méthodes d'acquisition automatiques des éléments constitutifs de ce modèle à partir du corpus. Enfin, plusieurs solutions pour structurer ces éléments constitutifs conformément au modèle seront proposées.

2.1 Proposition : le formalisme TransTree

Le formalisme TransTree est au cœur de notre travail de recherche. Il est le fruit de nombreux tâtonnements dans notre effort pour structurer les correspondances interlingues au sein de bisegments. Ce formalisme a pour objectif de représenter ces correspondances d'une part et des schémas traductionnels d'autre part. Nous en proposons une expression XML précise, qui nous servira de référence dans toute la suite de ce travail. Une visualisation dynamique interactive est également proposée, qui donne un aperçu de ce que pourrait être une présentation ergonomique de TransTree intégrée à un environnement d'aide à la traduction humaine.

2.1.1 Premières définitions

L'objectif principal de la modélisation TransTree est la manipulation de correspondances « gigognes » précisant l'agencement simultané des unités dans les deux langues. C'est le sens que nous avons voulu souligner en adoptant l'appellation « TransTree » (« trans » comme « translation » ou « traduction » en français- et « tree » signifiant « arbre »). Nous espérons que ce nom véhicule l'idée d'une unique structure arborescente contenant des informations relatives à des bisegments exprimés en deux langues ou plus.

2.1.1.1 Architecture et expression XML

2.1.1.1.1 Amphigramme atomique

Le formalisme TransTree que nous utilisons pour représenter des correspondances fines dans le corpus multilingue s'appuie sur une construction arborescente. Nous présenterons d'abord le formalisme pour le cas d'un bitexte.

Nous appelons « amphigramme* » une correspondance à un niveau quelconque de cette construction arborescente. Un arbre d'amphigrammes, autrement dit, de correspondances traductionnelles gigognes, nous renseigne sur les correspondances entre les deux énoncés et l'ordre dans lequel elles s'agencent dans les deux langues.

Les correspondances élémentaires entre chaînes graphiques sont appelées « amphigrammes atomiques* » parce qu'elles ne sont pas subdivisées en correspondances plus fines. Dans un amphigramme atomique, il n'y a pas d'indication concernant l'agencement des composants, alors que c'est le cas dans un amphigramme en général.

N'importe quelle paire de chaînes textuelles appartenant à deux langues différentes, ici l'anglais et le français, peut constituer un amphigramme atomique. Nous supposons que les chaînes de caractères sont liées sémantiquement, mais cela n'est pas requis par le formalisme lui-même. Typiquement, les amphigrammes atomiques encodent des correspondances du type : **(dialog, dialogue)**[5].

L'expression XML[6] d'un amphigramme atomique est la suivante :

```
<amphigram type='atomic'>
  <text xml:lang='en'>dialog</text>
  <text xml:lang='fr'>dialogue</text>
</amphigram>
```

Figure 15 : un amphigramme atomique en XML

Cette représentation utilise un élément **<amphigram>** dont l'attribut **type** est optionnel. Il permet de préciser le type de l'amphigramme, ici : 'atomic'. Cet élément contient deux chaînes de caractères dans les balises **<text>**. Nous les appellerons « éléments textuels ». L'attribut XML standard **xml:lang** permet de spécifier la langue du contenu de l'élément textuel. Il n'y a pas de contrainte particulière sur le contenu des éléments textuels. On peut y trouver de la ponctuation, des espaces ou même des balises de mise en page, à condition d'utiliser un mécanisme d'échappement pour ne pas interférer avec la syntaxe XML. Il se peut que les textes soient mal délimités ou même sans aucun rapport linguistique. Il n'y a pas de nécessité de spécifier ce genre de contraintes pour l'instant.

2.1.1.1.2 Amphigramme complexe

Nous considérons maintenant les amphigrammes en général. Un amphigramme permet non seulement de mettre en correspondances deux sous-segments, mais encore de décrire les correspondances internes à ce sous-segment. Pour ce faire, un amphigramme inclut d'autres amphigrammes et forme avec eux une structure arborescente. Il est lui-même éventuellement inclus dans un autre amphigramme. Les feuilles de cette arborescence sont les amphigrammes atomiques. Ce que nous appelons amphigramme en général est donc un nœud de l'arborescence et, par extension, le sous-arbre qu'il domine.

Nous distinguerons différents types d'amphigrammes. Nous avons déjà vu ce qu'est un amphigramme atomique. L'amphigramme que nous décrivons ici est un « amphigramme

[5] L'ordre des langues est arbitraire, puisque nos objets sont symétriques. Nous nous tiendrons à l'ordre alphabétique des noms anglais des langues considérées.
[6] Nous avons adopté la langue anglaise, concise et non accentuée, dans l'expression XML de TransTree.

complexe* ». Il s'oppose en cela à l' « amphigramme compact* » et à l'« amphigramme générique* » que nous verrons plus tard, en 2.1.3.

L'exemple suivant illustre la structure d'un amphigramme complexe. Il s'agit d'encoder les correspondances entre les chaînes « dialog box » en anglais et « boîte de dialogue » en français.

```
<amphigram type='complex' loc='X'>
   <text xml:lang='en'><a loc='A'/> <a loc='B'/></text>
   <text xml:lang='fr'><a loc='B'/> de <a loc='A'/></text>
   <amphigram loc='A'>
      <text xml:lang='en'>dialog</text>
      <text xml:lang='fr'>dialogue</text>
   </amphigram>
   <amphigram loc='B'>
      <text xml:lang='en'>box</text>
      <text xml:lang='fr'>boîte</text>
   </amphigram>
</amphigram>
```

Figure 16 : un amphigramme complexe en XML

En général, un amphigramme contient non seulement des éléments textuels **<text>** mais aussi d'autres éléments **<amphigram>** fils. Le contenu des éléments textuels est mixte, c'est-à-dire qu'il contient à la fois du texte et des balises XML. En l'occurrence, les balises spéciales, **<a/>**, identifient les « points d'insertion* » des amphigrammes fils pour chaque langue. Nous les appelons des « monobalises d'insertion* ». Elles portent un attribut **loc**, repris par les amphigrammes fils pour rendre compte de l'agencement des correspondances fines. Nous appelons cet attribut « indicateur-repère local* ».

Les deux textes et les liens de correspondance entre les points d'insertion constituent l'« armature textuelle* » de l'amphigramme. Les amphigrammes fils ne font pas partie de l'armature textuelle.

L'attribut **type** des éléments **<amphigram>** est optionnellement indiqué. Il est dicté par la structure même de l'amphigramme et par conséquent redondant.

Le diagramme suivant permet de mieux comprendre cette représentation XML. Nous le présenterons avec plus de détails en 2.1.1.2.

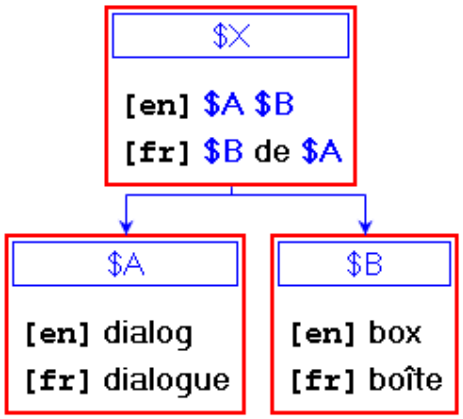

Figure 17 : dagramme d'un amphigramme complexe

Dans cet exemple, il y a trois amphigrammes en tout : un amphigramme complexe et deux amphigrammes atomiques fils.

Les amphigrammes atomiques encodent les deux paires **(box, boîte)** et **(dialog, dialogue)**. Ces derniers portent un indicateur-repère local marqué ici avec un dollar ($). Cette marque identifie aussi les points d'insertion au sein des éléments textuels de l'amphigramme complexe. L'ordre de présentation des amphigrammes fils est arbitraire puisqu'ils sont repérés par les indicateurs-repères locaux.

Nous avons supposé que cet amphigramme faisait partie d'un amphigramme plus large, à l'emplacement 'X'.

L'une des particularités de XML est sa capacité à encoder du contenu mixte, fait de texte et de balises. C'est ce qui nous permet de glisser la préposition "de" dans le texte français où elle prend place naturellement. Les éléments textuels, comme on le voit dans la partie anglaise de l'amphigramme ci-dessus, ne contiennent pas nécessairement de texte à proprement parler : ils peuvent être réduits à des monobalises d'insertion et à des espaces ou autres séparateurs.

La monobalise d'insertion **<a/>** et son attribut **loc**, indicateur-repère local, sont soumis à certaines contraintes :

1. Quel que soit leur ordre, il doit y avoir le même nombre de monobalises d'insertion **<a/>** dans chaque élément textuel **<text>**,
2. Un amphigramme admet autant d'amphigrammes fils que de monobalises d'insertion **<a/>** dans chacun de ses éléments textuels **<text>**,
3. Les indicateurs-repères locaux, attributs **loc** des monobalises d'insertion, doivent avoir des valeurs différentes au sein d'un même élément textuel **<text>**.

Une conséquence est que, pour une valeur donnée, l'indicateur-repère local apparaît trois fois : une fois dans chaque éléments textuels, où il repère un point d'insertion, et une fois dans un amphigramme fils, pour indiquer quel amphigramme utilise ces points d'insertion.

2.1.1.1.3 Caractérisation de TransTree

Nous pouvons donner maintenant une caractérisation d'un arbre TransTree. Il s'agit d'un bi-arbre abstrait, orienté et non ordonné, dont les nœuds sont des amphigrammes.

Il est "abstrait" car on ne peut pas lire directement les deux chaînes en correspondance par un parcours trivial[7]. L'arbre que nous construisons est double : il s'agit d'un arbre unique qui en contient deux, au sens où on peut générer à partir de lui un arbre (n-aire), cette fois-ci concret, pour chacun des deux segments en correspondance. Il est non ordonné parce que les deux langues sont dans des rôles symétriques. Par conséquent, l'ordre "horizontal" des nœuds n'a pas de signification intrinsèque, et l'ordre dans lequel il doit être lu est différent pour chaque langue. Les indicateurs-repères locaux véhiculent l'information nécessaire.

L'exemple suivant montre l'utilisation de TransTree pour encoder les correspondances au sein d'un bisegment constituant une énoncé complet. Dans une mémoire de traduction, nous avons le bisegment suivant :

```
<bisegment>
  <source>This task shows you how to change views.</source>
  <target>Dans cette tâche, vous apprendrez à modifier les
vues.</target>
</bisegment>
```

Figure 18 exemple de bisegment rencontré dans une mémoire

Le phrasé de ce bisegment est très dissemblable entre la version anglaise et la version française. Plus précisément, la première partie de la phrase anglaise est rendue de façon complètement différente en français. Nous pouvons représenter cela de la manière suivante :

[7] Un tel parcours suppose toujours un ordre entre les fils d'un même père, par exemple, le parcours des feuilles d'un arbre de constituants, ou le parcours infixé d'un arbre de dépendances syntaxiques.

```
<amphigram>
   <text xml:lang='en'><a loc='A'/>  <a loc='B'/> . </text>
   <text xml:lang='fr'><a loc='A'/>  <a loc='B'/> . </text>
   <amphigram loc='A'>
      <text xml:lang='en'><a loc='A'/>  <a loc='B'/></text>
      <text xml:lang='fr'><a loc='A'/>  <a loc='B'/></text>
      <amphigram loc='A'>
          <text xml:lang='en'>This  <a loc='A'/>  shows  <a loc='B'/>  how</text>
          <text xml:lang='fr'>Dans cette  <a loc='A'/>,  <a loc='B'/>
apprendrez</text>
        <amphigram loc='A'>
           <text xml:lang='en'>task</text>
           <text xml:lang='fr'>tâche</text>
        </amphigram>
        <amphigram loc='B'>
           <text xml:lang='en'>you</text>
           <text xml:lang='fr'>vous</text>
        </amphigram>
      </amphigram>
      <amphigram loc='B'>
         <text xml:lang='en'>to</text>
         <text xml:lang='fr'>à</text>
      </amphigram>
   </amphigram>
   <amphigram loc='B'>
      <text xml:lang='en'><a loc='A'/>  <a loc='B'/></text>
      <text xml:lang='fr'><a loc='A'/>  les  <a loc='B'/></text>
      <amphigram loc='A'>
         <text xml:lang='en'>change</text>
         <text xml:lang='fr'>modifier</text>
      </amphigram>
      <amphigram loc='B'>
         <text xml:lang='en'>views</text>
         <text xml:lang='fr'>vues</text>
      </amphigram>
   </amphigram>
</amphigram>
```

Figure 19 amphigramme complexe en XML

On trouvera un diagramme de cet amphigramme ci-dessous, en 2.1.1.2. Cet exemple a été construit automatiquement grâce à la méthode décrite dans le deuxième chapitre (voir 2.2). Les sous-segments "This task shows you how to" et "Dans cette tâche, vous apprendrez à" n'ont pas subi une analyse très poussée étant donné la grande disparité des expressions.

De nouveau, nous constatons que les éléments textuels ne contiennent pas toujours du texte à proprement parler : ils peuvent être réduits à une succession de monobalises d'insertion, séparés par des blancs ou des signes de ponctuation.

Le point important est que cette correspondance pourrait être utilisée par un traducteur alors qu'il serait très peu vraisemblable qu'elle soit produite par un système de traduction automatique. Il faut aussi remarquer que la virgule a été très simplement prise en compte.

Nous disposons à présent d'un arbre qui décrit les deux segments simultanément. La structure de chaque segment est déterminée par sa contrepartie dans l'autre segment. Un amphigramme peut contenir un nombre quelconque d'amphigrammes gigognes. Les feuilles de l'arbre sont les amphigrammes atomiques.

2.1.1.2 Représentation graphique de TransTree

2.1.1.2.1 Motivation

TransTree est donc une modélisation arborescente des alignements sous-phrastiques dans un bisegment. Il nous a paru intéressant de représenter graphiquement cet arbre afin de mettre en évidence sa structure. Il s'agit d'un arbre unique, dont la lecture est double (voire multiple s'il y a plus de deux langues) : il existe une lecture pour chaque langue dans laquelle l'énoncé est exprimé.

Cette représentation graphique devait pouvoir être générée automatiquement à partir de l'expression formalisée des correspondances, c'est-à-dire l'expression XML. La méthode qui nous a semblé la plus naturelle pour ce faire est d'utiliser le langage de transformation XSL associé à un langage de représentation graphique fondé lui-même sur XML. Nous avons choisi le Vector Markup Language (VML) de MicroSoft qui, à notre connaissance est le seul qui puisse être librement intégré à la volée dans un document HTML. Ce point est important dans la mesure où la disposition des nœuds de l'arbre repose, dans notre méthode, sur l'élément <table> du langage HTML, dont la disposition est calculée par le système en temps réel. Cette richesse permet de simplifier grandement le calcul de la disposition des nœuds de l'arbre dans le plan de l'écran.

Il existe un autre langage de représentation graphique fondé sur XML, Scalable Vector Graphics (SVG) qui ne permet pas cette liberté. Au contraire de SVG, VML semble n'avoir jamais été normalisé bien qu'il soit toujours présenté par le consortium W3C sur son site Web (http://w3c.org/vml.html). De fait, VML n'est plus utilisé que par MicroSoft à l'heure actuelle. Cependant, nous voulons souligner que cette représentation graphique n'est pas un format d'échange de données, et ne prétend donc pas à la portabilité. D'autres représentations graphiques à partir de l'expression XML sont envisageables dans d'autres environnements. Notre représentation graphique n'est visible qu'avec le navigateur Internet Explorer de Microsoft.

À l'aide d'une feuille de style XSL donnée en annexe, nous avons donc produit une représentation graphique d'un arbre TransTree dans le format VML à partir de l'expression XML.

2.1.1.2.2 Exemple simple

Le diagramme déjà rencontré va nous servir de point de départ pour présenter la lecture de cette représentation graphique.

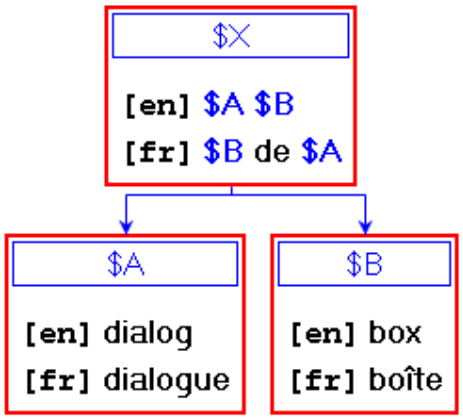

Figure 20 : diagramme de l'amphigramme complexe (dialog box / boîte de dialogue)

Sur ce diagramme, chaque boîte rouge représente un amphigramme. La boîte du haut est un amphigramme complexe, comprenant des amphigrammes fils. Les deux amphigrammes fils en bas sont des amphigrammes atomiques. Chaque amphigramme est porteur de deux éléments textuels, annoncés par la langue dans laquelle ils sont rédigés, entre crochets. Au sein des ces éléments textuels, les indicateurs-repères locaux (l'attribut **loc** de l'élément **<a/>**) sont indiqués avec des signes « dollar » (**$**). Les deux amphigrammes fils sont porteurs, dans un rectangle bleu, de leur indicateur-repère local, qui indique à quel point d'insertion il se rapporte dans l'amphigramme père. Celui-ci porte l'indicateur-repère local **$X**, par convention.

2.1.1.2.3 Exemple complexe

Le deuxième exemple donne lieu au diagramme suivant :

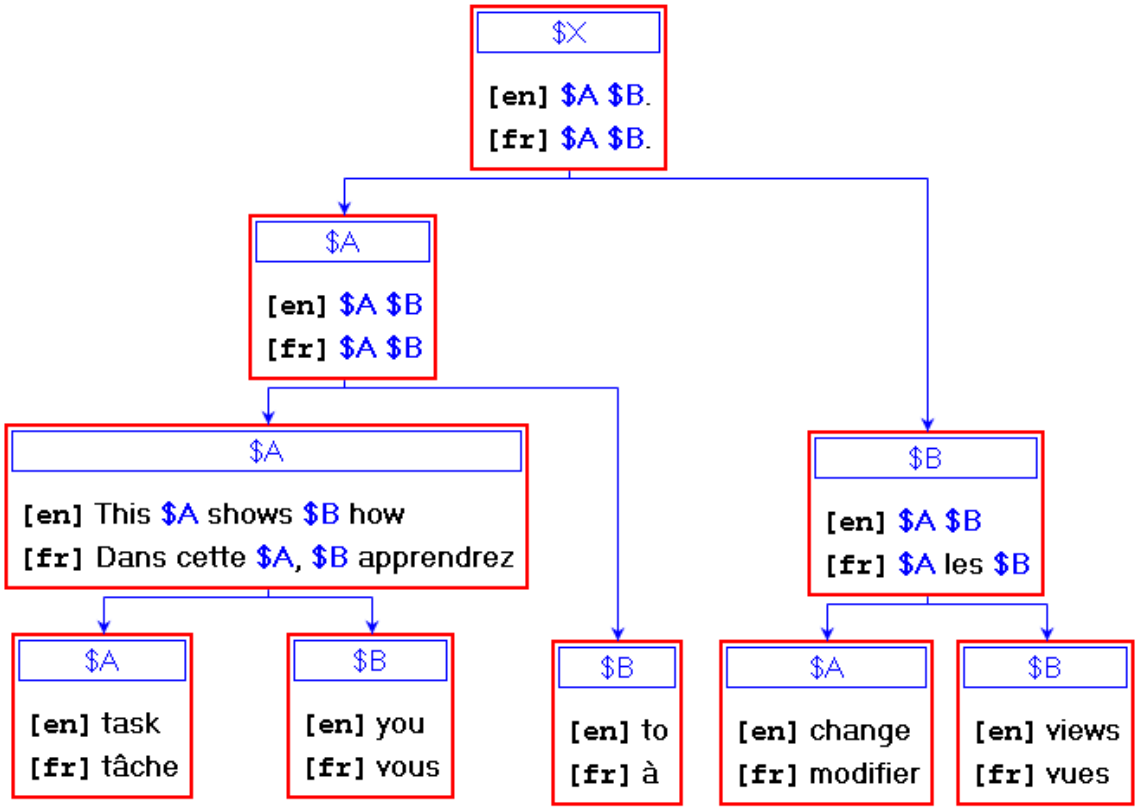

Figure 21 : diagramme de l'amphigramme complexe : (This task shows you how to change views / Dans cette tâche, vous apprendrez à changer les vues)

Cet exemple illustre comment plusieurs niveaux de correspondance peuvent être encodés grâce à TransTree. Comme nous l'avons vu, il n'y a pas d'ordre particulier des branches puisque l'ordre linéaire est donné par les indicateurs-repères locaux. Dans ce diagramme, nous avons choisi de faire figurer les amphigrammes atomiques sur la ligne la plus basse.

Rappelons que cette représentation sous forme de dessin vectorisé est équivalente à l'expression XML, à partir de laquelle elle a été calculée directement.

2.1.1.3 Discussion

2.1.1.3.1 Avantages

Le formalisme TransTree permet d'encoder les correspondances gigognes dans un bisegment (éventuellement un multisegment) de manière simple et souple. Il permet d'isoler des sous-parties dans les segments et d'exprimer les correspondances à chaque niveau. L'agencement dans chaque langue des parties en correspondance est traduit par l'utilisation des points d'insertion munis d'indicateurs-repères locaux.

Le langage XML est particulièrement approprié pour exprimer le formalisme TransTree. La structure arborescente intrinsèque d'XML est utilisée pour incarner l'arborescence de TransTree de façon naturelle. En outre, grâce à sa capacité à gérer du contenu mixte, les points d'insertion de correspondances fines sont facilement indiqués par les monobalises d'insertion **<a/>**munies de l'attribut **loc**.

2.1.1.3.2 Inconvénients

Les phénomènes de corrélation ou de dépendance à distance sont mal représentés avec TransTree. TransTree est sans doute mieux adapté aux langues isolantes, sans trop de contraintes grammaticales fortes.

2.1.1.3.3 Conclusion

Le formalisme TransTree est né spontanément comme sous-produit de nos efforts pour extraire des connaissances utilisables à partir de corpus bilingues, sans ressource linguistique autre. Il est donc particulièrement adapté pour représenter des correspondances traductionnelles établies de cette manière.

Nous pensons en outre qu'il fournit une base intéressante susceptible de donner lieu à différentes extensions. C'est ce que nous avons fait pour envisager un système de proposition de traductions (voir 2.1.3).

2.1.2 Visualisation interactive dynamique

Afin d'apprécier les correspondances à différentes profondeurs, nous avons imaginé une méthode de visualisation interactive dynamique des bisegments analysés selon le formalisme TransTree.

2.1.2.1 Présentation

2.1.2.1.1 Concept

Dans un environnement d'aide à la traduction, il est souhaitable d'éditer facilement et rapidement le texte à traduire. Les segments sont plus agréables à lire quand ils sont présentés chacun sur une même ligne de lecture, et non pas sous forme d'arbre. Autrement dit, l'utilisateur de la représentation abstraite proposée préférera sans doute pouvoir la lire d'une manière traditionnelle.

Afin de faciliter la mise en évidence des différents niveaux de segmentation tout en conservant un certain sens de l'ergonomie, nous proposons une visualisation interactive de TransTree, adaptée nous semble-t-il au travail du traducteur. Dans ce contexte, les amphigrammes seront vus comme des briques traductionnelles enchâssées.

L'idée principale est que le positionnement du pointeur de la souris sur un mot quelconque du segment cible ou du segment source déclenche la mise en évidence simultanée, dans les deux segments, de la hiérarchie des amphigrammes. Cette mise en évidence est réalisée grâce à l'emploi de différentes couleurs de fond pour chaque brique traductionnelle. La couleur la plus sombre délimite la brique traductionnelle la plus courte à l'endroit du segment pointé par l'utilisateur. Cette brique traductionnelle est elle-même incluse dans des briques traductionnelles plus de plus en plus longues, de couleurs de plus en plus claires. L'aspect gigogne de la construction est ainsi aisément perceptible par l'utilisateur.

2.1.2.1.2 Exemples

Sur la figure suivante, on voit ce qui se passe quand l'utilisateur fait glisser le pointeur en différents endroits du bisegment. On pourra retrouver ces exemples en cliquant sur le lien « Exemples d'alignement automatique de sous-segments enchassés » à la page :

http://www-clips.imag.fr/geta/User/christophe.chenon .

Les couleurs que l'on verra sur le site sont ici rendues par différentes nuances de gris. Le pointeur lui-même change de forme et indique le niveau de profondeur de la correspondance. Les niveaux les plus profonds sont indiqués par une couleur plus sombre et représentent des sous-segments plus petits.

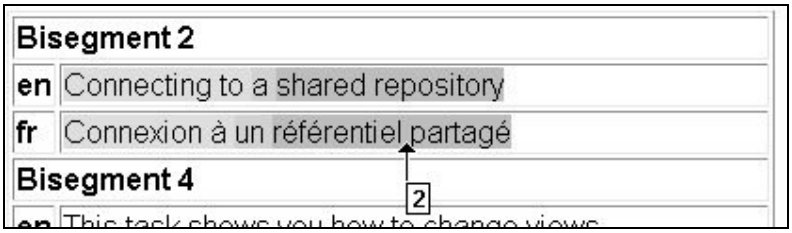

Figure 22 : sélection de la brique traductionnelle correspondant à « référentiel partagé »

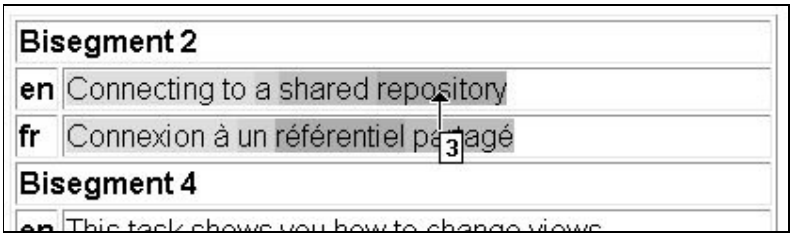

Figure 23 : sélection de la brique traductionnelle correspondant à « repository »

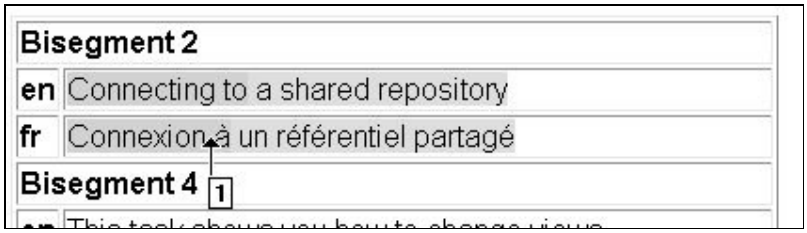

Figure 24 : sélection de la brique traductionnelle correspondant à « Connexion à »

En faisant glisser le pointeur de la souris sur les segments, indifféremment dans l'une ou l'autre langue, les briques traductionnelles sont mises en évidence simultanément avec la même couleur dans les deux textes.

Dans la Figure 22, l'utilisateur a positionné le pointeur au dessus de « référentiel partagé », ce qui a mis en évidence le sous-segment français correspondant. La partie anglaise de ce même amphigramme est mise en évidence avec la même couleur. L'ensemble est perçu comme une brique traductionnelle.

En faisant glisser sa souris, l'utilisateur peut ensuite sélectionner une sous-brique traductionnelle, comme on le voit dans la Figure 23. Dans la Figure 24, il s'est intéressé à une autre brique traductionnelle. En fait, il a sélectionné une autre branche de l'arbre TransTree. Les chiffres qui apparaissent en incrustation indiquent le niveau de l'amphigramme dans l'arbre.

2.1.2.1.3 Remarque

Le découpage en briques traductionnelles est porté par la représentation interne du bisegment : l'utilisateur ne peut pas le modifier. Dans l'exemple ci-dessous, la brique traductionnelle **(This task shows you how|Dans cette tâche, vous apprendrez)** est insécable.

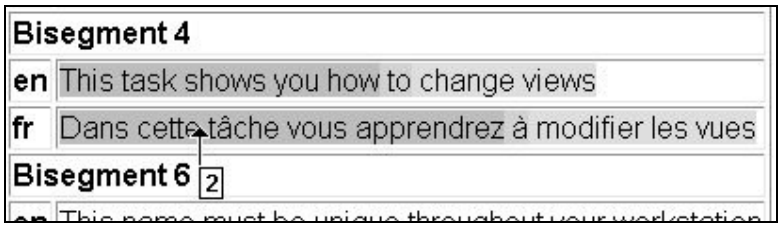

Figure 25 : brique complexe insécable contenant une sous brique

Elle contient néanmoins une brique enchâssée, **(task|tâche)** :

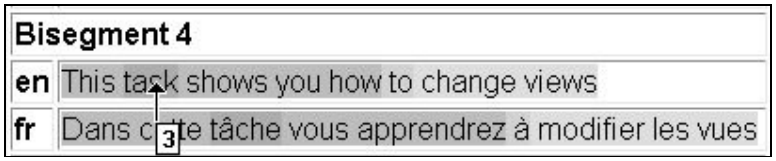

Figure 26 : sous brique enchâssée mise en relief sous le curseur

Enfin, cette dernière fait même partie d'une brique traductionnelle plus large : **(This task shows you how to|Dans cette tâche, vous apprendrez à)**

Cette visualisation pourrait être intégrée dans un environnement d'aide à la traduction. Un sous-segment proposé par le système pourrait être utilisé par le traducteur en l'important dans son environnement d'édition. Naturellement, si le traducteur n'est pas satisfait de ce découpage en sous-segments, il peut décider d'importer une partie quelconque de l'un des segments dans son environnement d'édition et la modifier à sa guise.

2.1.2.2 Mise en œuvre

Cette présentation dynamique a été réalisée en HTML et est conçue pour être utilisée à partir d'un navigateur courant, supportant JavaScript. Nous avons utilisé une feuille de style XSL pour transformer les bisegments formatés selon le formalisme TransTree en XML en un document HTML.

Cette visualisation interactive repose donc sur la formalisation XML des données. Les amphigrammes sont réécrits avec des balises **** munies des attributs **onMouseOver** et **onMouseOut** qui déclenchent les changements de couleur d'arrière-plan pour chaque brique traductionnelle.

Ce code est créé au moment de la transformation par la feuille de style XSL. Des identifiants sont créés dynamiquement pour lier les amphigrammes en correspondance. Le choix des couleurs a été effectué à l'avance, et prévoit dix couleurs pour dix niveaux de profondeur, ce qui est largement suffisant en pratique.

La couleur « transparent » est utilisée pour les éléments non-activés. De cette manière, la hiérarchie des amphigrammes depuis le nœud où la feuille activé(e) jusqu'à la racine de l'arbre est visible à tout moment.

On trouvera en annexe (3 Feuilles de style XSL) la feuille de style XSL permettant de transformer un document XTR (TransTree) en une fichier HTML mettant en œuvre la présentation dynamique.

2.1.2.3 Utilisation potentielle

2.1.2.3.1 Importance de l'ergonomie

Cette méthode de visualisation nous a été inspirée par le fait que le formalisme TransTree, qui nous semblait évident sous forme de schémas arborescents, restait pourtant relativement opaque à notre entourage ! Nous en avons donc cherché une présentation plus accessible pour en faire apprécier l'intérêt, et il nous a semblé plus ergonomique de rester dans un cadre de perception proche de la lecture habituelle.

L'avantage de cette méthode de visualisation est d'indiquer en temps réel les parties en correspondance dans un bisegment. Cela nous paraît particulièrement bien adapté non seulement au travail du traducteur, mais aussi à celui de l'apprenant d'une langue étrangère, ou d'une manière générale à toute personne travaillant interactivement sur le langage. Par contraste, on voit mal comment présenter l'amphigramme complet de façon aussi immédiatement perceptible.

2.1.2.3.2 Aide à la traduction

Dans un environnement de traduction fondé sur les mémoires de traduction de type TM, le traducteur se voit proposer plusieurs propositions de traduction correspondant à des segments source proches de celui qu'il a à traduire, dites coïncidences floues[8]. Généralement, ces propositions sont au nombre de trois.

Il en voit d'abord la version cible, qui est celle qui l'intéresse directement, et peut accéder à la version source s'il le demande. Autrement dit, s'il y a trois propositions, il voit d'emblée un segment source à traduire et trois segments cible déjà traduits, ressemblant peu ou prou à ce que sera sa propre traduction.

La détermination de ces propositions est faite par le système sur la base d'une distance d'édition. Mais dans le cas où les segments sont très différents, le traducteur peut ne pas immédiatement percevoir l'usage qu'il pourrait en faire. De fait, dans les outils usuels, les segments candidats ne traduisent pas tout le segment à traduire et la raison pour laquelle ils sont proposés ne saute pas aux yeux. Nous présentons ici une amélioreration de l'ergonomie du système dans cette situation.

S'il est aisé de déterminer les parties communes et les parties différentes entre les deux segments source, il est, en général plus difficile de connaître les traductions des parties communes, celles qui intéressent le traducteur. Avec une représentation TransTree des correspondances au sein des bisegments de la mémoire, le système pourrait fournir cette information au traducteur. Ainsi, les parties des segments cible candidats qui ne sont pas considérées comme pertinentes par le système pourraient apparaître grisées.

En effet, les parties cible pertinentes, celles qui correspondent à des sous-chaînes communes entre le segment à traduire dans une coïncidence floue, peuvent être mises en évidence, comme l'illustre la figure suivante.

[8] Fuzzy maches en jargon technique.

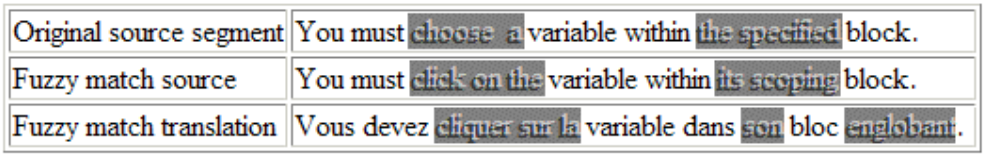

Figure 27 : présentation complète d'une proposition de traduction

Ici, les parties inutiles sont grisées, en profondeur. Du point de vue du traducteur, ce sont les parties à modifier.

La version source (2ème ligne ci-dessus) du bisegment proposé par le système peut alors être cachée afin d'alléger la quantité d'information à traiter par le traducteur. Il reste deux segments, le segment original à traduire et la traduction de la coïncidence floue.

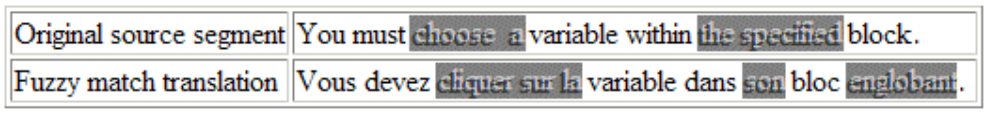

Figure 28 : présentation allégée d'une proposition de traduction

Les parties non grisées sont celles qui peuvent servir à la traduction.

En passant son pointeur sur le segment à traduire ou sur la proposition de traduction, le traducteur peut voir les sous-parties en correspondance.

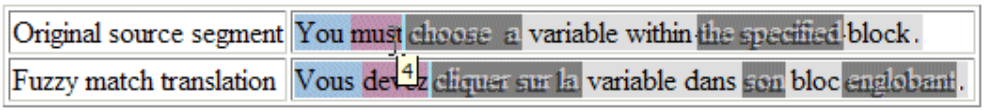

Figure 29 : mise en évidence de correspondances entre sous-chaînes

Avec trois segments candidats, les propositions concurrentes seront mises en évidence de la même façon.

Une mise en œuvre pratique serait la suivante : Le traducteur passe rapidement son curseur sur la ou les propositions de traduction, et détermine celle d'où il va partir pour traduire, par exemple en cochant un bouton radio. Cette proposition est copiée dans la zone de traduction éditable. Cliquer sur une zone grisée provoquerait l'apparition d'une liste de proposition (venant du dictionnaire et de la mémoire de traduction). et permettrait aussi l'écriture du reste.

Remarquons que l'information supplémentaire qui est ainsi fournie, quoique très riche, ne surcharge pas l'information que le traducteur a déjà à traiter. Au contraire, elle lui permet d'isoler rapidement ce qui est pertinent pour lui.

Par ailleurs, le fait d'utiliser ou de ne pas utiliser cette fonctionnalité n'entraînerait pas de modification de l'environnement de travail. Il s'agit bien d'un module qui viendrait s'ajouter au poste de travail du traducteur, à sa demande.

Bien entendu, il est possible d'envisager le même type de manipulation *via* le clavier. Ce fonctionnement paraît même nécessaire étant donné que l'utilisation de la souris ou d'un autre dispositif de pointage peut gêner la saisie de texte.

2.1.2.3.3 Apprentissage de langues étrangères

L'édition de livres bilingues connaît un succès croissant. La comparaison de la version originale et de la version traduite est une source très riche d'enseignement pour l'apprenant d'une langue étrangère (qui apprend, en général, la langue dans laquelle le texte qu'il lit a été rédigé).

Dans un environnement informatisé, un module de mise en correspondance sous-phrastique tel que celui que nous proposons semble pouvoir fournir une aide importante à un apprenant. Une édition bilingue sur cédérom ou un site Internet pourraient par exemple servir de support à ce module.

Ce type d'utilisateur peut être particulièrement sensible aux différents degrés de finesse qui lui sont proposés dans la mise en correspondance. D'une part, une correspondance « macroscopique » lui permet d'apprécier quelle phrase se traduit par quelle autre et il est libre, d'autre part, d'explorer des correspondances plus fines.

2.1.3 Compléments de définition

L'ambition que nous avons annoncée dans la première partie va au-delà de la seule analyse des mémoires de traduction. Nous voulons les voir comme un corpus bilingue aligné dont nous pouvons extraire de l'information réutilisable automatiquement. Dans ce chapitre, nous allons préciser dans un premier temps la forme que nous voudrions donner à cette information, puis nous donnerons une définition rigoureuse du formalisme. Enfin nous examinerons quelques extensions possibles.

2.1.3.1 Amphigrammes génériques et amphigrammes compacts

2.1.3.1.1 Ensemble d'amphigrammes

Dans la première partie (voir 1.3.2.1), nous avons évoqué l'axe paradigmatique de modélisation des données. Nous l'avons lié à la factorisation de l'information et avons souligné l'importance que nous lui accordions sur le plan cognitif. Nous allons maintenant compléter la modélisation TransTree pour nous permettre d'aborder cet aspect des choses.

Jusqu'à présent, nous avons considéré les amphigrammes comme des occurrences. Ils incarnent des correspondances dans les bisegments. Dans un corpus modélisé avec TransTree, nous pouvons compter les occurrences d'un même prototype d'amphigramme et éventuellement comparer ces prototypes entre eux.

Nous allons tout d'abord considérer les prototypes d'amphigrammes atomiques dont nous disposons dans un corpus particulier. Nous supposons qu'ils se répartissent en différents ensembles et nous voulons exprimer cette répartition en ensembles dans TransTree. Nous ne préjugeons pas ici de la façon dont ces ensembles sont constitués : nous cherchons juste à les représenter en présupposant leur existence.

Dans l'expression XML, nous ajoutons d'une part, à l'élément **<amphigram>**, un attribut **id**, identifiant qui nous permettra plus tard de faire référence à ce prototype d'amphigramme et d'autre part, un attribut **set**. Cet attribut est susceptible d'être renseigné avec une « étiquette d'ensemble*» ou éventuellement plusieurs séparées par des espaces : autrement dit, cet attribut est possiblement multivalué. Nous l'appelons « spécificateur d'ensemble ».

Par définition, un ensemble d'amphigrammes est constitué par tous les prototypes d'amphigrammes portant une même étiquette d'ensemble. Cette étiquette d'ensemble n'est donc rien d'autre qu'une marque portée par tous les amphigrammes appartenant à un ensemble particulier, lui-même identifié par cette étiquette.

```
<amphigrams section='lex'>
...
   <amphigram type='atomic id='#87BD53' set='#D9B# #EF0#'>
      <text xml:lang='en'>dialog</text>
      <text xml:lang='fr'>dialogue</text>
   </amphigram>
...
</amphigrams>
```

Figure 30 : un amphigramme atomique muni de son étiquette d'ensemble

Ici, nous avons un amphigramme atomique déjà connu. Nous lui avons ajouté un identifiant (#87BD53) et un attribut **set** multivalué (#D9B# #EF0#). Les autres amphigrammes ayant #D9B# parmi leurs étiquettes d'ensemble appartiennent *ipso facto* au même ensemble, dont la définition est ainsi donnée en extension. Ceux qui ont #EF0# appartiennent à un autre ensemble, et l'amphigramme atomique #87BD53 appartient à l'intersection de ces deux ensembles.

Une fois établie la notion d'ensemble d'amphigrammes atomiques, nous pouvons introduire le concept d'« amphigramme générique*».

2.1.3.1.2 Amphigramme générique

Les amphigrammes génériques possèdent des points d'insertion dans leurs éléments textuels mais pas d'amphigrammes fils, au contraire des amphigrammes complexes. En revanche, ces points d'insertion font indirectement référence à d'autres amphigrammes génériques ou atomiques, en spécifiant leur étiquette d'ensemble. En effet, les ensembles en question sont constitués d'amphigrammes atomiques ou génériques.

Nous ajoutons donc un attribut **set** dans les monobalises d'insertion **<a/>** des éléments textuels **<text>**. Sa signification, dans ce contexte, est différente de sa signification en tant qu'attribut de **<amphigram>** comme précédemment. Ici, il stipule que seuls les amphigrammes appartenant à l'ensemble indiqué par cet attribut peuvent occuper ce point d'insertion. Nous l'appellerons « indicateur d'ensemble* ». Cet attribut est potentiellement multivalué, ce qui introduit une contrainte forte : les amphigrammes atomiques ou génériques susceptibles d'occuper ce point d'insertion doivent appartenir simultanément à tous les ensembles indiqués, ce qui est compatible avec le fait que le spécificateur d'ensemble est lui-même multivalué.

Par le jeu de ces attributs **set**, nous avons donc, d'une part, la possibilité de définir des ensembles d'amphigrammes et d'autre part de faire référence à ces ensembles dans les monobalises d'insertion. Cependant, nous devons insister sur le fait que ces ensembles ne concernent que les amphigrammes atomiques ou génériques c'est-à-dire des prototypes et non pas des occurrences, et que ces amphigrammes ne contiennent pas d'amphigrammes fils. Autrement dit, ils ne concernent pas les amphigrammes complexes. Nous verrons plus loin comment le lien avec les amphigrammes complexes est établi.

Nous donnons ci-dessous un exemple d'amphigramme générique exprimé en XML, ainsi que deux amphigrammes atomiques susceptibles d'occuper ses points d'insertion. Cet amphigramme générique permet de décrire une structure linguistique de type « complémentation du nom » en anglais et en français. C'est la structure que l'on trouve dans la paire **(dialog box|boîte de dialogue)**.

```
<amphigrams section='lex'>
...
   <amphigram type='generic' id='#B6A82F' set='#3DE#'>
      <text xml:lang='en'><a loc='A' set='#D9B#'/>  <a loc='B' set='#1AC#'/></text>
      <text xml:lang='fr'><a loc='B'/>  de  <a loc='A'/></text>
   </amphigram>
...
   <amphigram type='atomic ' id='#87BD53' set='#D9B#'>
      <text xml:lang='en'>dialog</text>
      <text xml:lang='fr'>dialogue</text>
   </amphigram>
...
   <amphigram type='atomic' id='#90AE0A' set='#1AC#'>
      <text xml:lang='en'>box</text>
      <text xml:lang='fr'>boîte</text>
   </amphigram>
...
</amphigrams>
```

Figure 31 extrait de la section lex : un amphigramme générique et deux amphigrammes atomiques

Dans cet exemple, il y a donc trois amphigrammes différents. L'amphigramme générique porte un identifiant (#B6A82F) et appartient à un ensemble (#3DE#). Il fait référence, dans ces monobalises d'insertion, à d'autres ensembles d'amphigrammes (#D9B# et #1AC#).

Les amphigrammes atomiques **(dialog|dialogue)** et **(box|boîte)** appartiennent justement aux deux ensembles en question, grâce à leur attribut **set**.

Afin d'éviter toute redondance ou incohérence dans les amphigrammes génériques, les points d'insertion d'un seul élément textuel (ici le premier, exprimé en anglais) précisent les ensembles d'amphigrammes susceptibles de les occuper. La perte de symétrie entre les éléments textuels n'est donc qu'apparente.

Les amphigrammes génériques font partie de la base de connaissances du système. Ils sont recensés dans une section de document ne contenant que des amphigrammes génériques ou atomiques. C'est le sens de la balise **<amphigrams section="lex">**.

On constate la grande ressemblance entre amphigrammes génériques et amphigrammes complexes. La différence fondamentale tient au fait qu'un amphigramme complexe représente une instance et qu'un amphigramme générique représente un patron de traduction. La différence n'est pas tout à fait la même qu'entre occurrence et prototype : même si l'on trouve plusieurs occurrences d'un prototype d'amphigramme complexe dans un corpus, ce prototype ne constitue pas un patron de traduction. L'attribut **set** témoigne de cette distinction : un amphigramme générique appartient à un ensemble et appelle récursivement des amphigrammes génériques ou atomiques par le biais de cet attribut. Un amphigramme

générique n'appartient pas au corpus et la notion d'occurrence n'est pas directement pertinente à son sujet. Ce point est détaillé plus bas.

Remarquons que les ensembles jouent un rôle semblable à celui des catégories linguistiques utilisées dans une grammaire unilingue. Ici, les objets manipulés sont bilingues. Les amphigrammes génériques constituent une grammaire dont les amphigrammes atomiques sont les éléments terminaux.

2.1.3.1.3 Amphigrammes compacts

Dans le document lui-même, on peut décrire les bisegments en tenant compte des amphigrammes génériques. Un amphigramme complexe peut faire référence à un amphigramme générique, sous-entendu son armature textuelle, et détailler seulement les amphigrammes fils, spécifiques de l'occurrence du bisegment à décrire. L'armature textuelle étant celle de l'amphigramme générique référencé, il n'est plus nécessaire de l'expliciter.

Ce nouveau type d'amphigramme, sans armature textuelle explicite, est appelé « amphigramme compact*». Il s'agit d'une occurrence rencontrée dans un texte. Nous complétons donc une dernière fois le formalisme TransTree en ajoutant l'attribut **ref** sur l'élément **<amphigram>**.

```
<amphigrams section='doc'>
...
   <amphigram type='compact' ref='#B6A82F'>
      <amphigram loc='A' ref ='#87BD53'/>
      <amphigram loc='B' ref ='#90AE0A'/>
   </amphigram>
...
</amphigrams>
```

Figure 32 exemple d'amphigramme compact

Cet amphigramme compact encode la paire **(dialog box|boîte de dialogue)** rencontrée plus haut, comme on pourra le vérifier à l'aide des identifiants utilisés. Il contient deux amphigrammes fils, mais pas d'élément textuel. Pour expliciter l'agencement des amphigrammes fils, il fait référence à un amphigramme générique. Les amphigrammes fils font eux référence à des amphigrammes atomiques.

Une nouvelle contrainte apparaît ici : les amphigrammes (génériques ou atomiques) référencés par les amphigrammes fils d'un amphigramme compact A doivent satisfaire les conditions d'ensemble dictées par l'amphigramme générique référencé par A. Autrement dit, dans cet exemple, les amphigrammes fils sont effectivement admissibles parce que les amphigrammes atomiques identifiés #87BD53 et #90AE0A appartiennent bien aux ensembles d'étiquettes #D9B# et #1AC# respectivement, ainsi que l'exige l'amphigramme générique #B6A82F.

Un amphigramme compact est donc équivalent à un amphigramme complexe. La différence tient dans l'information supplémentaire que représente la référence à un amphigramme générique : il constitue une instance du patron de traduction décrit par l'amphigramme générique auquel il fait référence.

Nous plaçons les amphigrammes compacts dans la section « document » du fichier XML, comme le traduit la balise **<amphigrams section='**doc**'>.**

2.1.3.2 Récapitulation et définitions formelles de TransTree

Pour finir cette présentation du formalisme TransTree, nous donnons ici une représentation sous forme d'une DTD, d'un schéma et d'autres éléments de références.

2.1.3.2.1 DTD de TransTree

La DTD suivante permet de décrire un document TransTree complet. Elle permet de regrouper les deux sections « document' » et « lexique » dans un même document. Bien entendu, on peut envisager de placer ces deux sections dans deux documents séparés.

```
<!ELEMENT transtree ( amphigrams , amphigrams? ) >

<!ELEMENT amphigrams ( amphigram+ ) >
<!ATTLIST amphigrams section (doc | lex) 'doc' >

<!ELEMENT amphigram ( amphigram | text )* >
<!ATTLIST amphigram
     type (complex | compact | generic | atomic) #IMPLIED
     id ID #IMPLIED
     ref IDREF #IMPLIED
     loc (A | B | C | D | E | F) #IMPLIED
     set NMTOKENS #IMPLIED >

<!ELEMENT text ( #PCDATA | a )* >
<!ATTLIST text xml:lang NMTOKEN #REQUIRED >

<!ELEMENT a EMPTY >
<!ATTLIST a
   loc (A | B | C | D | E | F) #REQUIRED
   set NMTOKENS #IMPLIED >
```

Figure 33 DTD de TransTree

Un fichier XML TransTree est composéde deux sections, une section document et une section lexique[9] : Dans la DTD, l'élément **<transtree>** doit contenir au moins un élément **<amphigrams>**, éventuellement deux. L'attribut section peut prendre la valeur 'doc' ou la valeur 'lex'. S'il y a deux élément **<amphigrams>**, l'esprit de cette DTD est que les valeurs de l'attribut section soient différentes, mais cela dépasse la capacité d'expression d'une DTD. Si les deux attributs sont identiques, l'utilisateur (humain ou informatique) doit considérer qu'il n'y a qu'une seule section.

La section « doc » décrit un bitexte sous forme d'amphigrammes complexes ou compacts qui expriment des bisegments réels munis de correspondances internes concrètes.

Les amphigrammes compacts, dans la section « doc », font références à des amphigrammes génériques ou atomiques qui indiquent explicitement les ensembles d'amphigrammes auxquels ils appartiennent. Ces derniers sont placés dans la section « lex ».

Le fichier XML suivant résume cette organisation :

[9] Éventuellement, on peut organiser ces deux sections dans deux fichiers XML différents.

```
<? xml version='1.0' ?>
<transtree>
..<amphigrams section='doc'>
...
<!--
Ici, on trouve les amphigrammes complexes ou compacts. Chacun
d'entre eux représente un bisegment complet.
-->
...
..</amphigrams>
..<amphigrams section='lex'>
...
<!--
Ici, on trouve les amphigrammes génériques ou atomiques
nécessaires aux amphigrammes compacts de la section doc.
-->
...
..</amphigrams>
</transtree>
```

Figure 34 architecture d'un document TransTree

Le reste de la DTD explicite les éléments et attributs déjà décrits.

2.1.3.2.2 Schéma de TransTree

Comme une DTD ne peut pas exprimer toutes les contraintes portant sur les amphigrammes, nous proposons en outre un schéma pour TransTree.

Le schéma ci-dessous est en fait presque équivalent à la DTD. Il nous permet néanmoins de préciser, dans la définition de l'espace de noms, que DTD et schémas, fichiers d'exemples et autres ressources sont consultables sur la Toile à l'adresse suivante :

http://www-clips.imag.fr/geta/User/christophe.chenon/TransTree

Nous utilisons le symbole trilitère **xtr** comme préfixe de nom de domaine et comme suffixe d'extension des fichiers TransTree exprimés en XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xtr=" http://www-clips.imag.fr/geta/User/christophe.chenon/TransTree "
targetNamespace="http://www-clips.imag.fr/geta/User/christophe.chenon/TransTree">

  <xs:element name="transtree">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="amphigrams" maxOccurs="2" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="amphigrams">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="amphigram" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="section" use="optional" default="doc">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="doc" />
            <xs:enumeration value="lex" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name="amphigram">
    <xs:complexType>
      <xs:choice>
        <xs:element ref="amphigram" />
        <xs:element ref="text" />
      </xs:choice>
      <xs:attribute name="type" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="complex" />
            <xs:enumeration value="compact" />
            <xs:enumeration value="generic" />
            <xs:enumeration value="atomic" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="id" type="xs:ID" use="optional" />
      <xs:attribute name="ref" type="xs:IDREF" use="optional" />
      <xs:attribute name="set" type="xs:NMTOKENS" use="optional" />
      <xs:attribute name="loc" use="optional">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="A" />
            <xs:enumeration value="B" />
            <xs:enumeration value="C" />
            <xs:enumeration value="D" />
            <xs:enumeration value="E" />
            <xs:enumeration value="F" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

  <xs:element name="text">
    <xs:complexType mixed="true">
      <xs:choice>
        <xs:element ref="a" />
      </xs:choice>
      <xs:attribute name="xml:lang" type="xs:NMTOKENS" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="a">
    <xs:complexType>
      <xs:attribute name="set" type="xs:NMTOKENS" use="optional" />
      <xs:attribute name="loc" use="required">
```

```
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="A" />
            <xs:enumeration value="B" />
            <xs:enumeration value="C" />
            <xs:enumeration value="D" />
            <xs:enumeration value="E" />
            <xs:enumeration value="F" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Figure 35 proposition de schéma XML pour TransTree

2.1.3.2.3 Autres éléments

Tableau récapitulatif

Nous donnons ci-dessous un tableau récapitulatif qui résume certaines propriétés des amphigrammes.

Amphigramme	autonome	concret	référençable	amphigramme fils	element textuel	point d'insertion
atomique	oui	oui	oui/non	non	oui	non
complexe	oui	oui	non	oui	oui	oui
compact	non	oui	non	oui	non	non
générique	non	non	oui	non	oui	oui

Figure 36 : comparaison des différents types d'amphigrammes

Les différentes propriétés des amphigrammes sont données par colonnes. Un amphigramme peut être :

1. **autonome** selon qu'il fait ou non référence à d'autres amphigrammes ou ensemble(s) d'amphigrammes
2. **concret** s'il se réécrit sans ambiguïté sous forme d'un bisegment
3. **référençable** quand il porte un identifiant utilisé dans le formalisme TransTree
4. susceptible de contenir un **amphigramme fils** ou plus dans l'arborescence XML
5. susceptible de contenir un **élément textuel** ou plus.
6. susceptible de contenir un **point d'insertion** ou plus.

Il existe deux genres d'amphigramme atomique :

- l'amphigramme atomique référençable, qu'on va trouver dans la section « lex » de notre fichier XML, appartenant à un ou plusieurs ensembles utilisés par un ou plusieurs amphigrammes génériques. Il s'agit d'un prototype, défini par son armature textuelle : on ne peut avoir, dans la section « lex », deux amphigrammes atomiques identiques, c'est-à-dire ayant la même armature textuelle.
- l'amphigramme atomique non référençable, équivalent à un amphigramme complexe, que l'on trouvera dans la section « doc » de notre fichier XML, généralement comme feuille d'un amphigramme complexe. Il s'agit d'une occurrence, potentiellement répétée plusieurs fois dans la section « doc ».

Vérificateur de contraintes

Le schéma que nous avons donné en 2.1.3.2.2 ne permet pas de vérifier toutes les contraintes du formalisme TransTree. Nous avons donc développé une feuille de style XSL (TransTreeChecker.xsl) destinée à effectuer quelques-unes de ces vérifications.

On en trouvera le code en annexe 3.

Ce vérificateur parcourt un fichier XML TransTree et produit un compte rendu contenant des messages informatifs et des messages d'erreur.

Les points vérifiés appartiennent à plusieurs catégories :

Détermination ou vérification du type d'un amphigramme (amphigramme complexe ou compact dans la section 'doc', générique ou atomique dans la section 'lex'), en fonction de leurs caractéristiques formelles (attributs, amphigrammes fils),

Vérification des contraintes formelles sur les monobalises d'insertion et les indicateurs-repères locaux au sein d'un amphigramme (nombre, unicité des valeurs etc.),

Vérification des contraintes d'ensemble des amphigrammes compacts.

Ces trois catégories sont reflétées par dans cet exemple de compte-rendu de TransTreeChecker[10] :

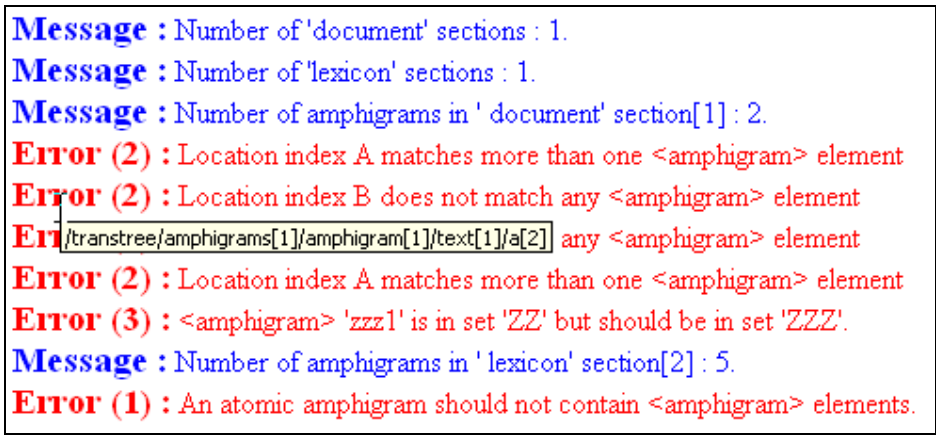

Figure 37 : exemple de compte rendu de TransTreeChecker

Les erreurs sont caractérisées par leur type entre parenthèses et sont suivies de leur libellé. Il s'agit d'un fichier HTML. Pour ne pas surcharger la mise en page, le chemin (dans le fichier XML) des amphigrammes où l'erreur se situe est donné à l'utilisateur en incrustation quand il positionne le curseur sur le message d'erreur. Ce chemin sert à identifier les amphigrammes évoqués dans le libellé.

2.1.3.3 Extensions possibles du formalisme

2.1.3.3.1 Multilinguisme

Une première extension envisageable consiste à modéliser un multitexte, comprenant plus de deux langues. Dans l'architecture de TransTree, il suffit de rajouter un élément **<text>** associé à chaque langue. Le diagramme ci-dessous permet d'imaginer la forme que de tels arbres pourraient revêtir.

