# Modèles de segmentation et ré-ordonnancement en traduction statistique

Samy Blusseau, Alexandre Allauzen, François Yvon

**NOTES et DOCUMENTS LIMSI N$^o$ : 2011 - 08**
**20 septembre 2011**

# Modèles de segmentation et ré-ordonnancement en traduction statistique

S. Blusseau, A. Allauzen, F. Yvon

**Notes et Documents LIMSI N° : 2011 - 08**
**20 septembre 2011**

**Auteurs (Authors) :** S. Blusseau, A. Allauzen, F. Yvon

**Titre :** Modèles de segmentation et ré-ordonnancement en traduction statistique

**Title:** Segmentation and Reordering Models for Statistical Machine Translation.

**Nombre de pages (Number of pages) :** 58

**Résumé :** *Lors d'une traduction, les positions des unités de langue (mots ou groupes de mots) qui se correspondent ne se retrouvent pas nécessairement dans le même ordre - par exemple, il y a souvent permutation entre nom et adjectif au passage du français ("la maison bleue") à l'anglais ("the blue house"). Ce dernier aspect de la traduction est nommé* ***ré-ordonnancement****. L'étape de segmentation et permutation vise à découper la phrase source* $\mathbf{f}$ *en segments (groupes de mots contigus)* $\widehat{f}_1\widehat{f}_2...\widehat{f}_K$ *représentant des unités de sens, et à les disposer dans un ordre* $\tilde{f}_1...\tilde{f}_K$ *plus naturel pour la langue cible. L'objectif de ce stage a été de trouver, malgré l'imprécision des alignements automatiques de mots, les permutations qui remettent les mots source dans le "bon" ordre, celui des mots de la phrase cible correspondante. En d'autres termes, il s'agissait de générer de bonnes données d'apprentissage du point de vue du ré-ordonnancement.*

**Mots clés :** Traduction statistique, ré-ordonnancement, segmentation, permutations

**Key words:** Statistical translation, reordering, segmentation, permutations

Rapport de stage de recherches
8 avril 2011 - 15 septembre 2011

# *Modèles de segmentation et de ré-ordonnancement en traduction statistique*


Samy BLUSSEAU

Ecole Normale Supérieure de Cachan - Master Recherche
"Mathématiques, Vision, Apprentissage"

Responsables de stage LIMSI :
**Alexandre ALLAUZEN**
**François YVON**

# Remerciements

Ces six mois de stage ont conclu le master "MVA" ainsi que ma formation d'ingénieur de façon très agréable. Je tiens à remercier tous mes collaborateurs, qui ont contribué à ce qu'il en soit ainsi.

Merci en particulier à Alexandre ALLAUZEN et François YVON, dont le savoir et l'expérience ont été des repères précieux, et qui m'ont néanmoins laissé une grande liberté d'initiative, me permettant d'apprendre beaucoup et de véritablement apprécier le travail de recherche. Leur constante bonne humeur, leur ouverture ainsi que leur compétence ont suscité mon estime et ont participé à me motiver au quotidien.
Enfin, je remercie tous mes collègues stagiaires, pour la bonne ambiance et la richesse culturelle qui ont été constantes dans notre bureau pendant six mois !

# Table des matières

# Chapitre 1

# Contexte et objectifs du stage

## 1.1 Présentation du laboratoire

Le Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur (**LIMSI**), est une unité propre de recherche du CNRS (UPR3251), associée aux universités Pierre et Marie Curie (UPMC) ainsi que Paris-Sud 11, et situé à Orsay (91).
Le laboratoire est constitué deux départements : Communication Homme-Machine (CHM) d'une part, et Mécanique - Energétique (MECA) d'autre part. Chacun de ces départements se divise en plusieurs groupes.