[10] Donnons ici un exemple en anglais : ce programme est prévu pour être facilement traduit.

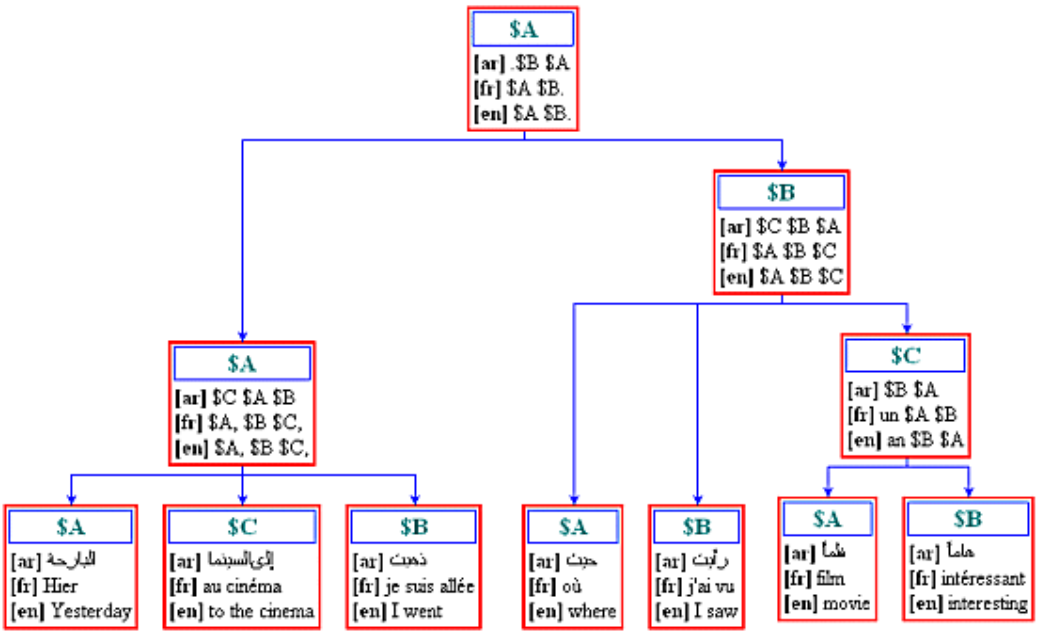

Figure 38 : TransTree appliqué à trois langues

Cet exemple met en évidence la possibilité de traiter une langue comme l'arabe, qui s'écrit de droite à gauche. Nous avons donc présenté les éléments textuels dans ce sens-là, y compris pour les points d'insertion ($A, $B etc), naturels pour l'arabe. Cette représentation est purement arbitraire et ne préjuge pas des mises en œuvre particulières des systèmes d'exploitation ou des applications utilisant le document XML sous-jacent.

Comme dans le cas de deux langues, l'ordre dans lequel se présentent les amphigrammes dans l'arbre n'est pas important, puisque l'agencement linéaire dans chaque langue est géré par les armatures textuelles.

2.1.3.3.2 Information linguistique

Il est également possible de traiter les amphigrammes comme des éléments structurels nommés, et d'adjoindre à leur description des traits syntaxiques (de type COD, pour un complément d'objet direct) ou morphosyntaxiques (de type NMS pour nom masculin singulier).

Cette connaissance peut être encodée dans les étiquettes d'ensemble des amphigrammes génériques et atomiques. Cependant, il faut bien avoir présent à l'esprit que les amphigrammes sont des objets bilingues. Autrement dit, les ensembles envisagés pourraient plutôt avoir une forme de type NOM/ADJ, par exemple, pour préciser le cas d'un nom rendu par un adjectif dans la traduction.

2.1.3.3.3 Dépendances à distance

Notre formalisme permet de tenir compte de corrélations syntaxiques diverses par le jeu de la granularité*. En effet, la taille des amphigrammes est "maximaliste", au sens où les dépendances lointaines peuvent être représentées au sein d'un amphigramme de taille sufisante. Les éléments liés par corrélation se réécrivent sous forme d'amphigrammes plus petits, dépendant du premier.

Cependant, ce formalisme risque d'être insuffisant en cas de dépendances croisées par exemple. On peut envisager d'ajouter de la souplesse à cette représentation en autorisant un mécanisme de référence entre amphigrammes hiérarchiquement indépendants. Cette option n'a pas été examinée à l'heure actuelle.

2.2 Eléments de construction de TransTree

TransTree n'est pas une modélisation vide. Elle a été conçue en fonction des algorithmes de structuration des bisegments tels que nous les avons peu à peu élaborés. Notre but, dans ce chapitre, est d'indiquer les éléments fondamentaux dans cette structuration pour un bisegment quelconque du corpus. Le chapitre suivant sera consacré à la description de différentes utilisations de ces éléments fondamentaux.

2.2.1 Alignements élémentaires

La première tâche que nous nous assignons est de travailler selon l'axe interlingue. Nous commençons par repérer des correspondances élémentaires dans un bisegment. Celles-ci nous permettront d'ancrer l'ensemble de la construction de TransTree. Nous appelons ces correspondances élémentaires entre mots typographiques des digrammes.

2.2.1.1 Digrammes

2.2.1.1.1 Notion de digrammes

Au moment d'aborder la structuration d'un bisegment selon le modèle TransTree, nous avons peu d'éléments à notre disposition : des bisegments constitués de mots typographiques, c'est-à-dire des chaînes finies de caractères comprises entre deux séparateurs.

Nous allons raffiner la connaissance que nous avons sur chaque mot typographique de chacun des segments en lui associant, si possible, un mot typographique de l'autre segment, et un seul. Nous appelons les couples ainsi constitués des « digrammes* ».

Un digramme est un objet bilingue symétrique. Il est constitué deux mots typographiques, un dans chaque langue, par exemple : **(computer,ordinateur)**[11].

En termes d'occurrences, un digramme est un lien de correspondance entre les segments dans un bisegment. Par exemple, un digramme peut être le lien entre le 5ème mot typographique du segment source et le 7ème mot typographique du segment cible.

En termes de prototypes, un digramme est un couple de mots typographiques (prototypes) dont on trouve un certain nombre d'occurrences dans le corpus.

Un de nos objectifs est de désambiguïser les données manipulées autant que faire se peut. En lexicologie, on distingue classiquement deux homonymes en leur adjoignant une étiquette distinctive. Nous avons par exemple **(affection,maladie)** formellement distingué de **(affection,tendresse).**

[11] L'ordre des langues est arbitraire, puisque nos objets sont symétriques. Les constituants du digramme sont donnés dans l'ordre alphabétique des noms anglais de leur langue respective.

C'est dans le même esprit que nous considérerons les digrammes. En effet, un digramme étant un couple formé de deux mots typographiques, on raffine l'un avec l'autre. Par exemple, le digramme **(record,enregistrer)** n'est pas le même que le digramme **(record,enregistrement)**. Symétriquement, les deux digrammes **(part,pièce)** et **(room,pièce)** sont différents.

Cette différenciation permet une connaissance beaucoup plus fine du comportement des occurrences manipulées. Le segment français et le segment anglais ne sont plus vus comme deux suites de mots typographiques mais comme deux suites d'occurrences digrammes, chacune dans un ordre qui lui est propre.

La notion de digramme participe de l'axe interlingue tel que nous l'avons évoqué au 1.3.2.2.1. Il s'agit en effet d'enrichir l'information que nous avons sur les formes graphiques simples rencontrées dans l'une des deux langues par une connaissance héritée de l'expertise humaine du traducteur.

Notre première tâche est donc d'apparier les mots typographiques des bisegments entre eux. Comme nous n'avons pas d'information sur la sémantique des mots typographiques, nous allons recourir à la statistique pour établir cette information. Nous allons pour ce faire étudier la cooccurrence des apparitions des mots typographiques sur l'ensemble du corpus.

2.2.1.1.2 Vrais digrammes, faux digrammes

Mécaniquement, ne serait-ce que parce que le nombre de mots diffère entre un segment et sa traduction, certains mots typographiques n'entrent pas dans un lien de correspondance. Par souci d'unifier les types de données utilisés, nous conserverons le terme de digramme : Nous dirons qu'ils constituent de « faux digrammes » et les opposerons aux « vrais digrammes ».

Conventionnellement, nous adoptons le symbole **#null#** pour dénoter l'absence de contrepartie dans un faux digramme. Nous aurons, par exemple, les deux faux digrammes suivants : **(#null#,de)** ou **(does, #null#)**.

Ainsi, dans un bisegment contenant « database » en anglais et « base de données » en français, si le digramme **(database,base)** est construit, les mots **de** et **données** vont se retrouver orphelins, donnant lieu aux faux digrammes **(#null#,de)** et **(#null#,données).**

Le digramme **(database,base)** différencie le mot typographique **base**,utilisé dans **base de données**, du mot typographique **base** utilisé dans une expression comme **de base**. Dans ce cas il donnera peut-être lieu au digramme **(basic,base)**.

Il ne faut donc pas confondre le mot typographique **données**, par exemple, avec le digramme **(#null#,données)**. Les nombres d'occurrences de l'un et de l'autre dans le corpus sont potentiellement très différents, celui du mot typographique étant, évidemment, supérieur à celui de ce digramme particulier. Plus précisément, le nombre d'occurrences du mot typographique est égal à la somme des nombres d'occurrences de tous les digrammes dans lequel il entre : le faux digramme et les vrais digrammes.

Les comportements statistiques des tous ces digrammes étant différents, il nous sera possible de reconstituer le lien entre **database** et **base de données** ultérieurement (voir le chapitre 2.2.2 sur la sécabilité et les arbres binaires.).

2.2.1.1.3 Digrammes et amphigrammes atomiques

Dans TransTree, il n'y a pas de restriction particulière sur les chaînes de caractères dont sont constitués les amphigrammes atomiques.

Par ailleurs, nous venons de définir les digrammes, qui sont, en général, des couples de mots typographiques liés statistiquement.

Nous faisons à présent un choix d'implémentation particulier : les vrais digrammes incarneront les amphigrammes atomiques[12].

Nous avons choisi de distinguer formellement amphigrammes atomiques et vrais digrammes pour la raison suivante : les digrammes couvrent une notion liée à l'algorithme d'analyse et non pas au formalisme de représentation des bisegments comme les amphigrammes. En outre, nous pouvons parler d'amphigramme en général (complexe, atomique etc.) et de digramme en général (vrai digramme ou faux digramme) sans que ces notions ne se recoupent.

2.2.1.2 Constitution des digrammes

2.2.1.2.1 Présentation

Nous constituons les digrammes à partir des informations trouvées dans les segments eux-mêmes : ce sont les mots typographiques rencontrés dans les segments qui sont mis en correspondance afin de former des digrammes. Nous avons pris le parti de construire des correspondances biunivoques. Autrement dit, dans un bisegment, chaque mot typographique ne peut appartenir qu'à un seul digramme.

Les digrammes sont établis par des moyens statistiques : on associera en digrammes les mots typographiques qui se retrouvent « relativement souvent » ensemble, « relativement souvent », signifiant plus souvent que si leur répartition était totalement aléatoire.

Un digramme est un couple de mots **(a,b)**, où **a** est un mot de la langue **A** et **b** un mot de la langue **B**. C'est un élément d'une grande matrice creuse $\mathbf{(a_i,b_j)}$ associant tous les mots typographiques du corpus bilingue. Si **N** est le nombre total de bisegments du corpus et **N(x)** le nombre de bisegments contenant **x** (dans la langue de **x**), nous nous intéressons à la probabilité d'un bisegment de contenir **a**, soit :

$$\frac{N(a)}{N}$$

et pour **b** :

$$\frac{N(b)}{N}$$

Parmi les bisegments qui contiennent **a**, certains contiennent également **b**. La probabilité d'un bisegment contenant **a** de contenir également **b**, s'écrit :

$$\frac{N(a,b)}{N(a)}$$

Les mots **b** de la langue **B** qui ont une relation privilégiée avec **a** sont relativement plus probables en présence de **a** que dans les corpus en général.

Les plus fortement liés à **a** sont les **b** pour lesquels cette probabilité relative est la plus importante, autrement dit pour lesquels l'indice suivant est le plus important :

[12] Ce choix n'est possible que pour les langues dont le système d'écriture prévoit des séparateurs.

$$\frac{\frac{N(a,b)}{N(a)}}{\frac{N(b)}{N}} = \frac{N(a,b) \cdot N}{N(a) \cdot N(b)}$$

Comme **N** est invariable dans l'ensemble du corpus, nous aboutissons à l'expression suivante :

$$I = \frac{N(a,b)}{N(a) \cdot N(b)}$$

où nous reconnaissons une forme simplifiée de l'information mutuelle entretenue par les bisegments contenant au moins une occurrence du mot typographique source **a** et du mot typographique cible **b.**

Il existe d'autres indices permettant de repérer les mots en correspondance, tant statistiques que morphologiques. En particulier l'observation de cognats morphologiques est utilisée dans certains travaux. Nous n'avons pas cherché à nous en servir afin de traiter des langues morphologiquement éloignées. Pour une étude approfondie des autres indices et de la cognation, on pourra se référer à la thèse d'Olivier Kraif par exemple([7]).

2.2.1.2.2 Collecte des cooccurrences

Dans un premier temps, on passe donc en revue tous les bisegments du corpus. Pour chaque mot typographique source $\mathbf{S_i}$, on compte le nombre de bisegments où il apparaît, $\mathbf{N_{si}}$, et on établit la liste de tous les mots typographiques cible apparaissant dans les bisegments où ce mot source est utilisé. Pour chacun de ces mots, on consigne également le nombre de bisegments contenant à la fois le mot source et le mot cible.

Dans le sens inverse, de la cible vers la source, on décompte le nombre total de bisegments contenant chaque mot cible **Tj : $\mathbf{N_{tj}}$**.

Nous obtenons donc une matrice de type **($\mathbf{N_{ij}}$)** où $\mathbf{N_{ij}}$ est le nombre de bisegments du corpus contenant à la fois les mots typographiques **Si** (côté source) et **Tj** (côté cible). On calcule alors la matrice **($\mathbf{I_{ij}}$)** avec :

$$I_{ij} = \frac{N_{ij}}{N_{Si} \cdot N_{Tj}}$$

Cette matrice nous sert de base pour la stratégie d'association en couples qui suit.

2.2.1.2.3 Stratégie d'association en couples

L'algorithme comprend plusieurs passes. Nous l'appellons la méthode des champions / cochampions. Nous dirons qu'un mot cible (ou plusieurs en cas d'*ex æquo*) est le champion d'un mot source s'il donne lieu à l'information mutuelle la plus grand avec le mot source. Chaque mot source a donc son (ou ses) champion(s) côté cible, et inversement.

La première passe consiste à considérer, pour chaque bisegment, les couples de mots typographiques source et cible tels que chacun est un champion pour l'autre. Nous dirons que

de tels mots typographiques constituent un couple de champions / cochampions, ou qu'ils se cooptent.

Soit par exemple un bisegment composé de la suite de mots S1 S2 S3 S4 S5 S6 S7 côté source et T1 T2 T2 T3 T4 T5 T6 côté cible.

Dans le tableau suivant, nous avons indiqué les à gauche les champions de chaque mot source et à droite les champions de chaque mot cible. Quand il y a plus d'un champion, il s'agit d'*ex æquo*.

Champions des mots source	Champions des mots cible
S1 → T2, T1	T1 → S1
S2 → T3	T2 → S2, S1
S3 → T6	T3 → S2
S4 → T4	T4 → S1
S5 → T2	T5 → S4, S7
S6 → T5, T6	T6 → S7
S7 → T5	

Tableau 2 : première table des champions source et cible

Nous voyons que les mots typographiques se cooptent dans les couples suivants : (S1, T1), (S1,T2), (S2,T3) et (S7, T5). Nous enregistrons ces couples. Les deux premiers couples représentent des situations équivalentes : nous les enregistrons l'un et l'autre. Nous retirons aussi des segments les mots typographiques utilisés. Nous obtenons, par exemple, le tableau suivant :

Champions des mots source	Champions des mots cible
S3 → T6	
S4 → T4	T4 → **S4**
S5 → **T6**	
S6 → T6	T6 → **S3**

Tableau 3 : deuxième table des champions source et cible

Certains mots ayant été retirés, d'autres prennent leur place et deviennent éventuellement les champions des mots restants dans chaque langue[13]. Nous avons constitué ce nouvel exemple arbitrairement. Nous avons donc deux nouveaux couples : (S3, T6) et (S4, T4), que nous enregistrons.

Comme il ne reste plus de mot cible, le traitement de ce bisegment prend fin. D'une manière générale, le traitement prend fin quand il n'y a plus de mot d'un côté ou de l'autre.

L'ensemble du corpus est ainsi parcouru et les couples de mots sources et de mots cibles sont enregistrés au fur et à mesure.

Nous reprenons alors l'ensemble de cette méthode, mais en utilisant à présent le nombre d'occurrences des couples en lieu et place de la matrice des informations mutuelles. Nous établisssons alors une nouvelle liste couples de de champions / cochampions, différente de la précédente. En effet, de nouveaux couples sont enregistrés et certains disparaissent. La liste des digrammes admissibles évoluant, il arrive fréquement que certains mots typographiques du segment le plus court ne trouvent pas de champion, ce qui diminue le nombre de vrais digrammes fautifs.

L'ensemble de cette étape est réitéré, en utilisant à chaque fois les résultats de la passe précédente, jusqu'à ce que les décomptes de couples obtenus soient stabilisés.

2.2.1.3 Alignement

2.2.1.3.1 Alignement simple

Nous disposons donc à présent, pour chaque prototype de mot source (respectivement cible) du corpus, d'une liste de prototypes de mots cible (respectivement source) associés, avec le nombre d'occurrences pour chaque couple. Il nous reste à les replacer dans les bisegments du corpus. Le cas le plus fréquent est le cas le plus simple, où un mot typographique du segment cible ne peut être associé, d'après la liste, qu'à un mot typographique du segment source et inversement.

La figure suivante présente un exemple des alignements obtenus. Les coefficients indiqués sont un indice de confiance lié à l'information mutuelle.

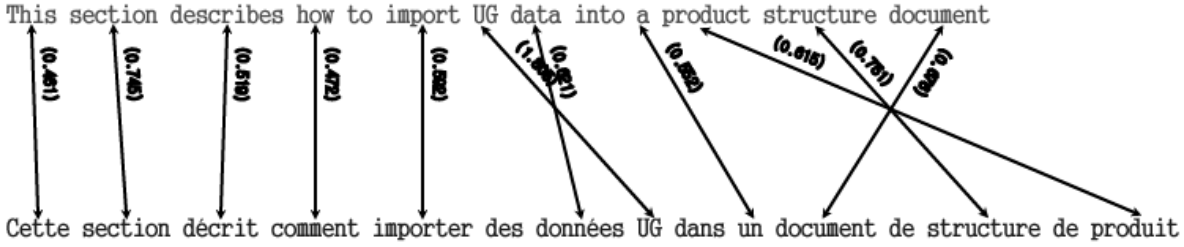

Figure 39 : correspondances établies sur un bisegment

On constatera qu'il y a un hapax : le mot « UG », dont l'indice de confiance vaut 1. Par ailleurs, le mot anglais « into » n'a pas trouvé son partenaire, le mot français « dans », parce que le digramme n'a pas été statistiquement mis en évidence sur l'ensemble du corpus.

[13] Ces cas sont indiqués en gras dans le tableau.

2.2.1.3.2 Résolution des cas difficiles

Certaines situations d'alignement sont plus complexes. Par exemple, un segment contient plusieurs occurrences du même mot typographique, en particulier les mots très fréquents comme « the » en anglais ou « de » en français. Il y également les cas d'information mutuelle égales conduisant à des *ex æquo*, qui concernent en particulier les hapax.

Ces situations ont été gérées en appliquant l'heuristique des moindres croisements : ayant établi les correspondances non équivoques, nous avons exploré l'ensemble des solutions possibles pour les cas litigieux et avons conservé les solutions qui entraînaient le moins de croisements.

Cependant, cette méthode n'est pas complètement satisfaisante. Dans le deuxième exemple ci-dessous, la résolution de l'attachement du mot typographique anglais « the » (avec une minuscule) n'a pas été faite. Il y a deux candidats dans la version française du bisegment, deux occurrences du mot « la » (avec une minuscule). Chacune des deux possibilités entraîne deux croisements. Nous avons préféré ne pas chercher à trancher dans ce type de cas afin d'introduire le moins de bruit[14] possible dans le reste du traitement.

Figure 40 : alignement incomplet

Cet exemple illustre aussi une situation où un seul mot typographique anglais, « updated », correspond linguistiquement à plusieurs mots typographiques français, « mise à jour ». Le digramme établi associe le mot typographique « updated » et le mot typographique « jour ». Nous verrons plus loin comment l'ensemble du syntagme « mise à jour » sera finalement associé à « updated ».

Nous avons imaginé une autre méthode que l'heuristique des moindres croisements pour gérer les cas d'ambiguïté. Cette méthode, fondée sur les arbres binaires, est brièvement décrite en 3.2.1.2. Elle est en cours de test au moment de cette rédaction.

2.2.1.3.3 Conclusion

Finalement, lorsque les associations sont figées dans l'ensemble du corpus, chacune d'entre elles constitue une occurrence de vrai digramme. Les mots typographiques non associés constituent les faux digrammes.

Ce traitement nous a permis d'établir des correspondances élémentaires entre les mots typographiques du corpus. Dorénavant, nous ne considérerons plus les segments comme des suites de mots typographiques mais comme des suites de digrammes, dont nous connaissons le nombre d'occurrences dans le corpus, qu'il s'agisse de vrais digrammes ou de faux digrammes.

[14] C'est-à-dire des correspondances fautives

2.2.2 Sécabilité et arbres binaires

Nous allons à présent nous efforcer de structurer les segments selon l'axe syntagmatique. Pour ce faire, nous allons mettre en évidence des sous-segments « gigognes », grâce à l'indice de sécabilité et aux arbres binaires. Quelques propriétés utiles de ces arbres binaires seront ensuite précisées.

2.2.2.1 Sécabilité

2.2.2.1.1 Notion de sécabilité

Dans une analyse logique de type linguistique, on découpe en général un énoncé en propositions, puis en syntagmes etc. On admet, ce faisant, que certains mots sont plus soudés entre eux que certains autres. C'est ce qui inspire la notion de sécabilité. Nous utiliserons cette propriété locale entre deux objets pour déterminer une hiérarchie des frontières à l'intérieur du segment.

Les segments des mémoires de traduction sont vus comme des suites finies de digrammes, dans chaque langue. Certains digrammes, les vrais digrammes, sont communs aux deux segments d'un bisegment. Ils se présentent en général dans un ordre différent dans chacun des segments.

Sur l'axe linéaire de l'énoncé, nous considérons les séparateurs entre chaque digramme et les munissons d'un indice, la « sécabilité* », caractérisant la cohésion du segment à cet endroit. Autrement dit, elle indique dans quelle mesure le segment est sécable entre deux digrammes quelconques du segment.

Quand la sécabilité d'un séparateur est faible, c'est que les parties droite et gauche du segment sont souvent en coocurrence dans le corpus : le segment est donc peu sécable à cet endroit. Quand elle est forte, les parties droite et gauche ne se présentent pas particulièrement souvent ensemble dans le corpus et sont donc indépendantes l'une de l'autre.

2.2.2.1.2 Comment calculer la sécabilité ?

Pour calculer la sécabilité d'un séparateur, nous évaluons l'information mutuelle entre les deux parties du segment de chaque côté du séparateur en en calculant une estimation sur une fenêtre glissante, par exemple deux digrammes avant et après le séparateur.

Nous affectons alors un rang aux séparateurs, allant de 1, où l'information mutuelle est la plus faible, à n-1 (pour n digrammes), là où elle est la plus forte. C'est ce rang que nous appelons sécabilité. En cas d'égalité, c'est le séparateur le plus à gauche qui reçoit arbitrairement la sécabilité la plus forte.

La sécabilité nous donne une indication numérique sur la dépendance statistique qu'entretiennent les deux parties séparées au point où elle est calculée. Pour de plus amples informations sur le rapport entre l'information mutuelle entre deux unités du segment et la cohésion de la phrase, on pourra se reporter au travaux sur les *distituents* de David Magerman et Mitchell Marcus, notamment : [29].

Il nous faut insister sur le fait que nous considérons les rangs, c'est-à-dire des nombres entiers, et non pas l'information mutuelle elle-même. Un indicateur équivalent, autre que

l'information mutuelle, peut être utilisé sans changer fondamentalement la définition de la sécabilité.

Par exemple, sur ce schéma, où **x** représente les digrammes dans le segment, les index de sécabilité sont placés sous chaque séparateur.

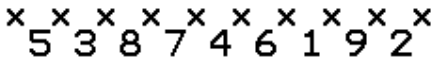

Figure 41 : exemple d'indices de sécabilité sur un segment

Au point **9**, la sécabilité est la plus grande : les parties gauche et droite sont pratiquement indépendantes. Au point **1**, la sécabilité est la plus faible : nous ne couperons pas le segment en ce point.

2.2.2.1.3 Calcul effectif de la sécabilité

Lors du calcul effectif, nous supposons que la sécabilité est inversement liée à l'information mutuelle qu'entretiennent les parties droite et gauche du segment et que cette information mutuelle peut être approchée sur une fenêtre glissante de quatre digrammes.

Ce calcul peut s'effectuer à l'aide de bigrammes et de trigrammes, qui dans notre contexte seront des suites de deux ou trois digrammes[15].

Par exemple, dans la sous-chaîne linéaire **[a,b,c,d]**, extraite d'un segment donné, on s'intéresse à la sécabilité entre le digramme **b** et le digramme **c**. Cette chaîne est couverte par deux trigrammes **[a,b,c]** et **[b,c,d]** et un bigramme : **[b,c]**. En combinant l'information mutuelle entre les chaînes **[a,b]** et **[c]** dans **[a,b,c]**, l'information mutuelle de **[b]** et **[c,d]** dans **[b,c,d]** et l'information mutuelle de **[b]** et **[c]** dans **[b,c]** on obtient un indice de sécabilité.

En notant **n(x)** le nombre d'occurrences de **x** dans le corpus, **InfMutInt()** l'information mutuelle interne d'un bigramme ou d'un trigramme, et **sec([xy])** l'indice de sécabilité entre **x** et **y**, nous avons calculé :

$$InfMutInt([ab|c]) = \frac{n([abc])}{n([ab]) \cdot n([c])}$$

$$InfMutInt([b|cd]) = \frac{n([bcd])}{n([b]) \cdot n([cd])}$$

$$InfMutInt([b|c]) = \frac{n([bc])}{n([b]) \cdot n([c])}$$

La formule que nous avons retenue pour la sécabilité est donc la suivante :

$$sec([bc]) = \frac{n([ab]) \cdot n([bc]) \cdot n([cd])}{n([abc]) \cdot n([bcd])}$$

[15] Nous soulignons pour insister sur la différence entre les digrammes et les bigrammes…

L'indice de sécabilité est une mesure locale. Nous l'avons calculé sur quatre digrammes successifs ; il ne tient donc pas compte de la réalité de tout le segment. On peut sans doute imaginer d'autres manières de calculer l'indice de sécabilité, mais cette méthode nous donne des résultats expérimentaux très satisfaisants.

2.2.2.2 Arbres binaires de sécabilité

2.2.2.2.1 Parenthésage

L'évaluation des indices de sécabilité sur un segment revient à parenthéser le segment. Nous coupons le segment en deux parties au point de sécabilité maximale, puis nous séparons récursivement chaque sous-segment en deux parties, au point de sécabilité maximale de chacun, jusqu'à rencontrer des sous-segments réduits à un seul digramme. Voici le résultat obtenu à partir de l'exemple précédent :

$$(((x_5\ (x_3\ x))_8\ (x_7\ ((x_4\ x)_6\ (x_1\ x))))_9\ (x_2\ x))$$

Figure 42 : parenthésage binaire obtenu à partir des indices de sécabilité

Sur ce schéma, nous voyons donc que le segment est complètement partagé au point de sécabilité **9** et qu'inversement plusieurs associations de deux digrammes se sont formées. C'est le cas au point **1** comme prévu, mais aussi aux points **3**, **4** et **2**. Nous arrivons ainsi à mettre en évidence une certaine cohésion locale sur le segment.

2.2.2.2.2 Arbres binaires de sécabilité

Nous pouvons exprimer cette cohésion locale sous la forme d'un « arbre binaire[16] de sécabilité* », dont les nœuds sont les séparateurs du segment.

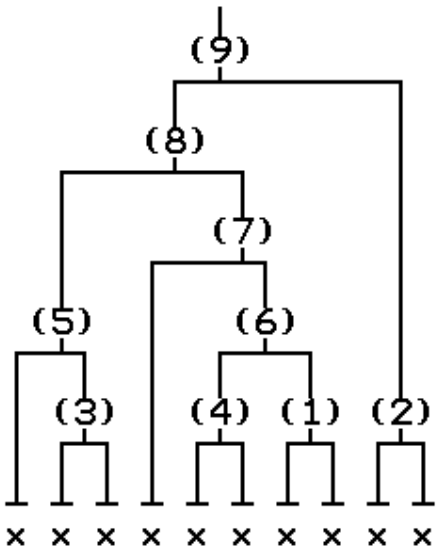

Figure 43 : arbre binaire correspondant

[16] Un arbre binaire admet deux sous-arbres binaires en chacun de ses nœuds, ou est vide.

Chaque nœud de l'arbre porte un index de sécabilité. Les deux parties du segment séparées en cet endroit correspondent aux deux branches de ce nœud. Les feuilles correspondent aux digrammes eux-mêmes. Nous appelons un tel arbre un « arbre binaire de sécabilité* »

Dans ce qui suit, nous dirons que les noeuds les plus élevés sont ceux qui sont proches de la racine, et les plus bas ceux qui sont proches des feuilles. La métaphore reflète plus les schémas présentés ici que l'ordre naturel... Nous dirons également qu'un nœud domine d'autres nœuds ou feuilles quand il est au dessus d'eux, c'est à dire plus proche de la racine.

Un nœud quelconque dans un arbre binaire domine deux nœuds. Nous dirons qu'ils sont voisins.

2.2.2.2.3 Importance des digrammes

Dans le segment, nous avons distingué les vrais digrammes et les faux digrammes. Nous allons maintenant approfondir la conséquence de cette distinction sur les arbres que nous avons construits.

Les digrammes sont porteurs d'une information plus riche que les mots typographiques qui les composent. Dans une certaine mesure, on peut considérer que les mots typographiques sont désambiguïsés en digrammes. Même les faux digrammes, qui sont réduits à des mots typographiques sans contrepartie dans l'autre langue, sont porteurs d'une information plus riche que si la construction en digrammes n'était pas faite. Ce surplus d'information réside dans le contraste avec les vrais digrammes.

Cela a une conséquence statistique sur la sécabilité observée entre les digrammes et, du même coup, sur la géométrie des arbres binaires de sécabilité. On constate, et ce n'est pas surprenant, que les syntagmes du type « base de données » ou « mis à jour » présentent une sécabilité interne faible. De fait, ils se retrouvent groupés sous forme de sous-arbres dans les arbres binaires de sécabilité.

Cependant, cette bonne propriété des digrammes a ses limites. Nous avons calculé les indices de sécabilité sur une fenêtre locale. Même en supposant une certaine propagation des propriétés syntaxiques du segment, ces indices perdent de leur pertinence pour les grandes sécabilités. Autrement dit, la fiabilité des sous-arbres diminue lorsque l'on s'approche de la racine.

Les propriétés que nous allons dégager à présent sont également étroitement dépendantes de la prise en compte des digrammes. S'il l'on calcule des arbres binaires de sécabilité sur les mots typographiques seulement, on ne pourra pas discerner les propriétés en question.

2.2.2.3 Quelques propriétés des arbres binaires de sécabilité

2.2.2.3.1 Saturation

Nous allons maintenant définir la propriété de « saturation* », propre à certains nœuds des arbres binaires de sécabilité. L'idée qui sous-tend la notion de saturation consiste à repérer les nœuds qui dominent le plus grand nombre possible de faux digrammes pour un ensemble de vrais digrammes donné. D'où la définition suivante : un nœud qui domine un ensemble donné de vrais digrammes, de telle sorte que le nœud immédiatement au dessus de lui domine un sur ensemble strict de vrais digrammes, sera dit saturé.

Une conséquence immédiate est qu'un nœud est saturé s'il domine au moins un vrai digramme et si le nœud voisin domine lui aussi au moins un vrai digramme.

Nous étendons la définitions de la saturation aux feuilles, c'est-à-dire aux digrammes : un faux digramme n'est jamais saturé. Un vrai digramme est saturé si le nœud (ou feuille) voisin domine au moins (ou est) un vrai digramme.

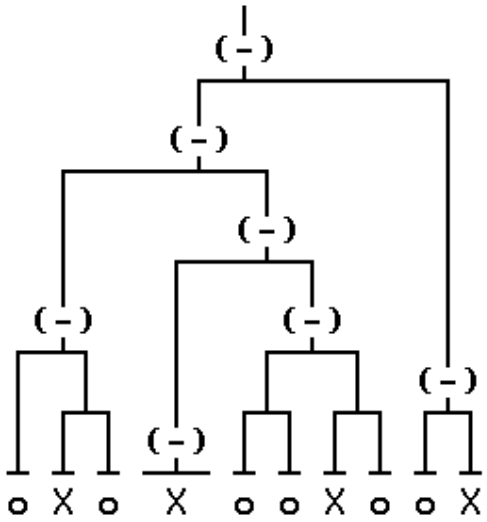

Figure 44 : vrais digrammes et faux digrammes

Sur ce schéma, les **X** représentent des vrais digrammes et les **o** des faux digrammes. Nous avons mis des tirets entre parenthèses sur les nœuds saturés. Il y a trois nœuds et neuf feuilles non saturés. On voit qu'un nœud quelconque domine ou bien deux nœuds ou feuilles saturés, ou bien deux nœuds ou feuilles non saturés. La racine de l'arbre elle-même est toujours considérée comme saturée. Il est possible qu'un nœud non saturé domine deux nœuds saturés (ce dernier cas n'est pas illustré sur la figure).

L'utilité de la propriété de saturation apparaîtra plus loin. Elle est liée au découpage du segment en sous-segments autonomes par traduction. Autrement dit, elle est incarne le principe de non compositionnalité interne des unités utilisées à un niveau supérieur de granularité.

2.2.2.3.2 Congruence des nœuds

Les arbres binaires donnent pour chaque digramme du segment une caractérisation de sa dépendance statistique vis-à-vis de son voisinage dans un segment. Une occurrence de vrai digramme apparaît par définition dans les deux segments d'un bisegment et donne lieu à un lien de correspondance dans le bisegment.

En comparant les arbres obtenus pour les deux langues et en utilisant les correspondances interlingues entre les deux segments, on peut extraire des sous-arbres cohérents vis à vis des deux langues.

Deux nœuds, un dans chaque arbre, seront dits « congruents* » s'ils dominent les mêmes occurrence de vrais digrammes. Par extension, on parlera également de sous-arbres congruents.

La congruence est une propriété qui ignore les faux digrammes.

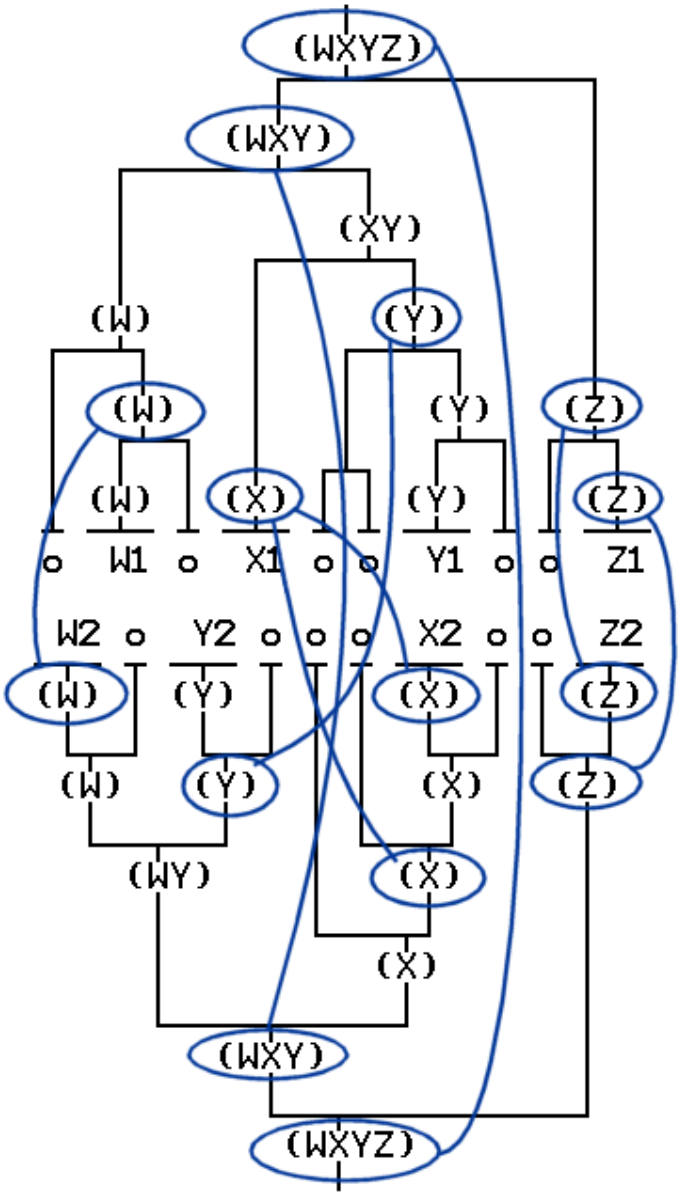

Figure 45 : nœuds congruents

Sur ce schéma, on a représenté deux arbres binaires de sécabilité l'un au dessus de l'autre, orientés inversement. Ils correspondent aux deux segments d'un bisegment. Ce bisegment contient quatre vrais digrammes, symbolisés par W, X, Y et Z. Ils relient les mots typographiques étiquetés de W1 à Z2 dans les langues numérotées 1 et 2. Nous avons indiqué entre parenthèses, sur chaque nœud de la figure, l'ensemble des vrais digrammes dominés par ce nœud.

Les liens de congruence apparaissent en bleu (lignes courbes). Nous n'en avons indiqué qu'un petit nombre, pour ne pas surcharger la figure.

Comme on le voit, tous les nœuds dans un arbre donné, n'ont pas, en général, de contre-partie congruente. Par exemple, le nœud {X,Y}, côté source, ou le nœud {W,Y}, côté cible, n'entrent ni l'un ni l'autre dans une relation de congruence. La seule possibilité de congruence concerne l'ensemble {W,X,Y}. La propriété de congruence, et c'est son intérêt, s'applique aux seuls nœuds qui dominent les mêmes ensembles de digrammes, s'ils existent.

2.2.2.3.3 Nœuds saturés et congruents : les amphigrammes

Revenons au formalisme TransTree. Les arbres binaires de sécabilité vont nous servir de canevas pour structurer le bisegment en amphigrammes.

Pour déterminer la granularité que nous voulons atteindre à une étape quelconque du calcul, nous cherchons à préserver la non-compositionnalité interne des unités que nous manipulons. Les vrais digrammes sont les points de repère dont nous disposons pour baliser l'énoncé. Ces vrais digrammes sont disséminés dans l'énoncé, au milieu de mots typographiques pour lesquels on n'a pas d'équivalent dans l'autre langue, les faux digrammes.

Ce qui nous intéresse, c'est de regrouper ces faux digrammes, non transposables, autour des vrais digrammes. On découpe ainsi l'énoncé en fragments centrés sur un vrai digramme, ce qui nous donne un pointeur sur l'image de ce fragment par traduction. Pour opérer ces regroupements de faux digrammes, nous comparons les deux arbres binaires afin de déterminer des points de rupture dans le segment qui coïncident entre l'arbre binaire de sécabilité du segment source et celui du segment cible.

Fondamentalement, toute paire de nœuds congruents peut être réécrite sous forme d'un amphigramme complexe, sans recourir à une structuration plus avancée. Par exemple, les nœuds dominant l'ensemble {W,X,Y}, évoqué dans l'exemple précédent, peuvent donner lieu à l'amphigramme suivant :

```
<amphigram>
 <text xml:lang='L1'>o <a loc='A'/> o <a loc='B'/> o o <a
loc='C'/> o</text>
 <text xml:lang='L2'><a loc='A'/> o <a loc='C'/> o o o <a
loc='B'/> o</text>
 <amphigram loc='A'>
  <text xml:lang='L1'>W1</text>
  <text xml:lang='L2'>W2</text>
 </amphigram>
 <amphigram loc='B'>
  <text xml:lang='L1'>X1</text>
  <text xml:lang='L2'>X2</text>
 </amphigram>
 <amphigram loc='C'>
  <text xml:lang='L1'>Y1</text>
  <text xml:lang='L2'>Y2</text>
 </amphigram>
</amphigram>
```

Figure 46 amphigramme créé avec les arbres binaires de sécabilité

Cependant, ce choix est arbitraire, et nous allons examiner, au chapitre 2.3, quelques stratégies de structuration des segments selon le formalisme TransTree, à partir des éléments précédents. Les briques pertinentes seront délimitées par l'analyse et non pas en fonction d'un besoin extérieur (recherche en texte plein, traduction etc.). C'est l'analyse qui détermine la granularité, c'est à dire la taille et le découpage des sous-segments considérés, sur le critère de leur non-compositionnalité interne.

2.2.3 Classification

Nous abordons maintenant l'axe paradigmatique. Le formalisme TransTree prévoit de manipuler des ensembles d'amphigrammes. C'est à leur constitution que nous allons consacrer ce chapitre. Dans un premier temps, nous préciserons notre choix de nous en tenir à une implémentation particulière des ensembles d'amphigrammes sous forme de classes d'amphigrammes, puis nous passerons en revue deux stratégies d'obtention de telles classes.

2.2.3.1 Constitution d'ensembles par classification

2.2.3.1.1 Ensembles et classes

Dans le formalisme TransTree, nous avons introduit la notion d'ensemble d'amphigrammes à l'occasion de la présentation des amphigrammes génériques. Nous n'avons rien dit de la nature de ces ensembles que nous avons volontairement modélisés en conservant un haut degré de généralité. En particulier, nous avons choisi ce terme neutre d'« ensemble » (« set » en anglais).

Notre modélisation n'est pas sans restriction cependant : nous avons dit qu'il s'agissait d'ensembles de prototypes, ce que sont les amphigrammes atomiques et les amphigrammes génériques, et non pas d'ensembles d'occurrences comme les amphigrammes complexes et compacts. De fait, les amphigrammes atomiques et génériques ne contiennent pas d'amphigramme fils, contrairement aux amphigrammes complexes et compacts ; ils ne sont pas liés à tel ou tel bisegment particulier. Il en existe généralement plusieurs occurrences dans notre corpus : il ne s'agit donc pas, en général, d'hapax. C'est d'ailleurs pour cela que l'on peut, en ce qui les concerne, distinguer occurrences et prototypes (Cf. 1.3.2.1.3.). Toutes ces caractéristiques en font des objets potentiellement réutilisables.

On peut imaginer de nombreuses implémentations de la notion d'ensemble d'amphigrammes. Nous avons choisi de construire des relations d'équivalence sur les prototypes d'amphigrammes et donc d'implémenter les ensembles prévus par notre modèle sous forme de classes d'équivalence. Nous ne parlerons donc plus d'ensemble dorénavant, mais uniquement de classes. Ces classes, par construction, seront disjointes et concerneront des prototypes et non pas des occurrences.

Cette distinction entre classes de prototypes et classes d'occurrences recoupe celle que l'on entend parfois entre « classification dure » et « classification molle ». On caractérise ainsi le fait qu'un élément donné appartient à une seule classe, dans le premier cas, ou bien qu'il appartient éventuellement à plusieurs classes, dans le deuxième cas.

Dire qu'un prototype appartient à plusieurs classes peut signifier deux choses : ou bien que les différentes occurrences de ce prototype appartiennent à des classes éventuellement différentes, chaque occurrence appartenant à une seule classe, ou bien que les occurrences elles-mêmes sont susceptibles d'appartenir à plusieurs classes.

Dans ce dernier cas, on peut considérer que les classes en question expriment diverses propriétés *in extenso*, et qu'une même occurrence peut, par exemple appartenir simultanément à la classe des mots de 7 lettres, à la classe des termes de jardinage, et à la classe des verbes. Autrement dit, on considère une superposition de plusieurs classifications. Nous évoquons cette situation uniquement par souci de complétude.

Une véritable classification molle des prototypes consiste plutôt à effectuer une classification dure des occurrences : On répartit les occurrences en classes et on constate *a posteriori* que, par exemple, il existe des occurrences de **boucher** en tant que nom et d'autres occurrences de **boucher** en tant que verbe. La classification est donc dure du point de vue des occurrences et molle du point de vue des prototypes, car une occurrence donnée ne peut pas être à la fois nom et verbe.

La classification des prototypes en classes dures, c'est-à-dire en classes d'équivalence, est finalement la plus restrictive des options que nous avons évoquées. Elle consiste à dire qu'un prototype donné appartient à une unique classe. Cela vaut donc pour toutes ses occurrences. C'est elle que nous avons choisi de mettre en œuvre parce qu'elle nous est apparue comme la plus accessible et la plus exploitable. En effet, les objets que nous manipulons sont beaucoup plus fins que de simples mots typographiques. Nous ne cherchons pas à les raffiner mais au contraire à les généraliser.

Nous utiliserons dans la suite le terme de « classification » pour désigner tout processus au terme duquel des objets sont répartis en classes. Le terme anglais correspondant est habituellement « clustering ».

2.2.3.1.2 Factorisation et extrapolation

Dans notre présentation de l'axe paradigmatique, nous avons souligné que l'objectif de la classification est double :

Factoriser les données rencontrées dans le corpus

Extrapoler les connaissances rencontrées dans le corpus

Réaliser le premier objectif permettra de lutter contre la rareté des données, phénomène inhérent au corpus quelle que soit sa taille. En effet, le nombre d'hapax est toujours très important, tant sur les formes rencontrées que sur les suites de formes ou autres constructions. Si l'on compare la représentativité de deux corpus de tailles différentes, on constate que les données présentes dans le petit corpus sont mieux représentées statistiquement dans le plus grand corpus, mais que le plus grand corpus contient aussi d'autres données, rares et donc mal représentées. Une bonne factorisation des formes ou autres constructions nous fournira des informations statistiques plus fiables, en particulier sur ces données rares.

Le deuxième objectif est lié à la constitution des amphigrammes génériques. Nous les avons introduits dans le modèle TransTree avec l'idée de construire automatiquement des propositions de traduction. Etant donné un segment exprimé dans une des langues du corpus mais non présent dans le corpus, nous voudrions en extrapoler une représentation TransTree et donc produire un bisegment. Les arbres d'amphigrammes complexes, recueillis dans le corpus, sont étroitement liés à ce dernier, et nous ne cherchons pas à les réutiliser en tant que tels pour représenter un segment inconnu du corpus. En revanche, le but des amphigrammes génériques, extrapolés à partir du corpus, est de fournir une connaissance potentiellement utilisable pour analyser un segment source inconnu et générer un segment cible.

Cette dualité n'est pas sans conséquence sur les traitements que nous allons mettre en œuvre, comme nous allons le voir à présent.

2.2.3.1.3 Classes monolingues, classes bilingues

La simultanéité de ces deux ambitions pose une autre question : quelle est la nature des objets classés ? Nous avons dit que nous classions des prototypes, et non pas des occurrences,

et nous avons parlé de prototypes d'amphigrammes. Le souci de lutter contre la rareté des données est du domaine du calcul, et n'a pas de rapport direct avec le formalisme de représentation des données.

Nous voulons en particulier classer les digrammes, qu'il s'agisse de vrais digrammes ou de faux digrammes, absents, en tant que tels, du formalisme. De fait, si les vrais digrammes, ou amphigrammes atomiques, sont des objets bilingues, tout comme les amphigrammes génériques, les faux digrammes sont des objets monolingues, puisqu'ils ne sont constitués que d'un mot typographique dans l'une des deux langues du corpus.

À partir de ce constat, on peut anticiper deux types de classification :

Classification monolingue, pertinente dans une seule des deux langues du corpus,

Classification bilingue, pertinente simultanément dans les deux langues du corpus.

Les classes monolingues regroupent des éléments selon des propriétés observées dans une seule des langues du corpus. Ces éléments peuvent être monolingues, autrement dit des faux digrammes, ou bilingues, c'est-à-dire des amphigrammes atomiques ou génériques. La nature des éléments n'est pas pertinente : ce qui compte, c'est l'utilisation qui en est faite dans une seule des langues du corpus. L'objectif poursuivi dans la création de telles classes est essentiellement une amélioration du calcul des indices de sécabilité, par suite de la construction des arbres binaires et donc de la factorisation des données.

Les classes bilingues, quant à elles, mettent en jeu les deux langues du corpus et ne regroupent que des éléments bilingues. Il ne peut donc pas s'agir de faux digrammes. Le calcul de classes bilingues sur les amphigrammes atomiques a pour objectif l'émergence d'amphigrammes génériques, voire d'ensembles d'amphigrammes génériques, tels qu'ils ont été définis dans TransTree.

Pour une langue donnée, nous allons donc effectuer une classification monolingue des éléments, qui eux peuvent être soit monolingues (faux digrammes), soit bilingues (vrai digrammes) en fonction de considérations sur leur contexte d'apparition dans la langue. Insistons sur le fait que cette classification est monolingue, même si elle est effectuée sur des éléments bilingues. Par exemple, on distingue d'emblée des digrammes comme (**number**, **nombre**) et (**number**, **numéro**) et l'on considère la distribution statistique en anglais de ces deux objets parmi les autres digrammes pour les classer.

On voit que les digrammes sont très raffinés par rapport à de simples mots typographiques, ce qui nous encourage à nous restreindre à des classes dures, avec une seule classe admissible par digramme. Cela reste vrai dans le cas de faux digrammes, différent des vrais digrammes construits avec le même mot typographique. Par exemple, les digrammes (**of**, **de**) et (**#null#**, **de**) sont différents et leurs distributions statistiques en français parmi les autres digrammes sont différentes. Cette classification comprend donc également les faux digrammes propres à la langue en question. Naturellement, on conduit ce type de classification dans chacune des deux langues, ce qui conduit à deux jeux de classes.

Nous allons maintenant exposer différentes stratégies de classification conduisant à différents jeux de classes.

2.2.3.2 Méthodes de classification

2.2.3.2.1 Classes simplement contextuelles

Nous commençons par présenter le premier *modus operandi* que nous avons mis en œuvre pour la constitution de ces classes. Nous avons imaginé et implémenté de nombreuses variations sur ce schéma de base.

Les classes que nous avons obtenues se fondent sur les contextes d'apparition des objets du corpus. Il s'agit de digrammes, vrais et faux.

La classification s'effectue alors en fonction du contexte d'apparition des unités. La logique sous-jacente est que ce contexte ne dicte pas la présence d'un unique objet à cet endroit, mais plutôt d'un type d'objets, autrement dit, d'une classe. Tous les objets d'une même classe sont donc équivalents à cet endroit. La factorisation induite en découle.

Les classes sont établies à partir des trigrammes collectés dans le corpus. Les trigrammes sont les suites **[abc]** de 3 digrammes dans un segment, pour une langue donnée. Les segments sont étendus à gauche et à droite par deux symboles spéciaux, **"#left#"** et **"#right#"**. Un trigramme **[abc]** modélise un digramme **b** dans son contexte, **[a_c].**

La relation d'équivalence est simple : tous les digrammes **b**, apparaissant dans un même contexte, **[a_c]**, dans l'ensemble des trigrammes de référence, sont équivalents. Les digrammes, les trigrammes et les contextes s'entendent ici comme des prototypes. Le fait d'apparaître concerne une occurrence au moins.

L'un des écueils liés à cette méthode simple est que la dispersion des digrammes peut-être telle, au sein du corpus, qu'en définitive une seule classe de digrammes (et de contextes) émerge par ce procédé. Cet écueil est lié à la taille du corpus considéré. Pour palier cet inconvénient, nous avons choisi pour un corpus donné, de restreindre le nombre de trigrammes pris en compte dans la détermination des classes d'équivalence en appliquant un seuil de pertinence lié à leur représentativité dans le corpus.

Le jeu de classes obtenu varie considérablement en fonction du seuil appliqué dans notre méthode. Nous discutons ce seuil et donnons le détail des expérimentations faites dans la troisième partie : voir 3.2.3.1

2.2.3.2.2 Classes optimisées par algorithme génétique

Nous avons imaginé un algorithe génétique pour effectuer une classification visant une plus grande adéquation des classes à leur utilisation. En effet, les classes simplement contextuelles sont des classes *a priori*, qui ne nous garantissent pas de factoriser l'information de manière cohérente avec leur utilisation.

Dans un algorithme génétique, une fonction de performance est régulièrement testée sur une population de génomes pour sélectionner les meilleurs individus. Ceux-ci sont ensuite croisés entre eux et sont éventuellement sujets à des mutations spontanées. De cette manière, la population est constamment renouvelée. Si ce type d'algorithme est bien adapté au problème posé, la performance des génomes sélectionnés croît régulièrement, en raison de la création des nouveaux génomes à partir des meilleurs de la génération précédente. L'algorithme s'arrête quand la population des meilleurs génomes vieillit au-delà d'un certain seuil, sans être remplacée par des génomes plus performants.

Nous présentons en détails les caractéristiques en 3.2.3.2.

Nous avons implémenté une version « à vide » de cet algorithme, sur des données factices et avec une fonction de performance simplissime. Nous avons observé une bonne convergence en un temps admissible pour des ensembles relativement grands.

Il nous reste à préciser quels objets nous désirons classer et la fonction de performance à utiliser.

2.2.3.2.3 Fonctions de performance

Nous présentons ici un exemple de fonction de performance utilisable dans l'algorithme génétique décrit plus haut. Nous nous intéressons à l'ensemble des trigrammes du corpus, qui vont être transformés en triclasses après la classification de leurs éléments constitutifs.

L'idée repose sur la comparaison des deux indices de sécabilité à l'intérieur d'un trigramme sachant que nous avons une liste des trigrammes récoltés dans le corpus pour une langue donnée.

Dans une première option l'indice de sécabilité est simplifié : entre **a** et **b**, il ne tient compte que des nombres d'occurrences de **a**, de **b** et du bigramme **[ab]**. Dans un trigramme **[abc],** on considère les deux points de sécabilité entre **a** et **b** d'une part et entre **b** et **c** d'autre part. L'indice de sécabilité est soit supérieur à gauche (entre **a** et **b**) soit supérieur à droite (entre **b** et **c**), éventuellement égal des deux côtés. Dans les deux premiers cas, de loin les plus fréquents, chaque trigramme **[abc]** est donc orienté, à droite dans le premier cas, à gauche dans le second.

La classification nous amène non plus à considérer des digrammes **a**, **b** et **c** mais des classes de digrammes : **A**, **B** et **C**. Le calcul des indices de sécabilité peut être effectué sur les biclasses, autrement dit sur les suites de classes de type **[AB]**. L'orientation à l'intérieur des triclasses **[ABC]** peut être évaluée. Chaque triclasse recouvre plusieurs trigrammes, avec chacun une orientation.

L'optimisation de notre classification consiste à faire en sorte que les trigrammes représentés par une triclasse aient tous la même orientation (éventuellement nulle) et que cette orientation soit celle calculée sur la triclasse. Autrement dit, nous voulons que la classification des constituants conserve l'orientation des trigrammes.

La fonction elle-même peut être le rapport entre le nombre de trigrammes en accord avec l'orientation de leur triclasse avec le nombre de trigrammes représentés pas la triclasse.

Cette optimisation nous semble suffisamment souple pour autoriser la convergence de l'algorithme et suffisamment contraignante pour classer effectivement les digrammes. En effet, si la succession linéaire des orientations des triclasses est identique à celle des trigrammes à l'intérieur d'un segment, il est vraisemblable (mais pas vrai en général) que les arbres binaires résultant seront similaires. Cette vraisemblance s'appuie, bien sûr, sur la taille du corpus.

Une variante intéressante, plus lourde, consisterait à classer les occurrences des digrammes. Cette classification serait alors dure du point de vue des occurrences de digrammes mais molle du point de vue des « prototypes » de digrammes. Un même digramme pourrait en effet appartenir à deux classes différentes, lors de deux occurrences différentes.

Cette option permettrait de faire un calcul plus complet de l'indice de sécabilité de référence pour chaque occurrence de trigramme. On utiliserait alors le calcul fait dans le corpus qui consiste à utiliser deux trigrammes successifs **[abc]** et **[bcd]** pour calculer l'indice de sécabilité entre **b** et **c**.

Le calcul des deux indices de sécabilité dans les triclasses **[ABC]** serait, lui, restreint à l'utilisation de biclasses **[AB]** et **[BC]**.

Dans ce cas, la conservation de l'orientation des sécabilité conduirait non seulement à factoriser les données, mais aussi à concentrer à l'intérieur des classes une part de l'information diffuse le long du segment. Nous pouvons proposer comme métaphore, l'augmentation du contraste d'une image.

Ce deuxième indice de sécabilité n'a pas été testé au moment ou nous rédigeons ce rapport.

2.2.3.3 Utilisation effective des classes obtenues

2.2.3.3.1 Des classes monolingues aux classes bilingues

Nous pouvons construire une classification bilingue des vrais digrammes à partir des deux classifications monolingues par produit cartésien des partitions. Pour ce faire, on s'intéresse aux seuls éléments bilingues (les vrais digrammes). Si un élément appartient à la classe C1 de la première partition et à la classe C2 de la deuxième partition, il appartient du même coup à la classe (C1, C2) élément du produit cartésien des deux partitions, par intersection des classes C1 et C2. Quand l'intersection est vide, (C1, C2) est en fait une classe vide. La nouvelle partition est donc incluse dans le produit cartésien des deux partitions d'origine, dont on a exclu les classes vides.

De cette manière, nous obtenons deux types de classes : d'une part des classes monolingues, exploitables dans la construction des arbres binaires de sécabilité dans chaque langue ; et d'autre part des classes bilingues que l'on pourra utiliser comme ensembles dans la définition des amphigrammes génériques.

Dans le formalisme TransTree, nous avons prévu que les spécificateurs d'ensembles et les indicateurs d'ensembles peuvent être multivalués. On peut donc utiliser cette facilité d'encodage pour préciser l'appartenance simultanée à deux classes monolingues.

2.2.3.3.2 Des classes bilingues aux amphigrammes génériques

À l'aide des méthodes décrites ci-dessus, nous pouvons donc parvenir à rassembler les amphigrammes atomiques, c'est-à-dire les vrais digrammes, identifiés dans notre corpus en classes bilingues, qui tiennent lieu d'ensemble dans notre modèle TransTree.

De ce fait, les amphigrammes complexes dont l'armature textuelle est identique et qui admettent des amphigrammes atomiques fils de même classes forment un prototype nouveau. Tous les amphigrammes complexes rencontrés dans le corpus en sont des instances particulières, dont les amphigrammes atomiques fils appartiennent aux ensembles particuliers décrit par ce prototype en ses points d'insertion. Ce prototype nouveau est un amphigramme générique.

Les amphigrammes génériques obtenus peuvent eux-mêmes être rassemblés en ensembles selon leur contexte d'apparition. Les amphigrammes complexes du corpus admettant les mêmes ensembles d'amphigrammes génériques en leur point d'insertion constituent à leur tour un nouveau type de prototype : un amphigramme générique de niveau supérieur.

On voit donc que les amphigrammes génériques ne sont pas établis directement par classification, mais par classification des amphigrammes référencés en leurs points d'insertion. On peut considérer qu'ils constituent de fait des ensembles d'amphigrammes

complexes, mais ces ensembles ne sont pas pertinents en tant que tels dans TransTree et n'ont pas d'identifiant.

2.2.3.3.3 D'autres types de classes

Comme nous l'avons vu, plusieurs types de critères peuvent guider la constitution de classes. Nous pouvons envisager de rassembler des occurrences sur des critères extrinsèques, liés au contexte dans lequel elles apparaissent, comme sur des critères intrinsèques, liés à la forme des prototypes.

Typiquement, la plupart des paradigmes flexionnels du français se reconnaissent à une certaine similitude formelle entre leurs composantes. Par exemple, « chantes », « chantera » et « chantions » font partie d'un même paradigme, celui du verbe « chanter ». Cependant, « chantier » ne fait pas partie de ce paradigme, non plus que « chantre », malgré leur ressemblance de surface. Ces constats connus nous amènent à ne pas chercher à effectuer de classification sur un critère intrinsèque pour ce qui concerne les mots typographiques.

En revanche, pour les vrais digrammes, bilingues par essence, l'expérience peut être envisagée du fait de la plus grande richesse d'information. De même, pour les amphigrammes, bilingues eux aussi, et dont la structure est plus élaborée, on peut penser qu'une classification selon un critère intrinsèque pourrait avoir une certaine pertinence en termes d'extrapolation. Cependant, nous n'avons pas eu le temps d'expérimenter cette voie de constitution de classes par cognation.

2.3 Différentes méthodes de construction de TransTree

Les éléments de construction présentés dans le chapitre précédent doivent à présent être articulés pour modéliser un bisegment selon le modèle TransTree. À l'aide des digrammes, des arbres binaires de sécabilité et des classes, nous allons donc explorer différentes méthodes de construction de l'arbre TransTree. Le passage de deux arbres binaires à un arbre n-aire peut être fait de plusieurs manières. Dans ce chapitre, nous dégageons d'abord une démarche générale et nous étudions ensuite différents protocoles d'application, chacun en mettant en œuvre différents aspects.

2.3.1 Démarche générale

Pour établir un arbre de correspondances TransTree entre deux segments en traduction mutuelle, nous allons exploiter statistiquement l'ensemble du corpus à notre disposition. Les observations que nous faisons donnent lieu à une réécriture progressive du corpus avec des éléments nouveaux, eux-mêmes consignés dans une base de connaissances.

2.3.1.1 Récapitulatif des éléments de construction de TransTree

2.3.1.1.1 Digrammes : vrais digrammes, faux digrammes

Rappelons que les digrammes sont en général des couples de mots typographiques appartenant aux deux langues du corpus. Au-delà du lien de traduction mutuelle de ces

couples, il faut garder à l'esprit le fait que chaque mot désambiguïse l'autre, raffinant ainsi notre étude dans chacune des contextes monolingues. Une fois que les digrammes ont été établis dans un bisegment, nous considérons chacun de ses segments constitutifs non plus comme une suite de mots typographiques mais comme une suite de digrammes.

Nous avons introduit les notions de vrais digrammes et de faux digrammes. Les vrais digrammes sont constitués de deux mots typographiques exactement. Les faux digrammes sont réduits à l'un de ces deux mots seulement, dans une langue ou dans l'autre. La notion de vrais et faux digrammes permet de distinguer les occurrences d'un mot typographique ayant une contrepartie (vrai digramme) de celles qui n'en ont pas (faux digrammes). De ce fait, le faux digramme est porteur d'une information plus riche que le seul mot typographique qui lui correspond.

Dans notre construction les vrais digrammes s'identifient aux amphigrammes atomiques prévus par le modèle TransTree.

2.3.1.1.2 Arbres binaires de sécabilité : saturation, congruence

À l'aide de l'index de sécabilité, nous avons organisé les segments en arbres binaires. La sécabilité est un index local, comme nous le repréciserons plus loin. Une conséquence est que la pertinence des arbres binaires est plus grande près des feuilles qu'elle ne l'est à la racine.

Les arbres binaires sont établis sur des suites de digrammes. Les feuilles sont constituées soit de vrais digrammes, soit de faux digrammes, ce qui permet d'introduire la notion de saturation dans l'arbre binaires. Un nœud saturé se caractérise par l'ensemble des vrais digrammes qu'il domine. Il est saturé parce que son père domine au moins un autre vrai digramme.

Quand on constitue les arbres binaires de sécabilité des deux segments d'un bisegment, les mêmes occurrences de vrais digrammes sont mises en jeu. Deux nœuds sont congruents, entre ces arbres binaires, s'ils dominent le même ensemble d'occurrences de vrais digrammes.

Les propriétés de congruence et de saturation sont indépendantes l'une de l'autre et sont à la base de notre construction des amphigrammes complexes prévus dans le modèle TransTree.

2.3.1.1.3 Classes : classes monolingues, classes bilingues

Les différentes constructions de classes que nous avons imaginées sont toutes monolingues à la base. Elles reposent en effet sur le contexte d'apparition dans une langue donnée des objets à classer. Au niveau des digrammes, ces classes monolingues concernent indifféremment les vrais digrammes, bilingues, et les faux digrammes, monolingues.

À partir du produit cartésien des deux jeux de classes obtenus, restreints aux seuls objets bilingues, nous construisons des classes bilingues par intersection de classes. Ces intersections peuvent être également mises en œuvre en utilisant la multivaluation des spécificateurs d'ensembles et des indicateurs d'ensembles.

Ces classes bilingues incarnent les ensembles d'amphigrammes atomiques prévus par le modèle TransTree, donnant ainsi corps aux amphigrammes génériques. Ceux-ci peuvent être à leur tour rassemblés en classes et donner ainsi lieu à une hiérarchie d'amphigrammes génériques.

2.3.1.2 Etapes principales du traitement du corpus

2.3.1.2.1 Remarques liminaires

Nous présentons ici les principes généraux de la construction des arbres TransTree à partir d'un bisegment. Il s'agit d'un cadre général. Trois propositions de construction concrète sont décrites au chapitre 2.3.2 Elles prennent progressivement en compte les différents principes exposés ici. Notamment, les deux premières constructions ne mettent pas en œuvre la constitution de classes. L'ensemble des idées rassemblées dans ce chapitre n'est véritablement repris que par la troisième proposition de construction.

Nous organisons une progression dans l'analyse du segment grâce à des coupures successives dans les arbres binaires de sécabilités. Ces coupures permettent de contrer la mauvaise fiabilité des arbres binaires pour les nœuds proches de la racine.

D'autres méthodes de construction de Transtree, statistiques ou non, plus ou moins riches etc. peuvent naturellement être envisagées. De toute façon, TransTree ne garantit pas l'unicité de la structuration pour un bisegment donné et différentes méthodes de calcul peuvent conduire à des résultats différents.

2.3.1.2.2 Principes de base de la méthode envisagée

Nous allons mettre en œuvre la démarche épistémologique proposée au 1.3.2. L'analyse du corpus est fondée sur le cycle illustré par le schéma suivant :

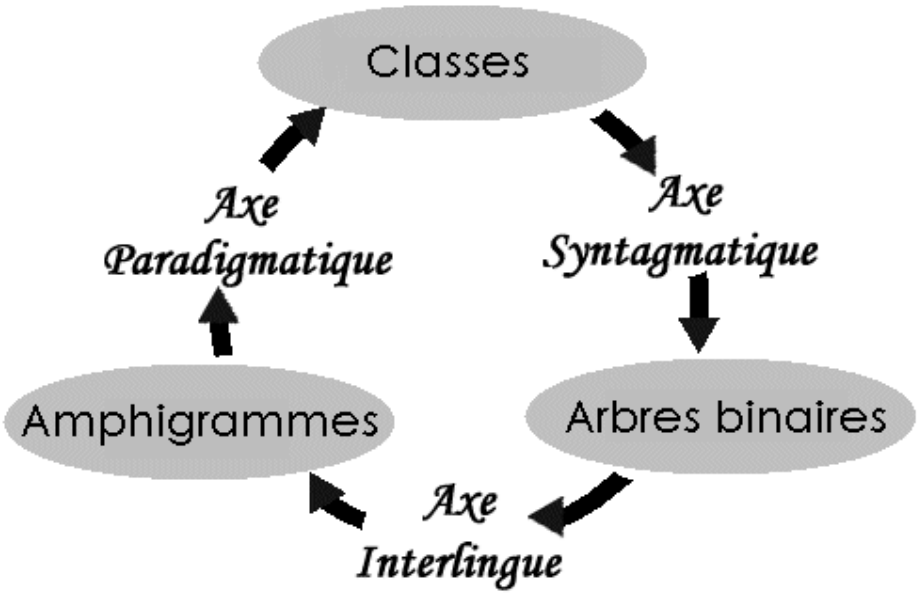

Figure 47 : schéma de la démarche épistémologique adoptée

Détaillons ce schéma.

Partons des amphigrammes. Il s'agit d'objets concrets, plus ou moins complexes, que l'on sait reconnaître dans un bisegment du corpus. De fait, les segments sont vus comme des suites d'amphigrammes, établies par coupure dans les arbres binaires comme nous allons le voir. Au

départ, il s'agit de digrammes[17]. Nous réunissons ces amphigrammes en ensemble par classification, selon l'axe paradigmatique.

Dans le corpus, les amphigrammes que nous reconnaissons sont maintenant remplacés par les identifiants de classes auxquelles ils appartiennent. Chaque segment est donc constitué d'une suite de classes d'amphigrammes. L'étude de leur agencement conduit à une structuration en arbres binaires de sécabilité selon l'axe syntagmatique.

L'application du principe d'harmonie entre granularité et compositionnalité, concrétisé par les propriétés de saturation et de congruence, nous permet de découper ces arbres en isolant des correspondances autonomes. Les parties isolées constituent de nouveaux amphigrammes en comparant les sous-arbres, selon l'axe interlingue.

Cette vision des choses est assez idéale, mais elle nous sert de guide méthodologique.

2.3.1.2.3 Récriture progressive du corpus

Au fur et à mesure que l'on parcourt ce cycle, l'ensemble du corpus est réécrit avec les objets nouvellement créés, ou du moins des identifiants de ces objets. Les objets eux-mêmes sont enregistrés dans une base de connaissances. À chaque cycle, les amphigrammes observés au cycle précédent sont donc remplacés par d'autres.

Ces amphigrammes nouveaux, issus de la structuration de chaque bisegment, représentent des paires de sous-segments de plus en plus longs. L'analyse d'un bisegment s'arrête quand il s'identifie tout entier à un amphigramme unique. Cet amphigramme ultime constitue la racine d'un arbre TransTree.

Selon la longueur des segments, la fin de l'analyse intervient après plus ou moins de cycles ; par conséquent, au fur et à mesure du traitement du corpus, le nombre de segments restant à analyser décroît progressivement. Le traitement du corpus prend fin quand tous les bisegments sont réécrits sous la forme d'un arbre TransTree.

2.3.1.3 Base de connaissance

2.3.1.3.1 Le texte et les nombres

L'analyse effectuée sur chaque bisegment dépend de l'observation du corpus tout entier. Les informations nécessaires à cette analyse sont collectées sur l'ensemble du corpus et enregistrées dans une « base de connaissance* »[18]. Comme nous l'avons vu, la réécriture du corpus se fait par substitution de nouveaux identifiants. Ceux-ci renvoient aux formes enregistrées dans la base de connaissances.

Nous distinguerons essentiellement deux types de données dans cette base :

- les données textuelles proches du modèle TransTree,
- les données numériques constituant un modèle de langage.

La base de connaissances peut être implémentée sous forme de base de données, au sens informatique. Nous verrons dans la troisième partie que nous l'avons simplement mise en œuvre sous forme de fichiers textes.

[17] Remarquons que les digrammes sont eux-mêmes construits selon l'axe interlingue, à partir des mots typographiques.
[18] Et non pas « base de données » car l'implémentation informatique n'est pas notre propos ici.

2.3.1.3.2 Les données textuelles

Les données textuelles sont équivalentes[19] à des amphigrammes ou à des digrammes, vrais ou faux.

Les amphigrammes enregistrés sont les prototypes de ceux calculés dans le corpus. Nous verrons dans la troisième partie que différentes options sont possibles dans la définition de ce que nous appelons prototype ici, en particulier concernant la conservation ou non du parenthésage liés aux arbres binaires. Pour l'instant, nous nous contenterons de dire qu'une certaine description de ces objets est faite et qu'elle est associée à un identifiant. Les identifiants des amphigrammes sont ceux avec lesquels on effectue la réécriture progressive du corpus.

Les digrammes font également partie des données textuelles, quoiqu'ils n'appartiennent pas directement au modèle TransTree. Les prototypes de vrais digrammes et de faux digrammes reçoivent eux aussi des identifiants permettant la réécriture du corpus.

Les identifiants produits dans la base de connaissances sont également utilisés à l'intérieur même de la base, de manière à décrire l'aspect gigogne des amphigrammes enregistrés.

Enfin, dans le cas où des classes ont été calculées, on consignera aussi des amphigrammes génériques. D'après notre modélisation, les ensembles d'amphigrammes, atomiques et génériques, font partie intégrante de la représentation TransTree.

2.3.1.3.3 Les données numériques

Les identifiants affectés aux données textuelles servent de support aux données numériques. Les nombres d'occurrences de chaque prototype sont consignés ici, ainsi que les suites de deux ou trois identifiants successifs à une étape donnée de réécriture du corpus : ce sont des bigrammes et des trigrammes. Ces termes recouvrent des suites de deux ou trois identifiants, qu'il s'agisse de faux digrammes, de vrais digrammes d'amphigrammes ou de classes de ces objets.

Les bigrammes et les trigrammes sont utilisés d'une part pour calculer les indices de sécabilité et d'autre part, à l'aide de l'algorithme de Viterbi, pour construire un arbre TransTree à partir d'un seul segment, comme nous le verrons au chapitre 2.3.3.2.

Les nombres de prototypes et d'occurrences recensés à chaque étape du cycle de l'analyse font également partie des données numériques que l'on enregistre.

[19] Nous disons « équivalentes » parce que dans nos expérimentations (voir troisième partie), nous n'avons pas utilisé la syntaxe XML pour le stockage de ces données.

2.3.2 Constructions effectives

Trois méthodes de construction d'un arbre TransTree modélisant les correspondances dans un bisegment sont présentées ic. La plus simple ne met en œuvre qu'une une seule passe et ne suppose pas de classes. La deuxième met en jeu plusieurs passe s et la dernière, la plus complète, utilise les classes d'amphigrammes génériques.

2.3.2.1 Construction simple

2.3.2.1.1 De deux arbres binaires à un arbre n-aire

Dans cette première méthode, nous construisons directement l'arbre TransTree d'un bisegment à partir des deux arbres binaires de sécabilité établis à l'aide des digrammes. Nous disposons donc d'une structuration statistiquement motivée pour chacun des deux segments, et nous cherchons à obtenir une unique structure décrivant les deux segments simultanément.

Les nœuds des arbres TransTree sont des amphigrammes complexes. Pour déterminer l'étendue et les frontières de chacun, nous nous fondons sur le principe d'harmonie entre granularité et compositionnalité. Pour cela, nous utilisons les propriétés de congruence et de saturation dans les arbres binaires de sécabilité.

Grâce à la congruence, nous identifions les sous-arbres en correspondance sur le critère des vrais digrammes partagés. Grâce à la saturation, nous rassemblons autour de ces vrais digrammes les faux digrammes que nous ne saurions pas autrement mettre en correspondance. De fait, les faux digrammes dans les sous-arbres congruents forment ce que l'on pourrait appeler une « correspondance collective floue» autour des vrais digrammes. Aucun des faux digramme n'a de contrepartie propre : c'est l'ensemble du sous-arbre saturé dans le premier segment que nous rapprochons d'un sous-arbre saturé dans le deuxième segment.

La propriété de congruence, alliée à la notion de nœud saturé, nous fournit donc une indication sur la détermination de la granularité optimale dans l'analyse du segment qui se trouve ainsi fondée à la fois sur des contraintes unilingues et interlingues.

Pour fixer les idées, détaillons un cas de construction d'amphigrammes. Le diagramme suivant reprend l'exemple utilisé plus haut. Seuls les nœuds saturés ont été représentés, ainsi que les liens de congruence possibles sur ces nœuds.

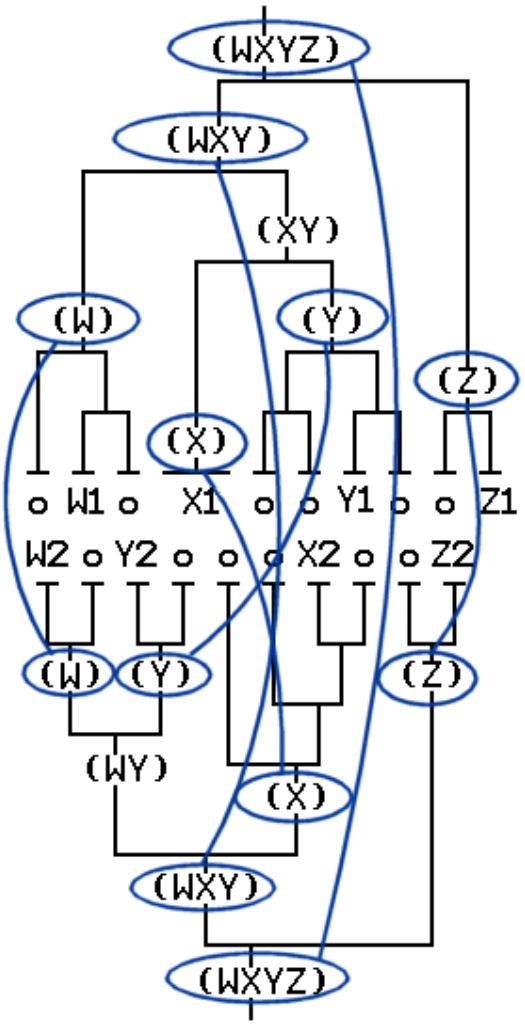

Figure 48 : nœuds saturés dans le même exemple

Les deux nœuds étiquetés **(W)** permettent de rapprocher les deux sous-segments suivants : **«o W1 o»** et **«W2 o»**, et donc de constituer l'amphigramme $\mathcal{A}_W$ = **(«o W o» , «W o»)** où **W=(W1,W2)** est un amphigramme atomique, autrement dit un vrai digramme.

De même, nous avons les trois amphigrammes complexes : $\mathcal{A}_X$ = **(«X» , «o o X o»)**, $\mathcal{A}_Y$ = **(«o o Y o» , «Y o»)** et $\mathcal{A}_Z$ = **(«o Z» , «o Z»)**.

À partir de ces amphigrammes de base, les deux nœuds **(WXY)** nous permettent de construire l'amphigramme $\mathcal{A}_{WXY}$ = **(«**$\mathcal{A}_W$ $\mathcal{A}_X$ $\mathcal{A}_Y$**», «**$\mathcal{A}_W$ $\mathcal{A}_Y$ $\mathcal{A}_X$**»)** et pour finir les nœuds **(WXYZ)** donnent l'amphigramme $\mathcal{A}_{WXYZ}$ = **(«**$\mathcal{A}_{WXY}$ $\mathcal{A}_Z$**», «**$\mathcal{A}_{WXY}$ $\mathcal{A}_Z$**»)**.

Les noeuds **(XY)** qui n'appartienent qu'à l'arbre du haut (langue L1) et **(WY)** sur l'arbre du bas (langue L2) sont saturés mais n'admettent pas de contrepartie congruente. Ils ne donnent pas lieu à un amphigramme et nous les laissons de côté.

2.3.2.1.2 Expression XML et représentation graphique

L'expression XML de l'arbre TransTree correspondant à l'exemple détaillé ci-dessus est :

```
<transtree>
<amphigrams section='doc'>
…
<amphigram loc='A'>
 <text xml:lang='L2'><a loc='A'/> <a loc='B'/></text>
  <amphigram loc='A'>
   <text xml:lang='L1'><a loc='A'/> <a loc='B'/> <a
loc='C'/></text>
   <text xml:lang='L2'><a loc='A'/> <a loc='C'/> <a
loc='B'/></text>
   <amphigram loc='A'>
    <text xml:lang='L1'> o <a loc='A'/> o</text>
    <text xml:lang='L2'><a loc='A'/> o</text>
    <amphigram loc='A'>
     <text xml:lang='L1'>W1</text>
     <text xml:lang='L2'>W2</text>
    </amphigram>
   </amphigram>
   <amphigram loc='B'>
    <text xml:lang='L1'><a loc='A'/></text>
    <text xml:lang='L2'>o o <a loc='A'/> o</text>
    <amphigram loc='A'>
     <text xml:lang='L1'>X1</text>
     <text xml:lang='L2'>X2</text>
    </amphigram>
   </amphigram>
   <amphigram loc='C'>
    <text xml:lang='L1'>o o <a loc='A'/> o</text>
    <text xml:lang='L2'><a loc='A'/> o</text>
    <amphigram loc='A'>
     <text xml:lang='L1'>Y1</text>
     <text xml:lang='L2'>Y2</text>
    </amphigram>
   </amphigram>
  </amphigram>
 <amphigram loc='B'>
  <text xml:lang='L1'>o <a loc='A'/></text>
  <text xml:lang='L2'>o <a loc='A'/></text>
  <amphigram loc='A'>
   <text xml:lang='L1'>Z1</text>
   <text xml:lang='L2'>Z2</text>
  </amphigram>
 </amphigram>
</amphigram>
…
</amphigrams>
</transtree>
```

Figure 49 : expression XML de l'amphigramme

Cette expression XML conduit à la représentation graphique suivante :

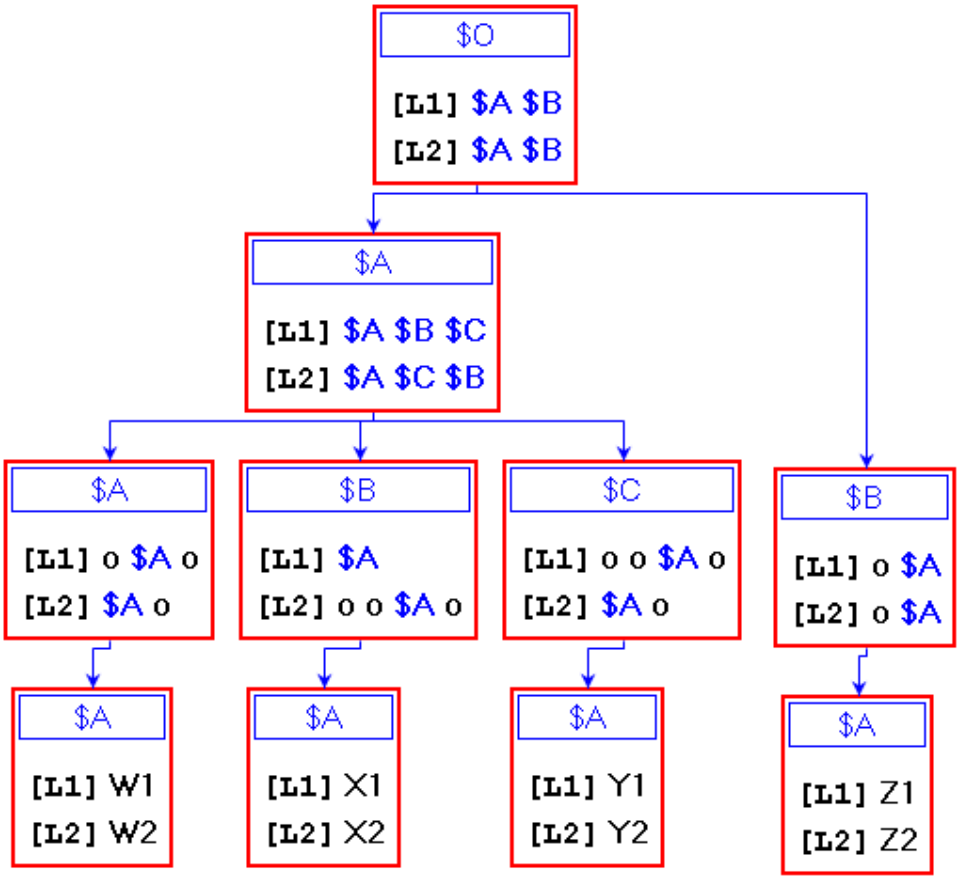

Figure 50 : diagramme VML d'une expression possible du bisegment selon TransTree

Cette figure montre comment les deux arbres binaires de sécabilité ont été finalement concentrés en un seul arbre n-aire.

2.3.2.1.3 Limites de la construction simple : la localité des informations

La méthode de construction de TransTree exposée jusqu'à présent conduit à des représentations du bisegment conformes au formalisme. Cependant, on observe souvent qu'elle mène à des arbres contenant un petit nombre de nœuds. Nous avons donc étudié plusieurs méthodes pour améliorer ce canevas de base.

La sécabilité est un index local qui a été calculé sur une fenêtre étroite. Nous l'utilisons en comparant ses valeurs en différents points du segment. Quand les points choisis sont proches, la comparaison des valeurs est plus pertinente que lorsqu'ils sont éloignés.

De même, dans les arbres binaires, la pertinence de la géométrie est plus grande pour les nœuds proches des feuilles, qui isolent des sous-segments courts, et diminue au fur et à mesure qu'on se rapproche de la racine, où les nœuds déterminent des sous-segments plus longs. La construction simple ne tient pas compte de cela.

2.3.2.2 Construction intermédiaire à plusieurs passes

2.3.2.2.1 Alignement par niveaux

Ce que nous appelons la « construction intermédiaire » prend cet aspect en compte. Cette construction s'opère par couches successives. Dans un premier temps, on établit les amphigrammes complexes les plus petits, correspondant aux nœuds saturés qui ne contiennent qu'un seul vrai digramme. C'est à cette étape qu'une construction comme « mise à jour » s'établira autour du digramme (updated, jour). De cette manière, nous établissons en fait une première coupure dans les arbres binaires de sécabilité.

Ces amphigrammes de premier niveau sont alors considérés comme les éléments terminaux du segment réécrit. Les faux digrammes de chaque langue qui ne sont pas entrés dans un amphigramme sont laissés tels quels. Tous les segments du corpus sont réécrits de cette façon. Un nouveau jeu de bigrammes et de trigrammes est alors collecté, et les indices de sécabilité établis avec ces nouveaux objets élémentaires sont calculés.

Pour calculer le niveau supérieur, on considère les amphigrammes correspondant aux nœuds saturés congruents contenant au moins deux amphigrammes du niveau inférieur, autrement dit au moins deux vrais digrammes. Ces amphigrammes contiennent, éventuellement, plus de deux vrais digrammes en fonction de la géométrie des arbres binaires et des règles de congruence.

C'est à partir de ce niveau que se rencontrent des structures plus complexes et qu'on peut enregistrer en particulier les ponts dans l'ordre linéaire de chaque langue. Par exemple, l'inversion des digrammes (dialog, dialogue) et (box, boîte) dans les sous-segments « dialog box » et « boîte de dialogue », sera prise en compte.

Les niveaux supérieurs de l'arbre TransTree sont établis de la même manière, en cherchant à chaque fois les nœuds congruents saturés les plus bas dans les arbres binaires, contenant au moins deux amphigrammes du niveau inférieur. Autrement dit, on reconstruit des arbres binaires à chaque passe, on établit une nouvelle coupure et l'on ne conserve que la partie la plus proche des feuilles sous forme d'amphigrammes.

2.3.2.2.2 Amphigrammes, bigrammes et trigrammes

Le calcul des index de sécabilité est toujours fondé sur l'information mutuelle entre deux éléments successifs, établis sur une fenêtre glissante de longueur réduite.

Cette deuxième méthode nous amène donc à rassembler des bigrammes et des trigrammes constitués d'amphigrammes du niveau inférieur, et ce pour chaque niveau et dans les deux langues. Il peut également y avoir des faux digrammes non inclus dans des amphigrammes, et ce à n'importe quel niveau. En pratique, ils sont inclus dans un amphigramme dès la deuxième passe et très exceptionnellement à la troisième, comme nous le verrons dans la troisième partie.

L'avantage de cette deuxième méthode de construction réside donc dans le fait de calculer des indices de sécabilité pertinents des feuilles des la racine. En effet, ceux-ci sont calculés non plus entre des digrammes, équivalents par la taille aux mots typographiques, mais entre des amphigrammes, qui recouvrent des briques traductionnelles entières. Autrement dit, au fur et à mesure que l'on ajoute des niveaux supérieurs, on modélise des structures de plus en plus élaborées.

2.3.2.2.3 Limite de la construction intermédiaire : la rareté des données

Notre construction souffre encore du problème de la rareté des données. Les amphigrammes sont des structures complexes dont les occurrences diminuent dans le corpus au fur et à mesure qu'on avance dans la construction de l'arbre TransTree. Les indices de sécabilité que l'on calcule sont donc de plus en plus pertinents mais de moins en moins précis.

Un premier moyen de lutter contre ce phénomène de rareté des données consiste à ne pas tenir compte de la structure interne des amphigrammes et à ne les distinguer qu'en tant que briques traductionnelle "à plat". Autrement dit, deux amphigrammes sont considérés comme identiques (sont des occurrences d'un même prototype) s'ils représentent les mêmes chaînes typographiques dans chaque langue. On constitue ainsi des bigrammes et des trigrammes plus représentatifs.

Bien entendu, lors de la réécriture du segment dans son ensemble sous forme d'un arbre TransTree, on utilise les amphigrammes d'origine, munis de leurs structure interne, afin de rétablir la construction gigogne.

Cette méthode a l'avantage de conserver une certaine précision des calculs pour les premiers niveaux, même si elle perd un peu en pertinence, puisqu'on ne tient plus compte de la spécificité structurelle des amphigrammes.

Cependant, même en normalisant les amphigrammes en briques traductionnelles à plat, les indices de sécabilité sont calculés sur des occurrences de moins en moins nombreuses lorsque l'on s'éloigne des feuilles. Ce phénomène est encore plus sensible pour les segments les plus longs. Dans ce cas, lors du calcul des niveaux les plus élevés, les occurrences sont de plus en plus rares, voire uniques. Les situations d'égalité entre indices de sécabilité en deux points du segment sont de plus en plus fréquentes et la pertinence de la géométrie de l'arbre binaire obtenue est de plus en plus faible.

2.3.2.3 Construction générale

2.3.2.3.1 Utilisation des amphigrammes génériques

La construction générale que nous présentons ici reprend l'ensemble des étapes du cycle présenté en 2.3.1.2.2. Nous allons pour y parvenir utiliser les amphigrammes génériques et les ensembles d'amphigrammes génériques. Cette méthode nous permet de remédier véritablement à la rareté des données liées à la taille croissante des amphigrammes dans la construction des arbres binaires de sécabilité.

En effet, les amphigrammes génériques, tels que nous les avons prévus dans le formalisme TransTree, représentent plusieurs amphigrammes, génériques ou atomiques. Leur représentativité est donc bien plus importante dans le corpus.

Avant la première construction d'arbres binaires de sécabilité, les classes seront établies au niveau des digrammes (vrais digrammes et faux digrammes). Ces classes regroupent un nombre variable de digrammes. Il se peut même qu'un unique digramme soit vu comme une classe.

Les arbres binaires de sécabilité sont alors calculés sur les suites de classes. Les sous-arbres congruents saturés ne contenant qu'un seul vrai digramme sont mis en correspondance, constituant ainsi les amphigrammes complexes de premier niveau.

Les amphigrammes complexes identiques par leur armature textuelle et contenant des vrais digrammes appartenant à la même classe sont généralisés en amphigrammes génériques.

De ce point de vue, les amphigrammes génériques sont des prototypes d'amphigrammes complexe. En tant qu'amphigrammes génériques, ils ne contiennent pas de vrais digrammes comme éléments fils, mais pointent vers des classes de vrais digrammes.

Le segment analysé à ce niveau est donc à présent constitué d'une suite d'amphigrammes génériques et éventuellement des faux digrammes résiduels.

2.3.2.3.2 Alignement par niveaux

La méthode d'alignement par niveaux décrite dans la présentation de la construction intermédiaire est globalement identique. La différence essentielle tient dans le fait qu'avant le calcul des index de sécabilité, il est nécessaire de constituer des classes avec les éléments du niveau courant.

L'ensemble du corpus ayant été réécrit à un niveau donné, il est possible d'opérer une nouvelle classification à ce niveau. Les amphigrammes génériques sont regroupés en classes en fonction du contexte dans lequel ils apparaissent. Les indices de sécabilité au niveau supérieur sont alors calculés sur ces segments constitués de classes bilingues d'amphigrammes génériques et monolingues de faux digrammes.

Cette méthode peut être réitérée aux niveaux supérieurs de l'analyse. Comme précédemment, l'analyse pour un segment donné s'arrête lorsque l'ensemble du bisegment constitue un unique amphigramme. Celui-ci peut être généralisé en amphigramme générique, représentant un certain type de bisegment. Il appartient à la classe des amphigrammes génériques indépendants, c'est-à-dire couvrant l'ensemble d'un bisegment.

2.3.2.3.3 Variations sur la construction générale

La méthode générale ouvre la voie à plusieurs variations.

Par exemple, Il est possible de ne pas chercher à classer les amphigrammes génériques au-delà d'un certain niveau. Cette éventualité n'affecte pas la nature de l'arbre TransTree. Cette construction reste une hiérarchisation des correspondances gigognes d'un bisegment. Les amphigrammes génériques ne participent à cette construction que pour établir des index de sécabilité.

En outre, les classes de faux digrammes étant monolingues, les regroupements effectués ne sont pas aussi solides que ceux que l'on trouve pour les vrais digrammes, étayés par les contextes trouvés dans les deux langues. Il est donc possible de ne pas chercher à les classer avant d'effectuer les calculs de sécabilité. La encore, il s'agit d'une variation n'affectant pas la démarche générale, ni surtout la nature de l'arbre TransTree.

Ces variations produisent donc toutes des arbres TransTree modélisant les correspondances sous-segmentales gigognes dans le bisegment. Si la nature d'arbre n-aire bilingue ne change pas, la géométrie de l'arbre est en revanche différente selon les options particulières que l'on aura choisi d'appliquer.

2.3.3 Application à des bisegments et segments inconnus

Dans cette section, nous décrivons diverses utilisations de la base de connaissances acquise à l'occasion de l'analyse, selon le formalisme TransTree, d'un corpus bilingue aligné.

2.3.3.1 Analyse d'un bisegment hors corpus

2.3.3.1.1 Motivation

La construction que nous avons proposée jusqu'à présent permet de produire une certaine analyse d'un corpus bilingue. Cette analyse effectuée, nous avons consigné un nouveau savoir dans la base de connaissances, notamment des prototypes d'amphigrammes complexes et, éventuellement, selon le type d'analyse, des ensembles d'amphigrammes et des amphigrammes génériques.

Cette analyse constitue en soi-même une application utile de notre méthode. Dans un environnement d'aide à la traduction, elle permet d'expliciter pour le traducteur en quoi les coïncidences floues qui lui sont proposées sont pertinentes. D'autres applications sont possibles, comme l'aide à l'apprenant d'une langue étrangère etc. Nous avons proposé en 2.1.2 une méthode de visualisation dynamique interactive des correspondances gigognes applicable à ces situations.

Dans ce chapitre, nous allons brièvement exposer la situation où le bisegment n'appartient pas au corpus d'apprentissage. Nous distinguerons deux cas, selon que l'apprentissage met ou non en œuvre les ensembles d'amphigrammes.

2.3.3.1.2 Sans ensemble d'amphigrammes

Dans le cas où la base de connaissances ne contient pas d'ensemble d'amphigrammes, il n'est possible d'envisager qu'une analyse simple, comme nous l'avons exposée en 2.3.2.1.

En effet, même si les amphigrammes de niveaux bas contenus dans le segment à analyser sont tous connus, c'est-à-dire si tous les syntagmes relativement simples apparaissent dans le corpus, ce qui, sans être certain, est possible, il existe nécessairement un niveau à partir duquel leur arrangement n'apparaît pas dans le corpus. En effet, si tous les amphigrammes étaient présents dans le corpus, alors l'amphigramme racine, c'est-à-dire le bisegment lui-même appartiendrait au corpus, ce qui serait contradictoire. Nous ne pouvons donc pas envisager une analyse intermédiaire, par niveau dans ce cas.

En revanche, selon la taille de notre base de connaissances, il est possible d'établir des digrammes avec les données à notre disposition, voire d'en constituer de nouveaux en appariant d'éventuels hapax. Nous pouvons en outre utiliser les bigrammes et trigrammes recensés dans le corpus et construire les arbres binaires de sécabilité sur chaque segment. La construction simple de TransTree en découle. Si la richesse de la base de connaissances le permet, il est possible de commencer une construction intermédiaire et de la compléter par une construction simple au-delà d'un certaine niveau.

2.3.3.1.3 Avec ensemble d'amphigrammes

L'hypothèse où l'on aurait établi des ensembles d'amphigrammes est sensiblement différente de la précédente. En effet, les données manquantes peuvent être remplacées, dans

l'analyse, par l'ensemble d'amphigrammes le plus probable, au vu du contexte, dans chaque langue, ce qui permet de mener l'analyse à son terme.

Cependant, comme dans le cas précédent, on est continellement amené à associer des armatures textuelles venant des deux segments ce qui entraîne une grande variabilité des structures observées. On ne peut pas garantir qu'on y retrouvera les amphigrammes génériques consignés dans la base de connaissances. Il est donc possible, ici aussi, que la base de connaissances soit défaillante ce qui obligerait à terminer l'analyse par une construction simple.

2.3.3.2 Vers un système de traduction automatique

2.3.3.2.1 Algorithme général

Précisons tout de suite qu'il n'est pas dans notre idée de présenter ici un système complet de traduction automatique. Nous avons seulement l'ambition d'aider le traducteur humain dans sa tâche en lui présentant un segment cible fictif, constitué le plus densément possible de sous-segments réutilisables.

Nous voulons nous appuyer pour ce faire sur l'analyse dans le formalisme TransTree de nombreuses mémoires de traductions et réutiliser le matériau ainsi recueilli. Paradoxalement, la construction que nous allons exposer semble moins difficile que l'analyse générale d'un bisegment complet exposée en 2.3.3.1. En effet, comme nous synthétisons entièrement le segment cible, nous sommes moins contraints par la prise en compte simultanée des deux langues.

Pour réutiliser ce matériau dans des situations nouvelles, non rencontrées dans le corpus d'apprentissage, une certaine forme de généralisation des données observées sera nécessaire. C'est pourquoi nous avons recours aux amphigrammes génériques. Nous supposons en outre que les ensembles d'amphigrammes génériques ont été mis en œuvre avec des classes dures.

Le diagramme suivant présente l'ensemble de l'algorithme de traduction proposé. On y distingue une phase d'analyse, à gauche, suivie d'une phase de synthèse, à droite. Chacune de ces phases met en œuvre une boucle algorithmique.

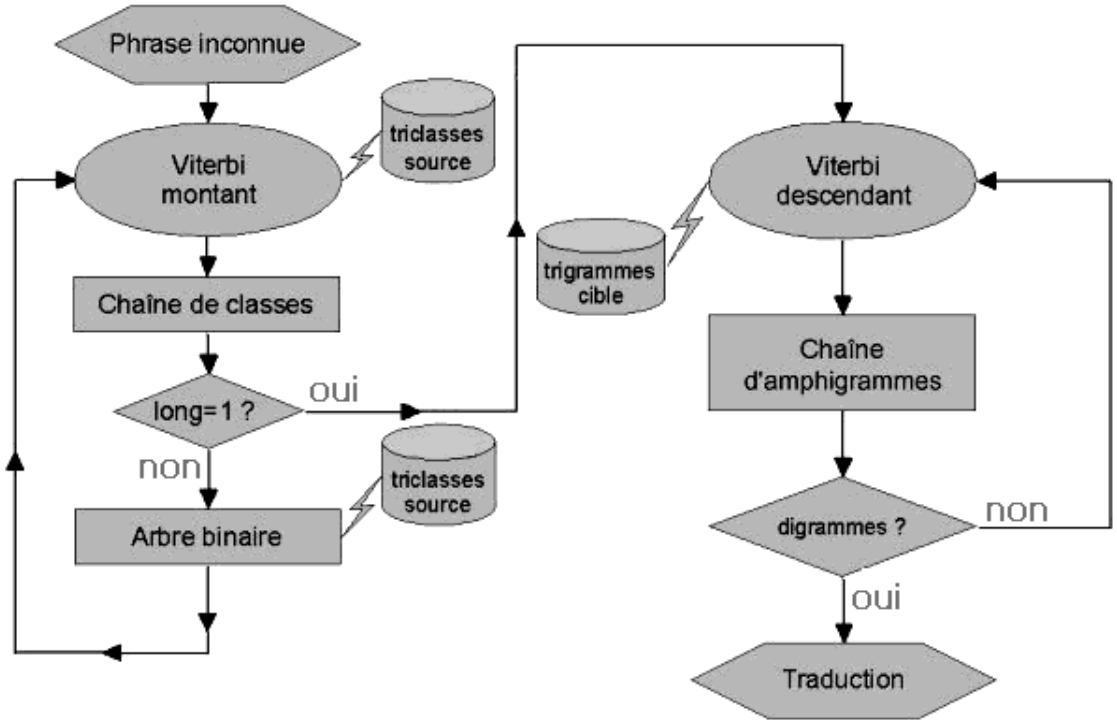

Figure 51 : algorithme général de traduction

Sur ce schéma, les deux hexagones horizontaux correspondent à l'entrée et à la sortie de l'algorithme. Les deux ellipses correspondent à une réécriture du segment à l'aide de l'algorithme de Viterbi. Nous avons représenté les résultats intermédiaires sous forme de rectangles. Les cylindres symbolisent la base de connaissances, reliée par un éclair au bloc où elle est utilisée. Enfin, les losanges indiquent les branchements conditionnels.

Les flèches montrent le parcours de l'algorithme Le principe fondamental consiste à recréer un arbre TransTree compatible avec la phrase source et n'utilisant que les amphigrammes génériques rencontrés dans le corpus. De cette manière, nous pouvons traiter, par extrapolation, des arrangements de briques traductionnelles non réalisés dans le corpus. Si l'on voit TransTree comme une grammaire bilingue, on peut considérer que l'on cherche un arbre de dérivation de la phrase source. Ce même arbre de dérivation décrit alors la phrase cible.

Les deux sections qui suivent détaillent ce schéma dans l'ordre de l'exécution de l'algorithme.

2.3.3.2.2 Analyse d'un segment inconnu

Notre idée principale est de construire un arbre binaire de sécabilité à partir de la phrase source afin d'y retrouver des sous-arbres connus de notre base de connaissances. Ces sous-arbres binaires sont alors réécrits sous forme d'amphigrammes génériques.

Les mots typographiques de la phrase inconnue sont d'abord étiquetés avec des classes de digrammes, grâce à l'algorithme de Viterbi utilisé avec un modèle de langage triclasse. Ce modèle de langage a été produit par la partie source du corpus et conservé dans la base de connaissances. Cet algorithme est utilisé pour « monter » conceptuellement des instances à

leur classes, qui regroupent plusieurs instances. C'est pourquoi nous avons écrit « Viterbi montant » dans ce schéma. La sortie de l'algorithme de Viterbi est donc une chaîne de classes.

Les classes de digrammes comprennent des classes de vrais digrammes et des classes de faux digrammes. On fait donc un choix dès cette étape, qui consiste à déterminer les mots typographiques qui donnent lieu à de vrais digrammes. Les autres constitueront l'armature textuelle des amphigrammes que nous allons déterminer ensuite.

Dans le cas où un mot typographique n'est pas connu, on lui affecte la classe bilingue la plus probable en fonction du contexte afin de continuer l'analyse. Nous verrons en 2.3.3.2.3, les stratégies de traduction au moment de la synthèse.

En général, le traitement ne s'arrête pas ici, comme pourrait le laisser supposer le schéma. Cependant, c'est à cette étape, après plusieurs boucles, que le passage à la synthèse peut éventuellement se faire si le segment est réduit à une seule classe.

Si la chaîne est constituée de plusieurs classes, on établit l'arbre binaire de sécabilité. Le calcul des coefficients de sécabilité utilise les mêmes triclasses que celles qui constituent le modèle de langage mentionné ci-dessus.

La propriété de saturation de certains nœuds (voir 2.2.2.3.1) nous est de nouveau accessible puisque nous distinguons les classes bilingues et les classe monolingues. En revanche, nous ne pouvons pas exploiter la propriété de congruence, puisque nous ne disposons que d'un seul arbre binaire.

Afin de choisir, parmi les nœuds saturés, ceux qui correspondent à des amphigrammes génériques connus de la base de connaissances nous effectuons un deuxième passage dans la boucle d'analyse et avons de nouveau recours à l'algorithme de Viterbi montant.

Lors de ce deuxième passage, l'algorithme de Viterbi utilisé est plus compliqué que l'algorithme habituel, à plat. En effet, il faut ici rechercher une coupure dans l'arbre binaire, c'est-à-dire délimiter les sous-arbres binaires à réécrire en amphigrammes génériques. On utilise donc un modèle triclasse d'amphigrammes génériques de premier niveau. Les sous-arbres sélectionnés correspondent à l'armature textuelle source des amphigrammes génériques connus de la base de connaissances. À l'issue de cette phase, on dispose donc de nouveau d'une suite de classes d'amphigrammes.

On réitère donc la phase d'analyse à chaque boucle avec des classes d'amphigrammes représentant des sous-segments de plus en plus longs. L'analyse s'arrête lorsque l'ensemble de la phrase est réduit à une seule classe d'amphigrammes génériques, contenue dans la base de connaissances et choisie par l'algorithme de Viterbi.

À l'issue de cette phase d'analyse, on a donc obtenu un arbre de classes d'amphigrammes génériques. Les feuilles de cet arbre sont des classes d'amphigrammes atomiques. Chaque classe contient plusieurs amphigrammes. Tous ne sont pas admissibles pour un nœud donné : ne sont admissibles que ceux dont l'armature textuelle côté source correspond au segment source de ce nœud.

Cette description du segment inconnu ayant été faite, il nous reste à synthétiser une proposition de segment cible.

2.3.3.2.3 Synthèse d'un segment cible

Le but de la synthèse est de réécrire l'arbre de classes en un arbre TransTree, en sélectionnant au sein des classes les meilleurs amphigrammes. Le choix est constamment guidé par l'armature textuelle du segment source. En effet, parmi les amphigrammes

regroupés au sein d'une classe, seuls certains d'entre eux correspondent effectivement aux contraintes données par le segment source.

À la racine de l'arbre, l'amphigramme générique sélectionné fait référence, dans ses points d'insertions, à des classes d'amphigrammes génériques qui sont à leur tour réécrits. Il s'agit d'un processus « descendant » ce qui justifie l'étiquette « Viterbi descendant » sur le schéma.

Selon la nature des classes, il est ou il n'est pas nécessaire d'utiliser l'algorithme de Viterbi. En général, il faut choisir parmi les seuls amphigrammes génériques (ou amphigrammes atomiques, niveau le plus bas) de la classe à instancier dont le texte source correspond au segment à traduire.

On peut avoir établi des classes de telle sorte que les différents amphigrammes génériques correspondant à un même texte source appartiennent à des classes différentes. Dans ce cas, un texte source et une classe suffisent à déterminer un amphigramme générique particulier et l'on n'a pas besoin d'utiliser l'algorithme de Viterbi.

La synthèse se poursuit de manière récursive jusqu'à ce que les amphigrammes obtenus soient atomiques : ils ne contiennent plus d'élément variable. Les armatures textuelles cible des amphigrammes atomiques contiennent éventuellement des mots typographiques qui sont bien entendu conservés lors de cette synthèse. La traduction obtenue est donc constituée d'une chaîne de mots typographiques provenant de l'armature textuelle cible des amphigrammes génériques et atomiques.

Dans le cas où la phrase de départ contient un mot inconnu, ou plus généralement, si l'on ne trouve pas de chemin constitué d'items connus de la base de connaissances, il est néanmoins possible d'effectuer l'analyse avec la classe la plus vraisemblable à cette position comme nous l'avons indiqué lors de l'analyse. Pour établir une traduction, il nous faut fabriquer un amphigramme de substitution. Deux cas se présentent alors :

- La classe utilisée est une classe de faux digrammes, monolingue par essence, nécessaire au calcul des indices de sécabilité,
- La classe utilisée est une classe bilingue.

Dans le premier cas, il n'y a pas de calcul supplémentaire à mener. La synthèse ignore les éléments non traduits.

Dans le deuxième cas, bien sûr, la situation est plus complexe. On pourra tenter de mettre en quadrilatère analogique trois éléments connus de cette même classe afin résoudre le système analogique. Par cette méthode, on peut fabriquer une chaîne de substitution.

Si cela n'est pas possible, on se résoudra à traduire a minima, c'est-à-dire à proposer le mot source lui-même[20] ou, plus généralement, une traduction par juxtaposition, dans l'ordre du segment source, des éléments bilingues.

2.3.3.3 Autres possibilités et conclusions

2.3.3.3.1 Analyse d'un bisegment en vue de sa traduction dans une troisième langue

L'étude qui précède ouvre peut-être vers un nouveau paradigme de traduction automatique dans le cas encore peu traité où l'on dispose d'un corpus trilingue aligné et où l'on veut traduire un bisegment dans la troisième langue. Par exemple, on disposerait d'un corpus

[20] Avec une marque d'avertissement adéquate.

français-anglais-allemand et on voudrait traduire en allemand une phrase anglaise déjà traduite en français.

L'idée est est d'établir une analyse selon le formalisme TransTree dans les trois couples de langues, ou même dans les trois langues en même temps comme nous l'avons présenté en 2.1.3.3.1.

Nous utilisons alors l'information sémantique contenue dans le bitexte source, beaucoup plus riche que celle apportée par chaque langue source. En effet, un grand nombre d'ambiguïtés langagières propres à une seule langue disparaissent par confrontation avec une deuxième langue. L'apport d'information est dû à l'expertise du traducteur qui connaît le monde réel et résout, sans même y penser le plus souvent, bien des ambiguïtés insolubles par l'analyse automatique.

Cependant, il est probable que la connaissance d'un texte exprimé dans deux langues différentes apporte aussi un certain bruit statistique.

Autrement dit, il serait intéressant de mesurer le gain de désambiguïsation en fonction des couples de langues et de le comparer à l'augmentation éventuelle du bruit. Nous pensons que notre formalisme peut contribuer à cette étude et à l'utilisation effective de deux versions linguistiques d'un même énoncé pour le traduire vers une troisième langue.

2.3.3.3.2 Mesure de la réutilisabilité des MT

Si l'on considère une mémoire de traduction comme un matériau à rentabiliser, il est légitime de chercher à quantifier le rendement que l'on en retire, par comparaison avec un rendement idéal propre à cette mémoire.

Nous nous intéressons donc à la réutilisabilité d'une mémoire de traduction. Il existe plusieurs moyens d'utiliser cette mémoire, en proposant au traducteur :

- des coïncidences floues brutes,
- des coïncidences floues alignées selon le modèle TransTree, comme nous l'avons présenté en 2.1.2.3.2,
- des traductions calculées à partir du segment source, comme nous l'avons présenté ci-dessus, en 2.3.3.2.

Pour chacune de ces possibilités, nous considérons deux indices : le gain de temps pour le traducteur, et la proportion de la mémoire utilisée pour traduire le même texte.

Si **T** est le temps nécessaire au traducteur pour traduire le texte sans l'aide de la mémoire et si **t** est le temps qu'il met avec l'aide de la mémoire, le gain de temps **g** est donné par la formule suivante :

$$g = \frac{T - t}{T}$$

Par ailleurs, si **N** est le nombre total de segments dans la mémoire, **n** le nombre de segments effectivement utilisés, et **p** la proportion de la mémoire utilisée, en pourcentage, on a :

$$p = \frac{N - n}{N}$$

Le nombre de segments effectivement utilisés dans le cas de la troisième méthode est établi de la manière suivante : on ne retient que les segments contenant les digrammes utilisés dans les propositions de traduction.

Les différentes méthodes de traduction donnent donc lieu à différents couples (g, p).

- (0, 0) si l'on n'utilise pas du tout la mémoire,
- (g1, p1) si on utilise la première méthode,
- (g2, p2) avec la seconde méthode,
- (g3, p3) avec la troisième.

Cela est illustré par le graphique suivant :

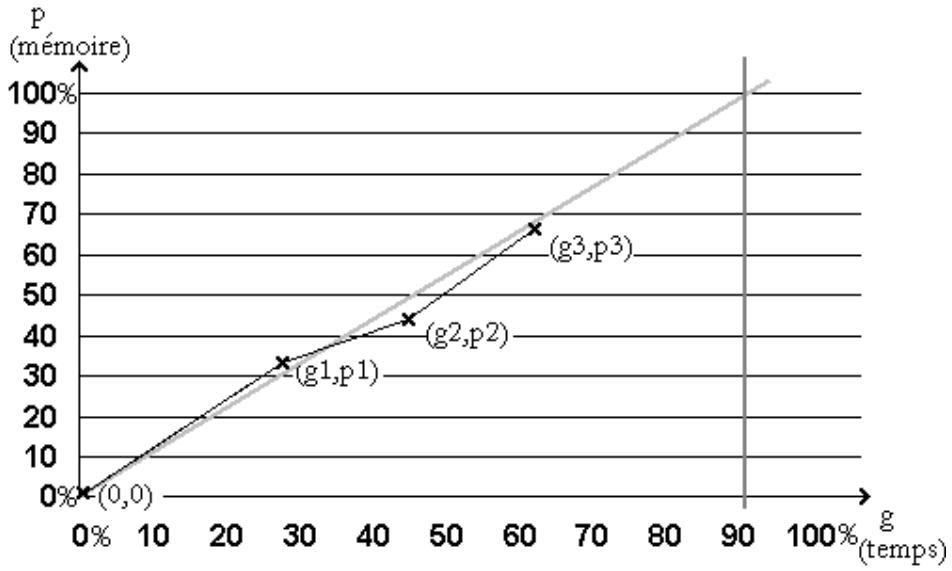

Figure 52 : estimation de la réutilisabilité d'une mémoire pour un document

Les points représentés sur cette courbe ont été placés arbitrairement. Nous ne savons pas quel type de courbe est le plus à même d'interpoler les points obtenus. Ces points dépendent étroitement du couple [document à traduire / mémoire de traduction]. Sur ce graphique, nous avons interpolé linéairement, et nous arrivons à la conclusion que la réutilisabilité de cette mémoire pour ce document avoisine 90% : si nous utilisons toute la mémoire au mieux, le gain de temps pour le traducteur ne peut pas excéder ce taux.

D'autres couples [document à traduire / mémoire de traduction] pourraient conduire à des conclusions différentes : par exemple, si tous les segments du document sont dans la mémoire, le gain est de 100% avec la seule première méthode.

2.3.3.3.3 Conclusions

Le modèle TransTree permet non seulement de modéliser finement les correspondances enchâssées dans les bisegments d'une mémoire de traduction ou un corpus bilingue aligné en général, mais encore d'extrapoler ces correspondances pour fournir ce que l'on peut appeler une grammaire de traduction.

Une telle grammaire de traduction ouvre la voie à différentes applications, de la structuration des correspondances enchâssées à l'intérieur d'un bisegment nouveau à la production de proposition de traduction dans un système de traduction humaine assistée par ordinateur.

Ces applications potentielles n'ont pas été complètement mises en œuvre dans le cadre de cette thèse. Nous avons néanmoins fait quelques expérimentations que nous présentons dans la troisième partie.

3 Réalisations et expérimentations

La mise en œuvre des solutions proposées dans la partie précédente fait l'objet de la troisième partie de cette thèse. Nous exposerons d'abord les conditions dans lesquelles ces expérimentations ont eu lieu. Puis nous détaillerons les travaux préparatoires effectués sur les mémoires de traduction avec lesquelles nous avons travaillé, ainsi que la mise en œuvre de l'analyse du corpus et l'acquisition de données. Enfin, nous exposerons quelques résultats relatifs aux applications présentées en fin de partie II.

3.1 Matériel et matériau

Tout d'abord, il convient de présenter les moyens et l' environnement au sens large dans lesquels les travaux qui suivent ont été menés. Nous évoquerons aussi le corpus utilisé ainsi que le filtrage effectué sur celui-ci.

3.1.1 Moyens informatiques

3.1.1.1 Plates-formes

Les développements effectués ont été successivement entrepris sur Windows 95, Windows NT, Windows XP, équipant différents modèles de la gamme ThinkPad d'IBM.

Le dernier ThinkPad utilisé est un T40, avec un processeur Intel de 1,5 GHz et une mémoire RAM de 1 Go. Le disque dur fait 40 Go.