Groupes CHM :

- Audio et Acoustique (AA)
- Architecture et Modèles pour l'Interaction (AMI)
- Information, Langue Ecrite et Signée (ILES)
- Perception, Cognition et Usages (CPU)
- Réalité Virtuelle et Augmentée (RV& A) ou Virtualité et Environnement Immersif pour la Simulation et l'Expérimentation (VENISE)
- Traitement du Langage Parlé (TLP)

Groupes MECA :

- Aérodynamique Instationnaire (AERO)
- Convection et Rotation (CORO)
- Transferts Solide-Fluide (TSF)

Le groupe TLP couvre plusieurs thèmes de recherches en rapport avec le langage, tels que la reconnaissance de la parole, de la langue et du locuteur, l'indexation audio et la traduction. Il est spécialisé dans des approches de traitement *statistique* du langage. J'ai effectué mon stage de recherche au sein de l'équipe de traduction, dans le cadre du programme européen **QUAERO** [1].

## 1.2 La traduction automatique

Le problème de la traduction automatique est de concevoir un système capable de construire une phrase cible $\mathbf{e} = e_1 e_2 ... e_I$ à partir d'une phrase source $\mathbf{f} = f_1 f_2 ... f_J$ donnée en entrée [2]. Aujourd'hui, les corpus parallèles (ensembles de textes déjà traduits en plusieurs langues par des humains, avec

1. www.quaero.org
2. les $e_i$ et $f_j$ étant les mots des phrases.

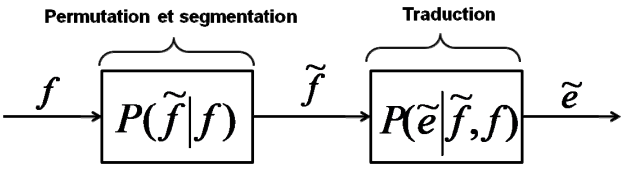


FIGURE 1.1: Processus de traduction

correspondances phrase à phrase) sont nombreux, volumineux et facilement accessibles. C'est pourquoi les approches statistiques, qui utilisent ces exemples pour apprendre le processus de traduction, constituent désormais l'état de l'art en traduction automatique [8].

Dans le cadre statistique, une façon de modéliser le processus de traduction serait de construire $\mathbf{e}$ en associant à chaque mot $f_j$ de $\mathbf{f}$ sa traduction $e_j$ la plus fréquemment observée dans l'ensemble d'apprentissage [2]. Ce modèle de traduction mot-à-mot est très naïf et ne permet pas de construire de bonnes traductions (i.e. qui paraitraient naturelles à un être humain) pour deux raisons principales : tout d'abord, la traduction d'un mot dépend de son contexte, et donc de ses mots (voire phrases) voisin(e)s ; ensuite, lors d'une traduction correcte, les positions des unités de langue (mots ou groupes de mots) qui se correspondent ne se retrouvent pas nécessairement dans le même ordre - par exemple, il y a souvent permutation entre nom et adjectif au passage du français ("*la maison bleue*") à l'anglais ("*the blue house*"). Ce dernier aspect de la traduction est nommé **ré-ordonnancement**.

## 1.3 Ré-ordonnancement et segmentation

Une approche plus élaborée que la traduction mot à mot consiste à concevoir la construction d'une phrase cible comme la succession de **deux étapes** : **la première**, dite de segmentation et permutation, vise à découper la phrase source $\mathbf{f}$ en *segments* (groupes de mots contigus) $\widehat{f}_1\widehat{f}_2...\widehat{f}_K$ représentant des unités de sens, et à les disposer dans un ordre $\tilde{f}_1...\tilde{f}_K$ plus naturel pour la langue cible ; **la deuxième** étape consiste alors à traduire ces segments source par les segments cible $\tilde{e}_1...\tilde{e}_K$ les plus vraisemblables d'après ce qui a été observé lors de l'apprentissage. On bâtit ainsi une traduction qui correspond aux segmentations (de la source et de la cible) et à l'ordre les plus probables. Autrement dit la phrase cible $\mathbf{e}^*$ vérifie l'équation :