3.1.1.2 Langage de développement

Après plusieurs tâtonnements, le choix du principal langage de développement s'est porté sur Perl. Ce choix est motivé en premier lieu par la richesse des outils de traitement de chaînes offerte par ce langage. La souplesse de développement alliée à la rapidité d'exécution a confirmé l'auteur dans son choix.

Parmi les « tâtonnements » en question, Java a été longuement envisagé. Mais la modélisation objet préalable, nécessaire dans tout développement en Java, s'accommodait mal de la constante évolution des idées et principes décrits dans la présente thèse.

XML a d'abord été considéré comme un langage de spécification simple et efficace. La possibilité de transformer l'expression XML de TransTree vers des représentations diverses, y compris graphiques, sans « rompre la chaîne XML » nous a rapidement convaincu de l'adéquation de ce standard à notre objectif.

Nous avons donc également utilisé des langages du monde XML, en particulier XHTML, XSL, SVG et VML. L'utilisation de VML mérite une attention spéciale : parce qu'il s'intègre finement dans le langage XHTML, VML permet, entre autres, de bénéficier de la disposition dynamique des tableaux XHTML pour mettre en place simplement différents types de graphiques, dont, notamment, des arbres. Malheureusement, les travaux de standardisation du

langage VML n'ont pas été poursuivis et seules les plate-formes Microsoft utilisent ce langage.

Javascript a également été utilisé dans le développement la visualisation dynamique des correspondances gigognes.

3.1.1.3 Stockage

Le stockage des données, qu'il s'agisse des mémoires de traduction originelles ou des données statistiques dérivées, a été effectué sans l'aide de système de gestion de base de données. La solution mise en œuvre, plus simple et plus souple, est un stockage en fichiers.

En fonction de l'usage qui en est fait, cette base de connaissances est enregistrée de deux manières différentes :

- Soit sous forme d'un seul fichier non indexé, destiné à être intégralement lu en mémoire vive au début de l'exécution d'un programme, lors de certains traitement de masse. C'est en particulier le cas pour les phases d'analyse du corpus et d'extraction de connaissances.
- Soit sous forme de fichiers multiples dont les entrées sont ordonnées selon le premier champ, autorisant une recherche par tri dichotomique. Cette méthode est utile pour les calculs en temps réels.

La recherche par tri dichotomique est implémentée par un programme perl donné en annexe.

Depuis ces expérimentations, Fabien Cromières a montré dans sont projet de deuxième année de master que les tableaux de suffixes étaient particulièrement adaptés au stockage des n-grammes.

Précisons enfin qu'un essai d'utilisation de Postgres avec le langage Java sur serveur s'est révélé trop peu souple dans le contexte d'évolution permanente des développements.

3.1.2 Corpus de départ

3.1.2.1 Description

Les corpus utilisés sont des mémoires de traduction d'IBM, dans leur format d'exportation, constituées par IBM en France. Ces mémoires sont en général utilisées lors de la traduction des produits IBM ou de leur documentation de l'anglais vers le français. Il s'agit donc de mémoires de traduction bilingues anglais/français.

Nous n'avons pas utilisé les mémoires de traduction des produits eux-mêmes, c'est-à-dire des logiciels. En effet, les segments sont le plus souvent constitués de termes isolés, employés dans des menus, des boutons, et des titres de fenêtres ou de figures. Quand ils sont plus complexes, ces segments contiennent très souvent des variables et des signes conventionnels connus des développeurs, mais souvent inintelligibles pour le non-initié. En outre, les conventions utilisées diffèrent entre deux produits et dépendent du laboratoire où ils ont été créés ou même, quelquefois, de l'équipe de développement, voire du développeur lui-même. D'une manière générale, les segments que l'on trouve dans les mémoires de traduction des produits sont très peu représentatifs des mécanismes langagiers que nous cherchons à modéliser.

En revanche, la documentation d'un produit est rédigée. Les segments de ces mémoires sont le plus souvent des phrases complexes. Le nombre de variables ou d'éléments

substituables y est beaucoup plus restreint. Ce matériau se prête donc mieux à l'étude de la traduction langagière.

Les mémoires de traduction que nous avons utilisées comme corpus sont donc exclusivement liées à la traduction de la documentation, jamais des produits eux-mêmes.

3.1.2.2 Chiffres

Le corpus que nous avons utilisé tient sur un cédérom. Il n'est pas compressé.
Voici quelques indices quantitatifs :

1. Taille des données : 565 Mo
2. Nombre de mémoires : 453
3. Nombre de bisegments : 1 785 684

Cependant, ces chiffres doivent être largement revus à la baisse, compte tenu de la qualité langagière du matériau en question. Un filtrage très important a dû être opéré, qui n'a finalement permis de récupérer qu'environ 3,6% du volume total. Le filtrage effectué est présenté en détail plus bas.

3.1.2.3 Exemple

Nous donnons ici un extrait d'une mémoire de traduction avant le filtrage.

```
<Segment>0000000091
<Control>
000160□□000000001045744131□English(U.S.)□FRENCH(NATIONAL)□□EQFANSI□README¶TX.000□Readme.txt
</Control>
<Source>bos.dlc.token @ 4.3.3.11        bos.dlc.ether @ 4.3.3.11        bos.dlc.8023  @ 4.3.3.11
   bos.dlc.fddi  @ 4.3.3.11        bos.dlc.sdlc  @ 4.3.3.11        bos.dlc.qllc  @ 4.3.3.1
sx25.comio    @ 1.1.5.6</Source>
<Target>bos.dlc.token @ 4.3.3.11        bos.dlc.ether @ 4.3.3.11        bos.dlc.8023  @ 4.3.3.11
   bos.dlc.fddi  @ 4.3.3.11        bos.dlc.sdlc  @ 4.3.3.11        bos.dlc.qllc  @ 4.3.3.1
sx25.comio    @ 1.1.5.6</Target>
</Segment>
<Segment>0000000092
<Control>
000162□□000000001045744131□English(U.S.)□FRENCH(NATIONAL)□□EQFANSI□README¶TX.000□Readme.txt
</Control>
<Source>B4)     REQUIRED GSKIT LEVEL FOR CS/AIX V6.1 SSL FUNCTIONS</Source>
<Target>B4)     NIVEAU GSKIT REQUIS POUR LES FONCTIONS SSL CS/AIX V6</Target>
</Segment>
<Segment>0000000093
<Control>
000164□□000000001045744131□English(U.S.)□FRENCH(NATIONAL)□□EQFANSI□README¶TX.000□Readme.txt
</Control>
<Source>The CS/AIX v6 SSL functions require gskkm.rte at the 5.0.4.29 level        or later.
</Source>
<Target>Les fonctions CS/AIX v6 SSL requièrent le niveau 5.0.4.29 ou suivant de gskkm.rte.</Target>
</Segment>
<Segment>0000000094
<Control>
000165□□000000001045744131□English(U.S.)□FRENCH(NATIONAL)□□EQFANSI□README¶TX.000□Readme.txt
</Control>
<Source>The installp image for gskkm.rte at this level is shipped        on the CS/AIX v6.1 install
media (CD or tape). </Source>
<Target>        L'image installp pour ce niveau de gskkm.rte est livrée sur le support
d'installation de CS/AIX v6.1 (CD-ROM ou bande). </Target>
</Segment>
```

Figure 53 : Quatre segments dans une mémoire de traduction

Ces quatre segments sont extraits d'un fichier Readme, tels qu'on les trouve dans une mémoire, avant tout traitement. On voit que la mémoire suit un balisage XML, même si le fichier ne fait pas état d'une référence à une DTD ou à un schéma. Cependant, il peut arriver que des balises XML fassent partie des segments enregistrés (pas sur cette figure), ce qui interdit toute analyse XML de la mémoire sans préparation.

Le texte compris dans la balise **<Control>** est structuré avec des séparateurs implicites, non XML et non UNICODE. Seule l'application, Translation Manager en l'occurrence, peut exploiter cette balise. Les données qu'elle contient font référence à l'organisation du projet et au fichier auquel appartient le segment. Les valeurs numériques indiquent les dates de traduction ou de révision.

3.1.3 Préparation du corpus

3.1.3.1 Motivation

Une mémoire de traduction peut avoir été créée lors d'une traduction nouvelle, dans un environnement de traduction *ad hoc,* et dans ce cas, le découpage en segments est généralement assez fiable.

Il se peut aussi que l'on récupère une traduction effectuée dans un autre environnement et que l'on utilise un logiciel d'alignement pour établir des bisegments.

Lors du découpage d'un document textuel en segments, le système se fonde sur une connaissance *a priori* du format de fichier. Il est alors capable d'isoler les segments et de leur attribuer un identifiant. Cette opération est effectuée automatiquement.

Ces découpages se fondent sur des critères typographiques, qui peuvent parfois être pris en défaut, notamment du point de vue de la complétude des éléments de mise en forme. S'il s'agit d'un format fondé sur des balises, rien ne dit que le segment extrait constituera un tout cohérent au regard de ces balises

Par ailleurs, les segments ainsi isolés ne constituent pas nécessairement des phrases, ni des énoncés complets.

En plus de phrases correctement formées du point de vue langagier, on trouve beaucoup de segments courts, d'un seul mot parfois, comme des éléments de menu ou des entrées d'index. On trouve aussi des segments non linguistiques contenant, par exemple, des déclarations SGML ou du code.

C'est pourquoi les traducteurs ont la possibilité de réarranger les segments à leur guise. Il est ainsi possible de les découper en plus petits segments, ou au contraire d'en réunir plusieurs.

Cependant, si l'on a créé la mémoire à partir d'une traduction existante, le traducteur n'aura généralement pas l'occasion de faire ces ajustements. Le matériau produit n'est pas toujours rigoureusement parallèle. Il arrive qu'un segment couvre deux phrases, ou qu'une phrase soit découpée en plusieurs segments. La succession des segments étant respectée, les parties du document liées à ces mémoires restent cohérentes. L'information qu'elles contiennent n'est jamais réutilisée mais cela ne se voit pas.

À l'issue de la traduction, les bisegments sont enregistrés en l'état dans la mémoire de traduction. Beaucoup de segments peuvent rester inchangés. Il se peut que, tout simplement, leur traduction n'appelle pas de formulation différente en français, mais aussi qu'ils soient en fait réduits à une déclaration SGML ou autre… Selon les différents avatars de la traduction d'un document, il se peut également qu'à l'occasion d'une correction tardive, le texte cible à corriger ait remplacé le texte source originel. On risque donc de prendre pour de l'anglais ce qui est en fait du français.

Par ailleurs, beaucoup de segments sont répétés au sein d'une même mémoire ou éventuellement dans des mémoires différentes, ce qui entraîne une surreprésentation arbitraire de certaines parties du corpus.

Ce matériau est donc très hétérogène, dès le départ.

Pour toutes ces raisons, il est souhaitable de nettoyer ce corpus avant de l'utiliser. Il semble à peu près impossible de réparer les segments non exploitables. En effet, si des critères peuvent être imaginés dans la langue source, supposée connue, nous n'avons pas de garantie concernant la langue cible supposée inconnue. C'est pourquoi nous avons eu recours à un filtrage assez sévère. En fait, même le filtrage est sujet à caution, en tout cas si la langue source n'est pas véritablement connue.

3.1.3.2 Critères de filtrage

Le filtrage que nous avons mis en place utilise les règles suivantes :

1. **Éviter les segments répétés en langue source.** Nous avons délibérément choisi de limiter l'ambiguïté dans l'apprentissage. La redondance en langue cible, sans être nulle, est beaucoup moins fréquente, compte tenu du foisonnement spontané de la traduction.
2. **Le segment ne contient pas de signe inconnu, de balises etc**. Ce point est plus délicat. Il nous a semblé dans un premier temps envisageable de réparer des segments interrompus par des balises de mise en page. Mais en fait, ces balises correspondent à différents logiciels ou normes et leur prise en compte suppose une connaissance précise et profonde, complètement extérieure au corpus. En outre, les informations de mise en page ne sont pas toujours cohérentes entre la source et la cible. Nous avons donc choisi d'éliminer les segments contenant ces marques.
3. **Les deux segments commencent par une majuscule et finissent par un point.** En effet, soit à la suite d'une erreur d'interprétation du système automatique de segmentation, soit comme effet du redécoupage des segments par les traducteurs, de nombreux segments ne sont en fait que des morceaux d'énoncés, non complets syntaxiquement. Il ne s'agit même pas de proposition indépendante, par exemple. En outre, le segment traduit peut avoir été découpé à d'autres endroits, et n'avoir aucun rapport avec le segment source. En exigeant que le segment commence par une majuscule et finisse par un point, on réduit très fortement ces inconvénients.
4. **Le segment source et le segment cible sont différents.** Cette règle peut sembler arbitraire, puisque de nombreux segments courts sont identiques entre l'anglais et le français. Mais on trouve aussi dans une mémoire de traduction de nombreux segments qui n'ont pas été traduits pour diverses raisons.La prise en compte de ces segments introduirait un bruit très important dans les calculs statistiques.
5. **Pas de caractères accentués côté source.** Ce dernier critère, rudimentaire, est lié au précédent et permet d'éliminer la plupart des segments contenant du français côté source.

3.1.3.3 Résultats

Ce filtrage, assez rigoureux, nous a permis de dégager quelque 65 000 bisegments fiables, dont le détail est résumé dans le tableau suivant

	Segments	Mots (occurrences)	Mots (prototypes)	Hapax

SOURCE	64 658	691 532	18 727	7 376
CIBLE	64 658	758 896	20 334	7 981

Tableau 4 : Données chiffrées du corpus bilingue aligné de départ

On voit que le foisonnement entre la langue source et la langue cible est d'environ 10% en nombre d'occurrences, 8,5% en nombre de prototypes, et que le nombre d'hapax rencontrés est supérieur d'environ 8,2% en français par rapport à l'anglais.

La longueur moyenne des segments anglais est donc de 10,70 mots et vaut 11,74 mots pour les segments français. La longueur des segments retenus varie de 1 mot à 30 mots. Le graphique suivant illustre la répartition des segments en fonction de leur longueur.

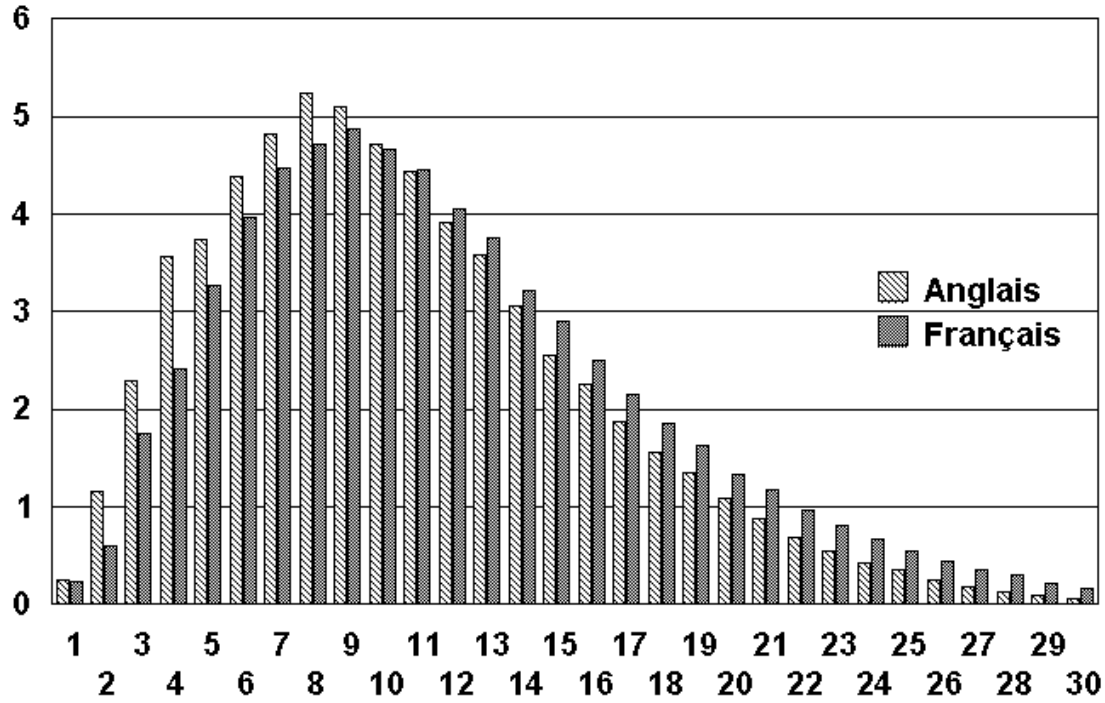

Figure 54 : Nombre de segments (en milliers) en fonction de leur longueur

On voit sur ce graphique que les segments anglais les plus nombreux (5 245) comportent 8 mots, 9 mots pour les segments français (4 869). En outre, la médiane s'établit entre 9 et 10 mots pour l'anglais, entre 10 et 11 pour le français.

L'exemple ci-dessus, qui contenait quatre bisegments, est réduit à deux après filtrage. Sur la figure suivante, nous avons supprimé les lignes de contrôle et normalisé les espaces.

```
<bisegment segnbr="93">
<source>The CS/AIX v6 SSL functions require gskkm.rte at the 5.0.4.29 level or later.</source>
<target>Les fonctions CS/AIX v6 SSL requièrent le niveau 5.0.4.29 ou suivant de gskkm.rte.</target>
</bisegment>
<bisegment segnbr="94">
<source>The installp image for gskkm.rte at this level is shipped on the CS/AIX v6.1 install media
(CD or tape).</source>
<target>L'image installp pour ce niveau de gskkm.rte est livrée sur le support d'installation de
CS/AIX v6.1 (CD-ROM ou bande).</target>
</bisegment>
```

Figure 55 : deux segments filtrés et reformatés

Un dernier traitement a consisté à ôter tous les signes de ponctuation. Là encore, cette décision peut sembler arbitraire. Elle a été prise dans un but de simplification du problème. En effet, la ponctuation n'est pas systématique en français comme en anglais. Elle n'est pas parallèle entre les deux langues et apporte un bruit supplémentaire dans les phénomènes que nous cherchons à dégager.

3.1.3.4 Alignement cible-cible

Il s'agit de mettre en œuvre l'exploitation des mémoires de traduction en sens transverse comme nous l'avons présenté en 1.3.3.2.2. Le principe de cet alignement repose sur le fait que les bisegments du corpus ont tous pour langue source l'anglais et concernent les mêmes documents traduits. Le segment source peut donc servir de pivot pour aligner les segments cible contenus dans les différentes mémoires.

Cependant, des mémoires différentes, même si elles ont été constituées à partir du même texte source, ne présentent pas nécessairement la même distribution de segments. En effet, la segmentation faite à l'origine peut avoir été modifiée en cours de traduction. Aussi est-il nécessaire effectuer un filtrage sur des mémoires différentes pour ne conserver que les segments effectivement disponibles dans toutes les langues étudiées.

Remarquons qu'il ne s'agit que d'un travail de vérification, éventuellement de recalage d'index en cas d'insertion ou de suppression de segments dans une mémoire.

En dépit de l'intérêt que nous avons porté à cette étude, nous n'avons pas pu rassembler suffisamment de matériau pour effectuer ce travail multilingue dans le cadre de cette thèse.

3.2 Acquisition/extraction d'informations

Dans la deuxième partie, nous avons exposé les méthodes que nous préconisons pour mettre en correspondance les deux parties d'un bisegment dans une mémoire de traduction, à l'aide d'informations statistiques récoltées sur un ensemble de segments. Nous allons présenter ici la mise en œuvre que nous en avons faite.

3.2.1 Correspondances élémentaires

3.2.1.1 Digrammes

Nous appelons correspondances élémentaires des liens explicites entre certaines occurrences de mots typographiques source et cible dans les bisegments. Un tel lien constitue une occurrence de vrai digramme. Si l'on considère toutes les occurrences identiques de ces correspondances, sans tenir compte de leur position au sein du bisegment, on obtient un prototype de vrai digramme.

Nous cherchons à établir des digrammes en nous donnant un double objectif : maximiser le nombre d'occurrences de vrais digrammes dans notre corpus tout en en minimisant le nombre de prototypes. Rappelons que les mots typographiques de l'une ou l'autre langue, non associés de cette manière, constituent ce que nous avons appelé de faux digrammes.

La méthode que nous avons utilisée pour établir ces liens repose sur l'information mutuelle entre les mots typographiques. À l'époque où nous avons mené ces expériences, cet indice nous a semblé naturel et nous a permis d'avancer dans notre construction de TransTree.

Nous n'avons donc pas cherché à le raffiner. Nous savons à présent qu'il en existe d'autres, éventuellement mieux adaptés à notre problématique. En effet, Fabien Cromières a mené une étude comparative de certains de ces indices [30]. Ces travaux l'amènent à privilégier l'indice de Dice :

$$Dice(w1, w2) = \frac{2 * N(w1, w2)}{N(w1) + N(w2)}$$

où $N(w1)$ est le nombre d'occurrences du mot typographique $w1$. Cet indice semble donner de meilleurs résultats pour les happax, particulièrement en rappel. D'autres indices sont envisageables : on pourra en trouver un bon exposé dans la thèse d'Olivier Kraif [7].

3.2.1.2 Heuristique utilisée

L'heuristique utilisée a été présentée au chapitre 2.2.1.2.3. Le cœur de cette heuristique est la séquence suivante :

```
Pour chaque bisegment {     // dont aucun segment n'est vide
   Pour chaque segment {
      Pour chaque graphie {
         Ordonner les graphies du segment opposé par
         information mutuelle décroissante
         }
   }
   pour chaque couple de graphies{
     SI elles sont au 1er rang l'une pour l'autre (cooptation)?
     ALORS : on enregistre le couple, et on retire ces
             graphies des segments
     SINON : rien
     }
}
```

Figure 56 : heuristique d'association des digrammes

On fait courir cette heuristique récursivement jusqu'à ce que l'un des segments soit vide. Puis on passe au segment suivant. On récupère donc un nouveau jeu de digrammes. L'ensemble de cette procédure est itéré jusqu'à figement du jeu de digrammes. On réduit alors la profondeur maximale de la récursion à chaque nouvelle passe jusqu'à la profondeur 1, c'est à dire que toutes les associations se font du premier coup.

On constate qu'à chaque passe le nombre de prototypes de digrammes diminue, sans que le nombre de graphies traduites diminue notablement. On réduit donc le bruit à chaque passe.

Nous avons évoqué en 2.2.1.3.2 une méthode non encore testée permettant de résoudre les cas litigieux d'associations en digrammes. Elle consiste à calculer les arbres binaires de sécabilité pour chaque configuration à tester et à compter le nombre de nœuds congruents que nous appelons « indice de congruence* ». Comme dans chaque cas, les digrammes utilisés sont différents, l'indice de congruence en dépend en général. La configuration retenue sera celle dans laquelle cet indice est le plus grand. On n'utilise la méthode des moindres croisements qu'en dernier ressort, en cas d'égalité de plusieurs solutions optimales.

3.2.1.3 Résultats de l'alignement en digrammes

L'alignement en digrammes que nous avons effectué avec le matériau décrit plus haut nous a donné les résultat suivants : En terme de volume, nous avons obtenu 532 098 correspondances entre occurrences de mots typographiques sur l'ensemble du corpus, soit une moyenne de 8,23 par segment, à rapprocher du nombre de mots par segment déjà indiqué, soit 10,70 pour l'anglais et 11,74 pour le français.

En terme de prototypes, autrement dit de digrammes différents, le tableau suivant résume quelques résultats indicatifs :

Digrammes	66 994
Vrais digrammes	47 796
Faux digrammes anglais	9 275
Faux digrammes français	9 923

Tableau 5 : Données chiffrées des digrammes obtenus

Enfin, il faut constater que ces données contiennent de nombreux hapax. Nous donnons ci-dessous un tableau récapitulatif des hapax obtenus.

Digrammes hapax	37 879
Vrais digrammes hapax	26 046
Faux dig. anglais hapax	5 532
Faux dig. français hapax	6 301
Vrais digrammes d'hapax	3 796

Tableau 6 : Données chiffrées des digrammes hapax

Voici un extrait de notre corpus, réécrit avec des digrammes, correspondant à un exemple utilisé dans la deuxième partie. Nous avons indexés les digrammes avec des étiquettes binaires précédées d'un prefixe. Ce préfixe est **X0#** pour les vrais digrammes, **S0#** pour les faux digrammes source et **T0#** pour les faux digrammes cible.

```
- <bisegment>
    <segmentEn>This task shows you how to change views</segmentEn>
    <segmentFr>Dans cette tâche vous apprendrez à modifier les vues</segmentFr>
    <digramEn>T0#00b59 X0#00807 T0#02420 X0#001ea T0#01028 X0#00065
      X0#0041f X0#00df7</digramEn>
    <digramFr>T0#00b57 T0#00b57 X0#00807 X0#001ea T0#01b29 X0#00065
      X0#0041f T0#0003c X0#00df7</digramFr>
    <bridge posEn="2" posFr="3" />
    <bridge posEn="4" posFr="4" />
    <bridge posEn="6" posFr="6" />
    <bridge posEn="7" posFr="7" />
    <bridge posEn="8" posFr="9" />
  </bisegment>
```

Figure 57 : extrait d'un mémoire réécrite en digrammes

Les digrammes observés sont consignés dans la base de connaissances. Voici quelques extraits des fichiers de digrammes :

Figure 58 : quelques vrais digrammes de la base de connaissances

On peut observer que le digrammes **X0#001ea** a été trouvé 4752 fois dans le corpus. Parmi les autres digrammes, on trouve quelques hapax, dont un vraisemblablement fautif, le digramme **X0#00706**.

Pour les faux digrammes source, on à l'extrait suivant :

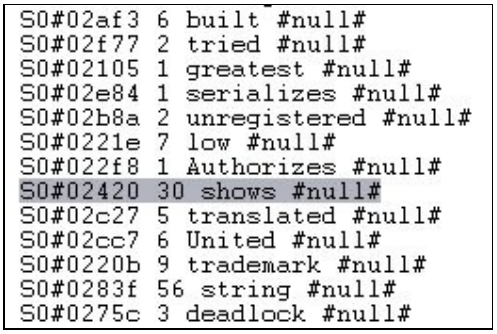

Figure 59 : quelques faux digrammes source de la base de connaissances

Et pour les faux digrammes cible, on à l'extrait suivant :

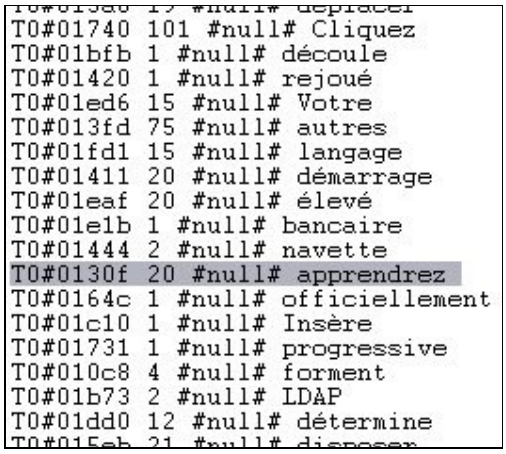

Figure 60 : quelques faux digrammes cible de la base de connaissances

Il peut sembler surprenant que certains mots typographiques, dont le sens est plein et pour lesquels nous connaissons intuitivement une traduction, soient considérés comme de faux digrammes. Il s'agit le plus souvent d'un mot typographique entrant dans un syntagme dont la traduction est portée par d'autres (vrais) digramme, comme c'est le cas de cet exemple. Ces faux digrammes seront rapprochés du syntagme auxquels ils appartiennent par le jeu des sécabilités. Il peut s'agir aussi, naturellement, d'une insuffisance de notre méthode.

Nous avons décompté les nombres de n-grammes de notre corpus avec les digrammes ainsi formés. Les résultats obtenus sont donnés dans les tableaux suivants. Les nombres d'occurrences n'apporte pas d'information nouvelle puisque nous connaissons déjà les indications relatives à la taille du corpus et au nombre de segments (voir 3.1.3.3).

Bigrammes (prototypes)	316 708
Bigrammes (occurrences)	756 190
Trigrammes (prototypes)	516 969
Trigrammes (occurrences)	691 532

Tableau 7 : Données chiffrées des n-grammes de niveau 0 en anglais

Bigrammes (prototypes)	298 376
Bigrammes (occurrences)	823 554
Trigrammes (prototypes)	517 074
Trigrammes (occurrences)	758 896

Tableau 8 : Données chiffrées des n-grammes de niveau 0 en français

On voit que le nombre de prototypes de bigrammes (2-grammes de digrammes) est d'environs 300 000, alors que le nombre de prototypes de trigrammes s'établit autour de 500 000, c'est-à-dire moins du double. Ce rapport semble faible : il signifie que les digrammes sont très nombreux (en prototypes) par rapport à la taille du corpus, et que leur dispersion dans ce dernier est très grande. Ce résultat suggèrerait bien sûr d'améliorer l'association en digrammes. Nous avons néanmoins utilisé ces n-grammes pour établir les arbres binaires de sécabilité dans notre corpus.

3.2.2 Correspondances complexes

3.2.2.1 Généralités

3.2.2.1.1 Principe

Les amphigrammes sont obtenus par confrontation des arbres binaires de sécabilité dans un bisegment. Nous avons retenu l'approche par niveaux, sans utiliser de classes.

Autrement dit, cet algorithme réduit les conséquences du fait que la sécabilité est un index local. En revanche, il ne contrecarre pas le phénomène de rareté croissante des données, c'est-à-dire des amphigrammes établis à chaque niveau. Ceux-ci représentent en effet des chaînes de caractères de plus en plus longues et donc de plus en plus rares dans le corpus.

L'approche par niveau consiste à utiliser plusieurs arbres binaires de sécabilité. On commence par calculer les arbres binaires de sécabilité sur le bisegment réécrit en digrammes, ce qui permet d'établir des amphigrammes de premier nivau. Puis on utilise les amphigrammes de premier niveau comme nouveaux symboles pour calculer un nouvel arbre binaire de sécabilité sur chacun des segments du bisegment. On en déduit alors des amphigrammes ce deuxième niveau et ainsi de suite. Le nombre d'amphigrammes établis à chaque niveau décroît jusqu'à obtenir un unique amphigramme.

Au terme de ce traitement, le bisegment est réduit à un unique amphigramme et l'analyse est terminée. Le nombre de niveaux nécessaires dépend de la longueur des segments du bisegment.

3.2.2.1.2 Différence entre le premier niveau et les niveaux supérieurs.

Nous avons élaboré deux méthodes d'acquisition des amphigrammes. L'une s'applique au premier niveau et l'autre aux niveaux supérieurs. Par « niveaux supérieurs », nous entendons l'exploitation des bisegments réécrits avec les identifiants des symboles de premier niveau, ou plus généralement de niveau inférieur. Ces identifiants renvoient à des amphigrammes ou, éventuellement, à des arbres binaires de faux digrammes.

La différence entre premier niveau et niveaux supérieurs est la suivante : au premier niveau, les amphigrammes construits rassemblent chacun les faux digrammes autour d'un vrai digramme, considéré comme pivot ; aux niveaux supérieurs, on enregistre l'agencement dans chaque langue des amphigrammes de niveau immédiatement inférieur.

Par conséquent, les amphigrammes de premier niveau ne contiennent qu'un seul vrai digramme, autour duquel on trouve éventuellement de faux digrammes, dans chacune des deux langues.

Les amphigrammes des nivaux supérieurs contiennent en général deux vrais digrammes, ou plus, ou éventuellement un seul, en fonction de la géométrie des arbres binaires. Nous donnons des précisions à ce sujet en 3.2.2.3.

3.2.2.1.3 Indice de sécabilité

Pour établir les arbres binaires de sécabilité, quel que soit le niveau considéré, il nous faut décider d'un indice de sécabilité. Nous avons calculé les indices de sécabilité entre **b** et **c** dans un quadrigramme **[abcd]** avec la formule suivante, déjà donnée en 2.2.2.1.3 :

$$sec([bc]) = \frac{n([ab]) \cdot n([bc]) \cdot n([cd])}{n([abc]) \cdot n([bcd])}$$

Cet indice de sécabilité nous a permis d'établir les arbres binaires à tous les niveaux de l'analyse.

3.2.2.2 Premier niveau

3.2.2.2.1 Algorithme

Au premier niveau, nous enregistrons des paires de sous-arbres binaires contenant exactement un vrai digramme accompagné éventuellement de faux digrammes. Ces paires constituent des amphigrammes de premier niveau.

D'autre part, nous enregistrons aussi des sous-arbres binaires de faux digrammes « résiduels ». En effet, il arrive que des sous-arbres de faux digrammes apparaissent ainsi que l'illustre l'exemple suivant :

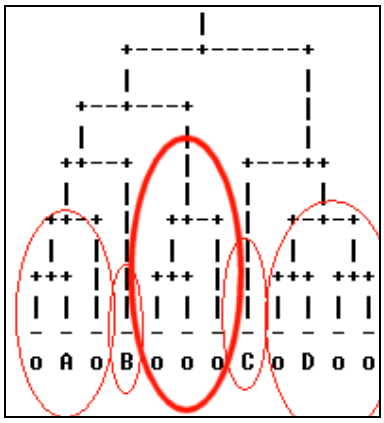

Figure 61 : exemple d'arbre de faux digrammes

Sur ce schéma, les lettres **A**, **B**, **C** et **D** figurent de vrais digrammes. Les petits **o** sont de faux digrammes. Comme il y a quatre vrais digrammes, on construit quatre sous-arbres saturés qui seront nécessairement congruents avec un sous-arbre dans l'autre arbre binaire de sécabilité du bisegment, puisqu'à ce niveau, ils ne contiennent chacun qu'un seul vrai digramme. Nous avons entouré ces sous-arbres avec un trait fin.

L'ellipse en trait plus épais entoure tout un arbre résiduel, ne contenant que des faux digrammes. Dans l'algorithme que nous avons utilisé, nous avons considéré que cet arbre entier formait un faux digramme de premier niveau. Il reçoit donc un identifiant et entrera en tant que tel dans le calcul des arbres binaires de sécabilité d'ordres supérieurs.

De même, on voit sur ce schéma qu'il est très possible qu'un arbre binaire saturé soit réduit à un vrai digramme (cas **B** et **C**). Si la situation est analogue dans l'autre arbre binaire, un amphigramme peut donc être réduit à un vrai digramme établi au niveau inférieur.

Nous avons utilisé l'algorithme suivant pour établir les arbres binaires de sécabilité, et construire les amphigrammes de premier niveau :

```
pour chaque bisegment
{
  pour chaque segment
  {
     • ordonner les séparateurs en fonction de leur sécabilité
     • construire l'arbre binaire de sécabilité induit
     • repérer les sous-arbres saturés dominant 1 vrai digramme
  }
  • enregistrer les paires de sous-arbres saturés congruents
  • enregistrer les sous-arbres binaires résiduels
  • réécrire le bisegment avec les identifiants obtenus
}
```

Figure 62 : algorithme de construction des amphigrammes de premier niveau

L'indexation des amphigrammes reprend celle qui a été utilisée pour les digrammes, avec un préfixe **X1#**, **S1#**ou **T1#**.

3.2.2.2.2 Exemple

Dans notre exemple, le segment n'est pas beaucoup modifié. Seul le mot typographique « vues » a été rapproché de « les » dans les arbres binaires, pour donner l'amphigramme **X1#13ece.**

```
- <bisegment>
    <segmentEn>This task shows you how to change views</segmentEn>
    <segmentFr>Dans cette tâche vous apprendrez à modifier les vues</segmentFr>
    <amph1En>S1#13a8e X1#13c60 S1#13f21 X1#13b9e S1#143d9 X1#138f4 X1#14c4c
      X1#13ece</amph1En>
    <amph1Fr>T1#13eff T1#13a91 X1#13c60 X1#13b9e T1#14f74 X1#138f4 X1#14c4c
      X1#13ece</amph1Fr>
    <bridge posEn="2" posFr="3" />
    <bridge posEn="4" posFr="4" />
    <bridge posEn="6" posFr="6" />
    <bridge posEn="7" posFr="7" />
    <bridge posEn="8" posFr="8" />
  </bisegment>
```

Figure 63 : extrait de la mémoire de traduction avec des amphigrammes de niveau 1

Nous présentons ci-dessous un extrait du fichier qui réunit les amphigrammes de premier niveau. On trouve à gauche l'index de l'amphigramme, son nombre d'occurrences dans le corpus, suivi d'une version parenthésée de son expression.

```
X1#13ec1        123     X0#007b9 X0#007b9
X1#13ec4        28      X0#007c4 X0#007c4
X1#13ec5        27      X0#007c2 X0#007c2
X1#13ec6        8       X0#007b2 (X0#007b2,T0#00083)
X1#13ece        21      X0#00df7 (T0#0003c,X0#00df7)
X1#13ecf        1       (X0#007ca,S0#007cb) (X0#007ca,T0#007cc)
X1#13ed3        8       (X0#007d0,S0#00077) (X0#007d0,T0#007d2)
X1#13edb        88      X0#007d6 X0#007d6
X1#13edc        10      X0#007d7 X0#007d7
X1#13edd        1       X0#007d8 X0#007d8
```

Figure 64 : extrait de la base de connaissances pour les amphigrammes de niveau 1

Nous voyons que l'amphigramme **X1#13ece** est présent 21 fois dans le corpus. Nous allons à présent détailler le volume des données traitées.

3.2.2.2.3 Données numériques

Nous avons réuni les chiffres suivants pour rendre compte des volumes traités à ce niveau. Dans ce tableau, nous avons fait figurer les sous-arbres binaires source et les sous-arbres binaires cible.

Objet \ Unité	Prototype	Occurrence	Hapax
Amphigrammes	11 466	77 804	5 797
Ss-arbres source	4 311	22 377	2 257
Ss-arbres cible	4 857	31 107	2 584

Tableau 9 : Données chiffrées des amphigrammes et des sous-arbres binaires

Le nombre total d'occurrences d'objets dans notre corpus est donc de 77 804+22 377+31 107 = 131 288. Nous voyons que les hapax représentent un peu plus de la moitié du nombre de prototypes recensés dans chaque catégorie.

Concernant les n-grammes, nous avons obtenus les résultats suivants :

Bigrammes (prototypes)	316 708
Bigrammes (occurrences)	756 190
Trigrammes (prototypes)	516 969
Trigrammes (occurrences)	691 532

Tableau 10 : Données chiffrées des n-grammes de niveau 1 en anglais

Bigrammes (prototypes)	298 376
Bigrammes (occurrences)	823 554
Trigrammes (prototypes)	517 074
Trigrammes (occurrences)	758 896

Tableau 11 : Données chiffrées des n-grammes de niveau 1 en français

3.2.2.3 Niveaux supérieurs

3.2.2.3.1 Algorithme

Aux niveaux supérieurs, les sous-arbres considérés dominent un nombre variable d'amphigrammes du niveau immédiatement inférieur.

L'algorithme utilisé considère d'abord les nœuds saturés congruents dominant exactement deux amphigrammes de niveau inférieur. Toutes ces paires de nœuds étant repérées et enregistrées sous forme d'amphigrammes, l'algorithme considère ensuite les nœuds saturés congruents dominant trois amphigrammes, à l'exception de ceux qui domineraient des nœuds déjà enregistrés, puis quatre amphigrammes etc. Ensuite, les nœuds congruents saturés résiduels, ne dominant qu'un seul amphigramme, sont pris en compte. Enfin, les sous-arbres binaires de faux digrammes non encore considérés sont enregistrés en tant que tels.

Nous pouvons exprimer cet algorithme de la manière suivante :

```
pour chaque bisegment
{
   pour chaque segment
   {
      • ordonner les séparateurs en fonction de leur sécabilité
      • construire l'arbre binaire de sécabilité induit
      • ordonner les nœuds saturés selon le nombre
        d'amphigrammes qu'ils dominent : 2,3,...,max,1,0   (•)
   }
   pour (dans l'ordre •) chaque nœud saturé source
   {
       s'il ne domine pas de nœud déjà enregistré à ce niveau
       et s'il admet un nœud congruent (saturé) cible
        {
            enregistrer cette paire de sous-arbres
        }
   }
   • enregistrer les sous-arbres binaires résiduels
   • Réécrire le bisegment avec les identifiants obtenus
}
```

Figure 65 : algorithme de construction des amphigrammes de niveaux supérieur

La encore, l'indexation des amphigrammes reprend celle qui a été utilisée pour les digrammes, avec un préfixe **Xn#**, **Sn#** ou **Tn#**, où **n** représente le niveau d'analyse en cours.

3.2.2.3.2 Données numériques

Dans le tableau suivant, nous avons reporté les nombres de prototypes, d'occurrences et d'hapax d'amphigrammes établis dans le corpus. Nous y avons adjoint les nombres de sous-arbres binaires source et cible que l'on trouve dans le corpus à ce niveau.

Objet \ Unité	Prototype	Occurrence	Hapax
Amphigrammes 2	17 816	48 148	12 399
Ss-arbres source	1 962	8 367	1 207
Ss-arbres cible	2 140	11 350	1 348

Tableau 12 : Données chiffrées des amphigrammes de niveau 2

Les chiffres présentés ici reflètent plusieurs phénomènes. Le nombre total d'occurrences d'objets est maintenant de 48 148+8 367+11 350 = 67 865. On voit que ce nombre est pratiquement réduit de moitié par rapport à ce qu'on avait établi au premier niveau (131 288)

La rareté des données commence à se faire sentir dès ce niveau, ce qui explique la grande proportion d'hapax, qui avoisine les deux tiers des prototypes.

Ces observations sont encore plus sensibles au troisième niveau, où l'on constate que le nombre total d'occurrence est encore divisé par deux et que le nombre d'hapax pour les amphigrammes est maintenant proche des trois quarts des prototypes.

Objet \ Unité	Prototype	Occurrence	Hapax

Amphigrammes 3	12 413	25 859	9 266
Ss-arbres source	810	3 198	518
Ss-arbres cible	888	4 452	573

Tableau 13 : Données chiffrées des amphigrammes de niveau 3

En pratique, les sous-segments modélisés par les amphigrammes sont de plus en plus rares dans le corpus, et l'indice de sécabilité perd de sa pertinence pour les niveaux supérieurs.

3.2.2.3.3 Fin de l'analyse

Au fur et à mesure que l'on analyse le corpus, les bisegments les plus courts sont réécrits avec un unique amphigramme. Il n'y a pas lieu de pousser l'analyse plus loin. À chaque niveau, on observe donc une augmentation du nombre de bisegments complètement analysés. Le nombre de niveau d'analyse pour parvenir à réécrire un bisegment sous forme d'un unique amphigramme, autrement dit la hauteur de l'arbre TransTree, dépend de la complexité du bisegment et de sa longueur. Globalement, ce sont les segments les plus longs qui se réécrivent les derniers en un unique amphigramme.

La courbe reproduite ci-dessous donne, en pourcentage, la proportion du corpus analysé en fonction du niveau d'analyse. On voit qu'après neuf niveaux d'analyse, la quasi totalité des bisegments (99,48%) a été analysée.

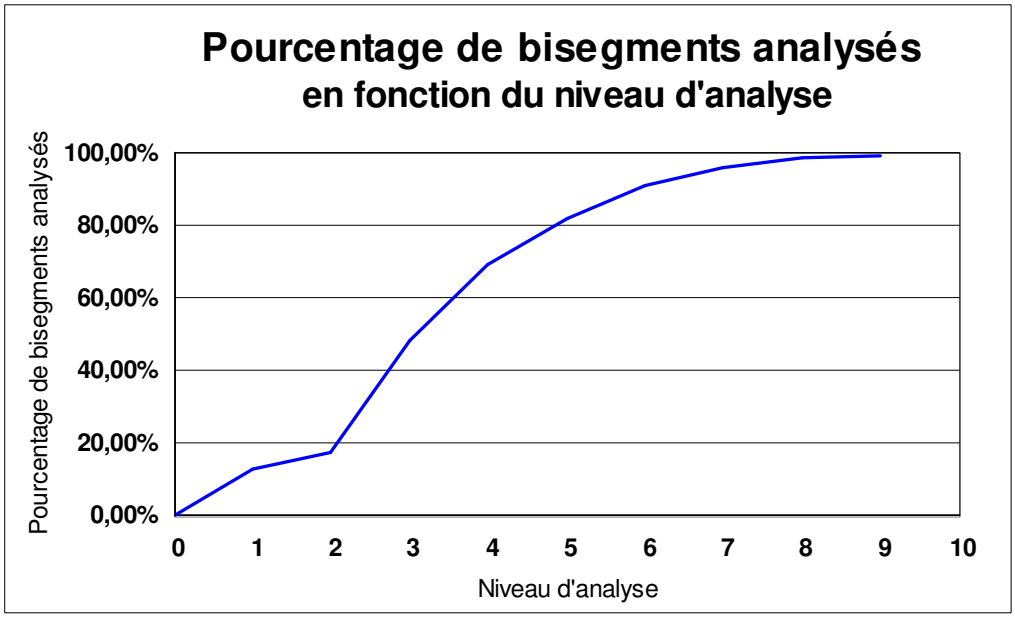

Figure 66 : Pourcentage de bisegments analysés en fonction du niveau d'analyse

Sur ce graphique, on voit aussi que la différence de méthode entre le premier niveau d'analyse et les suivants est tout à fait sensible. Rappelons que ce mode opératoire permet dans un premier temps de repérer les amphigrammes ne contenant qu'un seul digramme, afin d'agréger autour la plupart des faux digrammes.

L'analyse de premier niveau ne consomme qu'un seul vrai digramme. Elle ne traite donc complètement que les segments réduits à un seul vrai digramme complété éventuellement par quelques mots typographiques constituant des faux digrammes. On pourra se reporter aux

listes de digrammes et d'amphigrammes obtenues lors de l'analyse du corpus. On en trouvera quelques extraits à l'annexe 4 (Dictionnaires de termes simples obtenus).

Les amphigrammes de niveaux supérieurs permettent essentiellement d'enregistrer les modalités de réagencement des unités entre les deux langues, pour autant que ces unités sont suffisamment représentées dans le corpus, ce qui n'est pas vrai en général pour les derniers niveaux de l'analyse. Nous n'avons pas eu le temps de mener complètement à son terme la méthode que nous préconisons pour remédier à cette situation, c'est-à-dire la classification des amphigrammes et l'utilisation d'amphigrammes génériques.

3.2.3 Classification

Les classes permettent de factoriser la connaissance acquise dans le corpus. Même avec un grand corpus, chaque suite de mots est extrêmement rare. En revanche, une suite de classes représentant chacune plusieurs objets permet d'enregistrer une information factorisée de plus haut niveau. La difficulté consiste à établir un jeu de classes adéquates.

3.2.3.1 Classes contextuelles

3.2.3.1.1 Relation d'équivalence sur les trigrammes

Nous donnons ici l'algorithme que nous avons utilisé pour mettre en application la méthode classification conduisant à des classes simplement contextuelles, telle que nous l'avons présentée en 2.2.3.2.1.

Rappelons la relation d'équivalence : tous les digrammes **b**, apparaissant dans un même contexte, **[a_c]**, dans l'ensemble des trigrammes de référence, sont équivalents. Le fait d'apparaître concerne une occurrence au moins.

Un prototype **b** est susceptible d'apparaître dans plusieurs contextes différents. Comme nous avons choisi de constituer des classes d'équivalence, il s'ensuit que tous les contextes **[a_c]** contenant un même prototype **b** sont eux-mêmes équivalents. On établit ainsi la propriété de transitivité, nécessaire pour obtenir une relation d'équivalence.

Nous construisons donc deux relations d'équivalence, l'une sur les contextes, l'autre sur les digrammes. Tous les digrammes d'une même classe s'inscrivent dans les contextes d'une même classe, et les contextes d'une même classe délimitent tous les digrammes d'une même classe. Autrement dit, il y a autant de classes de digrammes que de classes de contextes et l'on peut utiliser les mêmes identifiants pour désigner classes de contextes et classes de digrammes.

L'utilisation simultanée de ces deux relations conduit à l'algorithme récursif suivant :

```
pour tout b
   si classId(b) existe, next    // passer au b suivant
   créer un classId pour b
   listeDesElements(classId(b))
    = b + listeDesElementsDeMemeClassId(b)
finpour
```

```
fonction : listeDesElementsDeMemeClassId (b)
listElem est vide
pour tout contexte [a_c] contenant b
   si classId([a_c])existe, next   // passer au [a_c] suivant
   classId([a_c]) = classId(b)
   pour tout b' apparaissant dans [a_c]
     si classId(b') existe, next // passer au b' suivant
     classId(b') = classId(b)
     listElem
           = listElem + b' + listeDesElementsDeMemeClassId(b')
     finpour
   finpour
retour de listElem
finfonction
```

Figure 67 : algorithme utilisé pour établir la relation d'équivalence contextuelle

Nous avons opéré en plusieurs passes : les classes obtenues à l'occasion d'une passe permettent de réécrire les contextes lors de la passe suivante.

Les classes concernant les vrais digrammes, pour être reportées dans les amphigrammes génériques, doivent être bilingues. Or les contextes qui nous servent de définition sont monolingues par essence. Nous obtenons donc deux partitions de l'ensemble des vrais digrammes, et considérons les intersections des classes de chacune de ces partitions.

3.2.3.1.2 Restrictions sur les trigrammes utilisés

Tous les trigrammes rencontrés dans les segments ne sont pas retenus pour établir les classes. Seuls sont considérés ceux qui satisfont les deux critères suivants.

- on ne retient pas les trigrammes de type **[#let#,b,#right#]**
- un certain indice de pertinence entre le contexte **[a_c]** et l'élément intérieur **b** est supérieur à un certain seuil.

La première restriction a pour objet de ne pas retenir les segments constitués d'un seul mot typographique. Le contexte y est complètement vide, et donc très peu informatif. C'est à la suite d'essais infructueux que nous sommes arrivés à cette nécessité.

L'indice de pertinence utilisé pour la deuxième restriction a été calculé ainsi :

$$\text{pertinence} = \frac{N([abc])^3}{N([a\_c])^2 \cdot N([b])}$$

où **N([abc])** représente le nombre d'occurrences du trigramme **[abc]**, **N([a_c])** le nombre d'occurrences du contexte **[a_c]**, et **N([b])** le nombre d'occurrences de l'objet **b** dans le corpus.

Le jeu de classes obtenu diffère considérablement en fonction du seuil appliqué sur l'indice de pertinence.

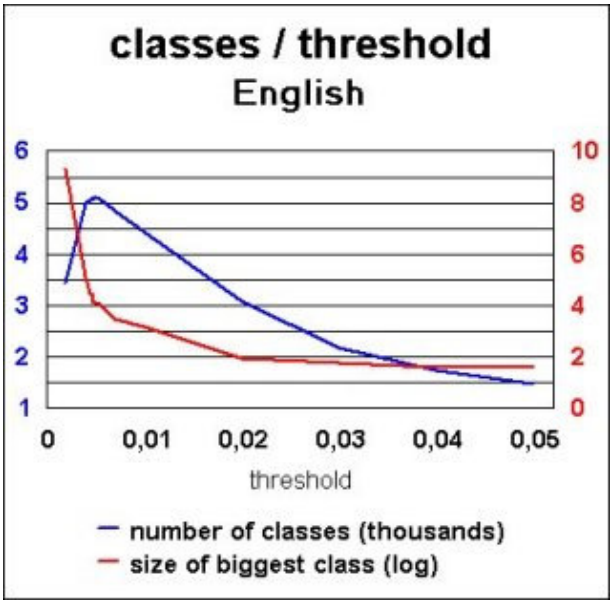

Figure 68 : seuil d'effondrement du nombre de classes (En)

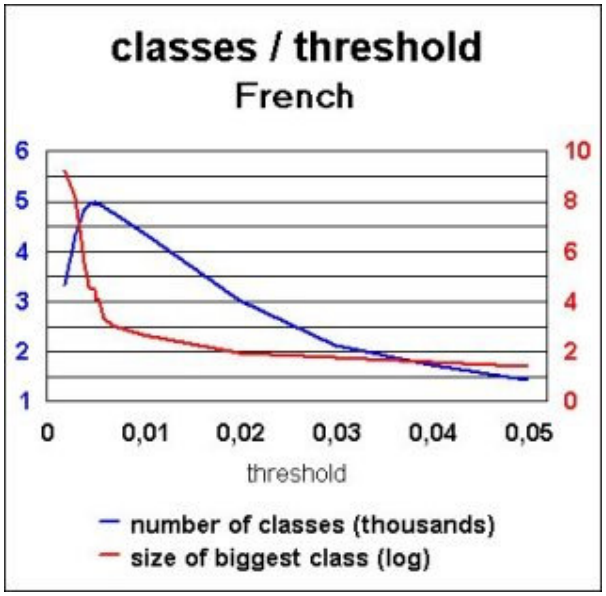

Figure 69 : seuil d'effondrement du nombre de classes (Fr)

Ces deux graphiques[21]. donnent le nombre de classes et la taille de la plus grande classe en fonction du seuil utilisé, pour l'anglais et pour le français.

Le nombre de classes est deonné par la courbe présentant un maximum, dont l'échelle, en milliers de classes, est à gauche du graphique. L'autre courbe donne le logarithme décimal de la taille de la plus grande classse, à droite du graphique.

On constate que le nombre de classes atteint un pic pour des valeurs proches de 0,005 (0,00485 pour l'anglais et 0,00509 pour le français).

Quand le seuil de pertinence est élevé (au dessus de 0,005), le nombre de trigrammes retenus est faible, le nombre de classes en résultant aussi. Remarquons que les occurrences de

[21] Les deux graphiques ont été établis en anglais

ces trigrammes sont sporadiquement réparties dans le corpus et ne se chevauchent que rarement. Il ne s'agit que de digrammes « **b** » c'est-à-dire intérieurs à un trigramme **[abc]**, et le fait de les classer ne modifie pas le contexte d'autres digrammes, même en effectuant plusieurs passes.

À l'inverse, quand le seuil de pertinence est bas (en dessous de 0,005), les trigrammes utilisés dans la classification sont nombreux, leurs occurrences se chevauchent dans le corpus, et, bien que le nombre de digrammes à classer soit important, le nombre de classes résultant est également faible. En effet, de nombreux trigrammes exceptionnels, non représentatifs, entrent dans la construction des classes. Des digrammes employés dans un contexte atypique sont mis en équivalence et les classes auxquelles ils appartiennent fusionnent, ce qui explique leur moindre nombre.

Entre ces deux extrêmes, il existe une valeur du seuil de pertinence telle que le nombre de classes est maximal. Le nombre de classes en fonction de ce seuil conduit à une courbe en cloche comme on le voit dans les graphique précédents.

L'autre courbe, dont l'échelle, est à droite, donne le logarithme décimal du nombre d'éléments de la classe la plus fournie. On voit que la courbe s'infléchit fortement aux environs de cette même valeur : une classe hypertrophiée apparaît quand le seuil utilisé est bas et que les classes fusionnent facilement. À l'inverse, quand il y a peu de trigrammes à classer, même la classe la plus fournie n'est pas très volumineuse.

3.2.3.1.3 Résultats indicatifs et conclusion

La meilleure valeur à adopter pour le seuil semble donc être celle où le pic se produit, soit 0,005 dans le cas des digrammes. Le nombre de classes d'au moins deux éléments obtenu pour l'anglais est de 5858 et de 5248 pour le français. La couverture est d'environ un digramme sur 3 dans le corpus. Le nombre de classes bilingues, d'au moins deux éléments, obtenues par produit cartésien est de 1207.

Nous donnons ici un aperçu de ces classes. Il s'agit de bons exemples, ou du moins de classes illustrant certains phénomènes :

Dans ce premier extrait (Figure 70), nous avons réuni quelques classes dont la cohérence sémantique et morphosyntaxique est tout à fait satisfaisante pour le jugement humain :

```
- <classes>
  - <class nbr="6" cnt="5">
      <dig src="User" trg="utilisateur" cnt="10" />
      <dig src="User" trg="User" cnt="21" />
      <dig src="Application" trg="application" cnt="2" />
      <dig src="Resource" trg="ressource" cnt="4" />
      <dig src="Group" trg="groupe" cnt="10" />
    </class>
  - <class nbr="946" cnt="3">
      <dig src="special" trg="spéciaux" cnt="3" />
      <dig src="alphanumeric" trg="alphanumériques" cnt="2" />
      <dig src="numeric" trg="numériques" cnt="3" />
    </class>
  - <class nbr="149" cnt="4">
      <dig src="minimum" trg="minimal" cnt="1" />
      <dig src="maximum" trg="maximum" cnt="3" />
      <dig src="minimum" trg="minimum" cnt="2" />
      <dig src="maximum" trg="maximal" cnt="7" />
    </class>
  - <class nbr="100" cnt="2">
      <dig src="physical" trg="physique" cnt="1" />
      <dig src="logical" trg="logique" cnt="4" />
    </class>
```

Figure 70 : Quelques classes très cohérentes

On peut constater sur cet exemple que le fait d'être très peu représenté dans le corpus n'empêche pas un digramme d'entrer dans une bonne classe ; il s'agit, par exemple, des digrammes (minimum/minimal) dans la classe 149 ou de (physical/physique) dans la classe 100, dont il n'existe qu'une occurrence dans le corpus.

Dans d'autres cas, on observe un transfert de catégorie.

```
<class nbr="306" cnt="2">
  <dig src="creating" trg="créer" cnt="2" />
  <dig src="loading" trg="charger" cnt="1" />
</class>
<class nbr="336" cnt="2">
  <dig src="group" trg="groupes" cnt="9" />
  <dig src="category" trg="catégories" cnt="2" />
</class>
<class nbr="548" cnt="3">
  <dig src="Add" trg="Ajouter" cnt="1" />
  <dig src="Browse" trg="Parcourir" cnt="3" />
  <dig src="Set" trg="Définir" cnt="1" />
</class>
<class nbr="597" cnt="2">
  <dig src="contains" trg="contenant" cnt="7" />
  <dig src="defines" trg="définissant" cnt="1" />
</class>
```

Figure 71 : Quelques classes présentant un transfert de catégories

Ces exemples concernent le passage d'une forme verbale à une autre ainsi que du singulier au pluriel.