$$\mathbf{e}^* = arg\max_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) = arg\max_{\mathbf{e}} \sum_{\tilde{\mathbf{f}}} \sum_{\tilde{\mathbf{e}}:\mathbf{e}} p(\tilde{\mathbf{e}}, \tilde{\mathbf{f}}|\mathbf{f}), \tag{1.1}$$

où "$\tilde{\mathbf{e}}$ : $\mathbf{e}$" signifie que $\tilde{\mathbf{e}}$ est une segmentation possible de $\mathbf{e}$. Dans la somme précédente on peut décomposer chaque terme en deux facteurs

$$p(\tilde{\mathbf{e}}, \tilde{\mathbf{f}}|\mathbf{f}) = p(\tilde{\mathbf{f}}|\mathbf{f})p(\tilde{\mathbf{e}}|\tilde{\mathbf{f}}, \mathbf{f}), \tag{1.2}$$

qui reflètent respectivement la première et la seconde étape : $p(\tilde{\mathbf{f}}|\mathbf{f})$ désigne le modèle de ré-ordonnancement (permutation et segmentation) et $p(\tilde{\mathbf{e}}|\tilde{\mathbf{f}}, \mathbf{f})$ représente le modèle de traduction. La figure 1.1 illustre le processus de traduction induit par l'equation 1.2.

Il existe dans la littérature différentes façons de mettre en œuvre l'équation 1.2. La plus répandue, et qui constitue l'état de l'art, est l'approche dite "**phrase based**" ( "à base de segments"), dont le système *Moses* est l'implémentation la plus utilisée [18, 9]).
Ces systèmes construisent les segments à partir des alignements de mots, en envisageant des unités de tailles variables, de 1 à une dizaine de mots. La figure 1.2 est une illustration de la méthode d'extraction

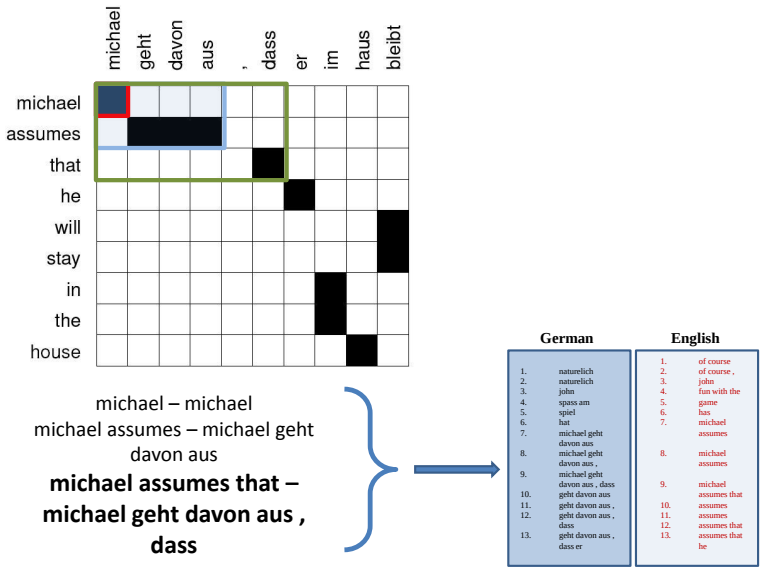


FIGURE 1.2: Extraction de trois segments (rectangles colorés) à partir d'alignements de mots (carrés noirs), par la méthode "phrase-based".

de segments.
En ce qui concerne le ré-ordonnancement, il peut par exemple s'effectuer de manière

- **monotone** : le système laisse le modèle de ré-ordonnancement évaluer et choisir, à chaque étape, le segment source à traduire parmi les différents choix possibles. Un tel modèle joue sur des considérations locales, comme les écarts entre la position d'un segment dans la phrase source et celle de son correspondant en cible (on parle de *distorsion*), ou encore des classes de déplacement (typiquement *monotone* - lorsque deux segments sources adjacents restent dans le même ordre, *"swap"* - lorsqu'ils échangent leurs positions, ou *discontinue* - lorsqu'ils ne sont plus juxtaposés dans le nouvel ordre).