Certaines classes présente une bonne homogénéité sémantique mais présentent une certaine variabilité morphosyntaxique :

```
<class nbr="626" cnt="2">
  <dig src="container" trg="conteneur" cnt="14" />
  <dig src="containers" trg="conteneurs" cnt="1" />
</class>
<class nbr="713" cnt="2">
  <dig src="changed" trg="modifiée" cnt="1" />
  <dig src="failed" trg="échoué" cnt="2" />
</class>
<class nbr="843" cnt="2">
  <dig src="registered" trg="enregistrée" cnt="6" />
  <dig src="included" trg="fourni" cnt="1" />
</class>
<class nbr="867" cnt="2">
  <dig src="criteria" trg="critères" cnt="6" />
  <dig src="result" trg="résultats" cnt="1" />
</class>
```

Figure 72 : Variabilité morpho syntaxique

Nous voyons ici que le regroupement s'est effectué sur un critère plus sémantique que syntaxique : le féminin n'étant pas marqué en anglais, on peut supposer que les contextes français conduisant à ces classes ont des traits morphosyntaxiques peu marqués.

D'autre classe encore contiennent du bruit au milieu d'un ensemble assez homogène. C'est le cas dans l'exemple suivant :

```
- <class nbr="771" cnt="7">
    <dig src="adding" trg="ajoutez" cnt="3" />
    <dig src="different" trg="différents" cnt="3" />
    <dig src="removing" trg="entre" cnt="1" />
    <dig src="specifying" trg="indiquez" cnt="2" />
    <dig src="changing" trg="modifiez" cnt="5" />
    <dig src="defining" trg="définissez" cnt="2" />
    <dig src="removing" trg="supprimez" cnt="3" />
  </class>
```

Figure 73 : classe présentant du bruit

Ici, le digramme apparemment fautif (removing/entre) est un hapax dans le corpus. Le digramme plus admissible pour le jugement humain (different/différents) donne lieu à trois occurrences dans le corpus. Ils se sont glissés dans une classe par ailleurs très homogène, présentant un transfert de forme verbale.

Ces expérimentations nous ont montré qu'il était très difficile d'obtenir un jeu de classes bilingues raisonnable. Nous entendons par "jeu raisonnable" un ensemble de classes qui permette d'obtenir une bonne couverture du corpus avec des classes de tailles comparables. Nous avons, en pratique, obtenu quelques classes hypertrophiées, contenant plusieurs centaines voire plusieurs milliers d'éléments, accompagnées de nombreuses classes très réduites, contenant deux ou trois éléments. Les jeux de classes obtenus, monolingues et bilingues confondues, ne permettaient pas une couverture de plus du tiers du corpus en volume.

3.2.3.2 Classes optimisées

3.2.3.2.1 Algorithme

Nous donnons ici la description de l'algorithme génétique esquissé au 2.2.3.2.2.

Nous voulons classer les éléments[22] d'un ensemble donné. Nous créons une liste de référence des éléments concernés : chaque élément reçoit un indice de position. Chaque génome est un tableau de même taille, dont chaque position est occupée par un identifiant de classe. Autrement dit, un individu dans notre population, un génome, est une application de l'ensemble des éléments à classer vers un jeu d'identifiants. Notre population de génomes constitue donc un assortiment de partitions de notre ensemble. Chaque génome est une classification potentielle.

La figure suivante illustre une population de génomes à une étape quelconque de l'algorithme.

Ensemble à classer	Population de génomes			
	Génome A	Génome B	Génome C	Génome
Elém 1	www	zzz	www	ww
Elém 2	xxx	www	xxx	xx
Elém 3	yyy	xxx	yyy	y
Elém 4	zzz	yyy	zzz	z
Elém 5	www	zzz	www	x
Elém 6	xxx	xxx	xxx	yy
Elém 7	yyy	www	yyy	z
Elém 8	www	zzz	yyy	ww
Elém 9	xxx	www	zzz	x
Elém 10	yyy	xxx	xxx	y
Elém 11	zzz	yyy	zzz	y
Elém 12	www	zzz	www	z
Elém 13	zzz	yyy	xxx	x
Elém 14	www	zzz	yyy	ww
Elém 15	xxx	xxx	zzz	yy
Elém 16	yyy	www	zzz	y
Elém 17	zzz	zzz	zzz	y
Elém 18	www	www	www	z
Elém 19	xxx	xxx	xxx	x
				w

Figure 74 : exemple de population génomique

Sur cet exemple, on voit, à gauche, les 19 premiers éléments à classer et, dans la partie principale, les 3 premiers génomes (**A**, **B** et **C**) de la population testée. Il y a 4 identifiants de classes : **www**, **xxx**, **yyy** et **zzz**. L'ensemble des éléments sera donc réparti en quatre classes.

[22] Nous préciserons plus loin de quels objets nous parlons.

Le premier génome donne les quatre classes {1,5,8,12,14,18…} , {2,6,9,15,19…} , {3,7,10,16…} et {4,11,13,17…} ; le deuxième génome donne les classes {1,5,8,12,14,17…}, {2,7,9,16,18…}, {3,6,10,15,19…} et {4,11,13…} etc.

Une fonction de performance permet de mesurer la performance de chaque génome, autrement dit la qualité de chaque classification.

Un génome issu du croisement de deux génomes parents s'effectue en affectant au hasard à chaque position l'un des deux identifiants porté par les parents à cette même position. La partition en classes définies par le nouveau génome hérite donc en partie de chacun des deux parents.

Le renouvellement du patrimoine génétique de la population est assuré par un certain taux de mutation spontanée (affectation d'un identifiant de classe aléatoire, indépendamment des parents, pour une position donnée). Ce taux est lui-même variable. Il est recalculé à chaque génération en fonction du nombre de mutations observées dans la population des meilleurs génomes.

La figure ci-dessous donne un aperçu de la création d'un nouveau génome.

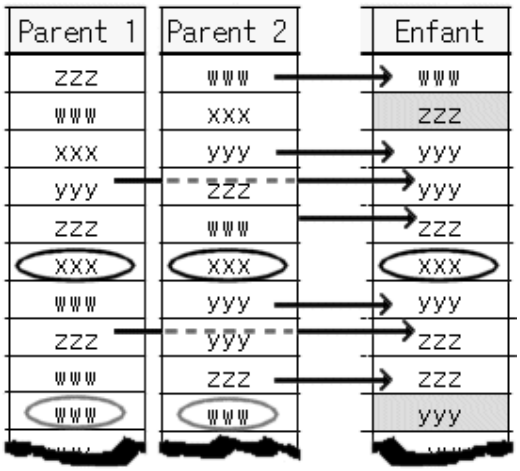

Figure 75 : création d'un nouveau génome

Sur cette figure, on voit que le premier parent a transmis deux identifiants de classes dans la partie visible du génome, et le deuxième parent en a transmis quatre. Deux cas de mutation sont indiqués par le fond grisé, où la valeur nouvelle n'est héritée d'aucun des deux parents. Les ellipses signalent les situations ou les valeurs des deux parents sont identiques.

Le nombre de classes (d'identifiants) est fixé au départ. La population initiale contient plusieurs génomes, constitués au hasard. À chaque génération, la performance de chaque génome (partition) est évaluée, les meilleurs sont conservés et croisés, les moins bons sont éliminés. Le nombre de génomes dans la population est constant.

Lors de la création d'un nouveau génome, on vérifie qu'il n'est pas, par hasard, déjà présent dans la population. Si c'est le cas, il est éliminé. Ainsi, la population est toujours constituée de génomes différents.

Remarquons que le croisement de deux gènes pourrait nécessiter une harmonisation préalable de leur partition respective par renommage des identifiants. En effet, les identifiants des classes sont arbitraires : l'identifiant d'une classe particulière pour l'un des génomes dénote une classe généralement totalement différente pour un autre génome.

Un renommage des classes entre deux partitions pourrait être fondé sur la formule suivante :

$$i = \frac{Card(\cup) - Card(\cap)}{Card(\cup)}$$

où **i** est l'indice de proximité entre deux sous-ensembles, $\cup$ l'union des sous-ensembles, $\cap$ leur intersection et **Card** le nombre d'éléments. Le renommage proprement dit consisterait alors à fournir le même identifiant pour les couples de classes les plus proches.

En pratique, nous avons constaté que les classes s'harmonisaient en peu de générations : au sein d'une population de génomes, les même identifiants se trouvent rapidement dénoter des classes proches, d'un génome à l'autre. D'autre part, cette fonction de renommage s'est avérée coûteuse. Nous l'avons implémentée puis abandonnée puisqu'elle était finalement inutile.

3.2.3.2.2 Expérimentations

Nous avons implémenté une version « à vide » de cet algorithme, sur des données factices et avec une fonction de performance simplissime.

L'idée est de prendre un ensemble d'objets pourvus chacun d'un numéro, c'est à dire un petit nombre entier compris entre **1** et **n**, **n** étant très inférieur au nombre d'éléments. Nous avons classé cet ensemble avec notre algorithme.

La fonction de performance d'une partition quelconque est la moyenne, sur l'ensemble entier, des écarts types des numéros des objets calculés au sein de chaque classe. Si les objets d'une classe ont tous le même numéro, l'écart type au sein de cette classe est nul. Si la moyenne des écarts types est nulle sur tout l'ensemble, c'est qu'ils sont tous nuls (puisqu'un écart type est non négatif). Plus la moyenne des écarts types est proche de zéro, plus la classification candidate est proche de la classification induite par les numéros caractérisant les objets. Remarquons que, si le nombre de classes fourni au départ à l'algorithme de classification est égal à **n**, la classification trouvée par l'algorithme est exactement celle qu'induisent les numéros portés par les objets. Dans cette simulation, il y a donc une solution optimale, unique, connue d'avance, au problème posé.

Nous avons testé de très nombreuses variations de cet algorithme. En particulier, nous avons apporté un soin particulier à régler le taux de mutation spontanée. Nos meilleurs résultats ont été obtenus avec un taux partiellement aléatoire dans une plage de variabilité dépendant elle-même à tout moment du taux moyen observé dans la sous-population la plus performante.

À titre d'exemple, nous reproduisons ci-dessous quelques courbes de convergence dans différents cas de figure. La fonction de performance utilisée est en fait la variance (carré de l'écart type) moyenne de la partition. On cherche donc à la minimiser. Ces courbes représentent l'évolution de la performance du meilleur génome dans la population, en fonction de la génération.

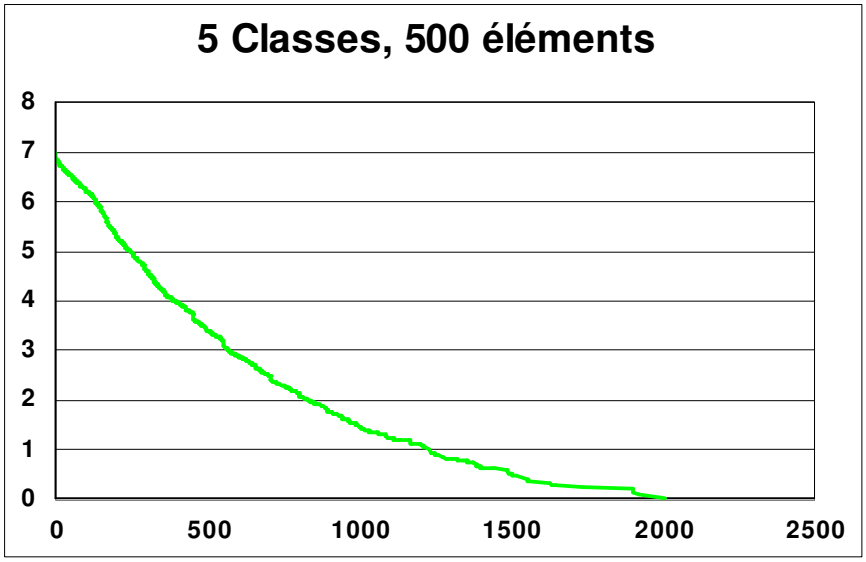

Figure 76 : partitionnement d'un ensemble de 500 éléments en 5 classes avec 40 génomes

Dans cette exécution du programme, la population contient 40 génomes, dont la moitié est renouvelée à chaque génération. Le nombre de générations nécessaires pour trouver la solution est de 2010 et la durée totale d'exécution est de 142 secondes.

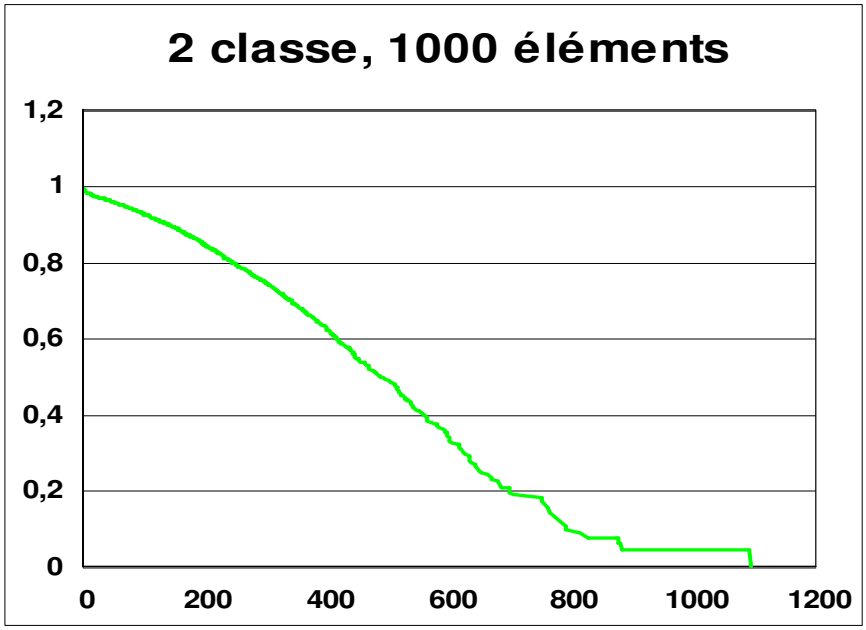

Figure 77 : partitionnement d'un ensemble de 1000 éléments en 2 classes avec 40 génomes

Dans cette exécution, on classe 1000 éléments en 2 classes seulement avec 40 génomes dans la population. Le temps de convergence est de 139 secondes et 1095 générations ont été calculées. On observe un large plateau juste avant la fin de l'exécution : le nombre de génomes n'étant pas très grand, la découverte de la solution repose largement sur les mutations en dernier recours.

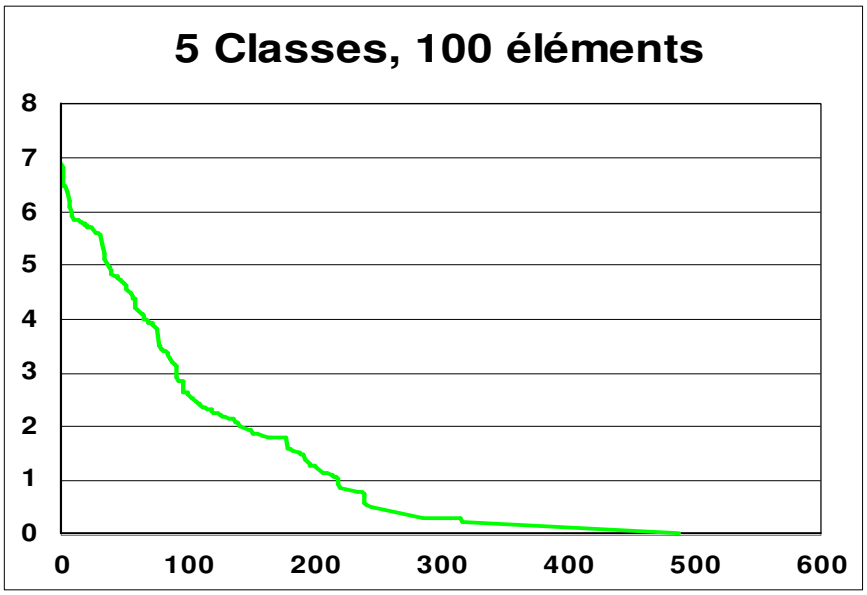

Figure 78 : partitionnement d'un ensemble de 100 éléments en 5 classes avec 40 génomes

Dans ce dernier exemple, il y a de nouveau 40 génomes dans la population. Le temps de convergence est très rapide par rapport aux exemples précédents : 18 secondes et 489 générations.

Ces courbes ne sont données qu'à titre indicatif. Nous avons observé une grande variabilité de la rapidité de convergence d'une exécution à l'autre, et ce d'autant plus que la population de génomes est réduite.

De plus, le nombre de générations par unité de temps dépend de la population. Le nombre de générations par seconde en fonction de la taille de la population est donné dans le graphique suivant. À titre d'exemple, nous avons utilisé les 1000 premières générations de la classification de 1000 éléments en 10 classes, pour des valeurs de population variant de 10 en 10, de 10 à 80 génomes.

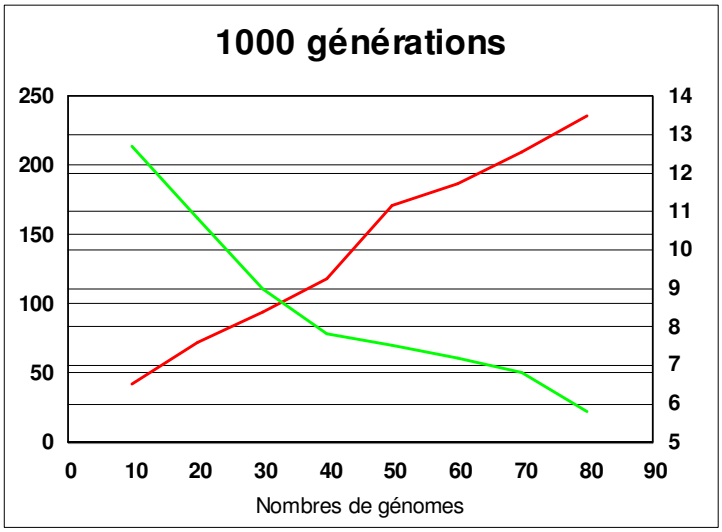

Figure 79 : Temps de calcul et performance pour 1000 générations, rapportés au nombre de génomes

La courbe croissante indique le temps de calcul, en secondes, sur l'échelle de gauche. On voit qu'il est sensiblement linéaire, les perturbations étant liées aux autres activités de l'unité centrale.

La courbe décroissante est la performance du meilleur génome, atteinte au bout de 1000 générations, à lire sur l'échelle de droite. L'évolution semble également linéaire, compte tenu de la grande variabilité de la convergence de l'algorithme.

On voit donc que l'on a un compromis à effectuer pour optimiser la rapidité de convergence : l'augmentation du nombre de génomes dans la population diminue le nombre de générations nécessaires pour converger mais augmente le temps de calcul.

3.2.3.2.3 Extrapolation

Pour tenter d'établir une loi empirique propre à extrapoler les valeurs nécessaires à la classification des éléments de notre corpus, nous avons mené une étude plus systématique sur la problématique de la classification en deux classes pour diverses valeurs du cardinal de l'ensemble à classer et de l'effectif de la population de génomes.

Pour chaque valeur, nous avons fait 8 mesures, afin d'établir une moyenne. Nous avons obtenu deux tableaux de valeurs : le premier pour le temps de convergence, le deuxième pour le nombre de générations. Dans ces tableaux, le cardinal de la population à classer est donné en tête de colonnes et l'effectif de la population de génomes est donné en tête de ligne.

pop\card	20	40	80	160	320	640	1280	2560
4	2,75	2,88	6,00	16,88	34,50	99,13	285,25	1047,88
8	1,00	1,86	4,38	8,75	20,13	62,43	190,13	704,63
16	1,00	1,50	2,38	6,88	15,63	50,50	177,63	695,88
24	0,75	1,75	2,75	5,75	14,25	53,50	222,88	727,13
32	0,75	1,88	2,88	6,25	16,50	57,88	207,13	907,63
64	0,86	2,00	4,00	7,38	18,50	67,25	231,63	1056,38

Tableau 14 : temps moyen pour classer un ensemble en deux classes

pop\card	20	40	80	160	320	640	1280	2560
4	269,38	234,75	505,50	1343,13	2446,13	5512,13	11573,00	27470,38
8	107,67	146,86	334,00	566,13	1099,75	2507,00	5005,25	10701,38
16	60,57	125,13	168,38	348,13	626,00	1293,75	2684,75	5552,63
24	65,25	93,25	161,25	249,88	425,00	998,50	2353,00	4112,75
32	74,88	109,63	141,13	238,50	405,25	851,38	1710,63	3908,38
64	35,71	87,63	132,88	172,63	265,00	533,75	992,75	2312,00

Tableau 15 : nombre moyen de générations pour classer un ensemble en deux classes

On constate que les valeurs concernant les petits ensembles (20 ou 40 éléments) sont assez erratiques. Cependant, le meilleur temps semble systématiquement s'établir pour une population d'une vingtaine de génomes.

Le deuxième tableau donne lieu aux courbes suivantes, tracées selon des échelles logarithmiques :

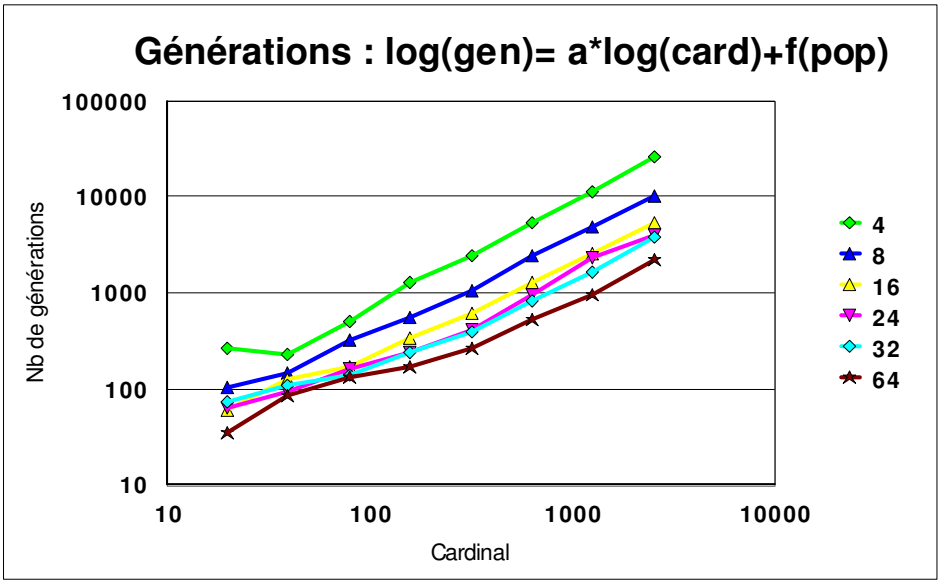

Figure 80 : nombre de générations en fonction de l'effectif de la population et du cardinal de l'ensemble

Si l'on modélise ces courbes avec une loi du type :

$$\log(gen) = \alpha \cdot \log(card) + f(pop)$$

on arrive à une valeur de α de 1,07563 donc assez proche de 1. En reportant la différence entre la partie affine de cette équation et les valeurs observées, on obtient la modélisation suivante de la fonction **f** :

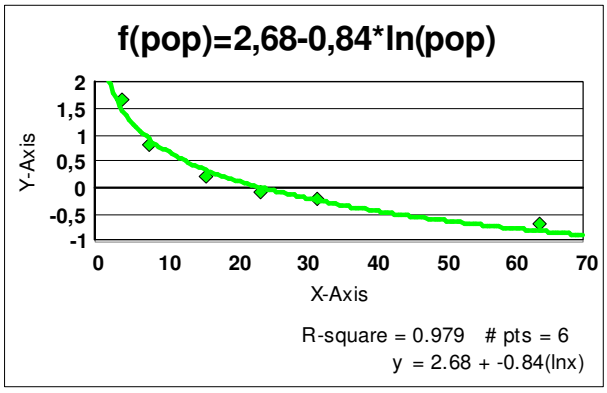

Figure 81 : détermination dela fonction f.

et finalement :

$$\log(gen) = 1{,}08 \cdot \log(card) - 0{,}84 \cdot \log(pop) + 2{,}68$$

Cette loi donne 115000 générations pour séparer en deux classes un ensembe de cardinal 50 000 avec une population de 24 génomes. Une classification plus élaborée, par dichotomies successives, pourrait être envisagée.

Les extrapolations en temps de calcul que nous avons obtenues ne nous semblent pas assez fiables pour en faire état dans ce travail. À l'évidence, il faudrait mener ces travaux sur une machine plus puissante.

Nous pensons qu'un algorithme de ce type peut faire émerger une répartition des amphigrammes en classes optimisant le pouvoir prédictif du modèle de langage, en particulier pour l'indice de sécabilité.

3.3 Quelques expérimentations

Nous allons présenter maintenant trois expérimentations que nous avons choisies parmi d'autres afin d'illustrer les développements que nous avons menés dans des domaines directement inspirés par le formalisme TransTree. Il s'agit d'une part d'exploiter la base de données des amphigrammes comme un dictionnaire bilingue de formes. Nous verrons ensuite les travaux effectués pour appréhender les systèmes d'écriture sans séparateur et finalement les essais de traduction que nous avons entrepris.

3.3.1 Systèmes d'écriture sans séparateurs

Notre méthode de segmentation repose très largement sur la notion de mot, chaîne de caractères comprise entre deux séparateurs. Nous avons exploré différentes méthodes pour nous affranchir de cette contrainte, tant pour être à même de travailler avec les systèmes d'écriture sans séparateur que pour essayer de dégager une éventuelle notion de pseudo morphèmes statistiques.

3.3.1.1 Travail de TER

Durant l'hiver 2004, Fabien Cromières (étudiant de 4° année) a fait un TER sur la possibilité de structurer un segment sans séparateur avec la méthode des arbres binaires. Il a utilisé un corpus bilingue anglais-japonais pour mener cette étude. L'analyse de la partie anglaise a été menée au niveau des caractères, sans utiliser la notion de mot typographique.

Ce faisant, il a étudié l'influence de la formule de calcul de la sécabilité sur l'analyse du segment en arbre binaire, notamment la taille de la fenêtre glissante à utiliser.

Nous lui avons suggéré d'étudier le nombre de sous-chaînes de caractères différentes isolées par les arbres binaires, dans l'ensemble du corpus, afin de déterminer une formule de calcul de la sécabilité qui minimise ce nombre. Cette idée est directement inspirée par la compression de données.

Il est arrivé à la conclusion que le calcul optimal des indices de sécabilité n'était pas le même dans le cas du japonais et de l'anglais.

Pour l'anglais, il semble qu'une formule de type :

$$\frac{N(a,b,c,d)^{\alpha}}{N(a,b) \cdot N(c,d)}$$

soit la plus efficace et pour le japonais :

$$\frac{N(a,b)^{\alpha}}{N(a) \cdot N(b)}$$

avec $\alpha \approx 1{,}2$ pour l'anglais et $\alpha \approx 1{,}9$ pour le japonais. Le gain en japonais par rapport à la valeur $\alpha \approx 1$ est de l'ordre de 10%, ce qui est tout à fait significatif.

Ces formules semblent en outre montrer que le rareté des données est plus pénalisante pour le japonais que pour l'anglais, ce qui n'est pas surprenant étant donné le beaucoup plus grand nombre de caractères différents en japonais.

Dans le but de contrecarrer la rareté des données, et de mieux tenir compte des hapax, F. Cromières a également testé le lissage de Good-Turing, qui consiste à substituer, au nombre réel d'occurrences d'un n-gramme quelconque, **Nb**, un terme **Nb'** dérivé de celui-ci :

$$Nb' = (Nb + 1) \cdot \frac{M_{Nb+1}}{M_{Nb}}$$

où M_{Nb} (respectivement M_{Nb+1})est le nombre de n-grammes apparus Nb fois (respectivement Nb + 1 fois) dans le corpus. Cependant, le gain, inférieur à 1%, ne semble pas significatif.

Il s'est en outre intéressé à la distribution des hapax parmi les n-grammes. Le tableau suivant récapitule les données quantitatives de son corpus d'environ 200 000 segments :

n	Anglais		Japonais	
	n-grammes différents	Hapax	n-grammes différents	Hapax
1	93	3	2858	247
2	2589	452	89460	30798
3	18885	5182	451169	234308
4	73967	24852	932373	580396
5	198454	74166		
6	431246	176947		

Tableau 16 : n-grammes et hapax en anglais-japonais

Pour étudier ce tableau, nous avons cherché d'éventuelles relations entre ces données. L'une d'elles nous a semblé intéressante.

La relation liant le logarithme du nombre de n-grammes différents et la proportion d'hapax parmi eux pour n donné est très sensiblement linéaire pour le texte anglais et présente une légère inflexion pour le texte japonais, comme on le voit sur les graphiques suivants :

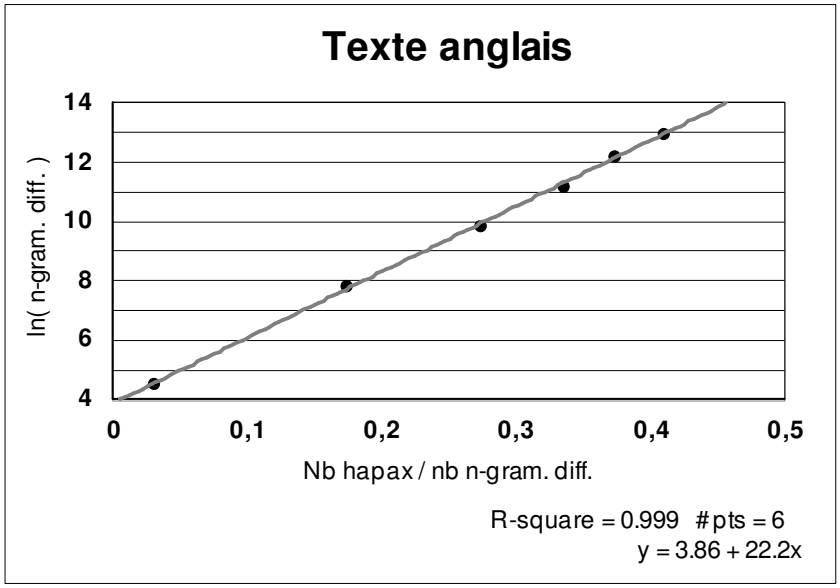

Figure 82 : relation entre longueur et nombre d'hapax en anglais

La relation linéaire est parfaitement nette entre les six types de n-grammes (n variant de 1 à 6) dans le corpus anglais. Elle est moins bien vérifiée entre les quatre types de n-grammes japonais décomptés, et la courbe présente une légère inflexion. Peut être faut-il y voir un effet de la différence profonde entre les caractères alphabétiques et le complexe système d'écriture japonais.

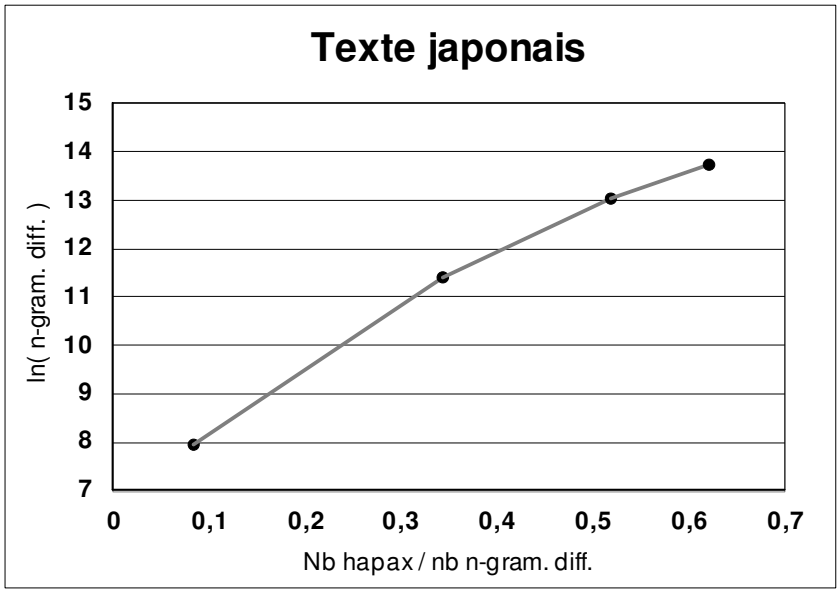

Figure 83 : relation entre nombre de hapax et de n-grammes différents en japonais

Cette relation concerne un seul et même corpus. Elle n'intègre donc pas la taille du texte. Nous ne l'avons pas vue décrite dans la littérature et ne savons pas si elle est liée à la loi de Zipf.

3.3.1.2 Langues orientales

Certaines langues orientales se caractérisent par leur système d'écriture sans séparateur entre les mots. C'est le cas pour le chinois, le japonais, le coréen et le thai, entre autres. Dans un corpus bilingue aligné, ces systèmes d'écriture ne nous permettent pas d'établir d'emblée des correspondances motivées entre sous-chaînes d'une langue à l'autre, comme on peut le faire entre les mots typographiques, grâce à l'hypothèse raisonnable d'équivalence sémantique.

Pour construire une correspondance dans notre formalisme TransTree, il est souhaitable de partir d'une segmentation en sous-chaînes *a priori*, afin de donner appui à une équivalence motivée entre sous-chaînes dans des deux langues. Quels moyens sont à notre disposition pour effectuer ce découpage ?

Les travaux menés par Fabien Cromières nous donnent quelques éléments de réponse. En partant d'une définition optimisée de la sécabilité, c'est-à-dire une définition qui, de manière empirique, conduit à un nombre minimal de sous-arbres différents on distingue, dans un segment donné, un nombre de sous-arbres égal au nombre de caractères du segment, moins un. Ce nombre est relativement réduit, et nous permet d'envisager de fabriquer un dictionnaire bilingue par voie statistique, tout comme nous l'avons fait en connaissant des mots typographiques *a priori*. Un algorithme multipasse fondé sur cette approche devrait permettre d'établir des correspondances pertinentes entre les deux segments.

3.3.1.3 Travailler sur les morphèmes

Si l'on envisage, poussé par l'étude des systèmes d'écriture sans séparateur, de travailler directement au niveau des caractères, la tentation est grande d'appliquer le même principe aux langues alphabétiques. Comme l'a démontré Fabien Cromières, il serait alors nécessaire d'utiliser une fenêtre plus large pour calculer la sécabilité. Les unités dégagées, par comparaison entre les segments des deux langues, s'apparenteraient plus vraisemblablement à des morphèmes qu'à des mots typographiques. Ce pourrait être une direction intéressante pour résoudre les difficultés liées aux langues fortement agglutinantes comme le hongrois, ou dans lesquelles la formation de mots composés est très peu contrainte, comme l'allemand.

3.3.2 Vers un système de TA

L'un des buts de cette thèse consiste à présenter au traducteur des segments cible chimériques, reconstitués avec les amphigrammes génériques renconus dans le segment à traduire. Les méthodes mises en œuvre et des résultats intermédiaires en l'état, tels qu'obtenus à la fin de cette thèse sont ici présentés.

3.3.2.1 Principe : utilisation de l'algorithme de Viterbi

L'idée principale qui sous-tend le système de traduction automatique que nous proposons est d'utiliser l'algorithme de Viterbi pour établir un arbre de dérivation avec la « grammaire » constituée par les amphigrammes génériques contenus dans la base de connaissance.

Cela est possible parce que nous pouvons calculer à tout moment un arbre binaire de sécabilité de notre segment de départ. De fait, cet arbre étant calculé, il nous reste à y reconnaître des sous-arbres dont l'armature textuelle corresponde à des amphigrammes génériques, et éventuellement, s'il y en a plusieurs, à choisir le plus probable en fonction du contexte dans lequel il s'inscrit. C'est cette détermination en fonction du contexte qui fait intervenir l'algorithme de Viterbi. Celui-ci est donc utilisé comme *tagger*, ou étiqueteur : il permet de déterminer la classe la plus probable d'un élément (mot typographique ou sous-arbre) en fonction du contexte d'apparition.

Pour ce faire, nous disposons à différents niveaux d'analyse du corpus des biclasses et triclasses observés dans le corpus. Ces données nous permettent de pondérer statistiquement les transitions entre deux classes le long de l'énoncé.

3.3.2.2 Expérimentations effectuées.

En pratique nous n'avons pas utilisé de classes pour faire les expérimentations dans ce domaine. Nous avons effectué un étiquetage en digrammes directement, à partir de la suite de mots typographiques source.

À titre indicatif, nous avons cherché à réétablir les digrammes du corpus d'apprentissage par cette méthode. Nous n'avons pas véritablement établi de protocole d'expérimentation : il s'agit juste d'un test. L'avantage, bien entendu, est que nous connaissons a priori une suite de digrammes correcte pour chaque segment du corpus. Les tests que nous avons menés nous ont permis de rétablir la suite de digrammes correcte pour 86 % des 5786 segments testés.

La suite de digrammes étant obtenue, nous avons reconstruit des arbres binaires de sécabilité, afin d'effectuer une analyse par niveau. Le premier niveau est défini sans ambiguïté car les sous-arbres binaires considérés contiennent au plus un vrai digramme. L'analyse consiste donc à retrouver les amphigrammes correspondant à ces sous-arbres binaires, éventuellement à choisir entre plusieurs amphigrammes de même armature textuelle source. Là aussi nous avons utilisé l'algorithme de Viterbi comme étiqueteur.

Pour les niveaux suivants, nous avons utilisé une version plus complète de l'algorithme de Viterbi pour établir le chemin de coupure de l'arbre binaire. Autrement dit tous les chemins d'états à envisager n'ont pas le même nombre de transitions, selon le parcours de la coupure dans l'arbre binaire.

Nous avons développé plusieurs programmes sur ces principes. On en trouvera un exemple dans l'annexe 2 (**trad.pl**) qui nous a permis de retraduire correctement environ deux tiers des segments d'un échantillon extrait du corpus de départ. Un aperçu des résultats est fourni dans l'annexe 5.

3.3.2.3 Présentation d'un exemple

Cet exemple est tiré de l'annexe. Nous l'utilisons ici pour illustrer le format de sortie des fichiers que nous avons obtenus.

```
Large assembly and bills of material management
la gestion des nomenclatures et des assemblages à grande échelle
```

Ces deux premières lignes reproduisent un bisegment de la mémoire. Nous retraduisons le segment source vers le segment cible.

```
[0](189635,15) :
N0#00001+X0#00045+S0#00046+X0#00031+X0#00047+X0#00048+S0#00049+X0#0004a+N0#00002
```

Cette ligne indique le résultat au niveau « zéro ». Un premier étiquetage a été effectué : les mots typographiques source sont remplacés par des identifiants de digrammes. Les valeurs numériques données en début de ligne indiquent le score de la solution proposée (la meilleure) ainsi que le nombre de solutions explorées. Les identifiants **N0#00001** et **N0#00002** dénotent les extrémités droite et gauche : on retrouvera leurs équivalents aux niveaux supérieurs.

```
[1] Arbre :
(((X0#00045,S0#00046),(X0#00031,((X0#00047,X0#00048),S0#00049))),X0#0004a)
[1] Positions : 0+2+3+4+6+3:4+2:3:4+0:2:3:4+0:2:3:4:6
[1] Découpage : (X0#00045,S0#00046)+X0#00031+X0#00047+X0#00048+S0#00049+X0#0004a
[1](8,13) : N1#00001+X1#138c6+X1#138b6+X1#138c7+X1#138c8+S1#138ca+X1#138c9+N1#00002
```

Ces quatre lignes concernent le niveau « un ». Tout d'abord, les indices de sécabilité ont conduit au parenthésage indiqué. Les noeuds saturés sont ensuite donnés par le deuxième ligne : chaque nœud est repéré par la chaîne des numéros de feuilles bilingues (commençant par 0 et séparés par des ':'). On a 9 nœuds saturés dans l'arbre, séparés par des '+'. Les deux lignes suivantes indiquent d'une part les regroupements effectués et d'autre part les identifiants des objets retenus à la fin du traitement de niveau « un ».

.

```
[2] Arbre : ((X1#138c6,(X1#138b6,((X1#138c7,X1#138c8),S1#138ca))),X1#138c9)
[2] Positions : 2:3+1:2:3+0:1:2:3+0:1:2:3:5+1+2+3+0+5
[2] Découpage : ((X1#138c6,(X1#138b6,((X1#138c7,X1#138c8),S1#138ca))),X1#138c9)
[2](3,3) : N2#00001+X2#1894b+N2#00002
[2] : N2#00001+X2#1894b+N2#00002
```

Ces quatre lignes concernent le niveau « deux ». Leur lecture est identique à celle que nous avons donnée pour le niveau « un ».On remarque qu'à la fin du niveau deux, il n'y a plus qu'un identifiant les symboles **N2#00001** et **N2#00002** constituant les extrémités de l'énoncé.

```
OK=> la gestion des nomenclatures et des assemblages à grande échelle
```

Ici, nous avons reconstitué l'énoncé cible de l'objet **X2#1894b** . Cette étape de synthèse n'est pas vraiment détaillé : il s'agit de l'assemblage des énoncés cible des objets déterminés lors de l'analyse. L'énoncé résultant est labellé « OK » pour indiquer qu'il correspond effectivement à ce qui était enregistré dans la mémoire. On pourra constater dans les annexes que ce n'est pas toujours le cas.

Cette expérience de « retraduction » montre essentiellement que l'ont peut réétiqueter les mots typographiques avec des identifiants de digrammes avec suffisamment de fiabilité pour que l'architecture des arbres binaires de sécabilité puisse être réétablie petit à petit.

L'expérience qu'il reste à conduire utiliserait des amphigrammes génériques afin d'extrapoler les observations faites en corpus à de nouveaux énoncés.

Conclusion

Notre objectif était de contribuer à la réutilisation automatique de la connaissance mise en œuvre au cours du processus de traduction humaine. Cet objectif est déjà partiellement réalisé dans les systèmes d'aide à la traduction fondés sur les mémoires de traduction. Notre travail a été constament guidé par la conviction que de nombreux gisements d'information traductionnelle restent encore largement inexploités au sein des corpus multilingues. Ces corpus étant de plus en plus abondants, nous avons cherché à améliorer leur réutilisabilité.

Dans ce but, nous avons développé une modélisation de correspondances traductionnelles sous-phrastiques associée à un formalisme XML propre à l'exprimer. L'ensemble forme le modèle TransTree. Parallèlement nous avons imaginé des méthodes automatisées de repérage et d'apprentissage de ces correspondances.

Notre démarche consiste à construire des arbres binaires sur des segments de texte au moyen de l'indice de sécabilité et à confronter ces arbres entre énoncés synonymes exprimés en plusieurs langues. Cette confrontation débouche sur la construction d'amphigrammes ou briques traductionnelles gigognes. Par ailleurs, nous explorons la possiblité de factoriser cette connaissance afin de l'extrapoler à des situations nouvelles.

Grâce au modèle TransTree et aux méthodes d'acquisition de connaissances associées, nous proposons une présentation originale des correspondances sous-phrastiques au sein d'un bisegment de manière immédiatement accessible à l'utilisateur. Cette visualisation dynamique améliore l'ergonomie du poste de travail du traducteur et peut contribuer à l'enseignement des langues étrangères. En outre, TransTree permet d'ouvrir de nouvelles voies d'investigation dans le domaine de la traduction automatique.

Plusieurs directions d'étude sont ouvertes à l'issue de ce travail. Notons en particulier l'adaptation des méthodes proposées aux systèmes d'écriture sans séparateurs propres à certaines langues asiatiques. À la fin de la rédaction de ce mémoire, cette étude est en cours. Une perspective plus large consiste bien entendu à réaliser un système complet de traduction automatique « approchée » au sein d'un système d'aide à la traduction humaine.

Au delà, l'élargissement des formalismes et méthodes décrits dans ce travail à la situation multilingue permet d'envisager l'analyse d'un texte écrit en plusieurs langues et l'exploitation simultanée d'informations complémentaires en vue de la traduction automatique de haut niveau dans une langue tierce. Cette direction d'étude nous paraît particulièrement prometteuse.

Glossaire

Les termes apparaissant dans ce glossaire sont marqués d'une astérisque (*) lors de leur première apparition dans le texte, éventuellement lors d'une réapparition, ainsi que dans les articles de ce glossaire.

Amphigramme : composant essentiel du formalisme arborescent TransTree*. Un amphigramme est en général un nœud dans cette arborescence et, par extension, le sous-arbre dominé par ce nœud. Un amphigramme comprend, directement (amphigramme atomique*, complexe* et générique*) ou indirectement (amphigramme compact*), deux éléments textuels* (éventuellement plus) qui forment ensemble l'armature textuelle* de l'amphigramme. Un amphigramme complexe ou compact contient d'autres amphigrammes.

Amphigramme atomique : amphigramme ne contenant ni amphigramme fils, ni point d'insertion* dans ses éléments textuels*. Un amphigramme atomique met en relations des chaînes de caractères liées sémantiquement dans deux (ou plus) langues différentes. Les amphigrammes atomiques sont susceptibles d'être regroupés en ensembles.

Amphigramme compact : amphigramme faisant référence à un amphigramme générique*, et ne contenant pas, de ce fait, d'élément textuel*. Les amphigrammes fils d'un amphigramme compact doivent vérifier les contraintes d'ensembles* imposées par l'amphigramme générique référencé. Un amphigramme compact est formellement équivalent à un amphigramme complexe*.

Amphigramme complexe : amphigramme qui encode les correspondances dans un bisegment* (ou en général un multisegment). Il est constitué de deux (ou plus) éléments textuels comprenant des points d'insertion* où peuvent se placer les d'amphigrammes fils de l'amphigramme complexe dans l'arborescence TransTree.

Amphigramme générique : amphigramme encodant un schéma traductionnel*. Il est composé d'élément textuels* dont les points d'insertion* font référence à des ensembles d'amphigrammes génériques* ou atomiques*. Un amphigramme générique appartient lui-même à un ensemble et ne contient pas d'amphigramme fils.

Armature textuelle : ensemble des deux (ou plus) éléments textuels d'un amphigramme, y compris leurs éventuels points d'insertion*. Dans un amphigramme complexe*, l'armature textuelle ne comprend pas les éléments textuels* des amphigrammes fils.

Arbre binaire de sécabilité : arbre construit sur un segment à partir des indices de sécabilité. Il s'agit bien d'un arbre binaire, dont les nœuds internes dominent deux autres nœuds ou éléments terminaux.

Base de connaissances : ensemble des connaissances extraites du corpus bilingue aligné. Il s'agit des digrammes*, des amphigrammes* et des ensembles d'amphigrammes*. Les suites

finies de deux ou trois objets de ces types, les n-grammes* constituent des modèles de langages et appartiennent aussi à la base de connaissances.

Bigramme, trigramme, n-grammes : suite de deux, trois ou n symboles, extraite du corpus dans l'ordre de surface pour une langue donnée. Selon la granularité* utilisée dans un traitement donné, les symboles que nous utilisons sont des mots typographiques*, des digrammes* amphigrammes* ou des ensembles d'amphigrammes.

Bisegment : ensemble de deux segments* sémantiquement équivalents exprimés dans deux langues différentes. S'il y a plus de deux langues, on parle de multisegment.

Classe : classe d'équivalence dans l'ensemble des amphigrammes, issue d'une relation d'équivalence établie par un processus classification. Les classes établies dans ce travail incarnent les ensembles d'amphigrammes génériques* et atomiques* utilisés dans les amphigrammes génériques*.

Congruence, sous-arbres congruents : Deux nœuds, dans les arbres binaires de chaque segment d'un bisegment, sont congruents s'ils dominent les mêmes occurrences de vrais digrammes*.

Contrainte d'ensemble : contrainte imposée sur les amphigrammes pouvant occuper les points d'insertion* des éléments textuels d'un amphigramme générique*. Elle est matérialisée dans la représentation XML par le jeu des attributs **set** des points d'insertion* de l'amphigramme générique. Cette contrainte exprime que seuls les amphigrammes appartenant aux ensembles indiqués sont susceptibles d'occuper les points d'insertion de l'amphigramme générique.

Correspondance gigogne : mise en relation de parties sémantiquement liées dans un bisegment ou un multisegment, contenant des sous-parties elles-mêmes mises en relation. Les correspondances gigognes sont incarnées par les amphigrammes* dans TransTree*.

Digramme : dans un contexte bilingue, couple de mots typographiques* liés statistiquement, (vrai digramme) ou éventuellement un mot typographique seul (faux digramme). Dans le cadre de ce travail, les vrais digrammes matérialisent amphigrammes atomiques* du formalisme TransTree*.

Élément textuel : chaîne de caractères dans un amphigramme* constitué par les sous-segments mis en correspondance et les éventuels points d'insertion* pour une langue donnée. L'ensemble des éléments textuels d'un amphigramme constitue son armature textuelle*.

Ensemble d'amphigrammes : Dans notre contexte, il s'agit toujours d'ensembles d'amphigrammes atomiques* ou génériques*. Les ensembles d'amphigrammes sont utilisés par les amphigrammes génériques*.

Étiquette d'ensemble : les ensembles d'amphigrammes utilisés dans TransTree sont toujours munis d'une étiquette. Dans la représentation XML de TransTree, ces étiquettes sont

les valeurs de l'attribut **set**, qu'il soit utilisé dans un élément **<amphigram>** (spécificateur d'ensemble) ou dans une monobalise d'insertion **<a/>** (indicateur d'ensemble).

Faux digramme : digramme* dont un des deux constituants est le symbole **#null#**.

Granularité : type et taille des symboles les plus petits utilisés dans un traitement informatique linguistique. La granularité est semblable à la « définition » d'une image.

Indicateur d'ensemble : indice utilisé dans un point d'insertion pour préciser les ensembles d'amphigrammes susceptible d'être insérés à cet endroit. Dans la représentation XML, il s'agit de l'attribut **set** utilisé dans la monobalise d'insertion **<a/>**. Cet attribut est possiblement multivalué. Ses valeurs sont des étiquettes d'ensembles*.

Indicateur-repère local : indice utilisé dans un point d'insertion* pour établir les correspondances entre éléments textuels. Il est également utilisé par l'amphigramme concerné par ce point d'insertion. L'indicateur-repère local est matérialisé par l'attribut XML **loc**.

Indice de congruence : nombre de nœuds congruents dans les arbres binaires de sécabilité d'un bisegment.

Indice de sécabilité : tout indice permettant de d'ordonner les séparateur d'un segment en fonction de la probabilité de la coocurrence des parties droite et gauche à chaque séparateur. La sécabilité est le rang du séparateur selon cet ordre.

Information mutuelle : quantité d'information que la connaissance d'une variable aléatoire apporte sur une autre. L'information mutuelle s'écrit :

$$I(x,y) = \log\left(\frac{Pr(x,y)}{Pr(x) \bullet Pr(y)}\right)$$

où Pr(•) désigne la probabilité d'une variable aléatoire.

Modèle de langage : ensemble de bigrammes et de trigrammes, de n-grammes en général. Le modèle de langage permet un calcul de dépendances entre les symboles d'un segment.

Mot typographique : dans un texte, chaîne de caractères comprise entre deux blancs ou séparateurs, ou délimitée par un signe de ponctuation. La notion de mot typographique n'est pas pertinente pour les langues à système d'écriture sans séparateur.

Point d'insertion : Dans un élément textuel d'un amphigramme non atomique, point devant être occupé par l'élément textuel de la même langue d'un autre amphigramme. Il porte obligatoirement un indicateur-repère local* et, dans le cas d'un amphigramme générique*, un pointeur vers un spécificateur d'ensemble*. Le point d'insertion est matérialisé par la monobalise d'insertion **<a/>** dans la représentation XML de TransTree.

Saturation, nœuds saturés : dans l'arbre binaire de sécabilité d'un segment, un nœud est saturé si son père (le nœud le dominant immédiatement) domine strictement plus de vrais digrammes* que lui-même. La racine d'un arbre est toujours saturée. Les faux digrammes* ne sont jamais saturés.

Sécabilité : numéro d'ordre, dans un segment*, porté par une frontière entre deux symboles, indiquant la possibilité relative (par rapport aux autres frontières) de couper le segment en deux morceaux à cette frontière-là.

Schéma traductionnel : armature textuelle* d'un amphigramme générique*. Un schéma traductionnel est caractérisé par les contraintes d'ensemble* portées par les points d'insertion*.

Segment : Une unité traduite dans une mémoire de traduction. Il peut s'agir d'une phrase ou d'un titre, d'un légende, d'une entrée de glossaire etc.

Spécificateur d'ensemble : marque portée par un amphigramme générique* ou atomique* pour spécifier à quel(s) ensemble(s) d'amphigrammes il appartient. Dans l'expression XML de TransTree, il est concrétisé par l'attribut **set** de l'élément **<amphigram>**. Cet attribut est possiblement multivalué et ses valeurs sont des étiquettes d'ensemble*.

Table de marquage : fichier contenant la description de la mise en page d'un document et permettant de découper ce dernier en segments*.

TransTree : formalisme proposé et utilisé dans ce travail pour décrire les correspondances gigognes* dans un bisegment*, voire un multisegment.

Vrai digramme : digramme* dont aucun des deux constituants n'est le symbole **#null#**.

Bibliographie

[1] V. Berment, "Méthodes pour informatiser les langues et les groupes de langues « peu dotées »," in *GETA CLIPS IMAG*. Grenoble, France: Université Joseph Fourier, 2004.

[2] E. Planas, "TELA : Structure et algorithmes pour la traduction fondée sur la mémoire," in *Clips-Imag*. Grenoble, France, 1998.

[3] J. Lacouture, *Champollion, une vie de lumières*. Paris: Grasset, 1988.

[4] E. Gaussier, "Flow Network Models for Word Alignment and Terminology Extraction from Bilingual Corpora," presented at ACL 1998, 1998.

[5] P. Langlais, M. Simard, J. Véronis, S. Armstrong, P. Bonhomme, F. Débili, P. Isabelle, E. Souissi, and P. Théron, "ARCADE : A cooperative research project on bilingual text alignment," presented at LREC, Grenade, Espagne, 1998.

[6] I. D. Melamed, "A Word-to-Word Model of Translational Equivalence," presented at Association for Computational Linguistics (ACL'97), Madrid, Spain, 1997.

[7] O. Kraif, "Constitution et exploitation de bi-textes pour l'aide à la Traduction," Université de Nice Sophia Antipolis, 2001.

[8] J. Nerbonne and J. Véronis, "Parallel Texts in Computer-assisted Language Learning," in *Parallel Text Processing*, K. A. P. Dordrecht, Ed., 2000, pp. 354 - 369.

[9] A. Berger, P. Brown, S. D. Pietra, V. D. Pietra, J. Lafferty, H. Printz, and L. Ures, "The candide system for machine translation," presented at ARPA Conference on Human Language Technology, Plainsboro, New Jersey, États Unis, 1994.

[10] P. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer, "The Mathematics of Statistical Machine Translation: Parameter Estimation," *Computational Linguistics*, vol. 19, 1993.

[11] F. J. Och and H. Ney., "Improved Statistical Alignment Models," presented at ACL, Hongkong, China, 2000.

[12] M. Franz, J. S. McCarley, and S. Roukos, "Ad hoc and Multilingual Information Retrieval at IBM," in *Text REtrieval Conference*, 1998, pp. 104-115.

[13] S. D. Richardson, W. B. Dolan, A. Menezes, and J. Pinkham, "Achieving commercial-quality translation with example-based methods," Microsoft Research, Redmond, Washington, Etats Unis 2001.

[14] A. Menezes and S. Richardson, "A best-first alignment algorithm for automatic extraction of transfer mappings from bilingual corpora," presented at MT Summit VIII, Santiago De Compostela, Spain, 2001.

[15] M. H. Al-Adhaileh, "Synchronous Structured String-Tree Correspondence (S-SSTC) and its applications for machine translation.," UTMK, Universiti Sains Malaysia, 2003, pp. 163.

[16] S. Chappuy, "Formalisation de la description des niveaux d'interprétation des langues naturelles.." Grenoble: UJF, 1983.

[17] Y. F. Yan, "Vers une ingénierie de la production de linguiciels. Spécification et réalisation d'un prototype de poste de travail linguistique pour la spécification de correspondances structurales." Grenoble: UJF, 1987.

[18] Y. Zaharin, "Strategies and heuristics in the analysis of a natural language in Machine Translation.." Penang: Universiti Sains Malaysia, 1986.

[19] R. D. Brown, "Automated Generalization of Translation Examples," presented at COLING 2000, 2000.

[20] R. D. Brown, "Transfer-Rule Induction for Example-Based Translation," presented at Machine Translation Summit VIII, Santiago de Compostella, 2001.

[21] Y. Lepage, "De l'analogie rendant compte de la commutation en linguistique," in *laboratoire CLIPS-IMAG*. Grenoble: Université Joseph Fourier, 2003.
[22] Y. Lepage, "Translation of sentences by analogy principle," presented at Language and technology, 2005.
[23] M. Simard and P. Langlais, "Sub-sentential Exploitation of Translation Memories," presented at Machine Translation Summit VIII, Santiago de Compostela, 2001.
[24] T. Watanabe, E. Sumita, and H. G. Okuno, "Chunk-Based Statistical Translation," presented at ACL, Sapporo, Japon, 2003.
[25] D. Wu, "Stochastic inversion transduction grammars and bilingual parsing of parallel corpora," *Computational Linguistics*, vol. 23, 1997.
[26] E. Planas, "Extending Translation Memories," NTT Cyber Solutions Laboratories, Japan 2000.
[27] E. Planas and O. Furuse, "Formalizing Translation Memories," presented at Machine Translation Summit VII, Singapore, 1999.
[28] F.-J. Och and H. Ney, "Statistical Multi-Source Translation," presented at MT Summit 2001, Santiago de Compostela, Spain, 2001.
[29] D. M. Magerman and M. P. Marcus, "Parsing a Natural Language Using Mutual Information Statistics," presented at National Conference on Artificial Intelligence, 1990.
[30] F. Cromières, "Recherche et exploitation d'alignements sous-phrastiques hiérarchiques," in *GETA CLIPS*. Grenoble: UJF, 2005.
[31] F. Cromières, "Étude d'arbres binaires statistiques pour l'analyse bilingue," in *GETA CLIPS*. Grenoble: UJF, 2004.
[32] C. Chenon, "Dynamical visualization of nested correspondences," presented at ASLIB, Londres, Royaume Uni, 2004.
[33] N. Chater and P. Vitanyi, "The generalized universal law of generalization," *Journal of Mathematical Psychology*, vol. 47, pp. 346-369, 2003.
[34] N. Chater and P. Vitanyi, "Simplicity: A unifying principle in cognitive science?," *Trends in Cognitive Sciences*, vol. 7, pp. 19-22, 2003.
[35] E. Charniak, "Statistical Techniques for Natural Language Parsing," in *AI Magazine*, vol. 18, 1997, pp. 33-35.
[36] M. Carl, "Inducing Translation Grammars From Bracketed Alignments," presented at Machine Translation Summit VIII, Santiago de Compostella, 2001.
[37] P. Brown, V. J. D. Pietra, P. V. d. Souza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language.," in *Comp. Linguistics*, vol. 18(4), 1992, pp. 467-479.
[38] C. Boitet, "Four technical and organizational keys for handling more languages and improving quality (on demand) in MT," presented at MTS2001 Workshop on MT2010 - Towards a Road Map for MT, Santiago de Compostela, 2001.
[39] V. Berment, "Several directions for minority languages computerization," Geta-Clips 2003.
[40] A. Berger, "Statistical machine learning for information retrieval," in *School of Computer Science*: CMU, 2001.
[41] A. Berger, S. D. Pietra, and V. D. Pietra, "A Maximum Entropy Approach to Natural Language Processing," in *Computational Linguistics*, vol. 22, 1996.
[42] D. Benedetto, E. Caglioti, and V. Loreto, "Language Tree and Zipping," in *Physical Review Letters*, vol. 88, 2002.
[43] A.-B. Al-Assimi and C. Boitet, "Management of Non-Centralized Evolution of Parallel Multilingual Documents.," presented at Proc. Internationalization Track, 10th International World Wide Web Conference, Hong Kong, 2001.

[44] A.-B. Al-Assimi, "Gestion de l'évolution non centralisée de documents parallèles multilingues.," in *Nouvelle thèse*. Grenoble: UJF, 2000.
[45] M. H. Al-Adhaileh and T. E. Kong, "Synchronous Structured String-Tree Correspondence (S-SSTC)," presented at IASTED02, Innsbruck, Austria, 2002.
[46] M. H. Al-Adhaileh and T. E. Kong, "Example-Based Machine Translation Based on the Synchronous SSTC Annotation Schema," presented at Machine Translation Summit VII '99, Singapore, 1999.
[47] M. H. Al-Adhaileh and T. E. Kong, "A Flexible Example-Based Parser Based on the SSTC," presented at 17th International Conference on Computational Linguistics (COLING'98), Montreal, Canada, 1998.
[48] T. Dunning, "Accurate Methods for the Statistics of Surprise and Coincidence," in *Computational Linguistics*, vol. 19, 1994, pp. 71-74.
[49] G. Foster, "Evaluating a Maximum Entropy Model," RALI - Université de Montréal 1999.
[50] G. Foster, *Toward An Empirical Evaluation of Maximum Entropy for Translation Modeling*. RALI - Université de Montréal, 1999.
[51] G. Foster, "A Maximum Entropy / Minimum Divergence Translation Model," RALI - Université de Montréal 2000.
[52] D. M. Goblirsch, "Viterbi Beam Search with Layered Bigrams," presented at ICSLP '96, 1996.
[53] J. T. Goodman, "Exponential Priors for Maximum Entropy Models,," Microsoft, Technical Report, 2001.
[54] J. T. Goodman, "A Bit of Progress in Language Modeling Extended Version," Microsoft, Technical Report 2001.
[55] F. Gow, "Metrics for Evaluating Translation Memory Software," in *School of Translation and Interpretation*: University of Ottawa, 2003.
[56] N. Hajlaoui and C. Boitet, "A pivot XML-based architecture for handling multilingual, multiversion documents : parallel monolingual documents aligned through a central correspondence descriptor," presented at International Conference on the Convergence of Knowledge, Culture, Language and Information Technologies, Alexandrian, Egypt, 2003.
[57] K. Harbusch and P. Poller, "Structural Translation with Synchronous Tree Adjoining Gramars in VERBMOBIL," Verbmobil - DFKI - Universität Koblenz-Landau 1996.
[58] S. Kahane, "Extraction dans une grammaire de dépendance lexicalisée à bulles," presented at TAL, 2000.
[59] S. D. Kamvar, D. Klein, and C. D. Manning, "Spectral Learning," presented at International Joint Conference of Artificial Intelligence, 2003.
[60] A. Kumano and H. Hirakawa, "Building an MT Dictionary from Parallel Texts Based on Linguistic and Statistical Information," presented at COLINC, 1994.
[61] P. Langlais and G. Foster, "Using Context-Dependent Interpolation to Combine Statistical Language and Translation Models for Interactive Machine Translation," presented at Content-Based Multimedia Information Access (RIAO), Paris, France, 2000.
[62] P. Langlais and M. Simard, "De la traduction probabiliste aux mémoires de traduction (ou l'inverse)," presented at TALN 2003, Batz-sur-Mer, 2003.
[63] S. Lappin and M. McCord, "Anaphora Resolution in Slot Grammar," in *Computational Linguistics*, vol. 16: IBM T.J.Watson Research Center, 1990.
[64] F.-H. Liu, Y. Gao, L. Gu, and M. Picheny, "Noise Robustness in Speech to Speech Translation," presented at Eurospeech, Geneva, Switzerland, 2003.

[65] S. Martin, J. Liermann, and H. Ney, "Algorithms for Bigram and Trigram Word Clustering.," presented at EUROSPEECH-95, Madrid, 1995.
[66] M. McCord, "Using Slots and Modifiers in Logic Grammars for Natural Language," in *Artificial Intelligence*, vol. 18, 1982, pp. 327-367.
[67] D. Munteanu and D. Marcu, "Processing Comparable Corpora with Bilingual Suffix Trees," Information Sciences Institute, University of Southern California 2002.
[68] D. Nogueira, "Translation Tools Today: A Personal View," in *Translation Aid Software*, vol. 6, 2002.
[69] F.-J. Och and H. Ney, "A Comparison of Alignment Models for Statistical Machine Translation," presented at Coling 2000, Saarbrucken, Germany, 2000.
[70] F.-J. Och, "Minimum Error Rate Training in Statistical Machine Translation," presented at ACL, Sapporo, Japan, 2003.
[71] K. A. Papineni, S. Roukos, T. Ward, and W. J. Zhu, "Bleu: a method for automatic evaluation of machine translation," IBM Research Division, Thomas J. Watson Research Center, Technical Report 2001.
[72] E. Planas, "A Case Study on Memory Based Machine Translation Tools," 2000.
[73] E. Planas and O. Furuse, "Multi-level Similar Segment Matching Algorithm for Translation Memories and Example-Based Machine Translation," presented at Coling 2000, 2000.
[74] S. Ranka and S. Sahni, "String editing on an SIMD hypercube multicomputer," *Journal of Parallel and Distributed Computing*, vol. 9, pp. 411 - 418, 1990.
[75] A. Ratnaparkhi, S. Roukos, and R. T. Ward, "A Maximum Entropy Model For Parsing.," presented at International Conference on Spoken Language Processing, Yokohama, 1994.
[76] A. Ratnaparkhi, "Trainable methods for surface natural language generation," presented at First North American ACL, Seattle, USA, 2000.
[77] S. D. Richardson, W. B. Dolan, A. Menezes, and J. Pinkham, "Achieving commercial-quality translation with example-based methods," presented at MT Summit VIII, Santiago de Compostella, Spain, 2001.
[78] R. Rosenfield, "Two Decades of Statistical Language Modeling: Where do we go from here?," presented at IEEE, 2000.
[79] R. Schäler, "Beyond Translation Memories," presented at MT Summit VIII, Spain, 2001.
[80] S. Sekine and R. Grishman, "A Corpus-based Probabilistic Grammar with Only Two Non-terminals," presented at Fourth IWPT, Prague, Czech Republic, 1995.
[81] W. M. Spears, *Evolutionary Algorithms: The Role of Mutation and Recombination*, 2000.
[82] K.-Y. Su, T.-H. Chiang, and J.-S. Chang, "An Overview of Corpus-Based Statistics-Oriented (CBSO) Techniques for Natural Language Processing," in *Intl. Journal of Computational Linguistics and Chinese Language Processing (CLCLP)*, vol. 1(1). Taipei, 1996, pp. 101-157.
[83] L. Vlad Lita, A. Ittycheriah, S. Roukos, and N. Kambhatla, "tRuEcasIng," presented at ACL, Sapporo, Hokkaido, Japan, 2003.
[84] F. Yvon, "Paradigmatic Cascades: a Linguistically Sound Model of Pronunciation by Analogy," presented at Thirty-Fifth Annual Meeting of the ACL and Eighth Conference of the EACL, 1997.
[85] Y. Zaharin, "On Formalisms and Analysis, Generation and Synthesis in Machine Translation," presented at 4th Conference of the European Chapter of the Association of Computational Linguistics, Manchester, 1989.

[86] M. M. v. Zannen, "Bootstrapping Structure into Language: Alignment Based Learning," in *Learning; Computation and Language*. Leeds: University of Leeds, 2002.
[87] B. Zhou, Y. Gao, J. Sorensen, Z. Diao, and M. Picheny, "Statistical Natural Language Generation for Speech-to-Speech Machine Translation Systems," presented at ICSLP-2002:Inter. Conf. on Spoken Language Processing, Denver, CO USA, 2002.
[88] B. Zhou, Y. Gao, J. Sorensen, D. Dechelotte, and M. Picheny, "A Hand-held Speech-to-speech Translation System," presented at IEEE ASRU, US Virgin Islands, 2003.

Annexes

Les annexes sont numérotées ainsi :

1. Données
2. Programmes en Perl
3. Feuilles de style XSL
4. Dictionnaires de termes simples obtenus
5. Quelques exemples de retraduction.

1. Données

Ce premier fichier est un extrait de mémoire brute (maquillée pour des raisons de confidentialités). Le format de cette mémoire est le format d'export de TM2. L'encodage de cette mémoire dépend de celui du projet : il n'est pas indiqué. On constate que beaucoup de bisegments sont totalement dépourvus d'intérêt sur le plan linguistique (0000000002 par exemple). De cet extrait de 9 bisegments (sur un total de 33 dans le fichier), seuls deux bisegments sont effectivement conservés (4 en tout) dans l'extrait filtré et nettoyé (voir plus bas)

```
<NTMMemoryDb>
<Description>
Annonce 33740 O. xxxxxxx
</Description>
<Segment>0000000001
<Control>
000004•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>:ivoryltr date='May 16, 2000'
ltrno='ZP00-0357'
</Source>
<Target>:ivoryltr date='16 mai 2000'
ltrno='ZP00-0357'
</Target>
</Segment>
<Segment>0000000002
<Control>
000005•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>ltrty=''
rfa='33740'
huid='MW'
hadate='20000516'
rvdate=''
srep='ULRICH K. XXXXXX  (0)7031-163608 IBMDE(XXXXXX)'
catg='OE100, OE200, OE300'
</Source>
<Target>ltrty=''
rfa='33740'
huid='MW'
hadate='20000516'
rvdate=''
srep='ULRICH K. XXXXXX  (0)7031-163608 IBMDE(XXXXXX)'
catg='OE100, OE200, OE300'
</Target>
</Segment>
<Segment>0000000003
<Control>
000013•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>Modification:  OS/390 Workload Transition Offering for VM and VSE
</Source>
<Target>Modification de l'offre Nouvelles applications OS/390 pour VM et VSE
</Target>
</Segment>
<Segment>0000000004
<Control>
000019•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>The OS/390&rm. Workload Transition Offering for VM and VSE is
being modified so it can be combined with
other offerings that specifically state that it may be combined
with those offerings.
</Source>
<Target>L'offre Nouvelles applications OS/390 pour VM et VSE est
modifi,e de fa‡on … pouvoir ^tre combin,e avec d'autres offres
pour lesquelles la compatibilit, est sp,cifiquement indiqu,e.
</Target>
</Segment>
<Segment>0000000005
<Control>
000020•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
```

```
</Control>
<Source>These offerings
reduce the expense for smaller system customers considering
transitioning workloads from VM and/or VSE to OS/390.
</Source>
<Target>Ces offres r,duisent les d,penses des clients utilisant
des plus petits systŠmes et souhaitant faire ,voluer leurs
applications VM et/ou VSE vers OS/390.
</Target>
</Segment>
<Segment>0000000006
<Control>
000021•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>This modification also corrects and clarifies the worldwide
education reimbursement terms and conditions of this offering.
</Source>
<Target>Par ailleurs, cette modification corrige et clarifie les
termes et conditions internationales de remboursement des d,penses
de formation applicables … cette offre.
</Target>
</Segment>
<Segment>0000000007
<Control>
000022•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>In addition, any OS/390 or MVS&tm. workload moved from a different
location to the Multiprise&rm. 3000 will not be eligible for waiver
of charges.
</Source>
<Target>En outre, le transfert de toute application OS/390 ou MVS d'un autre site
sur un Multiprise 3000 ne donnera pas droit … cette exon,ration.
</Target>
</Segment>
<Segment>0000000008
<Control>
000024•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>The following is related to this announcement and supersedes
the previously announced section of the
:hp2.Supplemental:ehp2. section
in
Marketing Announcement letter ZA99-0331
</Source>
<Target>Les informations ci-dessous concernent la pr,sente annonce
et remplacent la section
:hp2.Supplemental Information:ehp2. de la pr,c,dente lettre d'annonce
marketing ZA99-0331,
</Target>
</Segment>
<Segment>0000000009
<Control>
000026•0•0000000963490157•English(U.S.)•FRENCH(NATIONAL)••EQFWWAL•33740AFR.SCR•
</Control>
<Source>dated September 20, 1999.
</Source>
<Target>dat,e du 20 septembre 1999.
</Target>
</Segment>
……
</NTMMemoryDb>
```

Nous donnons ici le résultat du filtrage du fichier ci-dessus. Les segments sélectionnés sont ceux qui semblaient bien formés (Pas de balise ou de signes spéciaux, segments sources et segments cibles différents, pas de doublon source etc…) Les balises XML ont été changées, la conversion des caractères en **iso-8859-1** a été effectuée et les signes de ponctuation ont été enlevés.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<tm title="33740.EXP">
<segment nbr="6">
  <source>This modification also corrects and clarifies the worldwide education
reimbursement terms and conditions of this offering.</source>
  <target>Par ailleurs, cette modification corrige et clarifie les termes et
conditions internationales de remboursement des dépenses de formation applicables à
cette offre.</target>
</segment>
<segment nbr="9">
  <source>dated September 20, 1999.</source>
  <target>datée du 20 septembre 1999.</target>
</segment>
<segment nbr="15">
  <source>They will provide you the details on what type of education vehicle is
available to you as part of this offering.</source>
  <target>On vous indiquera le mode de paiement disponible dans le cadre de cette
offre.</target>
</segment>
<segment nbr="30">
  <source>Trademarks</source>
  <target>Marques</target>
</segment>
</tm>
```

2. Programmes en Perl

La routine qui suit est utilisée dans plusieurs de nos programmes : elle met en œuvre la recherche par tri dichotomique dans les fichiers **xxx.dicho** de notre base de connaissances.

```
#// La routine getinfo implémente une recherche dichotomique récursive.
#// Les lignes sont classées dans l'ordre hexa des caractères de la clé.
#// La première ligne du fichier est ignorée.
#// Les données sont séparées de la clé de recherche par un \t.
#// Seules les données sont renvoyées ou undef si la clé n'est pas dans le fichier.

sub getinfo{
   my $ref = shift; #// handle du fichier de data
   my $key = shift; #// clé à chercher
   my $min = shift; #// bas de la zone à chercher
   my $max = shift; #// haut de la zone à chercher
   return undef if $max-$min<2; #// renvoie undef si la clé est absente du fichier
   my $mid = int(($max+$min)/2);
   seek($ref, $mid, 0); #// fixe la nouvelle position de lecture.
   readline($ref); #// permet de se caler sur une ligne. La première ligne du
fichier est ignorée.
   my $line = readline($ref);
   my @line = split /\t/,$line; #// on sépare la clé des données.
   if ($line[0] gt $key) {
      return getinfo($ref,$key,$min,$mid); #// relance de la recherche dans la
moitié précédente
   }
   if ($line[0] lt $key) {
      return getinfo($ref,$key,$mid,$max); #// relance de la recherche dans la
moitié suivante
   }
return join("\t",@line[1..$#line]); #// renvoie les données.
}
```

collect.pl : Ce programme assigne à chaque mot de chaque langue tous les mots des segments en face desquels il a été rencontré. Ces mots candidats sont comptés et ordonnés. Il y a deux fichiers de sortie : un fichier volumineux avec tous les mots sources (décomptés) suivis chacun de tous les mots cibles candidats associé au nombre d'occurrences de ce mot cible dans les bisegments où ce mot source a été rencontré. Le deuxième fichier ne contient que les mots cibles avec leur nombre d'occurrences total.

```
#//################################################
#// Refonte de la collection des équivalents :
#// Pour chaque mot d'un segment, on collectionne tous mots du segment d'en face
#//(pas deux fois s'il y a deux occurrences)
#// On fait ca pour tous les segments du fichier.
#// Une seule passe, un seul fichier de sortie, pas d'indirection.
#// le fichier d'entrée est constitué de segments ponctués.
#// Ce fichier traite des répertoires de fichiers.
#// Doit être utilisé avec un batch pour ne pas dépasser la capacité mémoire
(=50.000 segments ,=100 fichiers);
#// Crée deux fichiers de données dont les noms contiennent les numéros de mémoires
concernées dans le répertoires.
#// Un autre perl Consolide.

my $begin = shift;
my $finish = shift;

$INdir = 'D:\These\dev\data\mem\EnFr\FilteredMem';
$S_DATAfile = 'Sequiv_'.$begin.'_'.$finish.'.data';
$T_DATAfile = 'Tequiv_'.$begin.'_'.$finish.'.data';

my %src2trg = ();
my %trg2src = ();
my %src;
my %trg;
my @src;
my @trg;

$nb_seg_tot = 0;

opendir(DIR,$INdir);
my $nb=0;
while($INfile=readdir(DIR))
{
#//next unless -f $INfile;
$nb++;
next if $nb<$begin;
print heure(),"N° $nb : $INfile \n";
process_file ($INdir,$INfile);
last if $nb>=$finish;
}


sub process_file {
$INdir = shift;
$INfile = shift;
print "traitement de $INfile\n";
open(FILE,$INdir.'\\'.$INfile);
my $nb_seg=0;
while(<FILE>)
{
   if (/<source>(.*)<\/source>/) {
      my $src = $1;
      @src = split(/\s+/,$src);
      }
   if (/<target>(.*)<\/target>/) {
      my $trg = $1;
      @trg = split(/\s+/,$trg);
      }
   if (/<\/bisegment>/) {
      foreach my $wrd (set(@src)){
         $src{$wrd} += grep {$_ eq $wrd} @src;
         push @{$src2trg{$wrd}}, @trg;
         }
      $nb_seg++;
      print heure(), " $nb_seg segments traités.\n" unless $nb_seg%100;
      }
   }
```

```
$nb_seg_tot += $nb_seg;
print heure(), " $nb_seg nouveaux segments traités : $nb_seg_tot en tout.\n";
close FILE;
}

print  heure()," Ecriture de $S_DATAfile\n";
open(S_OUT,'>'.$S_DATAfile);
foreach my $Swrd (keys %src2trg){
   my %trans=();
   foreach my $Ttr (@{$src2trg{$Swrd}}){
      $trans{$Ttr}++;
      }
   print S_OUT "$Swrd ",$src{$Swrd}," ";
   print S_OUT join ' ', map {($_ ,$trans{$_}) } sort {$trans{$b}<=>$trans{$a}}
(keys %trans);
   print S_OUT "\n";
   }
close S_OUT;

=inutile
print  heure()," Ecriture de $T_DATAfile\n";
open(T_OUT,'>>'.$T_DATAfile);
foreach my $Twrd (keys %trg2src){
   my %trans=();
   foreach my $Str (@{$trg2src{$Twrd}}){
      $trans{$Str}++;
      }
   print T_OUT "$Twrd ",$trg{$Twrd}," ";
   print T_OUT join ' ', map {($_,$trans{$_}) } sort {$trans{$b}<=>$trans{$a}} (keys
%trans);
   print T_OUT "\n";
   }
close T_OUT;
print  heure()," Fini.\n";
=cut
sub set{
   my %record = ();
   foreach $elem (@_){
      $record{$elem} = 1;
      }
   return keys %record;
}

sub heure{
   @t = gmtime(time()+3600);
   return
'['.substr($t[2]+100,1).':'.substr($t[1]+100,1).':'.substr($t[0]+100,1).']';
   }

##################################################
```

filter.pl : Ce programme doit tourner plusieurs fois. Il associe les digrammes dans le corpus, la première fois sur le critère des décomptes établis par le **collect.pl**, les fois suivantes sur le critère des associations faites dans la passe précédente. On constate que le nombre de digrammes différents diminue plus vite que le nombre d'occurrences d'associations, qui lui reste quasiment stable.

```
#//################################################
#// Filter applique les candidats de collect equiv au segments de NewMemOK.
#// On ne cherche pas à placer les digrammes, juste à les collectionner.
#// On n'enregistre pas les hapax qui seront traités ultérieurement quand on saura
placer avec des arbres.
#// On fait autant de passe que possible, jusqu'à épuisement du segment le plus
court.
#// Il faut relancer la routine plusieurs fois, avec min et max comme arguments : cf
.bat .

use strict;

my $min = shift;
my $max = shift;
my $BOTTOM = shift;
$BOTTOM = 100 unless $BOTTOM;

#// my $S_data = 'D:\These\dev\learn\digram\Sequiv.data';

my $S_data = "D:\\These\\CollectEquiv\\Sequiv.data";
my $T_data = "D:\\These\\CollectEquiv\\Tequiv.data";

my $MEMdir = 'D:\These\dev\data\mem\EnFr\XMLequiv';

my $DIGRAM_FILE = 'D:\These\dev\learn\digram\digrams1.data';

my %SWxTW_NOEC = ();
my %SW_NOEC = ();
my %TWxSW_NOEC = ();
my %TW_NOEC = ();
my %DIGRAM = ();
my $DEPTHMAX;

=douteux...
#// cette manière de charger les données ne fonctionne pas.
printf "%s Chargement de $S_data.\n", &heure;
open(S_FILE,$S_data);
while(<S_FILE>){
   my @line = split /\t/;
   my $Swrd = shift @line;
   $SW_NOEC{$Swrd} = shift @line;
   %{$SWxTW_NOEC{$Swrd}} = @line;
   foreach my $Tinfo (@line){
      my ($Twrd, $noec) = split / /, $Tinfo;
      $TW_NOEC{$Twrd} += $noec;
      $TWxSW_NOEC{$Twrd}{$Swrd} = $noec;
      }
   }
close S_FILE;
=cut

printf "%s Chargement de $S_data.\n", &heure;
open(S_FILE,$S_data);
while(<S_FILE>){
   my @line = split / /;
   my $Swrd = shift @line;
   $SW_NOEC{$Swrd} = shift @line;
   %{$SWxTW_NOEC{$Swrd}} = @line;
   }
close S_FILE;

printf "%s Chargement de $T_data.\n", &heure;
open(T_FILE,$T_data);
while(<T_FILE>){
      my @line = split / /;
      my $wrd = shift @line;
      $TW_NOEC{$wrd} += shift @line;
      %{$TWxSW_NOEC{$wrd}} = @line;
      }
```

```
close T_FILE;

printf "\n%s Traitement des fichiers.\n", &heure;

my $file_nb=0;
my $seg_nbr = 0;
opendir(DIR,$MEMdir)or die "pb de répértoire\n";
while(my $file_name=readdir(DIR)){
   my $file = $MEMdir."\\".$file_name;
   next unless -f($file);
   next if $file_nb+1 > $max or $file_nb+1 < $min;
   $file_nb++;
   print &heure, " N° $file_nb : $file_name\n";
   open(FILE,$file)or die "pb de fichier\n";
   my $src;
   my $trg;
   while(<FILE>){
      if (/<segment>/) {
         $seg_nbr++;
         }
      elsif (/<source>(.*)<\/source>/) {
         $src = $1;
         $src =~ s/(\"|\-|\,|\;|\:|\?|\!|\.|\')/ /g;
         $src =~ s/\s+/ /g;
         $src =~ s/^\s//g;
         $src =~ s/\s$//g;
         }
      elsif (/<target>(.*)<\/target>/) {
         $trg = $1;
         $trg =~ s/(\"|\-|\,|\;|\:|\?|\!|\.|\')/ /g;
         $trg =~ s/\s+/ /g;
         $trg =~ s/^\s//g;
         $trg =~ s/\s$//g;
         }
      elsif (/<\/segment>/) {
         next unless $src and $trg;
         my @src = split /\s+/, $src;
         my @trg = split /\s+/, $trg;
         next if $#src >30 or $#trg>30;
         my $depth = process(\@src,\@trg,0);
         $DEPTHMAX = $depth>$DEPTHMAX?$depth:$DEPTHMAX;
         printf "%s %7d segments.\n", &heure, $seg_nbr if ($seg_nbr % 100 == 0);
      }#// while(<FILE>)
   }#// while
   printf "%s %d segments, DEPTHMAX = $DEPTHMAX\n", &heure, $seg_nbr;
   close FILE;
}
closedir DIR;

printf "\n%s %d fichiers traités.\n", &heure, $file_nb;

my $nbwrd = 0;
my $nbtrad = 0;
open(DIG,'>'.$DIGRAM_FILE);
foreach my $sw (keys %DIGRAM){
   my $nbocc = 0;
   my $data = '';
   foreach my $tw (sort {$DIGRAM{$sw}{$b}<=>$DIGRAM{$sw}{$a}} (keys
%{$DIGRAM{$sw}})){
      $nbocc += $DIGRAM{$sw}{$tw};
      $data .= "\t$tw ".$DIGRAM{$sw}{$tw};
      $nbtrad++;
      }
   print DIG "$sw\t$nbocc$data\n";
   $nbwrd ++;
   }
close DIG;

my $moy = $nbtrad/$nbwrd;

printf "\n%s %d mots sources enregistrés avec %2f traductions en moyenne.\n",
&heure,$nbwrd,$moy;

sub process{
   my @src = @{shift()};
   return 0 if $#src<0;
   my %src = breakdown(@src);
```

```
   my @trg = @{shift()};
   return 0 if $#trg<0;
   my $depth = shift;
   my %trg = breakdown(@trg);
#// print join(' ',@src),"\n";
#// print join(' ',@trg),"\n";
   my %Swrd2Tlst =();
   my %Twrd2Slst =();
   my @Spos =();
   my @Tpos =();
   foreach my $Swrd (keys %src){

@{$Swrd2Tlst{$Swrd}}=sort{$SWxTW_NOEC{$Swrd}{$b}/$TW_NOEC{$b}<=>$SWxTW_NOEC{$Swrd}{$
a}/$TW_NOEC{$a}}(keys %trg);
      }
   foreach my $Twrd (keys %trg){

@{$Twrd2Slst{$Twrd}}=sort{$TWxSW_NOEC{$Twrd}{$b}/$SW_NOEC{$b}<=>$TWxSW_NOEC{$Twrd}{$
a}/$SW_NOEC{$a}}(keys %src);
      }
   my $stop = 1;
   foreach my $Swrd (keys %src){
      my $Twrd = $Swrd2Tlst{$Swrd}[0];
      if ($Swrd eq $Twrd2Slst{$Twrd}[0]){
         next unless $SWxTW_NOEC{$Swrd}{$Twrd} and $TWxSW_NOEC{$Twrd}{$Swrd};
         $stop = 0;
#print "[$Swrd] <-> [$Twrd]\n";
         my $qty = $#{$src{$Swrd}}<$#{$trg{$Twrd}}?$#{$src{$Swrd}}:$#{$trg{$Twrd}};
         push @Spos, @{$src{$Swrd}}[0..$qty];
         push @Tpos, @{$trg{$Twrd}}[0..$qty];
         $DIGRAM{$Swrd}{$Twrd}+=(1+$qty) if $SW_NOEC{$Swrd}>1 and $TW_NOEC{$Twrd}>1;
#// on enregistre, sauf hapax
         }#// if
      }#// foreach
   return 0 if $stop;
   foreach my $Spos (sort {$b <=> $a} (@Spos)){splice @src,$Spos,1}
   foreach my $Tpos (sort {$b <=> $a} (@Tpos)){splice @trg,$Tpos,1}
   return 1+process(\@src,\@trg,$depth+1) if $depth < $BOTTOM;
   }#// sub process

#-------------------------------------------------------------------------------
---------------
sub breakdown {
   my %items =();
   my $p = 0;
   foreach my $i (@_){
      push @{$items{$i}}, $p++ ;
      }
   return %items;
   }

#-------------------------------------------------------------------------------
---------------
sub heure{
   my @t = gmtime(time()+3600);
   return sprintf("[%02.0f:%02.0f:%02.0f]", $t[2],$t[1],$t[0]);
   }
```

tree.pl : Ce programme calcule les arbres binaires. Il prend en entrée le corpus muni de digrammes ainsi que les fichiers de bigrammes et de trigrammes. Le fichier de sortie est constitué de paires de nœuds congruents du corpus à un niveau donné (amphigrammes).

```
##################################################
#// 14/05/2003
#// Version intégrant les deux passes : on calcule les sous arbres maximaux, on les
comptes, on ne réécrit rien.
#// on fait une deuxième passe, et on choisit le di-tree le plus fréquent dans la
première passe (et non pas le maximal).
#// On récupère les laissés pour compte (tout est arbre) et on dumpe les XML et le
.dicho.
#//
#// 29/04/2003 XoF. Routine reprise de MapTrees (GetDiSubTrees1.pl)
#// noec : nombre d'occurrences en corpus.
#// Les changements concernent :
#// On ne s'intéresse qu'au sous arbres simples (un seul objet)
#// On reprend les identifiants de Learn, et les mémoires de Learn
#// Cette routine utilise 'D:\These\Learn\XMLmem' qui contient les segments écrits,
les ponts entre positions
#// incarnant les objets pleins et les objets eux mêmes. On connait donc les
positions des objets.
#// On utilise les fichiers NTokenS.data et NTokenT.data qui contiennent les
trigrammes.
#// Pour chaque disegment, l'arbre source et l'arbre cible sont montés sous forme
d'array à deux élements, chacun étant une
#// référence sur un array inférieur. Les feuilles sont des arrays à un élément :
l'index du objet. (et non pas
#// directement l'index dans l'array à deux élements supérieur, comme avant).
#// Au fur et à mesure que l'arbre est monté, on enregistre les sous-arbres maximaux
(avec toutes les dépendances vides)
#// ne contenant qu'un objet plein.
#// Ces enregistrements sont effectués dans deux hash (obj1PosToTree et
obj2PosToTree)
#// qui renvoient la référence du sous-arbre pour chaque position des objets pleins.
#// Les di-arbres enregistrés sont les paires présentes dans ces deux hash à la
fois.
##################################################

use strict;

my $L1 = shift or die "What is the first language?\n";
my $L2 = shift or die "What is the second language?\n";
($L1, $L2) = sort {$a cmp $b} ($L1, $L2);                    #// On rétablit l'ordre
alpha des langues.

my $inObj = shift or die "What object do we start from ?\n"; #// "digram" ou "class"
suivi du niveau

my $langDir = "$L1$L2";

my $lvl = 1+hex(substr($inObj,-1));       #// 1 pour une lettre (object ou class),
le chiffre++ si c'en est un.
my $outObj = sprintf("%s%1x",substr($inObj,0,-1),$lvl);    #// !!! Le level
conditionne le choix arbres simples / arbres complexes.

 my $min = shift;
 my $max = shift;

my $inDir  = "D:\\These\\dev\\data\\mem\\$langDir\\$inObj\\";
my $outDir = "D:\\These\\dev\\data\\mem\\$langDir\\$outObj\\";

#// les .dicho de données :

my $obj_file   = "D:\\These\\dev\\data\\extract\\$langDir\\$inObj.dicho";
my $Nobj1_file = "D:\\These\\dev\\data\\extract\\$langDir\\N$inObj$L1.dicho";
my $Nobj2_file = "D:\\These\\dev\\data\\extract\\$langDir\\N$inObj$L2.dicho";

#// le .dicho de sortie :

my $outObj_file = "D:\\These\\dev\\data\\extract\\$langDir\\$outObj.dicho" ;

#// les hash lus dans les fichiers de data : NOEC des objets, biObj et triObj.

my %objNoec = ();
my %biObjNoec1 = ();
my %biObjNoec2 = ();
```

```
my %triObjNoec1 = ();
my %triObjNoec2 = ();


#// GLOBAL : parce qu'on ne peut pas faire des my dans les boucles de lectures de
ligne du fichier

my @Xobj1X = ();   #// Les arrays des objets lus dans les mémoires.
my @Xobj2X = ();
my @pos1pos2 = (); #// Les arrays de ponts entre objets source et cible (lus dans
les mémoires)
my @pos2pos1 = ();
my @sec1 = ();     #// Les arrays de sécabilité. (Calculés)
my @sec2 = ();

my %diTreeNoec = ();    #// NOEC des ditrees provisoires, pour le choix. HASH

my $diTreeNum;          #// Numéro du dernier diTree définitif indexé. Initialisé
plus bas.
my %idxToNoec = ();     #// NOEC des ditrees définitifs, par l'index pour le dump.
ARRAY.
my %parenToIdx = ();    #// HASH : {parenS}{parenT} -> index
my %idxToParen = ();    #// index -> "parenS parenT"
my %p1p2 = ();
my %p2p1 = ();

#// les limites gauche et droite

my $idxL = sprintf("N%d#00001",$lvl-1);
my $idxR = sprintf("N%d#00002",$lvl-1);

my $newidxL = sprintf("N%d#00001",$lvl);;
my $newidxR = sprintf("N%d#00002",$lvl);
$idxToParen{$newidxL} = "$idxL $idxL";
$idxToParen{$newidxR} = "$idxR $idxR";
$idxToNoec{$newidxL} =1;
$idxToNoec{$newidxR} =1;

my $nu = "#null#";

#// Chargement des data

print &heure, " Loading $inObj from $obj_file...\n";
my $objNum = 0;
my $objI;
open OBJ, $obj_file;
<OBJ>;
while(<OBJ>){
   s/\x0D\x0A//;
   my @data = split /\t/,$_;
   $objNoec{$data[0]} = $data[1]; #// le noec de la clé est toujours en deuxième
position.
   $objNum++;
   $objI = hex(substr($data[0],3));
   }
close OBJ;
print &heure, " $objNum $inObj loaded.\n\n";

$diTreeNum = 1+$objI; #//On commence à indexer les ditrees à partir du dernier
objet.

print &heure, " Loading N$inObj$L1 from $Nobj1_file...\n";
my $biNb = 0;
my $triNb = 0;
open(TRI,$Nobj1_file);
<TRI>;
while(<TRI>){
   chomp;
   my @data = split /\t/,$_;
   my $biObj = shift @data;
   my @biObj = split / /,$biObj;
   $biObjNoec1{$biObj[0]}{$biObj[1]} = shift @data;
   $biNb ++;
   next if $#data < 0;
   foreach my $i (0..$#data){
      my $info = $data[$i];
      my @info = split / /, $info;
```

```
      $triObjNoec1{$biObj[0]}{$biObj[1]}{$info[0]} = $info[1];
      $triNb++;
      }
   }
close TRI;
print &heure, " $biNb bi$inObj$L1 and $triNb tri$inObj$L1 loaded.\n\n";

print &heure, " Loading N$inObj$L2 from $Nobj2_file...\n";
$biNb = 0;
$triNb = 0;
open(TRI,$Nobj2_file);
<TRI>;
while(<TRI>){
   chomp;
   my @data = split /\t/,$_;
   my $biObj = shift @data;
   my @biObj = split / /,$biObj;
   $biObjNoec2{$biObj[0]}{$biObj[1]} = shift @data;
   $biNb ++;
   next if $#data < 0;
   foreach my $i (0..$#data){
      my $info = $data[$i];
      my @info = split / /, $info;
      $triObjNoec2{$biObj[0]}{$biObj[1]}{$info[0]} = $info[1];
      $triNb++;
      }
   }
close TRI;
print &heure, " $biNb bi$inObj$L2 and $triNb tri$inObj$L2 loaded.\n\n";

#//-------------------------------------------------Première passe

print &heure, " First pass on $inDir...\n\n";
my $treeNb=0;
my $fileNb=0;
opendir(DIR,$inDir);
while(my $file_name = readdir(DIR)){
   my $file = $inDir."\\".$file_name;
   next unless -f($file);
   next if ++$fileNb >$max or $fileNb <$min; #// par prudence...
   print &heure, " N° $fileNb : $file_name\n";

open(FILE,$file) or die "Pas de fichier\n";
while(<FILE>)
{
   if (/<bisegment>/){
      @Xobj1X = ();
      @Xobj2X = ();
      @pos1pos2 = ();
      @pos2pos1 = ();
      }
   elsif (/<$inObj$L1>(.*)<\/$inObj$L1>/) {@Xobj1X = split(/ /,"$idxL $1 $idxR");}
   elsif (/<$inObj$L2>(.*)<\/$inObj$L2>/) {@Xobj2X = split(/ /,"$idxL $1 $idxR");}
   elsif (/<bridge pos$L1=\"([^\"]*)\" pos$L2=\"([^\"]*)\"\/>/) {
      $pos1pos2[$1-1]=$2-1; #// -1 parce que l'indice du fichier commencent à 1.
(utilisé dans le visualiseur par ailleurs)
      $pos2pos1[$2-1]=$1-1;
      }
   elsif (/<\/bisegment>/) {
      next if $#Xobj1X == -1 and $#Xobj2X == -1;          #// segment entièrement
traité.
      next if $#Xobj1X == 2 and $#Xobj2X == 2;            #// Cime de l'arbre
atteinte.

      my %Pos2pTreeMax1 = (); #// pointeur du sous-arbre max pour chaque pos pleine.
      my %Pos2pTreeMax2 = ();
      my %Pos2pTreeMin1 = (); #// pointeur du sous-arbre min pour chaque pos pleine.
(inutile pour la première passe).
      my %Pos2pTreeMin2 = ();

      my $pTree1 = makeTree(1, \@Xobj1X, \%Pos2pTreeMax1, \%Pos2pTreeMin1);
      my $pTree2 = makeTree(2, \@Xobj2X, \%Pos2pTreeMax2, \%Pos2pTreeMin2);

#// print join("\n",drawtree($pTree1)), "\n";
#// print "Max : ", join(' ', map {"$_->".$Pos2pTreeMax1{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMax1))), "\n";
```

```
#// print "Min : ", join(' ', map {"$_->".$Pos2pTreeMin1{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMin1))), "\n";

#// print join("\n",drawtree($pTree2)), "\n";
#// print "Max : ", join(' ', map {"$_->".$Pos2pTreeMax2{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMax2))), "\n";
#// print "Min : ", join(' ', map {"$_->".$Pos2pTreeMin2{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMin2))), "\n";
#// print "\n-------------------------------------------------------------------------
---------\n";

      if ($lvl == 1){             #// Cas simple.
         foreach my $pos1 (keys %Pos2pTreeMax1){
            my $pos2 = $pos1pos2[$pos1];             #// Positions homologues côté
cible.
            my $pSubTree1 = $Pos2pTreeMax1{$pos1};   #// sous arbre côté source
            my $pSubTree2 = $Pos2pTreeMax2{$pos2};   #// sous arbre côté cible
            $treeNb++ unless $diTreeNoec{paren($pSubTree1)}{paren($pSubTree2)}++;
#// Obj parenthésés.
            }#// foreach
         }#//if
      else{                        #// Cas complexe.
         my %posOK = ();     #// tableau des pos des ditrees retenus : pour éviter
de prendre les plus gros.
         foreach my $pos1 (sort {ordpos($a)<=>ordpos($b)} (keys %Pos2pTreeMax1)){
#// les petits arbres d'abord, les 1 ensuite.
            my @pos1 = split(/:/,$pos1);
            next if grep {$posOK{$_}} @pos1;               #// on saute si cet arbre
en contient de plus petits.
            my $pos2 = join(':',sort {$a<=>$b}(map {$pos1pos2[$_]} @pos1)); #// Pos.
homol. côté cible, dans ordre cible.
            my $pSubTree1 = $Pos2pTreeMax1{$pos1};         #// sous arbre max côté
source
            my $pSubTree2 = $Pos2pTreeMax2{$pos2};         #// sous arbre max côté
cible
            next unless $pSubTree2;                        #// C'EST LA CONDITION
DEFINITOIRE : la cible existe-t-elle ?
            $treeNb++ unless $diTreeNoec{paren($pSubTree1)}{paren($pSubTree2)}++;
#// soit l'un soit l'autre...
            map {$posOK{$_}=1} @pos1;                      #// on note les pos
retenues.
            }#//foreach
         }#//else
      } #// elsif
   }#// while file
}#// while dir
close DIR;
print &heure, " End of first pass on $inDir : $treeNb ditrees selected.\n\n";

#-------------------------------------------------------------------------------------
-----------------
#=second pass
#-------------------------------------------------------------------------------------
-----------------

print &heure, " Second pass on $inDir...\n";
my $segNb =0;
my $greenSegNb =0;   #// segments actifs
my $orangeSegNb =0;  #// segments terminés à cette passe.
my $redSegNb =0;     #// segments finis à une passe antérieure.
$treeNb=0;
$fileNb=0;
opendir(DIR,$inDir);
while(my $file_name = readdir(DIR)){
   my $file = $inDir."\\".$file_name;
   next unless -f($file);
   next if ++$fileNb >$max or $fileNb <$min; #// par prudence...
   print &heure, " N° $fileNb : $file_name\n";

   my $outfile = $outDir."\\".$file_name;
   open(OUT,">".$outfile);
   print OUT "<?xml version=\"1.0\" encoding=\"iso-8859-1\" ?>\n";
   print OUT "<bitext level='$lvl' type='object' S='$L1' T='$L2'>\n";

open(FILE,$file) or die "Pas de fichier\n";
while(<FILE>)
{
```

```
   chomp;
   if (/<bisegment>/){
      print OUT "<bisegment>\n";
      @Xobj1X = ();
      @Xobj2X = ();
      @pos1pos2 = ();
      @pos2pos1 = ();
      %p1p2 = ();              #// Correspondance diarbres.
      %p2p1 = ();              #// idem
      }
   elsif (/<segment$L1>(.*)<\/segment$L1>/) {print OUT
"<segment$L1>$1<\/segment$L1>\n";}
   elsif (/<segment$L2>(.*)<\/segment$L2>/) {print OUT
"<segment$L2>$1<\/segment$L2>\n";}
   elsif (/<$inObj$L1>(.*)<\/$inObj$L1>/) {@Xobj1X = split(/ /,"$idxL $1 $idxR");}
   elsif (/<$inObj$L2>(.*)<\/$inObj$L2>/) {@Xobj2X = split(/ /,"$idxL $1 $idxR");}
   elsif (/<bridge pos$L1=\"([^\"]*)\" pos$L2=\"([^\"]*)\"\/>/) {
      $pos1pos2[$1-1]=$2-1; #// -1 parce que l'indice du fichier commencent à 1.
(utilisé dans le visualiseur par ailleurs)
      $pos2pos1[$2-1]=$1-1;
      }
   elsif (/<\/bisegment>/) {
      $segNb++;
      if ($#Xobj1X == -1 and $#Xobj2X == -1){ #// Cime de l'arbre atteinte.
         print OUT "<\/bisegment>\n";
         $redSegNb++;
         next;
         }
      if ($#Xobj1X == 2 and $#Xobj2X == 2){ #// Cime de l'arbre atteinte.
         print OUT "<\/bisegment>\n";
         $orangeSegNb++;
         next;
         }
      $greenSegNb++;

      my %Pos2pTreeMax1 = (); #// pos est entendu au sens large : pour un arbre
simple, la position de son objet plein.
      my %Pos2pTreeMax2 = (); #// pour un arbre complexe, la concaténation des
positions de ses obj. pleins, séparés par ':';
      my %Pos2pTreeMin1 = ();
      my %Pos2pTreeMin2 = ();
      my $pTree1 = makeTree(1,\@Xobj1X, \%Pos2pTreeMax1, \%Pos2pTreeMin1);
      my $pTree2 = makeTree(2,\@Xobj2X, \%Pos2pTreeMax2, \%Pos2pTreeMin2);
=comment
 print join("\n",drawtree($pTree1)), "\n";
 print "Max : ", join(' ', map {"$_->".$Pos2pTreeMax1{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMax1))), "\n";
 print "Min : ", join(' ', map {"$_->".$Pos2pTreeMin1{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMin1))), "\n";
 print join("\n",drawtree($pTree2)), "\n";
 print "Max : ", join(' ', map {"$_->".$Pos2pTreeMax2{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMax2))), "\n";
 print "Min : ", join(' ', map {"$_->".$Pos2pTreeMin2{$_}} (sort {$a=~tr/:/:/ <=>
$b=~tr/:/:/}(keys %Pos2pTreeMin2))), "\n";
 print "-------------------------------------------------------------------------
----\n";
=cut
      if ($lvl == 1){           #// Cas simple.
         foreach my $pos1 (keys %Pos2pTreeMax1){
            my $pos2 = $pos1pos2[$pos1];                 #// Positions homologues
côté cible.
            my $pSubTreeMax1 = $Pos2pTreeMax1{$pos1};   #// sous-arbre max côté
source
            my $pSubTreeMax2 = $Pos2pTreeMax2{$pos2};   #// sous-arbre max côté
cible
            my $pSubTreeMin1 = $Pos2pTreeMin1{$pos1};   #// sous-arbre min côté
source
            my $pSubTreeMin2 = $Pos2pTreeMin2{$pos2};   #// sous-arbre min côté
cible
            recordBest($pSubTreeMax1,$pSubTreeMax2,$pSubTreeMin1,$pSubTreeMin2);
            } #// foreach
         } #// if ($lvl==1)
      else{                        #// Cas complexe.
         my %posOK = ();     #// hash des pos des ditrees retenus : pour éviter de
prendre les plus gros.
         foreach my $pos1 (sort {ordpos($a)<=>ordpos($b)} (keys %Pos2pTreeMax1)){
#// les petits arbres d'abord, les 1 ensuite.
```

```
            my @pos1 = split(/:/,$pos1);
            next if grep {$posOK{$_}} @pos1;               #// on saute si cet arbre
en contient de plus petits.
            my $pos2 = join(':',sort {$a<=>$b} (map {$pos1pos2[$_]} @pos1)); #//
Pos. homol. côté cible, dans ordre cible.
            my $pSubTreeMax1 = $Pos2pTreeMax1{$pos1};    #// sous arbre max côté
source
            my $pSubTreeMax2 = $Pos2pTreeMax2{$pos2};    #// sous arbre max côté
cible
            if (${$pSubTreeMax1}[2]==1 and ${$pSubTreeMax2}[2]==1){ #// on est au
sommet : on veut les max. (il faudrait le
               recordTree(paren($pSubTreeMax1),paren($pSubTreeMax2)); #// faire à
level =1 aussi.
               $p1p2{$pSubTreeMax1} = $pSubTreeMax2;
               $p2p1{$pSubTreeMax2} = $pSubTreeMax1;
               last;
               }
            next unless $pSubTreeMax2;                      #// C'EST LA CONDITION
DEFINITOIRE : la cible existe-t-elle ?
            my $pSubTreeMin1 = $Pos2pTreeMin1{$pos1};      #// sous-arbre min côté
source
            my $pSubTreeMin2 = $Pos2pTreeMin2{$pos2};      #// sous-arbre min côté
cible
#//print "[$pos1][$pos2]\n";
            recordBest($pSubTreeMax1,$pSubTreeMax2,$pSubTreeMin1,$pSubTreeMin2);
            map {$posOK{$_}=1} @pos1;                      #// on note les pos
retenues.
            }#//foreach
         }#//else

      my @treeChain1 = treeChain(1,$pTree1,\%p1p2);
      my @treeChain2 = treeChain(2,$pTree2,\%p2p1);

      my $objString1 = objString(1,\@treeChain1,\%p1p2);
      my $objString2 = objString(2,\@treeChain2,\%p2p1);
=comment
 print join(' ', map {$_.'->'.$p1p2{$_}} (keys %p1p2)),"\n";
 print join(" ", @treeChain1), "\n";
 print $objString1, "\n\n";

 print join(' ', map {$_.'->'.$p2p1{$_}} (keys %p2p1)),"\n";
 print join(" ", @treeChain2), "\n";
 print $objString2, "\n";
 print
"================================================================================\n"
;
=cut
      my %newPos = ();
      foreach my $i (0..$#treeChain1){
         my $p1 = $treeChain1[$i];
         foreach my $j (0..$#treeChain2){
            my $p2 = $treeChain2[$j];
            if ($p1p2{$p1} == $p2){
               $newPos{$i+1} = $j+1;
               last;
               }
            } #// foreach 2
         } #// foreach 1

#//       printf OUT "<paren1>%s<\/paren1>\n", paren($pTree1);
#//       printf OUT "<paren2>%s<\/paren2>\n", paren($pTree2);
      print  OUT "<$outObj$L1>$objString1<\/$outObj$L1>\n";
      print  OUT "<$outObj$L2>$objString2<\/$outObj$L2>\n";

      foreach my $pos (keys %newPos){
         print OUT "<bridge pos$L1=\"",$pos,"\" pos$L2=\"",$newPos{$pos},"\"/>\n";
         }
      print OUT "<\/bisegment>\n";
#//       print &heure, " $segNb segments processed (R=$redSegNb, O=$orangeSegNb,
G=$greenSegNb).\n" unless $segNb%1000 ;
      } #// if
   }#// while file
   print OUT "<\/bitext>\n";
   close OUT;
   print &heure, " $segNb segments processed (R=$redSegNb, O=$orangeSegNb,
G=$greenSegNb).\n";
   close FILE;
```

```
} #// while dir
closedir DIR;
print heure(), " End of second pass.\n";

my $ditreeNbr = $diTreeNum-$objI;
print heure(), " Dumping $ditreeNbr $outObj.\n";
open(DITREE,">$outObj_file");
print DITREE "This file was made by tree.pl level=$lvl type=object S=$L1 T=$L2\n";
foreach my $idx (sort {$a cmp $b}(keys %idxToNoec)){
   printf DITREE "%s\t%d\t%s\n", $idx, $idxToNoec{$idx}, $idxToParen{$idx};
   }
close DITREE;
print heure(), " $outObj dumped.\n";


#-------------------------------------------------------------------------------------------
------------------
#=cut second pass.
#-------------------------------------------------------------------------------------------
------------------


#===========================================================================================
=================
sub maxindex {
   my $maxindex = 0;
   foreach my $i (0..$#_){
      $maxindex = $i if $_[$i]>$_[$maxindex];
   }
   $maxindex;
}

#-------------------------------------------------------------------------------------------
----------------
sub makeTree{
   my $lang = shift;
   my @XobjX = @{shift()};
   my $pPos2pTreeMax = shift;          #// Arbre minimale (feuille pour simple) :
pointeur sur un hash de pointeurs.
   my $pPos2pTreeMin = shift;          #// Arbre maximale : pointeur sur un hash de
pointeurs.
   my @sec = ();
#// le fait de passer les gros hash de tri et bi ralentissait énormément, même par
référence.
   if ($lang ==1){
   foreach my $i (1..$#XobjX-2){
#=print
print "$XobjX[$i-1] $XobjX[$i] $XobjX[$i+1] : ", $triObjNoec1{$XobjX[$i-
1]}{$XobjX[$i]}{$XobjX[$i+1]},"\n";
print "$XobjX[$i-1] $XobjX[$i] : ", $biObjNoec1{$XobjX[$i-1]}{$XobjX[$i]},"\n";
print "$XobjX[$i+1] : ", $objNoec{$XobjX[$i+1]},"\n";
print "$XobjX[$i] $XobjX[$i+1] $XobjX[$i+2] : ",
$triObjNoec1{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]},"\n";
print "$XobjX[$i+1] $XobjX[$i+2] : ", $biObjNoec1{$XobjX[$i+1]}{$XobjX[$i+2]},"\n";
print "$XobjX[$i] : ", $objNoec{$XobjX[$i]},"\n";
#=cut
      my $secAB_C = $triObjNoec1{$XobjX[$i-
1]}{$XobjX[$i]}{$XobjX[$i+1]}**3/($biObjNoec1{$XobjX[$i-
1]}{$XobjX[$i]}**2*$objNoec{$XobjX[$i+1]});
      my $secB_CD =
$triObjNoec1{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]}**3/($objNoec{$XobjX[$i]}*$biObj
Noec1{$XobjX[$i+1]}{$XobjX[$i+2]}**2);
      push @sec, (nicecoef($secAB_C*$secB_CD));
      }}
   else {
   foreach my $i (1..$#XobjX-2){
=print
print "$XobjX[$i-1] $XobjX[$i] $XobjX[$i+1] : ", $triObjNoec2{$XobjX[$i-
1]}{$XobjX[$i]}{$XobjX[$i+1]},"\n";
print "$XobjX[$i-1] $XobjX[$i]: ", $biObjNoec2{$XobjX[$i-1]}{$XobjX[$i]},"\n";
print "$XobjX[$i+1] : ", $objNoec{$XobjX[$i+1]},"\n";
print "$XobjX[$i] $XobjX[$i+1] $XobjX[$i+2] : ",
$triObjNoec2{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]},"\n";
print "$XobjX[$i+1] $XobjX[$i+2] : ", $biObjNoec2{$XobjX[$i+1]}{$XobjX[$i+2]},"\n";
print "$XobjX[$i] : ", $objNoec{$XobjX[$i]},"\n";
=cut
```

```
      my $secAB_C = $triObjNoec2{$XobjX[$i-
1]}{$XobjX[$i]}{$XobjX[$i+1]}**3/($biObjNoec2{$XobjX[$i-
1]}{$XobjX[$i]}**2*$objNoec{$XobjX[$i+1]});
      my $secB_CD =
$triObjNoec2{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]}**3/($objNoec{$XobjX[$i]}*$biObj
Noec2{$XobjX[$i+1]}{$XobjX[$i+2]}**2);
      push @sec, (nicecoef($secAB_C*$secB_CD));
      }}
   my @obj = @XobjX[1..$#XobjX-1];
   return ${recurTree(\@obj,\@sec,0,1,$pPos2pTreeMax,$pPos2pTreeMin)}[0];
   }

#-------------------------------------------------------------------------------
---------------
sub recurTree{
   my @seg = @{shift()};
   my @sep = @{shift()};
   my $pos = shift;                    #// Position dans le segment d'origine du
premier élément passé en @seg
   my $pNode = shift;                  #// Pointeur sur le noeud supérieur.
   my $pPos2pTreeMax = shift;          #// pointeur sur un hash de pointeurs.
   my $pPos2pTreeMin = shift;          #// pointeur sur un hash de pointeurs.
   my $pSubTree = [];

   if ($#seg==0){ #// le segment est réduit à un objet. Sa position dans le segment
d'origine est $pos.
      push @{$pSubTree}, ($seg[0],$pNode);    #// !!! LA REF. DU NOEUD SUPERIEUR EST
À LA FIN DE L'ARRAY !!!
      if ($seg[0]=~/^X/){              #// cet objet est plein
         ${$pPos2pTreeMax}{$pos}=$pSubTree;  #// on enregistre sa position
         ${$pPos2pTreeMin}{$pos}=$pSubTree;  #// on enregistre sa position
         }
      else {$pos=-1}
      return [$pSubTree,$pos]; #// on renvoie la ref d'un array contenant l'objet et
sa position s'il est plein
      }
   my $m = maxindex(@sep); #// la plus grande coupure s'effectue sur le coef de
sécabilité le plus fort.
   my @segL=@seg[0..$m];
   my @sepL=@sep[0..$m-1];
   my @segR=@seg[$m+1..$#seg];
   my @sepR=@sep[$m+1..$#sep];

   my $pSubTreeL =
recurTree(\@segL,\@sepL,$pos,$pSubTree,$pPos2pTreeMax,$pPos2pTreeMin);
   my $pSubTreeR =
recurTree(\@segR,\@sepR,$pos+$m+1,$pSubTree,$pPos2pTreeMax,$pPos2pTreeMin);

   my $posL = ${$pSubTreeL}[1];
   my $posR = ${$pSubTreeR}[1];

   push @{$pSubTree}, (${$pSubTreeL}[0],${$pSubTreeR}[0],$pNode); #// !!! LA REF. DU
NOEUD SUPERIEUR EST À LA FIN DE L'ARRAY !!!

   if ($posL==-1 and $posR ==-1){return [$pSubTree,-1]} #// C'est vide
   if ($posR==-1) {$pos=$posL}      #// dépendance à droite
   elsif ($posL==-1) {$pos=$posR}   #// dépendance à gauche
   else {$pos = "$posL:$posR"}      #// 2 sous arbres pleins

   unless(1+index($pos,':') and $lvl==1){                              #// sauf si
on est au level 1 avec une pos multiple.
      ${$pPos2pTreeMin}{$pos}=$pSubTree unless ${$pPos2pTreeMin}{$pos}; #// Arbre
minimal.
      ${$pPos2pTreeMax}{$pos}=$pSubTree;                               #// Arbre
maximal.
      }
   return [$pSubTree,$pos];
  }

#-------------------------------------------------------------------------------
---------------
sub recordBest{
   my $pSubTreeMax1 = shift;
   my $pSubTreeMax2 = shift;
   my $pSubTreeMin1 = shift;
   my $pSubTreeMin2 = shift;
#//print "de $pSubTreeMin1 à $pSubTreeMax1 et de $pSubTreeMin2 à $pSubTreeMax2\n";
```

```
   my $noecMax = 0;
   my $bestParen1;
   my $bestParen2;
   my $pBestSubTree1;
   my $pBestSubTree2;
   my $pSubTree1 = $pSubTreeMin1;
   while (1){
      my $pSubTree2 = $pSubTreeMin2;
      while (1){
         my $paren1 = paren($pSubTree1);
         my $paren2 = paren($pSubTree2);
         if ($diTreeNoec{$paren1}{$paren2} >= $noecMax){ #// >= pour privilégier les
max.
            $noecMax = $diTreeNoec{$paren1}{$paren2};
            $bestParen1 = $paren1;
            $bestParen2 = $paren2;
            $pBestSubTree1 = $pSubTree1;
            $pBestSubTree2 = $pSubTree2;
            }#// if
#//print $pSubTree2;
         last if $pSubTree2==$pSubTreeMax2;
         last if ${$pSubTree2}[2] == 1;
         $pSubTree2 = ${$pSubTree2}[$#{$pSubTree2}]; #// on remonte dans l'arbre.
         } #// while
#//print "+\n";
      last if $pSubTree1==$pSubTreeMax1;
      last if ${$pSubTree1}[2] == 1;
      $pSubTree1 = ${$pSubTree1}[$#{$pSubTree1}]; #// on remonte dans l'arbre.
      } #// while
   my $index = recordTree($bestParen1, $bestParen2);
   $p1p2{$pBestSubTree1} = $pBestSubTree2; #// pour calculer les nouvelles pos.
   $p2p1{$pBestSubTree2} = $pBestSubTree1; #// pour calculer les nouvelles pos.
   }
#-------------------------------------------------------------------------------
--------------
sub recordTree{
   my $paren1 = shift;
   my $paren2 = shift;
   my $index;
   unless ($parenToIdx{$paren1}{$paren2}){         #// Si ce di-tree n'est pas déjà
connu.
      $diTreeNum++;                                        #// diTreeNum est une
variable globale.
      my $prefix = ($paren1 eq $nu or $paren2 eq $nu) ? ($paren2 eq
$nu?"S$lvl":"T$lvl") : "X$lvl";
      $index = sprintf("%s#%05x",$prefix,$diTreeNum);
      $parenToIdx{$paren1}{$paren2} = $index;          #// enregistrement d'un
nouveau di-tree.
      $idxToParen{$index} = "$paren1 $paren2";
      }
   $index = $parenToIdx{$paren1}{$paren2};
   $idxToNoec{$index}++;
#// print "$index\t",$diTreeNoec{$index},"\t$paren1 $paren2\n";
   return $index;
   }

#-------------------------------------------------------------------------------
---------------
sub paren{
   my $pTree = shift;
   return $nu unless $pTree;
   if (defined ${$pTree}[2]){ #// C'est bien un sous arbre à deux branches. (En [2],
on a la référence montante)
      return sprintf("(%s,%s)",paren(${$pTree}[0]),paren(${$pTree}[1]));
      }
   return ${$pTree}[0]; #// l'index du obj.
}

#-------------------------------------------------------------------------------
---------------
sub treeChain{
   my $lang = shift;
   my $ptree = shift;
   my $pp = shift;       #// réf de %p1p2 ou %p2p1

   my @tree = @{$ptree};
   my %pp = %{$pp};
```

```
   return ($ptree) if $pp{$ptree};

   if ($tree[2]){ #//contient une référence ou "1" donc deux branches.
      my @l = treeChain($lang,$tree[0],$pp);
      my @r = treeChain($lang,$tree[1],$pp);
      return  (@l, @r) if @l and @r;
      if (@l){
          $lang==1 ? recordTree(paren($tree[1]), $nu) : recordTree($nu,
paren($tree[1]));
          return (@l, $tree[1]);
         }
      if (@r){
          $lang==1 ? recordTree(paren($tree[0]), $nu) : recordTree($nu,
paren($tree[0]));
          return ($tree[0], @r);
         }
      }
      return ();
   }

#-------------------------------------------------------------------------------------
---------------
sub objString{
   my $l = shift() -1; #// language : 0 or 1
   my @treeChain = @{shift()};
   my %pp = %{shift()}; #// %p1p2 ou %p2p1

   return join(' ', map {  $parenToIdx{paren( $l ? $pp{$_} : $_ )}{paren( $l ? $_ :
$pp{$_} )}  } @treeChain);
   }

#-------------------------------------------------------------------------------------
---------------
sub ordpos{
   my $str=shift;
   my $cnt = $str=~tr/:/:/;
   return $cnt?$cnt:1000;
   }
#-------------------------------------------------------------------------------------
---------------
sub nicecoef{
   my $x = shift;
   return substr(log($x)/(log($x)-1),0,5);
   }
#-------------------------------------------------------------------------------------
---------------
sub drawtree{
   my $binode = shift;
   my @draw;
   my $deco ='';
#//   my $flagdeco = ($last > 0 and !ref(${$binode}[$last]));
   my $flagdeco = 1;
   if ($flagdeco){
#//      $deco = ${$binode}[$last];
      $deco = $binode;  #// affichage de la référence elle même.
#//      $deco = ($p1p2{$binode}?'+':'').'['.substr($binode,-
5,4).']'.($p2p1{$binode}?'+':'');  #// affichage de la référence elle même.
      $deco = ' 'x(1-length($deco)%2).'('.$deco.')'; #// longueur impaire
      }
#//   if (!ref(${$binode}[1])){
   if ($#{$binode}==1){
      my $data = ${$binode}[0];
      my $data = $data.' 'x(1-length($data)%2); #// longueur impaire
      my $width = length($data)>length($deco)?length($data):length($deco); #//
longueur impaire
      my $half = int $width/2 ; #// pair
      my $anchor = ' 'x $half.'|'.' 'x $half;
      my $justifdeco =  ' 'x int(abs($width-length($deco))/2);
      my $justifdata =  ' 'x int(abs($width-length($data))/2);
      my $line = '-'x $width;
      $deco = $justifdeco.$deco.$justifdeco;
      $data = $justifdata.$data.$justifdata;
      @draw = $flagdeco?($anchor,$deco,$line,$data):($anchor,$line,$data);
      return @draw;
      }
   my @pictL = drawtree(${$binode}[0]);
```

```
   my @pictR = drawtree(${$binode}[1]);
   my $widthL = length($pictL[0]);
   my $widthR = length($pictR[0]);
   my $half = ($widthL+$widthR)/2;
   my $halfL = int($widthL/2);
   my $halfR = int($widthR/2);
   my $anchor = ' 'x $half .'|'.' 'x $half;
   push @draw,($anchor);
   my $deco = ' 'x ($half-int(length($deco)/2)) .$deco.' 'x ($half-
int(length($deco)/2));
   push @draw,($deco) if $flagdeco;
   my $line = ' 'x $halfL.'+'.'-'x $halfR .'+'.'-'x $halfL.'+'.' 'x $halfR;
   push @draw,($line);
   my $height = $#pictL>$#pictR?$#pictL:$#pictR;
   my $deltaL = $height-$#pictL;
   my $deltaR = $height-$#pictR;
   foreach my $i (0..$height){
#//      push @draw, (($pictL[$i]?$pictL[$i]:' 'x $widthL).'
'.($pictR[$i]?$pictR[$i]:' 'x $widthR));
      push @draw, (($i>$deltaL?$pictL[$i-$deltaL]:$pictL[0]).'
'.($i>$deltaR?$pictR[$i-$deltaR]:$pictR[0]));
      }
   return @draw;
   }
#-------------------------------------------------------------------------------
---------------
sub heure{
   my @t = gmtime(time()+3600);
   return sprintf("[%02.0f:%02.0f:%02.0f]", $t[2],$t[1],$t[0]);
   }
```

trad.pl : Ce programme retraduit les segments source du corpus à partir des ressources collectées sur le corpus lui-même. Il n'utilise pas de classes et ne permet pas de faire de la traduction de segments inconnus. Les résultats de ce programme sont donnés dans l'annexe 5.

```
#// 26/05/2003 :
#// N appels à analys et un appel à synthes.
#// analys :
#// 1: montage des arbres (buildTree)
#// 2: détermination des unités (treeChain)
#// 3: Affectation des unités (pretagger et tagger)
#// synthes :
#//
#//
#// 12/05/2003 Placé et adapté pour l'arborescence raisonnée...
#//
#// 03/05/2003 XoF ESSAI avec addition plutôt que multiplication.
#// Digrammer : Deuxième programme permettant d'affecter une connaissance nouvelle à
des tokens en fonction de trigrammes.
#// Ce programme fonctionne avec la recherche dichotomique. Il traite 21 segments
par seconde et redigrammise à plus de 85 %.
#// Dans un premier temps, les bi/trigrammes possibles localement sont récupérés
avec leur nb d'oc en corpus.(NOEC)
#// Ensuite une routine récursive calcule les meilleures combinaisons globales, par
multiplication des scores de trigrammes
#// Le score d'un trigramme est simplement son NOEC.
#// En cas d'égalité, on test successivement le meilleur bidigramme puis le meilleur
digramme
#// L'application *au corpus d'apprentissage* retourne 86 % de segments correctement
redigrammisés.
#// Cela ne veut pas dire que les autres segments ne conduiraient pas à la bonne
traduction par un chemin détourné...
#// On ne peut rien tester à ce stade sur des segments inconnus.
#// Le vrai test ne peut être mené que sur la traduction elle-même.
#//-------------------------------------------------------------

use strict;

my $S = shift or die "What is the first language?\n";
my $T = shift or die "What is the second language?\n";
my ($L1, $L2) = sort {$a cmp $b} ($S, $T);                    #// On rétablit
l'ordre alpha des langues.
my $nu = "#null#";

my @S2Tfile;
my @S2Thandle=();
my @S2Tsize=();
my @NobjSfile;
my @NobjShandle=();
my @NobjSsize=();
my @STfile;
my @SThandle=();
my @STsize=();

foreach my $lvl (0..9){
   $S2Tfile[$lvl]   = "D:\\These\\dev\\data\\extract\\$L1$L2\\digram$lvl$S$T.dicho";
   $NobjSfile[$lvl] = "D:\\These\\dev\\data\\extract\\$L1$L2\\Ndigram$lvl$S.dicho";
   $STfile[$lvl]    = "D:\\These\\dev\\data\\extract\\$L1$L2\\digram$lvl.dicho";
   $S2Tsize[$lvl]   = (stat($S2Tfile[$lvl]))[7];
   $NobjSsize[$lvl] = (stat($NobjSfile[$lvl]))[7];
   $STsize[$lvl]    = (stat($STfile[$lvl]))[7];
   open($S2Thandle[$lvl],$S2Tfile[$lvl]);
   open($NobjShandle[$lvl],$NobjSfile[$lvl]);
   open($SThandle[$lvl],$STfile[$lvl]);
#print "$lvl : ",$S2Tfile[$lvl],$S2Thandle[$lvl],"\n";
#print "$lvl : ",$STfile[$lvl],$SThandle[$lvl],"\n";
   }

my $opt = shift;

if ($opt ne "-d"){
   my @input = @ARGV;
   my @XsrcX = ("#left#", $opt, @input, "#right#");
   my @analys = analys(0,@XsrcX);
   my $lvl = shift @analys;
   print "[$lvl] : ",join('+', @analys),"\n";
   my $trad = join(' ', map {synthes($lvl,$_)} @analys[1..$#analys-1]);
   print "$trad\n";
```

```
   exit;
   }

my $dir = shift;
my $min = shift;
my $max = shift;

my $sSegMarkup = "segment$S";
my $tSegMarkup = "segment$T";

my %noecA = ();   #// nb d'occurrences en corpus d'un digramme.
my %noecAB = ();  #// nb d'occurrences en corpus d'un bidigramme.
my %noecABC = (); #// nb d'occurrences en corpus d'un tridigramme.
my %digOk = ();   #// digrammes admissibles pour une graphie (hash double)
my %unknown = (); #// mots inconnus

my $nbr=0;
my $diff=0;
my $fileNb=0;

my $src;
#//my @XsrcX;
my $trg;

opendir(DIR,$dir);
while(my $file_name=readdir(DIR)){
   my $file = $dir."\\".$file_name;
   next if $file_name eq "PS459M01.xml";
   next if $file_name eq "AL007M02.xml";
   next if $file_name eq "AL012P55.xml";
#//   next unless $file_name eq "AL030P08.xml";
#//   next unless $file_name eq "AL021P13.xml";

   next unless -f($file);
   next if ++$fileNb >$max or $fileNb <$min; #// par prudence...
   print &heure, " N° $fileNb : $file_name\n";
   open(FILE,$file) or die "Pas de fichier $file !\n";
   my @XsrcX;
   while(<FILE>)
   {
      if (/<bisegment>/){
print "-----------------------------------------------------------------------\n";
         }
      elsif (/<$sSegMarkup>(.*)<\/$sSegMarkup>/) {
         $src = $1;
         @XsrcX = split / /,"#left# $src #right#";
         } #//elsif
      elsif (/<$tSegMarkup>(.*)<\/$tSegMarkup>/) {
         $trg = $1;
         } #//elsif
      elsif (/<\/bisegment>/) {
print "$src\n";
print "$trg\n";

         my $lvl = 0;
         my @analys = analys($lvl,@XsrcX);
         $lvl = shift @analys;
print "[$lvl] : ",join('+', @analys),"\n";
         my $trad = join(' ', map {synthes($lvl,$_)} @analys[1..$#analys-1]);
         my $ok = $trad eq $trg?'OK':'KO';
print "$ok=> $trad\n";
         $diff++ if $ok eq 'KO';
         $nbr++;
         print heure(), " $diff segments différents sur $nbr segments traités.\n" if
$nbr%100 == 0;
      }#// elsif
   }#// while FILE
   print heure(), " $diff segments différents sur $nbr segments traités.\n\n";
   close FILE;
   }#// while DIR
closedir DIR;
foreach my $lvl (0..9){
   close $S2Thandle[$lvl];
   close $NobjShandle[$lvl];
   close $SThandle[$lvl];
}
```

```
#//-------------------------------------------------------------------------------
--------------------
sub analys{
   my $lvl = shift;
   my @XchunkX = @_;
#print "Analyse de : ",join(' ',@XchunkX),"\n";
   if ($lvl){
      my @XsrcX = @XchunkX;
      my %Pos2pTreeMax = (); #// pos est entendu au sens large : pour un arbre
simple, la position de son objet plein.
      my %Pos2pTreeMin = (); #// pour un arbre complexe, la concaténation des
positions de ses obj. pleins, séparés par ':';
#print "Envoi de buidTree\n";
      my $pTree = buildTree($lvl,\@XsrcX,\%Pos2pTreeMax,\%Pos2pTreeMin);
print "[$lvl] Arbre : ", paren($pTree),"\n";
      my %pp = ();
      my %posOK = ();     #// hash des pos des ditrees retenus : pour éviter de
prendre les plus gros.
#//      my @posList = (sort {$a=~tr/:/:/<=>$b=~tr/:/:/} (keys %Pos2pTreeMax));
      my @posList = (sort {ordpos($a,$lvl)<=>ordpos($b,$lvl)} (keys %Pos2pTreeMax));
#// ordpos() met les pos simples devant ou derriere selon $lvl
print "[$lvl] Positions : ",join('+',@posList),"\n";
      foreach my $pos (@posList){
#//print "Test de la position $pos\n";
#//         next if $#posList and (1+index($pos,':') xor $lvl-1);  #// càd soit pos
simples (pour level 1) soit pos multiples (pour level>1)
#//print "type de pos autorisée\n";                                    #// mais ca passe
s'il n'y a qu'une pos.
         next if grep {$posOK{$_}} (split(/:/,$pos));           #// on saute si cet
arbre en contient de plus petits.
#//print "qui n'en contient pas de plus petites déjà vues\n";
         $Pos2pTreeMin{$pos} = $Pos2pTreeMax{$pos} unless ($#posList or $lvl<2); #//
au sommet (une seule pos) on veut le max.
         my @result = getBest($lvl,$Pos2pTreeMax{$pos},$Pos2pTreeMin{$pos});
#//print "resultat pour $pos :",join(' ',@result),"\n";
         next unless 1+$#result;
#//print "hello\n";
         my $pSubTree = shift @result;
         $pp{$pSubTree} = 1;
         map {$posOK{$_}=1} (split(/:/,$pos));                    #// on note les pos
retenues.
         }#// foreach
#// print join(' / ', (map {"$_=".$posOK{$_}} (keys %posOK))),"\n";
      my @treeChain = treeChain($pTree,\%pp);
      my @parenChain =  map {paren($_)} (@treeChain);
      return ($lvl-1,@XchunkX) if $#XchunkX-$#parenChain ==2 and $lvl>1; #// On
renonce si la chaine n'est pas réduite à lvl>1.
print "[$lvl] Découpage : ",join('+', @parenChain),"\n";
      @XchunkX = ("N".($lvl-1)."#00001",@parenChain,"N".($lvl-1)."#00002");
      }

#// print "Envoi de pretagger au niveau $lvl\n";
   pretagger($lvl,@XchunkX);
#// print "Envoi de tagger\n";
   my $pTagger = tagger(0,@XchunkX); #// renvoie une ref d'ARRAY : depth, score,
#left#, X,X,... X, #right#.
   my $depth = shift @{$pTagger};
   my $score = shift @{$pTagger};
   my @analys = @{$pTagger};
   print "[$lvl]($score,$depth) : ",join('+',@analys), "\n";
   return analys($lvl+1,@analys) if $#analys>2;
   return ($lvl,@analys);
   }

#//-------------------------------------------------------------------------------
--------------------
sub synthes{
   my $lvl = shift;
   my $id = shift;
# print "synthes de $id au niveau $lvl\n";
   return 'x' if $id eq "#null#";
   my $U = $L1 eq $S ? 'S' : 'T';
   return desperateTrans($id,$lvl) if $id=~/^$U/ ;
#// print "[$id]\n";
   my @info = split /\t/, getinfo($SThandle[$lvl],$id,0,$STsize[$lvl]);
print "[$lvl] Impossible de trouver $id dans ",$SThandle[$lvl],"\n" unless @info;
   return 'x' unless @info;
```

```
   my $result = (split / /, $info[1])[$T eq $L2]; #// astuce pour déterminer la
trad.
   $result=~s/\(|\)*//g;
   $result=~s/\,/ /g;
# print "[$lvl] $result\n";
=remplace par desperateTrans
   if ($result eq '#null#'){
      my $scorie = (split / /, $info[1])[$T eq $L1]; #// on repart de l'expression
elle meme.
      $scorie=~s/\(|\)*//g;
      $scorie=~s/\,/ /g;
      $result = join (' ',(map {
         (split / /, (split /\t/,
getinfo($S2Thandle[$lvl],$_,0,$S2Tsize[$lvl]))[1])[0];
         }(split / /, $scorie)));
      return $result;
      }
=cut
   return $result unless $lvl;
   my @result = split / /, $result;
   return join ' ', map {synthes($lvl-1,$_)} @result;
   }

sub desperateTrans{
   my $id = shift;
   my $lvl = shift;
   if ($lvl == -1){
      my $try = (split / /, (split /\t/,
getinfo($S2Thandle[0],$id,0,$S2Tsize[0]))[1])[$L2 eq $T];
      return "*$id*" unless $try;
      return $try unless $try eq '#null#';
      $try = (split / /, (split /\t/,
getinfo($S2Thandle[0],$id,0,$S2Tsize[0]))[2])[$L2 eq $T];
      return $try?$try:"*$id*";
      }
   return (split / /, (split /\t/, getinfo($S2Thandle[0],$id,0,$S2Tsize[0]))[1])[$L2
eq $T] if $lvl == -1;
   my $trans = (split / /, (split /\t/,
getinfo($SThandle[$lvl],$_,0,$STsize[$lvl]))[1])[$L2 eq $S];
print "[$lvl] Desperate : $id -> $trans\n";
#   return $trans unless $lvl+1;
   $trans=~s/\(|\)*//g;
   $trans=~s/\,/ /g;
   return map {desperateTrans($_,$lvl-1)} (split / /,$trans);
   }

#//-------------------------------------------------------------------------------
--------------------
#// Montage des arbres.
#// La plupart des infos nécessaires sont déjà connues : les trigrammes ont été
montés pour faire le taggage.
#// Mais il faut encore chercher le dernier bigramme, et les éventuels mots
inconnus. Les tri/bi grammes correspondants
#// aux mots inconnus sont déjà connus par construction.
#//-------------------------------------------------------------------------------
--------------------
sub buildTree{
   my $lvl = shift;
   my @XobjX = @{shift()};
   my $pPos2pTreeMax = shift;          #// Arbre minimale (feuille pour simple) :
pointeur sur un hash de pointeurs.
   my $pPos2pTreeMin = shift;          #// Arbre maximale : pointeur sur un hash de
pointeurs.
   my @sec;

   my @info = split('\t',getinfo($NobjShandle[$lvl-1],$XobjX[$#XobjX-1].'
'.$XobjX[$#XobjX],0,$NobjSsize[$lvl-1]));
   $noecAB{$XobjX[$#XobjX-1]}{$XobjX[$#XobjX]} = $info[0];

   foreach my $i (1..$#XobjX-2){
      if (!$noecA{$XobjX[$i]}){
         my @info = split('\t',getinfo($SThandle[$lvl-1],$XobjX[$i],0,$STsize[$lvl-
1]));
         $noecA{$XobjX[$i]} = $info[0];
         }
      if (!$noecA{$XobjX[$i+1]}){
```

```
         my @info = split('\t',getinfo($SThandle[$lvl-
1],$XobjX[$i+1],0,$STsize[$lvl-1]));
#//print "getinfo dans ",$SThandle[$lvl-1]," pour  ", $XobjX[$i+1]," : ",$info[0]
,"\n";
         $noecA{$XobjX[$i+1]} = $info[0];
         }

=print
print "$XobjX[$i-1] $XobjX[$i] $XobjX[$i+1] : ", $noecABC{$XobjX[$i-
1]}{$XobjX[$i]}{$XobjX[$i+1]},"\n";
print "$XobjX[$i-1] $XobjX[$i] : ", $noecAB{$XobjX[$i-1]}{$XobjX[$i]},"\n";
print "$XobjX[$i+1] : ", $noecA{$XobjX[$i+1]},"\n";
print "$XobjX[$i] $XobjX[$i+1] $XobjX[$i+2] : ",
$noecABC{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]},"\n";
print "$XobjX[$i+1] $XobjX[$i+2] : ", $noecAB{$XobjX[$i+1]}{$XobjX[$i+2]},"\n";
print "$XobjX[$i] : ", $noecA{$XobjX[$i]},"\n";
=cut

    if ($noecABC{$XobjX[$i-1]}{$XobjX[$i]}{$XobjX[$i+1]} and
$noecABC{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]}){
       my $sec1 = $noecABC{$XobjX[$i-
1]}{$XobjX[$i]}{$XobjX[$i+1]}**3/($noecAB{$XobjX[$i-
1]}{$XobjX[$i]}**2*$noecA{$XobjX[$i+1]});
       my $sec2 =
$noecABC{$XobjX[$i]}{$XobjX[$i+1]}{$XobjX[$i+2]}**3/($noecA{$XobjX[$i]}*$noecAB{$Xob
jX[$i+1]}{$XobjX[$i+2]}**2);
       push @sec, (nicecoef($sec1*$sec2));
       }
    elsif($noecAB{$XobjX[$i]}{$XobjX[$i+1]}){
       my $sec0 =
$noecAB{$XobjX[$i]}{$XobjX[$i+1]}**2/($noecA{$XobjX[$i]}*$noecA{$XobjX[$i+1]});
       push @sec, (nicecoef($sec0));
       }
    else {
       push @sec, (1);
       }
     }#// foreach
   my @obj = @XobjX[1..$#XobjX-1];
   return ${recurTree(\@obj,\@sec,0,1,$pPos2pTreeMax,$pPos2pTreeMin)}[0];
   }

#-------------------------------------------------------------------------------
---------------
sub recurTree{
   my @seg = @{shift()};
   my @sep = @{shift()};
   my $pos = shift;                    #// Position dans le segment d'origine du
premier élément passé en @seg
   my $pNode = shift;                  #// Pointeur sur le noeud supérieur.
   my $pPos2pTreeMax = shift;          #// pointeur sur un hash de pointeurs.
   my $pPos2pTreeMin = shift;          #// pointeur sur un hash de pointeurs.
   my $pSubTree = [];

   if ($#seg==0){ #// le segment est réduit à un objet. Sa position dans le segment
d'origine est $pos.
      push @{$pSubTree}, ($seg[0],$pNode);    #// !!! LA REF. DU NOEUD SUPERIEUR EST
À LA FIN DE L'ARRAY !!!
      if ($seg[0]=~/^X/){              #// cet objet est plein
         ${$pPos2pTreeMax}{$pos}=$pSubTree;  #// on enregistre sa position
         ${$pPos2pTreeMin}{$pos}=$pSubTree;  #// on enregistre sa position
         }
      else {$pos=-1}
      return [$pSubTree,$pos]; #// on renvoie la ref d'un array contenant l'objet et
sa position s'il est plein
      }
   my $m = maxindex(@sep); #// la plus grande coupure s'effectue sur le coef de
sécabilité le plus fort.
   my @segL=@seg[0..$m];
   my @sepL=@sep[0..$m-1];
   my @segR=@seg[$m+1..$#seg];
   my @sepR=@sep[$m+1..$#sep];

   my $pSubTreeL =
recurTree(\@segL,\@sepL,$pos,$pSubTree,$pPos2pTreeMax,$pPos2pTreeMin);
   my $pSubTreeR =
recurTree(\@segR,\@sepR,$pos+$m+1,$pSubTree,$pPos2pTreeMax,$pPos2pTreeMin);
```

```
   my $posL = ${$pSubTreeL}[1];
   my $posR = ${$pSubTreeR}[1];

   push @{$pSubTree}, (${$pSubTreeL}[0],${$pSubTreeR}[0],$pNode); #// !!! LA REF. DU
NOEUD SUPERIEUR EST À LA FIN DE L'ARRAY !!!

   if ($posL==-1 and $posR ==-1){return [$pSubTree,-1]} #// C'est vide
   if ($posR==-1) {$pos=$posL}      #// dépendance à droite
   elsif ($posL==-1) {$pos=$posR}   #// dépendance à gauche
   else {$pos = "$posL:$posR"}      #// 2 sous arbres pleins

#//   unless(1+index($pos,':') and $lvl==1){                                  #// sauf
si on est au level 1 avec une pos multiple.
      ${$pPos2pTreeMin}{$pos}=$pSubTree unless ${$pPos2pTreeMin}{$pos}; #// Arbre
minimal.
      ${$pPos2pTreeMax}{$pos}=$pSubTree;                                    #// Arbre
maximal.
#//      }
   return [$pSubTree,$pos];
  }


#//-------------------------------------------------------------------------------
-------------------
#// Pretagger fait toutes les recherches d'info extérieures. Elle en fait le moins
possible.
#// Elle prépare et restreint l'espace de recherche de tagger.
#// Pas franchement satifaisant... Il y a deux boucles successives sur le segment,
on n'élimine pas vraiment
#// les cas impossibles, on ne fait pas une recherche optimale en cas de mot inconnu
(meilleur C de ABC).
#// On doit pouvoir faire nettement mieux.

sub pretagger{
   my $lvl = shift;
   my @XsegX = @_;

#//print "Pretagger de :\n",join('+',@XsegX),"\n";
   %noecA = ();
   %noecAB = ();
   %noecABC = ();
   %digOk = ();
   my %dig = ();
   foreach my $i (0..$#XsegX){
      next if $dig{$XsegX[$i]}; #// On connaît déjà les digrammes de cette graphie
      my $diginfo = getinfo($S2Thandle[$lvl],$XsegX[$i],0,$S2Tsize[$lvl]); #// info
des digrammes pour cette graphie
#// print "($lvl) Recherche de ",$XsegX[$i], " dans ",$S2Thandle[$lvl], "
(",$S2Tsize[$lvl],") : $diginfo\n";
      if (!$diginfo) {
         $unknown{$XsegX[$i]}=1;
         next;
         }
      my @diginfo =  split(/\t/,$diginfo); #// array des infos
      shift @diginfo; #// le noec général de la source.
      my $best;
      my $max = 0;
      foreach my $info (@diginfo){
         my @info = split(/ /,$info);
         push @{$dig{$XsegX[$i]}}, $info[0]; #// nouveau digramme pour cette
graphie.
         $noecA{$info[0]} = $info[2]; #// compte pour ce digramme.
         $best  = $info[0] if $info[2]>$max;
         } #// foreach
      $digOk{$XsegX[$i]}{$best}=1; #// rajouté à la main pour avoir au moins une
hypothèse par mot, le dernier en particulier.
   }
   my @flag = ();
   my $noTrig = 1; #//
   my $noTrig2 = 1; #// Mémoire du NoTrig précédent.
   my $noTrig3 = 1; #// Mémoire du NoTrig d'avant.
   foreach my $i (0..$#XsegX-2){#// pour chaque position du segment
      $noTrig = 1;     #// Est-ce qu'on va trouver un trigramme convenable ?
      my $maxA   = 0 ;   #// Le meilleur score de digramme, calculé si on n'a aucun
bigramme.
      my $bestA  = ''; #// Le meilleur digramme, retenu si on a aucun bigramme.
      my $bestW1 = '';
```

```
      my $maxAB  = 0 ;   #// Le meilleur score de bigramme, calculé si on n'a aucun
trigramme.
      my $bestB  = ''; #// Le meilleur digramme (en bigramme), retenu si on a aucun
trigramme.
      my $bestW2 = '';
      foreach my $A (@{$dig{$XsegX[$i]}}){#// pour chaque digramme admissible à
cette position
         foreach my $B (@{$dig{$XsegX[$i+1]}}){#// pour chaque digramme admissible à
la position suivante
            next unless $noTrig2 or ${$flag[$i]}{"$A $B"}; #// Bidig non testé
(entre autre si $i=0) ou autorisé.
#// Recherche et ecriture de l'info.
            unless ($noecAB{$A}{$B}){ #// càd : sauf si on a déjà l'info.
                my $bidiginfo = getinfo($NobjShandle[$lvl],"$A
$B",0,$NobjSsize[$lvl]); #// info du bigramme : tous ses trig. connus
                next unless $bidiginfo; #// si pas d'info sur ce bigramme, on saute.
                my @bidiginfo = split(/\t/,$bidiginfo); #// array des infos
                $noecAB{$A}{$B} = shift @bidiginfo; #// NOEC du bigramme.
                foreach my $j (0..$#bidiginfo){
                   my @info = split(/ /,$bidiginfo[$j]);
                   $noecABC{$A}{$B}{$info[0]} = $info[1]; #// NOEC de chaque
trigramme de ce bigramme
                }
                }#// unless
#// fin de Recherche et écriture.

            if ($unknown{$XsegX[$i+2]}){ #// Ce mot est inconnu. Quid si mot inconnu
en 2e position ?
               my $bestC = (sort {$noecABC{$A}{$B}{$b} <=>
$noecABC{$A}{$B}{$a}}(keys %{$noecABC{$A}{$B}}))[0];
               push @{$dig{$XsegX[$i+2]}}, $bestC unless grep {$bestC eq $_}
@{$dig{$XsegX[$i+2]}};
#// print "$bestC ~> [", $XsegX[$i+2], "]\n";
               }
            foreach my $C (@{$dig{$XsegX[$i+2]}}){ #// pour chaque digramme
admissible pour la troisième position
               if ($noecABC{$A}{$B}{$C}){
                  ${$flag[$i+1]}{"$B $C"} = 1; #// Ce bigramme est la fin du
trigramme précédent.
                  $digOk{$XsegX[$i]}{$A} = 1;
                  $digOk{$XsegX[$i+1]}{$B} = 1;
                  $digOk{$XsegX[$i+2]}{$C} = 1;
#// print "$A $B $C -> [", $XsegX[$i], '][', $XsegX[$i+1], '][', $XsegX[$i+2], "] :
trigramme\n";
                  $noTrig = 0; #// On a un trigramme.
                  } #// if
               } #// foreach $C

            if ($noecAB{$A}{$B}>$maxAB){
               $maxAB  = $noecAB{$A}{$B};
               $bestW1 = $XsegX[$i];
               $bestA  = $A;
               $bestW2 = $XsegX[$i+1];
               $bestB  = $B;
            }#// if
         } #// foreach $B
         if ($noecA{$A}>$maxA){
            $maxA   = $noecA{$A};
            $bestW1 = $XsegX[$i];
            $bestA  = $A;
            }#// if
         } #// foreach $A
      if ($noTrig){
         $digOk{$bestW1}{$bestA} = 1 if ($noTrig3 and $noTrig2);
#// print "$bestA -> [", $bestW1,"] : pas de trigramme, noTrig, noTrig2 et no
Trig3.\n" if ($noTrig3 and $noTrig2);
         $digOk{$bestW2}{$bestB} = 1 if ($noTrig2 and $maxAB>0);
#// print "$bestB -> [", $bestW2,"] : pas de trigramme, noTrig2 et des bigramme.\n"
if ($noTrig2 and $maxAB>0);
         }
#// print "noTrig : $noTrig\n";
      $noTrig3 = $noTrig2; #//Statut de la recherche de tridigramme.
      $noTrig2 = $noTrig; #//Statut de la recherche de tridigramme.
      } #// foreach
   }
```

```
#//-------------------------------------------------------------------------------
--------------------
#// Optimisation récursive du tagage du segment. Pas trop mal. On se contente de la
somme des NOEC rencontrés,
#// élevés au cube pour un trigramme et au carré pour un bigramme. Le NOEC des
digrammes (unigrammes) est pris tel quel.
#// Les solution utilisant le plus de trigrammes (moins de recours aux bigrammes ou
digrammes -unigrammes-) est préférée.

sub tagger{
   my $depth = shift;  #// 0, 1, ou 2. ambition d'analyse. C'est également le nombre
de digrammes "given".
   my @given = splice(@_,0,$depth); #// les digrammes connus à ce niveau.
   my $wrd = shift;    #// le mot à déterminer
   my @seg = @_;       #// la suite du segment (en mots)
   my $depthSum;       #// la somme des profondeurs utilisées.
   my $dp_min = 0;     #// la profondeur à atteindre... $dp_min est négatif. On lui
compare -$depthSum
   my $sc_max = 0;     #// le score à battre !
   my $score;          #// le score courant.
   my $pTagger;        #// pointeur sur le resultat de tagger.
   my $pBest;          #// pointeur sur le meilleur résultat de tagger
   my $NoDig = 1;      #// flag : Est-ce qu'un des digrammes testés pour le mot va
convenir à scette profondeur ?

#// print "Tagger : $depth ", join(' ',@given) ," > $wrd < ", join(' ',@seg)
,"\n\n";

   foreach my $dig (keys %{$digOk{$wrd}}){   #// Digrammes retenus pour cette
graphie ($wrd)
      $score =
$depth==2?$noecABC{$given[0]}{$given[1]}{$dig}**3:($depth==1?$noecAB{$given[0]}{$dig
}**2:$noecA{$dig});
#// print "Score de $wrd avec $dig  : $score\n\n";

      next unless $score;               #// On saute si le score local est nul.
      $NoDig = 0;                       #// Le digramme convient à cette
profondeur.

#// récursivité si @seg non vide. $depth incrémenté (max à 2). @given avancé, reculé
ou vide (astuce). $dig est rajouté.
      $pTagger = $#seg<0 ? [0] : tagger($depth<2?$depth+1:2,$depth?@given[$depth-
1..$#given]:(),$dig,@seg);

      $depthSum = $depth + ${$pTagger}[0]; #// attention à ne pas accumuler $depth
dans la boucle.
      $score += ${$pTagger}[1];

      if (-$depthSum < $dp_min or -$depthSum == $dp_min && $score > $sc_max){  #//
Que faire en cas d'égalité ?
         $dp_min = -$depthSum;
         $sc_max = $score;
         splice @{$pTagger},0,2,($depthSum,$score,$dig); #// On substitue
($depthSum,$score,$dig) à ($depthSum,$score) de d.
         $pBest = $pTagger;                  #// On stocke ce résultat.
         }
      } #// foreach
   if ($NoDig){ #// Si aucun digramme de ce mot ne convient
#//       return [0] unless $depth; #// si $depth = 0, on ne peut plus descendre...
      return $#seg<0 ? [0] : tagger(0,@seg) unless $depth; #// Si depth = 0, on
saute.
      return tagger($depth-1,@given[1..$#given],$wrd,@seg); #// On relance tagger
avec une moindre ambition.
      }
   return $pBest;
   }

#-------------------------------------------------------------------------------
----------------
sub getBest{
   my $lvl = shift;
   my $pSubTreeMax = shift;
   my $pSubTreeMin = shift;
#// print "de $pSubTreeMin à $pSubTreeMax.\n";
   my $noecMax = 0;
   my @bestResult;
   my $pSubTree = $pSubTreeMin;
```

```
   while (1){
      my $paren = paren($pSubTree);
      my $info = getinfo($S2Thandle[$lvl],$paren,0,$S2Tsize[$lvl]);
#// printf "(%d) %s dans %s (%d) :
%s\n",$lvl,$paren,$S2Thandle[$lvl],$S2Tsize[$lvl],$info;
      if ($info){
         my @info = split /\t/ ,$info;
         my $noec = shift @info;
         if ($noec >= $noecMax){
            $noecMax = $noec;
            @bestResult = ($pSubTree,@info);
            }
         }
      last if $pSubTree==$pSubTreeMax;
      last if ${$pSubTree}[2] == 1;
      $pSubTree = ${$pSubTree}[$#{$pSubTree}]; #// on remonte dans l'arbre.
      } #// while
   return @bestResult;
   }
#-------------------------------------------------------------------------------
---------------
sub treeChain{
   my $ptree = shift;
   my $pp = shift;

   my @tree = @{$ptree};
   my %pp = %{$pp};

   return ($ptree) if $pp{$ptree};

   if ($tree[2]){ #//contient une référence ou "1" donc deux branches.
      my @l = treeChain($tree[0],$pp);
      my @r = treeChain($tree[1],$pp);
      return  (@l, @r) if @l and @r;
      return (@l, $tree[1]) if @l;
      return ($tree[0], @r) if @r;
      }
   return ();
   }


#-------------------------------------------------------------------------------
---------------
sub paren{
   my $pTree = shift;
   return $nu unless $pTree;
   if (defined ${$pTree}[2]){ #// C'est bien un sous arbre à deux branches. (En [2],
on a la référence montante)
      return sprintf("(%s,%s)",paren(${$pTree}[0]),paren(${$pTree}[1]));
      }
   return ${$pTree}[0]; #// l'index du obj.
}


#//-----------------------------------

#// La routine getinfo implémente une recherche dichotomique récursive.
#// Les lignes sont classées dans l'ordre hexa des caractères de la clé.
#// La première ligne du fichier est ignorée.
#// Les données sont séparées de la clé de recherche par un \t.
#// Seules les données sont renvoyées ou undef si la clé n'est pas dans le fichier.

sub getinfo{
   my $ref = shift; #// handle du fichier de data
   my $key = shift; #// clé à chercher
   my $min = shift; #// bas de la zone à chercher
   my $max = shift; #// haut de la zone à chercher
   return undef if $max-$min<2; #// renvoie undef si la clé est absente du fichier
   my $mid = int(($max+$min)/2);
   seek($ref, $mid, 0); #// fixe la nouvelle position de lecture.
   readline($ref); #// permet de se caler sur une ligne. La première ligne du
fichier est ignorée.
   my $line = readline($ref);
   chomp $line;
   $line=~s/\x0D//;
   my @line = split /\t/,$line; #// on sépare la clé des données.
   if ($line[0] gt $key) {
```

```
      return getinfo($ref,$key,$min,$mid); #// relance de la recherche dans la
moitié précédente
   }
   if ($line[0] lt $key) {
      return getinfo($ref,$key,$mid,$max); #// relance de la recherche dans la
moitié suivante
   }
return join("\t",@line[1..$#line]); #// renvoie les données.
}

#//-----------------------------------
#// routines ancilaires

sub maxindex {
   my $maxindex = 0;
   foreach my $i (0..$#_){
      $maxindex = $i if $_[$i]>$_[$maxindex];
   }
   $maxindex;
}

sub ordpos{
   my $str=shift;
   my $lvl = shift;
   my $cnt = $str=~tr/:/:/;
   return $cnt?$cnt:1000 if $lvl-1;
   return $cnt;
   }


sub nicecoef{
   my $x = shift;
   return '1' if $x==0;
   return sprintf("%1.3f", log($x)/(log($x)-1)); #// pour étaler les petits coef, il
faut remplacer 1 par 2, 3 ou même 3291...
   }                                             #// ce qui revient au même que
d'augmenter la base du log.

sub heure{
   my @t = gmtime(time()+3600);
   return sprintf("[%02.0f:%02.0f:%02.0f]", $t[2],$t[1],$t[0]);
   }
```

geneclass.pl Ce programme nous a permis de tester l'algorithme de classification génétique avec une fonction de performance de remplacement.

```
# Algorithme génétique de répartion en classes.
# @P est la population. @chromo = @{$P[n]} est le chromosome n
# Le chromosome 0 est temporaire : c'est un rejeton dans les limbes : singularité en
test, non évalué
# @{$chromo[i]} est la valeur du gène i (>0) du chromosome (un identifiant de
classe)
# @{$chromo[0]} contient les méta données du de @chromo
# 0) nom du chromosome
# 1) performance
# 2) nombre de classes
# 3) génération d'apparition
# 4) & 5) parents du chromosomes
# 6) flag de doublon en perf.
# 7) nombre de mutations pratiquées

# On lance l'application avec les paramètres séparés par des blancs.
# REMARQUES :
# La valeur par défaut s'appliquent pour les paramètres non renseignés à droite, et
pour les valeurs 0 et "".
# Les valeurs 0.0 (ou .0 ou 0.), '' (chaine vide) et " " (chaine de blancs)
suplantent les valeurs par défaut.
# ATTENTION : deux guillemets simples séparées (' ') sont interprétées comme deux
paramètres...

use strict;

my @E;                                          # ensemble d'éléments à partitionner
my @P;                                          # ensemble de chromosomes (population)

$#E      = (shift || 100);                     # Cardinal de l'ensemble (pas d'élément
0)
my $V    = (shift || 11) ;                       # nombre de valeurs possibles dans
l'ensemble.
my $K    = (shift || 11) ;                       # nombre de classes
my $G    = (shift || 1000000) ;                 # Nombre maximal de générations.
$#P      = (shift || 32);                       # Effectif de la population
my $T    = (shift || 1/2);                      # Proportion de la population conservée
à chaque génération (les meilleurs)
my $M    = (shift || 1/sqrt($#E));              # Nombre de mutations initiale pour un
chromosome
my $alpha= (shift || 3);                        # seuil d'arrêt :
my $beta = (shift || 100);                      # seuil d'arrêt : si génération
courante = alpha*Gen[P[1]] + beta
my $Ftr  = (shift || "geneclass.trace.$V.$#E.txt"); #Fichier de trace.

my $Pt   = int ($#P*$T);                        # Effectif des meilleurs
my $gen;                                        # generation courante

my $MutBest;                                    # taux de mutation observée parmi les
meilleurs chromosomes.
my $MutTot;                                     # taux de mutation observée parmi tous
les chromosomes.
my $MutP = $Pt;                                 # frontière pour la mutation entre les
meilleurs et les pires
my $deltaMut;                                   # delta entre le nombre réel de
mutations et le nombre probable de mutations
my $MutProb = $M*$#E;                           # Nombre de mutations probables pour un
gène dans un chromosome.
my $MutReel = 0;                                # Nombre de mutations reelles moyennes
des genes pour un chromosome.

my $epsi = 1/10000000;                          # epsilon pour juger de l'égalité de
deux perf.

my $trace;

generate_data();

populate();
printf "\n\n%s Genese\n", &heure;

open(OUT,">".$Ftr);
foreach $gen (0..$G){
   evaluation();
```

```
   evaluationMut();
   system "cls";
   printf "\n\n%s Card=%d NbValeurs=%d
NbClasses=%d\t\tGen=%d\t(MutReel=%.2f\tMutVoulue=%.2f)\n\n",
&heure,$#E,$V,$K,$gen,$MutReel,$MutProb;
   foreach my $chromo (1..$Pt){
      my @chromo = @{$P[$chromo][0]};
      printf "%04d)\tAge=%d\t[%s]\tPerf=%.6f\tClassNbr=%d\tGen=%d\t([%s,%s] flag=%d
Mut=%d)\n",$chromo,$gen-$chromo[3],@chromo;
   }
   if ($gen-$P[1][0][3] == 1){
      $trace = sprintf "%s\t%d\t%.7f\t%.5f\n",&heure, $gen, $P[1][0][1],$M;
      $trace =~s/\./\,/g; #Parce que EXCEL et 123 utilisent la virgule comme
séparateur décimal.
      printf OUT $trace;
   }
   if (($gen >= $alpha*$P[1][0][3]+$beta) || ($P[1][0][1]==0)){
      trace();
      exit;
    }
   generation($gen,$M);
}
close OUT;

#-------------------------------------------------------------------------------
------------

sub evaluation{
    foreach my $c (1..$#P){
       $P[$c][0][1]=perf($c);
    }
    $P[0][0][0]="CXX";
    $P[0][0][1]=-1;
    $P[0][0][2]=-1;
    $P[0][0][6]=0;

    @P = sort {${$a}[0][1] <=> ${$b}[0][1]
                            ||
               ${$a}[0][2] <=> ${$b}[0][2]
                            ||
               ${$a}[0][3] <=> ${$b}[0][3]  } @P;

    map {$P[1+$_][0][6] = ($P[1+$_][0][1]-$P[$_][0][1] <$epsi)}(1..$#P-1);

    @P = sort {${$a}[0][6] <=> ${$b}[0][6]
                            ||
               ${$a}[0][1] <=> ${$b}[0][1]
                            ||
               ${$a}[0][2] <=> ${$b}[0][2]  } @P;

}

#-------------------------------------------------------------------------------
------------
sub evaluationMut{

   $MutTot = 0;
   map {$MutTot += $P[$_][0][7]}(1..$#P);
   $MutReel = $MutTot/$#P;
   $deltaMut = $MutProb-$MutReel;

   $MutBest = $MutP*.5;                          # Tare

   map {$MutBest += $P[$_][0][7]}(1..$MutP);

   $M = ($MutBest/$MutP+$deltaMut)/$#E;       # Nouvelle probabilité de mutation
avec correctif anti dérive.
   $M = $M>0 ? $M : 1/(2*$#E);                # Garde fou anti zéro : 1 mutation
pour 2 chromosomes.
   $MutProb = $#E*$M;
}
#-------------------------------------------------------------------------------
------------

sub generation{
    my $gen = shift;
    my $M = shift;
```

```
    foreach my $c ($Pt+1..$#P){
       my ($chromo1,$chromo2);
       my $mut;
       my $classNb;

      ($chromo1,$chromo2) = draft($Pt,2);      # donne deux nombres différents entre
1 et Pt
       $mut = hybrid($c,$chromo1,$chromo2,$M); # Création d'un nouveau chromosome à
l'indice $c;

       $P[$c][0][1]=0;                           # 1) performance
#       $P[$c][0][2]=classNb($c);                 # 2) nombre de classes
       $P[$c][0][2]=-1;                          # 2) nombre de classes
       $P[$c][0][3]=$gen;                        # 3) génération d'apparition
       $P[$c][0][4]=$P[$chromo1][0][0];          # 4) & 5) parents du chromosomes
       $P[$c][0][5]=$P[$chromo2][0][0];
       $P[$c][0][6]=0;                           # 6) Flag de doublon (en perf)
       $P[$c][0][7]=$mut;                        # 7) Nombres de mutation
pratiquées.
    }
}

#-------------------------------------------------------------------------------
------------

sub hybrid{
    my ($c,$chromo1,$chromo2,$M) = @_;
    my $mut = 0;
    foreach my $e (1..$#E){
       my $r = rand();
       if ($r > $M){
          $P[$c][$e] = (rand > .5) ? $P[$chromo1][$e] : $P[$chromo2][$e];
          }
       else {
          $P[$c][$e] = int(1+ rand $K);
          $mut++;
       }
    }
    return $mut;
}

#-------------------------------------------------------------------------------
------------

sub classNb{
    my $chromo = shift;
    my @class= ();
    foreach my $e (1..$#E){
       $class[$P[$chromo][$e]]++;
    }
    my $q = grep {$class[$_]!=0} (1..$K);
    return $q
}

#-------------------------------------------------------------------------------
------------

sub diff{
  my ($chromo1,$chromo2) = @_;
  return (grep {$P[$chromo1][$_]!=$P[$chromo2][$_]}(1..$#E)); # nombre de
différences.
}

#-------------------------------------------------------------------------------
------------

sub draft{
    my $max = shift;
    my $nb = shift;
    my %check = ();
    my @lst = ();
    foreach my $i (0..$nb-1){
       do {
          $lst[$i] = int(1 + $max*rand()**2); #// le carré pour augmenter le choix
des tout meilleurs
#          $lst[$i] = int(1 + $max*rand());
       } until (!$check{$lst[$i]});
```

```
        $check{$lst[$i]} = 1;
    }
    return @lst;
}

#-------------------------------------------------------------------------------
---------

sub populate{
    foreach my $c (1..$#P){
       my $name = sprintf "C%02X",$c;
       foreach my $g (1..$#E){
          $P[$c][$g] = int(1+(rand $K));
       }
       $P[$c][0]=[$name,-1,classNb($c),0,"CXX","CXX",0,$M*$#E];
    }
}

#-------------------------------------------------------------------------------
------------

sub trace{
   my $str = join("\t",@E);
   print OUT "\t\t$str\n\n";
   foreach my $c (1..$#P){
      my @chromo = @{$P[$c]};
      shift @chromo;
      printf OUT "%s\t%.5f\t%d\t%s\n", $P[$c][0][0], $P[$c][0][1], $P[$c][0][2],
join("\t",@chromo);
   }
   print OUT "\n";
}

#-------------------------------------------------------------------------------
------------

sub generate_data{
    foreach my $e (1..$#E){
       $E[$e] = int(1+ rand $V);
    }
}

#-------------------------------------------------------------------------------
------------

sub perf{
   my $chromo = shift;
   my @chromo = @{$P[$chromo]};
   my @classSum = ();         #somme des élements de la classe
   my @classEltNbr = ();      #nombre d'éléments dans la classe
   my @classAvg = ();         #moyenne de la classe
   my @classSumDev = ();      #somme des carrés des écarts de la classe
   my @classStdDev = ();      #ecart type de la classe
   my $sumStdDev = 0;         #somme des ecarts types


   foreach my $e (1..$#E){      #// $chromo[0] contient les infos de ce chromosome.
      $classSum[$chromo[$e]] += $E[$e];
      $classEltNbr[$chromo[$e]] +=1;
      }
   foreach my $k (1..$K){
      $classAvg[$k] = $classSum[$k]/$classEltNbr[$k] unless $classEltNbr[$k]==0;
#     $classAvg[$k] = int($classSum[$k]/$classEltNbr[$k]) unless
$classEltNbr[$k]==0; #Partie entière : flop
      }
   foreach my $e (1..$#E){      #// $chromo[0] contient les infos de ce chromosome.
#      $classSumDev[$chromo[$e]] += ($classAvg[$chromo[$e]] - $E[$e])**2;
      $classSumDev[$chromo[$e]] += abs($classAvg[$chromo[$e]] - $E[$e]);
#      $classSumDev[$chromo[$e]] += ($classAvg[$chromo[$e]] == $E[$e]) ? 0 : 1 ;
#Partie entière : flop
      }
   foreach my $k (1..$K){
      $classStdDev[$k]=0;
#      $classStdDev[$k] = sqrt($classSumDev[$k]/$classEltNbr[$k]) if
$classEltNbr[$k];
      $classStdDev[$k] = $classSumDev[$k]/$classEltNbr[$k] if $classEltNbr[$k];
```

```
#      $classStdDev[$k] = $classSumDev[$k];
#Partie entière : flop
       $sumStdDev += $classStdDev[$k];
       }
   return $sumStdDev;
}


#-------------------------------------------------------------------------------------------
----------------
sub heure{
   my @t = gmtime(time()+3600);
   return sprintf("[%02.0f:%02.0f:%02.0f]", $t[2],$t[1],$t[0]);
   }
```

3. Feuilles de style XSL

TransTree.interactiveHTML.xsl Cette feuille de style XSL suivante permettant d'obtenir la visualisation dynamique interactive en HTML à partir d'un fichier XTR (TransTree). La méthode que nous utilisons n'utilise que les événements onMouseOver et onMouseOut. Ces événements déclenchent le changement de couleur des balises . La couleur « transparent » est utilisée pour les éléments non-activés. De cette manière, la hiérarchie des amphigrammes depuis le nœud où la feuille activé(e) jusqu'à la racine de l'arbre est visible à tout moment.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform' version='1.0'>
<xsl:output method='xml' encoding='utf-8' omit-xml-declaration='no' indent='yes'/>

<xsl:preserve-space elements='text'/>

<xsl:template match='/transtree'>
 <HTML>
  <HEAD>
   <TITLE>Visualisation interactive TransTree</TITLE>
   <META http-equiv='Content-Type' content='text/html; charset=utf-8'/>
  </HEAD>
  <BODY>
   <TABLE border='1'>
    <xsl:apply-templates select='amphigrams[@section="document"]'/>
   </TABLE>
  </BODY>
 </HTML>
</xsl:template>

<xsl:template match='amphigrams'>
 <xsl:apply-templates select='amphigram[@type="complex"]'/>
 </xsl:template>

<xsl:template match='amphigram'>
 <TR>
  <TD colspan='2'>
   <b><xsl:text>Amphigram </xsl:text>
    <xsl:value-of select='1+count(preceding-sibling::amphigram)'/>
   </b>
  </TD>
 </TR>
 <xsl:apply-templates mode='entry' select='text'>
  <xsl:with-param name='path' select='@id'/>
 </xsl:apply-templates>
</xsl:template>

<xsl:template match='text' mode='entry'>
 <xsl:param name='path'/>
  <xsl:param name='id' select='concat($path,@xml:lang)'/>
   <TR>
    <TD><b><xsl:value-of select='@xml:lang'/></b></TD>
    <TD>
     <SPAN>
      <xsl:attribute name='style'>cursor:url("up_rm.cur")</xsl:attribute>
      <xsl:attribute name='title'>
       <xsl:choose>
        <xsl:when test='../@deco'><xsl:value-of select='../@deco'/></xsl:when>
        <xsl:otherwise>0</xsl:otherwise>
       </xsl:choose>
      </xsl:attribute>
      <xsl:attribute name='id'><xsl:value-of select='$id'/></xsl:attribute>
      <xsl:attribute name='onmouseover'>
      <xsl:for-each select='parent::*/text'>
       <xsl:text>document.getElementById('</xsl:text>
       <xsl:value-of select='concat($path,@xml:lang)'/>
       <xsl:text>').style.backgroundColor='#DDDDDD'; </xsl:text>
      </xsl:for-each>
     </xsl:attribute>
     <xsl:attribute name='onmouseout'>
     <xsl:for-each select='parent::*/text'>
      <xsl:text>document.getElementById('</xsl:text>
```

```
      <xsl:value-of select='concat($path,@xml:lang)'/>
      <xsl:text>').style.backgroundColor='transparent'; </xsl:text>
     </xsl:for-each>
    </xsl:attribute>
    <xsl:apply-templates>
     <xsl:with-param name='depth' select='0'/>
      <xsl:with-param name='path' select='concat($path,../@loc)'/>
     </xsl:apply-templates>
    </SPAN>
   </TD>
  </TR>
</xsl:template>

<xsl:template match='text'>
 <xsl:param name='depth'/>
 <xsl:param name='path'/>
 <xsl:param name='id' select='concat($path,@xml:lang)'/>
 <xsl:param name='hexR' select='substring("AABCCDEFF",$depth,1)'/>
 <xsl:param name='hexG' select='substring("ECA975310",$depth,1)'/>
 <xsl:param name='hexB' select='substring("FEDBA9765",$depth,1)'/>
 <xsl:param name='color' select='concat("#",$hexR,"0",$hexG,"0",$hexB,"0")'/>
 <SPAN>
  <xsl:attribute name='id'><xsl:value-of select='$id'/></xsl:attribute>
  <xsl:attribute name='title'>
   <xsl:choose>
    <xsl:when test='../@deco'><xsl:value-of select='../@deco'/></xsl:when>
    <xsl:otherwise><xsl:value-of select='count(ancestor::*)-3'/></xsl:otherwise>
   </xsl:choose>
  </xsl:attribute>
  <xsl:attribute name='onmouseover'>
   <xsl:for-each select='parent::*/text'>
    <xsl:text>document.getElementById('</xsl:text>
    <xsl:value-of select='concat($path,@xml:lang)'/>
    <xsl:text>').style.backgroundColor='</xsl:text>
    <xsl:value-of select='$color'/>
    <xsl:text>'; </xsl:text>
   </xsl:for-each>
  </xsl:attribute>
  <xsl:attribute name='onmouseout'>
   <xsl:for-each select='parent::*/text'>
    <xsl:text>document.getElementById('</xsl:text>
    <xsl:value-of select='concat($path,@xml:lang)'/>
    <xsl:text>').style.backgroundColor='transparent'; </xsl:text>
   </xsl:for-each>
  </xsl:attribute>
  <xsl:apply-templates select='node()'>
   <xsl:with-param name='depth' select='$depth'/>
   <xsl:with-param name='path' select='$path'/>
  </xsl:apply-templates>
 </SPAN>
</xsl:template>

<xsl:template match='a'>
 <xsl:param name='depth'/>
 <xsl:param name='path'/>
 <xsl:apply-templates
select='../../amphigram[@loc=current()/@loc]/text[@xml:lang=current()/../@xml:lang]'
>
  <xsl:with-param select='$depth+1' name='depth'/>
  <xsl:with-param name='path' select='concat($path,@loc)'/>
 </xsl:apply-templates>
</xsl:template>

</xsl:stylesheet>
```

TransTree.checker.xsl : La feuille de style suivante vérifie la conformité d'un fichier XTR (TransTree) aux contraintes du formalisme non prises en compte par la DTD et le schéma.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE xsl:stylesheet SYSTEM "file:///D:/XML/DTDs/xsl.dtd" >
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:output method='xml' omit-xml-declaration='no' encoding='utf-8'/>

<!--Ce fichier permet de verifier quelques aspects non pris en compte par la DTD.
Il faudrait voir lesquels pourraient etre pris en compte dans les schema.
La coherence des set est partiellement prise en compte.
Il faudrait traiter les differents types avec mode. Cela permettrait d'inferer
le type d'un amphigramme sans type et de le reaiguiller.
Cependant, l'ambiguite persiste si complex est la valeur par defaut dans la dtd...
 -->

<xsl:template match='/transtree'>
 <xsl:processing-instruction name='xml-stylesheet'>
  <xsl:text>type="text/xsl" href="TransTreeChecker.viewer.xsl"</xsl:text>
 </xsl:processing-instruction>
 <TransTreeChecker>
  <message>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Number of 'document' sections :
   <xsl:value-of select='count(amphigrams[@section="document"])'/>.</message>
  <message>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Number of 'lexicon' sections :
   <xsl:value-of select='count(amphigrams[@section="lexicon"])'/>.</message>
  <xsl:apply-templates select='//amphigrams'/>
 </TransTreeChecker>
</xsl:template>

<xsl:template match='amphigrams'>
 <message>
  <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Number of amphigrams in '
  <xsl:value-of select='@section'/>' section[<xsl:value-of
select='1+count(preceding-sibling::*[name()=name(current())])'/>] :
  <xsl:value-of select='count(amphigram)'/>.</message>
  <xsl:apply-templates select='amphigram'/>
</xsl:template>

<xsl:template match='amphigram[@type="complex"]'>
 <xsl:param name='a_nb' select='count(amphigram)'/>
 <xsl:param name='t_nb' select='count(text)'/>
 <xsl:apply-templates select='text'>
  <xsl:with-param name='a_nb' select='$a_nb'/>
  <xsl:with-param name='t_nb' select='$t_nb'/>
 </xsl:apply-templates>
</xsl:template>

<xsl:template match='text'>
 <xsl:param name='a_nb'/>
 <xsl:param name='t_nb'/>
 <xsl:if test='not(count(a)=$a_nb)'>
  <error type='2'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Inconsistent number of &lt;a> elements.</error>
 </xsl:if>
 <xsl:apply-templates select='a'>
  <xsl:with-param name='t_nb' select='$t_nb'/>
 </xsl:apply-templates>
</xsl:template>

<xsl:template match='amphigram[@type="compact"]'>
 <xsl:param name='amphig_nb' select='count(amphigram)'/>
 <xsl:param name='ref' select='@ref'/>
 <xsl:param name='a_nb' select='count(//amphigram[@id=$ref]/text[1]/a)'/>
 <xsl:if test='t'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
```

```
   compact amphigrams cannot have a &lt;text> element</error>
 </xsl:if>
 <xsl:if test='not($amphig_nb=$a_nb)'>
  <error type='2'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Number of &lt;amphigram> elements (<xsl:value-of select='$amphig_nb'/>)
inconsistent with
referred generic amphigram (<xsl:value-of select='$ref'/>:<xsl:value-of
select='$a_nb'/>).</error>
 </xsl:if>
 <xsl:apply-templates mode='complex-daughter' select='amphigram'/>
</xsl:template>

<xsl:template match='amphigram' mode='complex-daughter'>
 <xsl:param name='loc' select='@loc'/>
 <xsl:param name='ref-parent' select='//amphigram[@id=current()/../@ref]'/>
 <xsl:param name='ref-current' select='//amphigram[@id=current()/@ref]'/>
  <xsl:if test='not($ref-current/@set=$ref-parent/text/a[@loc=$loc]/@set)'>
<!-- Il faudrait faire avec des chaines multivaluees...-->
  <error type='3'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   &lt;amphigram> '<xsl:value-of select='@ref'/>' is in set '<xsl:value-of
select='$ref-current/@set'/>' but should be in set '<xsl:value-of select='$ref-
parent/text/a[@loc=$loc]/@set'/>'.
   </error>
 </xsl:if>
</xsl:template>

<xsl:template match='amphigram[@type="generic"]'>
 <xsl:if test='count(text)&lt;2'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An generic amphigram should have at least two &lt;text> elements.</error>
 </xsl:if>
 <xsl:if test='not(@set)'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An generic amphigram should have a set-id attribute.</error>
 </xsl:if>
 <xsl:if test='not(@id)'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An generic amphigram should have an id attribute.</error>
 </xsl:if>
 <xsl:if test='amphigram'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An generic amphigram should not have daughter &lt;amphigram> elements.</error>
 </xsl:if>
 <!--  meme nb de a dans les textes -->
</xsl:template>

<xsl:template match='amphigram[@type="atomic"]'>
 <xsl:if test='count(text)&lt;2'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An atomic amphigram should have at least two &lt;text> elements.</error>
 </xsl:if>
 <xsl:if test='not(@set)'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An atomic amphigram should have a set-id attribute.</error>
 </xsl:if>
 <xsl:if test='not(@id)'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An not amphigram should have an id attribute.</error>
 </xsl:if>
```

```
 <xsl:if test='amphigram'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An atomic amphigram should not contain &lt;amphigram> elements.</error>
 </xsl:if>
 <xsl:if test='text/a'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An atomic amphigram should not have  &lt;a> elements in &lt;text>
elements.</error>
 </xsl:if>
 <!--  meme nb de a dans les textes -->
</xsl:template>

<xsl:template match='amphigram'>
 <xsl:if test='amphigram and @set'>
  <error type='1'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   An amphigram cannot have both &lt;text> elements and a set attribute.</error>
 </xsl:if>
</xsl:template>

<xsl:template match='a'>
 <xsl:param name='t_nb'/>
 <xsl:param name='loc' select='@loc'/>
 <xsl:if test='count(../a[@loc=$loc])>1'>
  <error type='2'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Element &lt;text> contains at least two &lt;a> elements with same location index
:
   <xsl:value-of select='$loc'/>.</error>
 </xsl:if>
 <xsl:if test='not(count(../../text/a[@loc=$loc])=$t_nb)'>
  <error type='2'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Location index <xsl:value-of select='$loc'/> does not appear once in each
&lt;text> elements</error>
 </xsl:if>
 <xsl:if test='not(../../amphigram[@loc=$loc])'>
  <error type='2'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Location index <xsl:value-of select='$loc'/> does not match any &lt;amphigram>
element</error>
 </xsl:if>
 <xsl:if test='count(../../amphigram[@loc=$loc])>1'>
  <error type='2'>
   <xsl:attribute name='path'><xsl:apply-templates select='.'
mode='path'/></xsl:attribute>
   Location index <xsl:value-of select='$loc'/> matches more than one &lt;amphigram>
element</error>
 </xsl:if>
</xsl:template>

<xsl:template match='*' mode='path'>
<xsl:choose>
 <xsl:when test='parent::*'><xsl:apply-templates select='parent::*'
mode='path'/>/<xsl:value-of select='name()'/>[<xsl:value-of
select='1+count(preceding-sibling::*[name()=name(current())])'/>]</xsl:when>
<xsl:otherwise>/<xsl:value-of select='name()'/></xsl:otherwise>
</xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

4. Dictionnaires de traduction simple obtenu

Briques traductionnelles : Liste des 100 amphigrammes les plus fréquents dont l'un des textes contient au moins deux mots.

Nombre	Anglais	Français
261	default	par défaut
198	database	base de données
161	Click	Cliquez sur
141	password	mot de passe
135	all	tous les
133	data	de données
112	dialog box	boîte de dialogue
108	process	de processus
79	Data	de données
73	all	toutes les
62	VisualAge for	VisualAge for
59	security	de sécurité
54	number	numéro de
54	Portal 3d	Portal 3d
52	work items	tâches élémentaires
49	current	en cours
49	Server	du serveur
48	Buildtime	Client de modélisation
48	control	de contrôle
47	filter	de filtrage
45	selected	cette option est sélectionnée
42	Management	de gestion
40	cannot	ne pouvez pas
40	concepts	Concepts de
37	click	cliquez sur
37	Process	de processus
36	All	Tous les
33	Copying	Copie des
33	Database	base de données
31	Trace	de trace
31	For example	Par exemple
30	Configuration	de configuration
30	will be	sera
28	Default	par défaut

28	Invalid	non valide
28	Security	de sécurité
27	type	Type de
27	Connection	de connexion
27	List	Liste des
27	Cannot	Impossible de
27	not found	introuvable
26	another	un autre
26	control	contrôle de
26	Buildtime	de modélisation
26	cannot	ne peut pas
25	key	de clés
24	About	A propos
24	After	Une fois
23	Commands	des commandes
23	properties	Propriétés correspondant
23	Password	mot de passe
23	are using	utilisez
22	input	en entrée
22	Once	Une fois
22	Rules	des règles
22	Log	de journalisation
21	bar	barre de
21	All	Toutes les
21	will be	seront
21	from	à partir du
21	from	à partir de
20	work item	tâche élémentaire
20	template	modèle de
20	at least	au moins
20	search	de recherche
20	definition	définition de
19	invalid	non valide
19	filter	des filtres
19	output	de sortie
18	viewpoint	point de vue
18	Importing	Importation de
18	pager	récepteur de radiomessagerie
17	dialog	boîte de dialogue
17	for example	par exemple
17	Could not	Impossible de

16	Password	Mot de passe
16	Portal 3d com	Portal 3D COM
16	staff	du personnel
16	names	les noms
15	Basic Tasks	Tâches de base
15	most	plupart des
15	example	par exemple
15	other countries	pays
14	Drag	Faites glisser
14	within	au sein
14	using	en utilisant
14	Defines the	Définit
14	carrier	des opérateurs
13	Distribution	de distribution
13	Command	de commande
13	Control	de contrôle
13	database	Base de données
13	online	en ligne
13	Loading	Chargement des
12	another	une autre
12	managing	gestion des
12	nonsecure	non sécurisées
12	log on	connecter
12	properties	Propriétés des
12	No	Pas de
12	template	de gabarits

Digrammes : Nous montrons ici un extrait d'un fichier de digrammes. Le premier champ est un identifiant, commençant par X lorsqu'il s'agit d'un vrai digramme, muni d'une traduction ou S lorsqu'il s'agit d'un faux digramme, en face duquel on trouve le symbole #null#. Le deuxième champ indique le nombre d'occurrences.
Attention : Les associations inattendues peuvent résulter de tournures difficiles à reconnaître. Il se peut donc que certaines associations prennent leur sens en contexte sans qu'il soit apparent dans ce fichier intermédiaire. Les véritables erreurs que l'on constate sont généralement peu attestées.

```
XX#0243a          2          mode          forme
XX#0b719          3          1252          1252
XX#0ecbc          1          accuracy          équivalents
XX#0108a          2          accuracy          Précision
SX#018cf          5          accuracy          #null#
XX#0108e          11          accuracy          précision
XX#0e69e          1          accuracy          nécessaire
XX#0ba73          2          accuracy          vocale
XX#04017          1          accuracy          la
XX#0df46          1          UNPROTECTED          UNPROTECTED
XX#08971          5          goals          objectifs
XX#08974          1          goals          fixer
XX#00c66          3          Sorting          Tri
XX#05937          1          Sorting          améliore
XX#08c51          1          pointer          souris
XX#00c1d          20          pointer          Pointeur
SX#04473          12          pointer          #null#
XX#009e4          81          pointer          pointeur
XX#0c158          1          pointer          périphérique
XX#02b94          1          CDMF          CDMF
XX#071e1          4          Samples          Exemples
XX#09d85          3          Samples          Samples
SX#023d7          2          Samples          #null#
XX#023d3          3          Samples          exemples
XX#06822          1          Rate          renferme
SX#0e49a          1          Rate          #null#
XX#044b3          1          Rate          Vitesse
XX#01bbd          1          broader          élargie
XX#0bc64          1          broader          commandée
XX#0b8f1          1          decrypt          chiffrer
XX#06ed0          2          decrypt          algorithme
XX#0b433          1          decrypt          déchiffrement
XX#0540d          6          decrypt          déchiffrer
XX#01f8c          1          decrypt          décoder
XX#102b5          1          Delayed          enlevées
SX#0f687          1          Delayed          #null#
XX#000b1          32          hardware          matériels
XX#0f813          1          hardware          Matériel
XX#0d95e          1          hardware          du
XX#0c132          1          hardware          dispositif
XX#0b09a          12          hardware          matérielles
XX#00125          89          hardware          matériel
XX#0efbf          1          hardware          plateformes
XX#03b4e          35          hardware          matérielle
XX#0e4e1          1          hardware          défectueuse
XX#0e283          1          hardware          automatique
XX#1054c          2          hardware          utilisent
SX#0881a          27          hardware          #null#
XX#0b7f8          1          unlink          lien
SX#082a6          1          unlink          #null#
XX#0f5eb          1          COALESCE          COALESCE
XX#0a316          2          MenuCustomerInfoPanel1          numberText
SX#0d727          1          spread          #null#
XX#09d3d          1          spread          répartis
XX#062c3          1          authentication          émis
XX#01b47          1          authentication          identité
SX#07068          2          authentication          #null#
XX#0292d          2          authentication          Authentification
XX#0638d          3          authentication          authentification
XX#026b0          1          authentication          optez
XX#1021a          1          tracks          unités
XX#0b3de          2          tracks          surveille
SX#0718a          2          tracks          #null#
XX#07078          1          tracks          analyse
XX#0b1e6          4          tracks          pistes
XX#0717a          3          tracks          suit
SX#0eddd          1          EICON          #null#
XX#0b004          1          AcctContainer          AcctContainer
XX#0916d          1          papers          bond
SX#02ac8          3          outbound          #null#
XX#02ccc          7          outbound          sortie
XX#02b12          5          outbound          sortant
XX#0f7f3          1          outbound          entrant
XX#02c76          6          outbound          sortants
XX#02a41          2          outbound          sortantes
SX#07251          1          sums          #null#
XX#0694c          12          Cassette          Cassette
```

XX#06523	2	Cassette	cassette
SX#0b2d2	1	Redundancy	#null#
XX#0c191	1	subset	fonctionnalités
XX#0f824	1	subset	Seules
XX#0523f	1	subset	saisies
SX#0523c	1	subset	#null#
XX#0b12e	2	subset	partie
XX#0ec27	1	subset	exportation
XX#045ec	1	who	personnes
XX#0414e	1	who	traiter
XX#0215a	2	who	personne
XX#00bf2	53	who	qui
XX#034cc	1	who	tâches
XX#06a02	1	who	sélectionnant
XX#06e8f	1	who	définis
XX#04a8f	1	who	biais
XX#091c2	1	who	collectant
XX#06379	1	who	affichage
SX#01b7f	47	who	#null#
XX#01379	15	who	ayant
XX#0217f	1	who	déconnecter
XX#04d92	1	who	posséder
XX#091c0	2	who	assurant
XX#03207	1	who	autorisés
SX#1038e	1	Segment	#null#
XX#10387	1	Segment	La
XX#06de0	1	motorcycles	motos
XX#07b42	9	SURFACES	surfaces
XX#0f7d1	1	why	néanmoins
XX#1024f	1	why	causes
XX#0e105	1	why	autorisent
XX#0a350	1	why	quoi
XX#08ead	4	why	pourquoi
XX#100a5	1	why	utilité
XX#0ff2c	1	why	explication
XX#0f881	1	why	amené
SX#0a597	1	why	#null#
XX#08e16	1	why	planifiez
XX#04c86	6	why	raison
XX#047d5	2	accomplished	accomplie
XX#071bf	1	accomplished	ouvrez
XX#09892	1	accomplished	faire
XX#09dcf	1	Even	Des
XX#027cf	10	Even	Même
XX#05ec8	1	StaticDataBean	StaticDataBean
SX#0ee38	1	WORK	#null#
XX#07f96	1	WORK	appliqué
XX#07bb6	1	WORK	Cependant
SX#0ad62	1	reinstallation	#null#
XX#0b114	1	OracleServiceORCL	OracleServiceORCL
XX#0cc6c	2	SAVE	SAUVEGARDE
SX#0cc7e	2	SAVE	#null#
XX#0cd01	2	SAVE	SAUVER
XX#0721f	1	refined	réalisé
SX#08c12	1	wil	#null#
SX#0eaa4	1	win	#null#
XX#04c4b	1	Determines	binaire
XX#04c46	3	Determines	déterminer
XX#030cc	42	Determines	Détermine
XX#0f7a7	1	Determines	détermine
XX#0e0f8	1	Determines	vérifie
XX#0310f	2	Determines	Indique
XX#06efc	1	Determines	située
XX#06eef	1	Determines	Définit
XX#07e3e	2	PRESENT	présente

5. Quelques exemples de retraductions

Ces exemples sont des retraductions des phrases du corpus, à l'aide des phrases source et des amphigrammes collectés. Il ne s'agit pas de segments inconnus mais des segments d'apprentissage. Treize segments sur dix-sept sont ici correctement retraduits, soit 77%. Sur l'ensemble du corpus, nous avons atteint 85%.

```
[02:14:17] N° 1 : 33087.xml
----------------------------------------------------------------------
Capabilities of all previously announced products are also doubled
Le nombre de fonctions proposées dans tous les produits déjà annoncés a également
doublé
[0](189557,19) :
N0#00001+S0#00003+X0#00004+X0#00005+X0#00006+S0#00007+X0#00008+S0#00009+X0#0000a+X0#
0000b+N0#00002
[1] Arbre :
(((S0#00003,X0#00004),(X0#00005,((X0#00006,S0#00007),X0#00008))),(S0#00009,(X0#0000a
,X0#0000b)))
[1] Positions : 7+1+2+3+5+8+3:5+7:8+2:3:5+1:2:3:5+1:2:3:5:7:8
[1] Découpage :
S0#00003+X0#00004+X0#00005+(X0#00006,S0#00007)+X0#00008+S0#00009+X0#0000a+X0#0000b
[1](10,17) :
N1#00001+S1#138a2+X1#1389e+X1#1389f+X1#138a0+X1#138a1+S1#138a3+X1#1389c+X1#1389d+N1#
00002
[2] Arbre :
((S1#138a2,X1#1389e),((X1#1389f,(X1#138a0,X1#138a1)),(S1#138a3,(X1#1389c,X1#1389d)))
)
[2] Positions : 6:7+3:4+2:3:4+2:3:4:6:7+1:2:3:4:6:7+4+6+7+1+2+3
[2] Découpage :
S1#138a2+X1#1389e+X1#1389f+X1#138a0+X1#138a1+(S1#138a3,(X1#1389c,X1#1389d))
[2](8,13) : N2#00001+S2#1893c+X2#18939+X2#1893a+X2#1893b+X2#18938+X2#18937+N2#00002
[3] Arbre : ((S2#1893c,X2#18939),(X2#1893a,(X2#1893b,(X2#18938,X2#18937))))
[3] Positions : 4:5+3:4:5+2:3:4:5+1:2:3:4:5+3+4+5+1+2
[3] Découpage : S2#1893c+X2#18939+X2#1893a+(X2#1893b,(X2#18938,X2#18937))
[3](6,9) : N3#00001+S3#1ded9+X3#1ded7+X3#1ded8+X3#1ded6+N3#00002
[4] Arbre : ((S3#1ded9,X3#1ded7),(X3#1ded8,X3#1ded6))
[4] Positions : 2:3+1:2:3+1+2+3
[4] Découpage : S3#1ded9+X3#1ded7+(X3#1ded8,X3#1ded6)
[4](5,7) : N4#00001+S4#215f8+X4#215f7+X4#215f6+N4#00002
[5] Arbre : (S4#215f8,(X4#215f7,X4#215f6))
[5] Positions : 1:2+1+2
[5] Découpage : (S4#215f8,(X4#215f7,X4#215f6))
[5](3,3) : N5#00001+X5#23543+N5#00002
[5] : N5#00001+X5#23543+N5#00002
OK=> Le nombre de fonctions proposées dans tous les produits déjà annoncés a
également doublé
----------------------------------------------------------------------
Competitive hybrid mechanical design capabilities support complex part design such
as power trains
Ses fonctions de conception mécanique hybrides extrêmement performantes permettent
de concevoir des pièces complexes telles que des groupes motopropulseurs
[0](189561,27) :
N0#00001+S0#00014+S0#00015+X0#00016+S0#00017+X0#00018+S0#00019+X0#0001a+S0#0001b+S0#
00017+X0#0001c+X0#0001d+S0#0001e+S0#0001f+N0#00002
[1] Arbre :
(((((((S0#00014,S0#00015),X0#00016),S0#00017),X0#00018),(S0#00019,((X0#0001a,S0#0001
b),S0#00017))),(X0#0001c,X0#0001d)),(S0#0001e,S0#0001f))
[1] Positions : 6+9+2+10+4+2:4+9:10+2:4:6+2:4:6:9:10
[1] Découpage :
(((S0#00014,S0#00015),X0#00016),S0#00017)+X0#00018+S0#00019+X0#0001a+S0#0001b+S0#000
17+X0#0001c+X0#0001d+(S0#0001e,S0#0001f)
[1](11,19) :
N1#00001+X1#138a8+X1#138aa+S1#138ae+X1#138ab+S1#138ac+S1#138ad+X1#138a7+X1#138a9+S1#
138af+N1#00002
[2] Arbre :
((((X1#138a8,X1#138aa),S1#138ae),((X1#138ab,S1#138ac),S1#138ad)),((X1#138a7,X1#138a9
),S1#138af))
[2] Positions : 6:7+0:1+0:1:3+0:1:3:6:7+3+6+7+0+1
[2] Découpage :
((X1#138a8,X1#138aa),S1#138ae)+((X1#138ab,S1#138ac),S1#138ad)+((X1#138a7,X1#138a9),S
1#138af)
[2](5,7) : N2#00001+X2#18941+X2#18942+X2#18940+N2#00002
[3] Arbre : (X2#18941,(X2#18942,X2#18940))
```

```
[3] Positions : 1:2+0:1:2+0+1+2
[3] Découpage : X2#18941+(X2#18942,X2#18940)
[3](4,5) : N3#00001+X3#1dede+X3#1dedd+N3#00002
[4] Arbre : (X3#1dede,X3#1dedd)
[4] Positions : 0:1+0+1
[4] Découpage : (X3#1dede,X3#1dedd)
[4](3,3) : N4#00001+X4#215fb+N4#00002
[4] : N4#00001+X4#215fb+N4#00002
OK=> Ses fonctions de conception mécanique hybrides extrêmement performantes
permettent de concevoir des pièces complexes telles que des groupes motopropulseurs
-----------------------------------------------------------------
Conferencing
Systèmes de conférence
[0](189559,3) : N0#00001+X0#0191f+N0#00002
[0] : N0#00001+X0#0191f+N0#00002
KO=> Conférence
-----------------------------------------------------------------
Generative fitting trajectories and generative kinematic simulations
Trajectoires de montage et simulations cinématiques génératives
[0](189555,15) :
N0#00001+S0#0002e+S0#0002f+S0#00030+X0#00031+S0#00032+S0#00033+S0#00034+N0#00002
[1] Arbre :
((S0#0002e,(S0#0002f,S0#00030)),(X0#00031,(S0#00032,(S0#00033,S0#00034))))
[1] Positions : 3
[1] Découpage :
(S0#0002e,(S0#0002f,S0#00030))+X0#00031+(S0#00032,(S0#00033,S0#00034))
[1](5,7) : N1#00001+S1#138b8+X1#138b6+S1#138b7+N1#00002
[2] Arbre : (S1#138b8,(X1#138b6,S1#138b7))
[2] Positions : 1
[2] Découpage : (S1#138b8,(X1#138b6,S1#138b7))
[2](3,3) : N2#00001+X2#18946+N2#00002
[2] : N2#00001+X2#18946+N2#00002
OK=> Trajectoires de montage et simulations cinématiques génératives
-----------------------------------------------------------------
Publishing
Publication
[0](189571,3) : N0#00001+X0#0003a+N0#00002
[0] : N0#00001+X0#0003a+N0#00002
OK=> Publication
-----------------------------------------------------------------
Introduces advanced Elfini options
options avancées Elfini
[0](189567,9) : N0#00001+S0#0003b+X0#0003c+X0#0003d+X0#0003e+N0#00002
[1] Arbre : ((S0#0003b,(X0#0003c,X0#0003d)),X0#0003e)
[1] Positions : 1+2+3+1:2+1:2:3
[1] Découpage : S0#0003b+X0#0003c+X0#0003d+X0#0003e
[1](6,9) : N1#00001+S1#138be+X1#138bb+X1#138bc+X1#138bd+N1#00002
[2] Arbre : ((S1#138be,(X1#138bb,X1#138bc)),X1#138bd)
[2] Positions : 1:2+1:2:3+1+2+3
[2] Découpage : (S1#138be,(X1#138bb,X1#138bc))+X1#138bd
[2](4,5) : N2#00001+X2#18947+X2#18948+N2#00002
[3] Arbre : (X2#18947,X2#18948)
[3] Positions : 0:1+0+1
[3] Découpage : (X2#18947,X2#18948)
[3](3,3) : N3#00001+X3#1dedf+N3#00002
[3] : N3#00001+X3#1dedf+N3#00002
OK=> options avancées Elfini
-----------------------------------------------------------------
Standard parts catalogs
les catalogues de pièces standard
[0](190574,7) : N0#00001+X0#0003f+X0#00040+X0#00041+N0#00002
[1] Arbre : (X0#0003f,(X0#00040,X0#00041))
[1] Positions : 0+1+2+1:2+0:1:2
[1] Découpage : X0#0003f+X0#00040+X0#00041
[1](40,7) : N1#00001+X1#138bf+X1#138c0+X1#138c1+N1#00002
[2] Arbre : (X1#138bf,(X1#138c0,X1#138c1))
[2] Positions : 1:2+0:1:2+0+1+2
[2] Découpage : (X1#138bf,(X1#138c0,X1#138c1))
[2](3,3) : N2#00001+X2#18949+N2#00002
[2] : N2#00001+X2#18949+N2#00002
OK=> les catalogues de pièces standard
-----------------------------------------------------------------
Structure design
la conception de structures
[0](189553,5) : N0#00001+X0#00042+X0#00043+N0#00002
[1] Arbre : (X0#00042,X0#00043)
[1] Positions : 0+1+0:1
```

```
[1] Découpage : X0#00042+X0#00043
[1](4,5) : N1#00001+X1#138c3+X1#138c4+N1#00002
[2] Arbre : (X1#138c3,X1#138c4)
[2] Positions : 0:1+0+1
[2] Découpage : (X1#138c3,X1#138c4)
[2](3,3) : N2#00001+X2#1894a+N2#00002
[2] : N2#00001+X2#1894a+N2#00002
OK=> la conception de structures
----------------------------------------------------------------------
Large assembly and bills of material management
la gestion des nomenclatures et des assemblages à grande échelle
[0](189635,15) :
N0#00001+X0#00045+S0#00046+X0#00031+X0#00047+X0#00048+S0#00049+X0#0004a+N0#00002
[1] Arbre :
(((X0#00045,S0#00046),(X0#00031,((X0#00047,X0#00048),S0#00049))),X0#0004a)
[1] Positions : 0+2+3+4+6+3:4+2:3:4+0:2:3:4+0:2:3:4:6
[1] Découpage : (X0#00045,S0#00046)+X0#00031+X0#00047+X0#00048+S0#00049+X0#0004a
[1](8,13) : N1#00001+X1#138c6+X1#138b6+X1#138c7+X1#138c8+S1#138ca+X1#138c9+N1#00002
[2] Arbre : ((X1#138c6,(X1#138b6,((X1#138c7,X1#138c8),S1#138ca))),X1#138c9)
[2] Positions : 2:3+1:2:3+0:1:2:3+0:1:2:3:5+1+2+3+0+5
[2] Découpage : ((X1#138c6,(X1#138b6,((X1#138c7,X1#138c8),S1#138ca))),X1#138c9)
[2](3,3) : N2#00001+X2#1894b+N2#00002
[2] : N2#00001+X2#1894b+N2#00002
OK=> la gestion des nomenclatures et des assemblages à grande échelle
----------------------------------------------------------------------
You can choose from among a variety of methods to leverage knowledge for achieving
productivity increases and cost reductions
Vous disposez de différentes méthodes pour tirer parti de ces compétences accroître
la productivité de votre entreprise et réduire les coûts
[0](5082749697,39) :
N0#00001+X0#0004e+X0#00460+S0#00050+S0#00051+S0#00052+S0#00053+X0#00054+S0#00055+X0#
00056+S0#00057+S0#00058+S0#00059+X0#0005a+S0#0005b+X0#0005c+S0#0005d+X0#00031+S0#000
5e+S0#0005f+N0#00002
[1] Arbre :
((X0#0004e,X0#00460),((S0#00050,((S0#00051,S0#00052),((S0#00053,X0#00054),S0#00055))
),(X0#00056,(S0#00057,((((S0#00058,S0#00059),(X0#0005a,((S0#0005b,X0#0005c),S0#0005d
))),X0#00031),(S0#0005e,S0#0005f))))))
[1] Positions :
8+0+1+16+12+6+14+0:1+12:14+12:14:16+8:12:14:16+6:8:12:14:16+0:1:6:8:12:14:16
[1] Découpage :
X0#0004e+X0#00460+(S0#00050,((S0#00051,S0#00052),((S0#00053,X0#00054),S0#00055)))+X0
#00056+S0#00057+(S0#00058,S0#00059)+X0#0005a+((S0#0005b,X0#0005c),S0#0005d)+X0#00031
+(S0#0005e,S0#0005f)
[1](15263714,18) :
N1#00001+X1#138cd+X1#13c1c+X1#138cf+X1#138cc+S1#138d3+S1#138d1+X1#138ce+X1#138cb+X1#
138b6+S1#138d2+N1#00002
[2] Arbre :
((X1#138cd,X1#13c1c),((X1#138cf,X1#138cc),((S1#138d3,(S1#138d1,(X1#138ce,X1#138cb)))
,(X1#138b6,S1#138d2))))
[2] Positions : 2:3+6:7+0:1+6:7:8+2:3:6:7:8+0:1:2:3:6:7:8+3+6+7+0+8+1+2
[2] Découpage :
(X1#138cd,X1#13c1c)+X1#138cf+X1#138cc+(S1#138d3,(S1#138d1,(X1#138ce,X1#138cb)))+X1#1
38b6+S1#138d2
[2](80053,8) :
N2#00001+X2#18b82+X2#18a45+X2#1b1c5+X2#1894d+X2#1894f+S2#18950+N2#00002
getinfo dans GLOB(0xa084444) pour  X2#18a45 : 63
[3] Arbre : ((X2#18b82,X2#18a45),(X2#1b1c5,(X2#1894d,(X2#1894f,S2#18950))))
[3] Positions : 3:4+0:1+2:3:4+0:1:2:3:4+3+4+0+1+2
[3] Découpage : (X2#18b82,X2#18a45)+X2#1b1c5+X2#1894d+X2#1894f+S2#18950
[3](984,4) : N3#00001+X3#1e37a+X3#204f5+X3#1dee1+S3#1dee3+N3#00002
[4] Arbre : (X3#1e37a,(X3#204f5,(X3#1dee1,S3#1dee3)))
[4] Positions : 1:2+0:1:2+0+1+2
[4] Découpage : X3#1e37a+X3#204f5+X3#1dee1+S3#1dee3
[4](150,6) : N4#00001+X4#2189a+X4#23123+X4#21617+X4#21997+N4#00002
getinfo dans GLOB(0xa0845f4) pour  X4#21997 : 5
[5] Arbre : (X4#2189a,((X4#23123,X4#21617),X4#21997))
[5] Positions : 1:2+1:2:3+0:1:2:3+1+2+3+0
[5] Découpage : X4#2189a+X4#23123+X4#21617+X4#21997
[5](111,6) : N5#00001+X5#23934+X5#23933+X5#2354f+X5#24330+N5#00002
getinfo dans GLOB(0xa0846cc) pour  X5#23933 : 2
[6] Arbre : ((X5#23934,X5#23933),(X5#2354f,X5#24330))
[6] Positions : 2:3+0:1+0:1:2:3+0+1+2+3
[6] Découpage : (X5#23934,X5#23933)+(X5#2354f,X5#24330)
[6](7,1) : N6#00001+X6#24769+X6#24bd1+N6#00002
[7] Arbre : (X6#24769,X6#24bd1)
[7] Positions : 0:1+0+1
[7] Découpage :
```

```
[7](2,0) : N7#00001+N7#00002
[7] : N7#00001+N7#00002
KO=>
--------------------------------------------------------------------
CATIA Photo Studio is a new communication tool for stylists and others involved in
the product design process in automotive and consumer goods
CATIA Photo Studio est un nouvel outil de communication destiné aux responsables du
processus de conception dans le secteur automobile ou des biens de consommation
[0](191177,47) :
N0#00001+X0#0006a+X0#0006b+X0#0006c+X0#0006d+X0#0006e+X0#0006f+X0#00070+X0#00071+S0#
00072+X0#00073+S0#00074+S0#00075+S0#00076+S0#00077+X0#00078+S0#00079+X0#00043+X0#000
7a+S0#00077+S0#0007b+S0#00074+S0#0007c+X0#0007d+N0#00002
[1] Arbre :
(((X0#0006a,(X0#0006b,X0#0006c)),(X0#0006d,(X0#0006e,((X0#0006f,((X0#00070,X0#00071)
,(S0#00072,((X0#00073,S0#00074),(S0#00075,S0#00076))))),S0#00077)))),(X0#00078,((S0#
00079,(X0#00043,X0#0007a)),(S0#00077,(S0#0007b,(S0#00074,(S0#0007c,X0#0007d)))))))
[1] Positions :
4+5+6+7+22+14+0+9+16+1+17+2+3+6:7+16:17+1:2+16:17:22+6:7:9+0:1:2+5:6:7:9+14:16:17:22
+4:5:6:7:9+3:4:5:6:7:9+0:1:2:3:4:5:6:7:9+0:1:2:3:4:5:6:7:9:14:16:17:22
[1] Découpage :
X0#0006a+X0#0006b+X0#0006c+X0#0006d+X0#0006e+X0#0006f+X0#00070+X0#00071+(S0#00072,((
X0#00073,S0#00074),(S0#00075,S0#00076)))+S0#00077+X0#00078+S0#00079+X0#00043+X0#0007
a+(S0#00077,(S0#0007b,(S0#00074,(S0#0007c,X0#0007d))))
[1](32,31) :
N1#00001+X1#138d9+X1#138da+X1#138db+X1#138dc+X1#138dd+X1#138de+X1#138df+X1#138e0+X1#
138e3+S1#138e5+X1#138e2+S1#138e6+X1#138c4+X1#138e4+X1#138e1+N1#00002
[2] Arbre :
(((X1#138d9,(X1#138da,X1#138db)),X1#138dc),(X1#138dd,((((X1#138de,(X1#138df,(X1#138e
0,X1#138e3))),S1#138e5),X1#138e2),((S1#138e6,X1#138c4),(X1#138e4,X1#138e1)))))
[2] Positions :
7:8+13:14+1:2+12:13:14+6:7:8+0:1:2+5:6:7:8+0:1:2:3+5:6:7:8:10+5:6:7:8:10:12:13:14+4:
5:6:7:8:10:12:13:14+0:1:2:3:4:5:6:7:8:10:12:13:14+10+5+0+6+12+7+13+1+8+14+2+3+4
[2] Découpage :
X1#138d9+(X1#138da,X1#138db)+X1#138dc+X1#138dd+X1#138de+X1#138df+X1#138e0+X1#138e3+S
1#138e5+X1#138e2+S1#138e6+X1#138c4+X1#138e4+X1#138e1
[2](97,24) :
N2#00001+X2#18957+X2#18954+X2#1895e+X2#1895f+X2#18956+X2#18958+X2#1895a+X2#1895c+S2#
18960+X2#18955+X2#1c0f4+X2#1a334+X2#1895b+X2#1895d+N2#00002
getinfo dans GLOB(0xa084444) pour  X2#1c0f4 : 11
[3] Arbre :
((((X2#18957,X2#18954),X2#1895e),(X2#1895f,(((X2#18956,(X2#18958,(X2#1895a,X2#1895c)
)),S2#18960),(X2#18955,X2#1c0f4)))),(X2#1a334,(X2#1895b,X2#1895d)))
[3] Positions :
6:7+0:1+12:13+9:10+5:6:7+11:12:13+0:1:2+4:5:6:7+4:5:6:7:9:10+3:4:5:6:7:9:10+0:1:2:3:
4:5:6:7:9:10+0:1:2:3:4:5:6:7:9:10:11:12:13+3+4+10+11+5+12+6+13+7+0+1+9+2
[3] Découpage :
(X2#18957,X2#18954)+X2#1895e+X2#1895f+X2#18956+X2#18958+X2#1895a+X2#1895c+S2#18960+X
2#18955+X2#1c0f4+X2#1a334+X2#1895b+X2#1895d
[3](204,9) :
N3#00001+X3#1dee6+X3#1dee8+X3#1df92+X3#2103e+X3#21155+S3#1e07f+X3#1df8e+X3#204ff+N3#
00002+X3#1e4f6+N3#00002
[4] Arbre :
((((X3#1dee6,X3#1dee8),X3#1df92),(X3#2103e,(X3#21155,((S3#1e07f,X3#1df8e),X3#204ff))
)),(N3#00002,X3#1e4f6))
[4] Positions :
6:7+0:1+4:6:7+0:1:2+3:4:6:7+0:1:2:3:4:6:7+0:1:2:3:4:6:7:9+4+6+7+0+1+2+9+3
[4] Découpage :
X3#1dee6+X3#1dee8+X3#1df92+X3#2103e+X3#21155+S3#1e07f+X3#1df8e+X3#204ff+N3#00002+X3#
1e4f6
[4](211,5) :
N4#00001+X4#215ff+X4#2160c+X4#21693+S4#217f9+X4#21675+X4#22ca5+N4#00002+X4#21b57+N4#
00002
[5] Arbre :
(X4#215ff,(X4#2160c,(X4#21693,((S4#217f9,(X4#21675,(X4#22ca5,N4#00002))),X4#21b57)))
)
[5] Positions : 4:5+4:5:7+2:4:5:7+1:2:4:5:7+0:1:2:4:5:7+5+7+0+1+2+4
[5] Découpage :
X4#215ff+X4#2160c+X4#21693+S4#217f9+X4#21675+(X4#22ca5,N4#00002)+X4#21b57
[5](129,0) : N5#00001+X5#23565+X5#2361f+S5#2390f+X5#23609+X5#23805+N5#00002
[6] Arbre : (X5#23565,(X5#2361f,(S5#2390f,(X5#23609,X5#23805))))
[6] Positions : 3:4+1:3:4+0:1:3:4+3+4+0+1
[6] Découpage : X5#23565+X5#2361f+S5#2390f+X5#23609+X5#23805
[6](51,0) : N6#00001+X6#24687+X6#2464d+X6#24674+X6#2471b+N6#00002
[7] Arbre : (X6#24687,(X6#2464d,(X6#24674,X6#2471b)))
[7] Positions : 2:3+1:2:3+0:1:2:3+0+1+2+3
[7] Découpage : X6#24687+X6#2464d+X6#24674+X6#2471b
[7](19,0) : N7#00001+X7#24d7c+X7#24e25+X7#24d6b+X7#24ece+N7#00002
```

```
[8] Arbre : (X7#24d7c,(X7#24e25,(X7#24d6b,X7#24ece)))
[8] Positions : 2:3+1:2:3+0:1:2:3+0+1+2+3
[8] Découpage : X7#24d7c+X7#24e25+X7#24d6b+X7#24ece
[8](10,0) : N8#00001+X8#25055+X8#25092+X8#2504d+X8#250db+N8#00002
[9] Arbre : (X8#25055,(X8#25092,(X8#2504d,X8#250db)))
[9] Positions : 2:3+1:2:3+0:1:2:3+0+1+2+3
[9] Découpage : X8#25055+X8#25092+X8#2504d+X8#250db
[9](6,0) : N9#00001+X9#2515e+X9#25176+X9#2515c+N9#00002
[10] Arbre : (X9#2515e,(X9#25176,X9#2515c))
[10] Positions : 1:2+0:1:2+0+1+2
[10] Découpage :
[10](,0) :
[10] :
KO=>
------------------------------------------------------------------
This fee service can help improve your productivity by providing voice and
electronic access to the Engineering Solutions Support Center
Ce service payant peut vous aider à améliorer votre productivité en fournissant un
accès vocal et électronique au centre Engineering Solutions Support Center
[0](1275338,41) :
N0#00001+X0#00085+S0#00086+X0#00087+X0#00088+X0#00089+X0#0008a+X0#0008b+X0#0005c+S0#
0008c+X0#0008d+X0#0008e+X0#00031+X0#0008f+X0#00090+X0#00091+S0#00092+X0#00093+X0#000
94+X0#00095+X0#00096+N0#00002
[1] Arbre :
((X0#00085,((S0#00086,X0#00087),(((X0#00088,X0#00089),(((X0#0008a,X0#0008b),((X0#000
5c,(S0#0008c,(X0#0008d,X0#0008e))),X0#00031)),X0#0008f)),X0#00090))),((X0#00091,S0#0
0092),(((X0#00093,X0#00094),X0#00095),X0#00096)))
[1] Positions :
11+6+12+7+13+14+9+16+17+18+19+0+2+3+4+10+5+9:10+3:4+5:6+16:17+16:17:18+7:9:10+16:17:
18:19+7:9:10:11+14:16:17:18:19+5:6:7:9:10:11+5:6:7:9:10:11:12+3:4:5:6:7:9:10:11:12+3
:4:5:6:7:9:10:11:12:13+2:3:4:5:6:7:9:10:11:12:13+0:2:3:4:5:6:7:9:10:11:12:13+0:2:3:4
:5:6:7:9:10:11:12:13:14:16:17:18:19
[1] Découpage :
X0#00085+S0#00086+X0#00087+X0#00088+X0#00089+X0#0008a+X0#0008b+X0#0005c+S0#0008c+X0#
0008d+X0#0008e+X0#00031+X0#0008f+X0#00090+X0#00091+S0#00092+X0#00093+X0#00094+X0#000
95+X0#00096
[1](11046,41) :
N1#00001+X1#138ea+S1#138fa+X1#138eb+X1#138ec+X1#138ed+X1#138ef+X1#138f0+X1#138f2+S1#
138fb+X1#138f5+X1#138ee+X1#138b6+X1#138f1+X1#138f3+X1#138f4+S1#138fc+X1#138f6+X1#138
f7+X1#138f8+X1#138f9+N1#00002
[2] Arbre :
((X1#138ea,((S1#138fa,X1#138eb),(((X1#138ec,(X1#138ed,X1#138ef)),(X1#138f0,(X1#138f2
,(S1#138fb,(X1#138f5,X1#138ee))))),(X1#138b6,(X1#138f1,X1#138f3))))),(X1#138f4,(S1#1
38fc,((X1#138f6,X1#138f7),(X1#138f8,X1#138f9)))))
[2] Positions :
18:19+16:17+9:10+4:5+12:13+11:12:13+7:9:10+3:4:5+16:17:18:19+6:7:9:10+14:16:17:18:19
+3:4:5:6:7:9:10+3:4:5:6:7:9:10:11:12:13+2:3:4:5:6:7:9:10:11:12:13+0:2:3:4:5:6:7:9:10
:11:12:13+0:2:3:4:5:6:7:9:10:11:12:13:14:16:17:18:19+4+10+5+11+0+6+12+7+13+14+9+16+1
7+2+18+3+19
[2] Découpage :
X1#138ea+S1#138fa+X1#138eb+X1#138ec+X1#138ed+X1#138ef+X1#138f0+X1#138f2+S1#138fb+X1#
138f5+X1#138ee+X1#138b6+X1#138f1+X1#138f3+X1#138f4+S1#138fc+(X1#138f6,X1#138f7)+(X1#
138f8,X1#138f9)
[2](7763,37) :
N2#00001+X2#18968+S2#18971+X2#1896f+X2#18970+X2#18965+X2#18967+X2#18969+X2#1896b+S2#
18972+X2#1896e+X2#18966+X2#1894f+X2#1896a+X2#1896c+X2#1896d+S2#18973+X2#18964+X2#189
63+N2#00002
[3] Arbre :
((X2#18968,((S2#18971,X2#1896f),(((X2#18970,(X2#18965,X2#18967)),(X2#18969,(X2#1896b
,(S2#18972,(X2#1896e,X2#18966))))),(X2#1894f,(X2#1896a,X2#1896c))))),((X2#1896d,S2#1
8973),(X2#18964,X2#18963)))
[3] Positions :
16:17+4:5+12:13+9:10+11:12:13+14:16:17+3:4:5+7:9:10+6:7:9:10+3:4:5:6:7:9:10+3:4:5:6:
7:9:10:11:12:13+2:3:4:5:6:7:9:10:11:12:13+0:2:3:4:5:6:7:9:10:11:12:13+0:2:3:4:5:6:7:
9:10:11:12:13:14:16:17+10+4+11+5+12+6+13+7+14+0+9+16+2+17+3
[3] Découpage :
X2#18968+S2#18971+X2#1896f+X2#18970+X2#18965+X2#18967+X2#18969+X2#1896b+S2#18972+X2#
1896e+X2#18966+X2#1894f+X2#1896a+X2#1896c+X2#1896d+S2#18973+(X2#18964,X2#18963)
[3](5059,35) :
N3#00001+X3#1def2+S3#1def6+X3#1def4+X3#1def5+X3#1deeb+X3#1deec+X3#1deee+X3#1def0+S3#
1def7+X3#1def3+X3#1deea+X3#1dee1+X3#1deed+X3#1deef+X3#1def1+S3#1def8+X3#1dee9+N3#000
02
[4] Arbre :
(X3#1def2,(((((S3#1def6,X3#1def4),(((X3#1def5,(X3#1deeb,X3#1deec)),(X3#1deee,(X3#1de
f0,(S3#1def7,(X3#1def3,X3#1deea))))),(X3#1dee1,(X3#1deed,X3#1deef)))),X3#1def1),S3#1
def8),X3#1dee9))
```

```
[4] Positions :
4:5+12:13+9:10+3:4:5+11:12:13+7:9:10+6:7:9:10+3:4:5:6:7:9:10+3:4:5:6:7:9:10:11:12:13
+2:3:4:5:6:7:9:10:11:12:13+2:3:4:5:6:7:9:10:11:12:13:14+2:3:4:5:6:7:9:10:11:12:13:14
:16+0:2:3:4:5:6:7:9:10:11:12:13:14:16+3+4+10+5+11+6+12+7+13+14+9+0+16+2

[4] Découpage :
X3#1def2+(((((S3#1def6,X3#1def4),(((X3#1def5,(X3#1deeb,X3#1deec)),(X3#1deee,(X3#1def
0,(S3#1def7,(X3#1def3,X3#1deea))))),(X3#1dee1,(X3#1deed,X3#1deef)))),X3#1def1),S3#1d
ef8),X3#1dee9)
[4](3367,5) : N4#00001+X4#21601+X4#21600+N4#00002
[5] Arbre : (X4#21601,X4#21600)
[5] Positions : 0:1+0+1
[5] Découpage : (X4#21601,X4#21600)
[5](3,3) : N5#00001+X5#23546+N5#00002
[5] : N5#00001+X5#23546+N5#00002
OK=> Ce service payant peut vous aider à améliorer votre productivité en fournissant
un accès vocal et électronique au centre Engineering Solutions Support Center
----------------------------------------------------------------------
Click on Services Offerings Engineering Support Services
Cliquez sur Services Offerings et Engineering Support Services
[0](6029889,15) :
N0#00001+X0#0009d+X0#0009e+X0#0009f+S0#000a0+X0#00093+X0#00095+X0#0009f+N0#00002
[1] Arbre :
((X0#0009d,X0#0009e),((X0#0009f,((S0#000a0,X0#00093),X0#00095)),X0#0009f))
[1] Positions : 4+5+6+0+1+2+4:5+0:1+2:4:5+2:4:5:6+0:1:2:4:5:6
[1] Découpage : X0#0009d+X0#0009e+X0#0009f+(S0#000a0,X0#00093)+X0#00095+X0#0009f
[1](4543,13) :
N1#00001+X1#13902+X1#13903+X1#13904+X1#13905+X1#138f8+X1#13904+N1#00002
[2] Arbre : ((X1#13902,X1#13903),(((X1#13904,X1#13905),X1#138f8),X1#13904))
[2] Positions : 2:3+0:1+2:3:4+2:3:4:5+0:1:2:3:4:5+1+2+3+4+5+0
[2] Découpage : (X1#13902,X1#13903)+((X1#13904,X1#13905),X1#138f8)+X1#13904
[2](260,7) : N2#00001+X2#18977+X2#18978+X2#18979+N2#00002
[3] Arbre : (X2#18977,(X2#18978,X2#18979))
[3] Positions : 1:2+0:1:2+0+1+2
[3] Découpage : X2#18977+(X2#18978,X2#18979)
[3](84,5) : N3#00001+X3#1defc+X3#1defb+N3#00002
[4] Arbre : (X3#1defc,X3#1defb)
[4] Positions : 0:1+0+1
[4] Découpage : (X3#1defc,X3#1defb)
[4](3,3) : N4#00001+X4#21602+N4#00002
[4] : N4#00001+X4#21602+N4#00002
OK=> Cliquez sur Services Offerings et Engineering Support Services
----------------------------------------------------------------------
Softcopy collection kits of translated publications for each language are available
four to six weeks after planned availability
Les versions électroniques des publications traduites sont disponibles quatre à six
semaines après la date de commercialisation prévue
[0](189566,37) :
N0#00001+S0#000a3+S0#000a4+S0#000a5+X0#00004+S0#000a6+X0#000a7+S0#00072+S0#000a8+S0#
000a9+X0#000aa+X0#000ab+X0#000ac+X0#00091+X0#000ad+X0#000ae+X0#000af+X0#000b0+S0#000
b1+N0#00002
[1] Arbre :
(((((S0#000a3,(S0#000a4,S0#000a5)),X0#00004),(S0#000a6,X0#000a7)),((S0#00072,S0#000a
8),S0#000a9)),((X0#000aa,X0#000ab),(X0#000ac,(X0#00091,((X0#000ad,X0#000ae),((X0#000
af,X0#000b0),S0#000b1))))))
[1] Positions :
5+12+13+14+15+9+16+3+10+11+3:5+15:16+9:10+13:14+13:14:15:16+12:13:14:15:16+11:12:13:
14:15:16+9:10:11:12:13:14:15:16+3:5:9:10:11:12:13:14:15:16
[1] Découpage :
(S0#000a3,(S0#000a4,S0#000a5))+X0#00004+(S0#000a6,X0#000a7)+((S0#00072,S0#000a8),S0#
000a9)+X0#000aa+X0#000ab+X0#000ac+X0#00091+X0#000ad+X0#000ae+X0#000af+X0#000b0+S0#00
0b1
[1](15,27) :
N1#00001+S1#13910+X1#1389e+X1#1390a+S1#13911+X1#1390e+X1#13908+X1#13909+X1#138f4+X1#
1390b+X1#1390c+X1#1390d+X1#1390f+S1#13912+N1#00002
[2] Arbre :
((S1#13910,(X1#1389e,(X1#1390a,S1#13911))),(X1#1390e,((X1#13908,(X1#13909,X1#138f4))
,((X1#1390b,X1#1390c),(X1#1390d,(X1#1390f,S1#13912))))))
[2] Positions :
6:7+8:9+10:11+1:2+5:6:7+8:9:10:11+5:6:7:8:9:10:11+4:5:6:7:8:9:10:11+1:2:4:5:6:7:8:9:
10:11+10+4+11+5+6+1+2+7+8+9
[2] Découpage :
S1#13910+X1#1389e+(X1#1390a,S1#13911)+X1#1390e+X1#13908+(X1#13909,X1#138f4)+(X1#1390
b,X1#1390c)+X1#1390d+(X1#1390f,S1#13912)
[2](11,19) :
N2#00001+S2#18981+X2#18939+X2#18980+X2#1897d+X2#1897f+X2#1897a+X2#1897b+X2#1897c+X2#
1897e+N2#00002
```

```
[3] Arbre :
(((S2#18981,(X2#18939,X2#18980)),X2#1897d),(X2#1897f,((X2#1897a,X2#1897b),(X2#1897c,
X2#1897e))))
[3] Positions : 5:6+7:8+1:2+1:2:3+5:6:7:8+4:5:6:7:8+1:2:3:4:5:6:7:8+3+4+1+5+6+7+8+2
[3] Découpage :
S2#18981+X2#18939+X2#18980+X2#1897d+X2#1897f+(X2#1897a,X2#1897b)+X2#1897c+X2#1897e
[3](10,17) :
N3#00001+S3#1df03+X3#1ded7+X3#1df02+X3#1defe+X3#1deff+X3#1defd+X3#1df00+X3#1df01+N3#
00002
[4] Arbre :
(((S3#1df03,(X3#1ded7,X3#1df02)),X3#1defe),(X3#1deff,(X3#1defd,(X3#1df00,X3#1df01)))
)
[4] Positions : 6:7+1:2+1:2:3+5:6:7+4:5:6:7+1:2:3:4:5:6:7+3+4+1+5+6+7+2
[4] Découpage :
(((S3#1df03,(X3#1ded7,X3#1df02)),X3#1defe),(X3#1deff,(X3#1defd,(X3#1df00,X3#1df01)))
)
[4](3,3) : N4#00001+X4#21603+N4#00002
[4] : N4#00001+X4#21603+N4#00002
OK=> Les versions électroniques des publications traduites sont disponibles quatre à
six semaines après la date de commercialisation prévue
----------------------------------------------------------------
dated November 16 1999
datée du 16 novembre 1999
[0](189618,9) : N0#00001+X0#000b8+X0#000b9+X0#000ba+X0#000bb+N0#00002
[1] Arbre : (((X0#000b8,X0#000b9),X0#000ba),X0#000bb)
[1] Positions : 3+0+1+2+0:1+0:1:2+0:1:2:3
[1] Découpage : X0#000b8+X0#000b9+X0#000ba+X0#000bb
[1](35,9) : N1#00001+X1#13917+X1#13918+X1#13919+X1#1391a+N1#00002
[2] Arbre : (((X1#13917,X1#13918),X1#13919),X1#1391a)
[2] Positions : 0:1+0:1:2+0:1:2:3+2+3+0+1
[2] Découpage : ((X1#13917,X1#13918),X1#13919)+X1#1391a
[2](4,5) : N2#00001+X2#18987+X2#18988+N2#00002
[3] Arbre : (X2#18987,X2#18988)
[3] Positions : 0:1+0+1
[3] Découpage : (X2#18987,X2#18988)
[3](3,3) : N3#00001+X3#1df0a+N3#00002
[3] : N3#00001+X3#1df0a+N3#00002
OK=> datée du 16 novembre 1999
----------------------------------------------------------------
Trademarks
Marques
[0](226853,3) : N0#00001+X0#000bc+N0#00002
[0] : N0#00001+X0#000bc+N0#00002
OK=> Marques
----------------------------------------------------------------
Windows and Windows NT are trademarks of Microsoft Corporation
Windows et Windows NT sont des marques de Microsoft Corporation
[0](2230387,19) :
N0#00001+X0#000bd+X0#00031+X0#000bd+X0#000be+X0#000aa+X0#000bf+S0#00055+X0#000c0+X0#
000c1+N0#00002
[1] Arbre :
(X0#000bd,((X0#00031,(X0#000bd,X0#000be)),((X0#000aa,(X0#000bf,S0#00055)),(X0#000c0,
X0#000c1))))
[1] Positions :
5+7+8+0+1+2+3+4+2:3+4:5+7:8+1:2:3+4:5:7:8+1:2:3:4:5:7:8+0:1:2:3:4:5:7:8
[1] Découpage :
X0#000bd+X0#00031+X0#000bd+X0#000be+X0#000aa+X0#000bf+S0#00055+X0#000c0+X0#000c1
[1](1057,18) :
N1#00001+X1#1391b+X1#138b6+X1#1391b+X1#1391c+X1#1390e+X1#1391d+S1#13961+X1#1391e+X1#
1391f+N1#00002
[2] Arbre :
(X1#1391b,((X1#138b6,(X1#1391b,X1#1391c)),(((X1#1390e,X1#1391d),S1#13961),(X1#1391e,
X1#1391f))))
[2] Positions :
2:3+4:5+7:8+1:2:3+4:5:7:8+1:2:3:4:5:7:8+0:1:2:3:4:5:7:8+4+3+0+5+7+1+8+2
[2] Découpage :
X1#1391b+X1#138b6+(X1#1391b,X1#1391c)+X1#1390e+X1#1391d+S1#13961+(X1#1391e,X1#1391f)
[2](592,8) : N2#00001+X2#1898c+X2#1894f+X2#18989+X2#1897d+S2#189ae+X2#1898a+N2#00002
[3] Arbre : ((X2#1898c,(X2#1894f,X2#18989)),(X2#1897d,(S2#189ae,X2#1898a)))
[3] Positions : 3:5+1:2+0:1:2+0:1:2:3:5+2+3+5+0+1
[3] Découpage : X2#1898c+X2#1894f+X2#18989+X2#1897d+S2#189ae+X2#1898a
[3](240,5) : N3#00001+X3#1df0c+X3#1dee1+X3#1df0d+X3#1defe+S3#1df1e+X3#1df2f+N3#00002
[4] Arbre : ((X3#1df0c,(X3#1dee1,X3#1df0d)),(X3#1defe,(S3#1df1e,X3#1df2f)))
[4] Positions : 3:5+1:2+0:1:2+0:1:2:3:5+2+3+5+0+1
[4] Découpage : X3#1df0c+X3#1dee1+X3#1df0d+X3#1defe+S3#1df1e+X3#1df2f
[4](293,1) : N4#00001+X4#2161e+X4#21617+X4#21642+X4#2171c+S4#21612+X4#21641+N4#00002
[5] Arbre : (X4#2161e,(X4#21617,(X4#21642,(X4#2171c,(S4#21612,X4#21641)))))
```

```
[5] Positions : 3:5+2:3:5+1:2:3:5+0:1:2:3:5+3+2+5+0+1
[5] Découpage : X4#2161e+X4#21617+X4#21642+X4#2171c+(S4#21612,X4#21641)
[5](92,1) : N5#00001+X5#23552+X5#2354f+X5#23568+X5#23662+N5#00002
[6] Arbre : (X5#23552,(X5#2354f,(X5#23568,X5#23662)))
[6] Positions : 2:3+1:2:3+0:1:2:3+0+1+2+3
[6] Découpage : X5#23552+X5#2354f+X5#23568+X5#23662
[6](30,0) : N6#00001+X6#245e3+X6#2499b+N6#00002
[7] Arbre : (X6#245e3,X6#2499b)
[7] Positions : 0:1+0+1
[7] Découpage : X6#245e3+X6#2499b
[7](13,0) : N7#00001+X7#24d25+N7#00002
[7] : N7#00001+X7#24d25+N7#00002
KO=> et
[02:14:21] 4 segments différents sur 17 segments traités.
```