- **hiérarchique** : on envisage alors la traduction comme la construction d'un arbre "syntaxique" de la phrase cible. Cette construction est synchronisée à l'analyse syntaxique de la phrase source. Cette dernière approche adopte une vision plus globale de la phrase, permettant d'envisager des déplacements de grande amplitude et pas seulement locaux (**ré-ordonnancement hiérarchique** [4]).

Dans tous les cas, les problèmes de segmentation et de permutation sont liés et doivent être traités. Il s'agit d'apprendre, de générer et d'évaluer des permutations de mots ou de segments de mots. Quelle que soit l'approche, on cherche toujours à réduire la complexité liée à l'énumération des permutations, qui ne peut être exhaustive dès que l'on dépasse certaines longueurs de phrase.

## 1.4 Objectifs du stage

Le système de traduction développé au LIMSI [12, 5], appelé $n$-code, propose une mise en oeuvre de l'équation 1.2 un peu différente de celles mentionnées dans la section précédente. En effet $n$-code envisage d'abord un ensemble de permutations de la phrase source - ensemble qu'il représente par un treillis, puis génère des traductions à partir des (ou de certaines des) permutations de la phrase source induites par le treillis. L'apprentissage s'articule alors de la manière suivante :

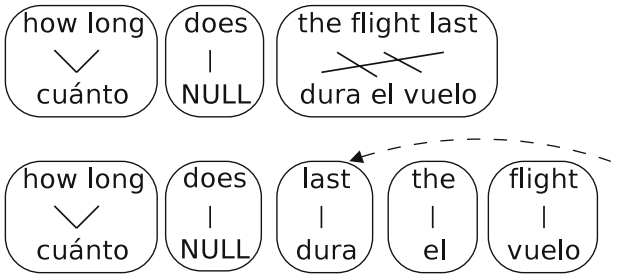


FIGURE 1.3: Segmentation en tuples et permutation des mots source (ici anglais), grâce aux alignements et à l' *unfolding* (figure : [5]).

- Estimer les alignements mot-à-mot pour chaque paire de phrases d'apprentissage (voir [14] pour plus de détails sur les estimations d'alignement).

- En fonction des liens d'alignement, réordonner la phrase source de manière à déplier au mieux les liens (*unfold*) pour ensuite segmenter la paire de phrase en unités bilingues ("tuples"). Cette étape correspond à la transformation $\mathbf{f} \rightarrow \tilde{\mathbf{f}}$. On peut signaler que les tuples, assez similaires aux segments des modèles "phrase-based", diffèrent de ces derniers car ils constituent une segmentation *unique* de la phrase, et le nombre de tuples est bien moindre que le nombre de segments possibles.

- Apprendre les déplacements des mots sources à l'aide de règles de réécriture basées sur les catégories grammaticales des mots (modèle de ré-ordonnancement).

- Apprendre le modèle de traduction ($\tilde{\mathbf{f}} \rightarrow \mathbf{e}$ ).

Une illustration des deux premières étapes est donnée à la figure 1.3. L'étape d'*unfolding* revient donc à permuter des mots source avant segmentation. Ainsi, en phase d'apprentissage, les patrons de ré-ordonnancement appris sont très dépendants de l'exactitude des alignements mot-à-mot, générés automatiquement.
Par conséquent, un objectif de mon stage a été de trouver, malgré l'imprécision des alignements automatiques, les permutations qui remettent les mots source dans le "bon" ordre, celui des mots de la phrase cible correspondante. En d'autres termes, il s'agissait de générer de bonnes données d'apprentissage du point de vue du ré-ordonnancement.
L'objectif suivant était de concevoir une stratégie pour représenter, apprendre et inférer ces ré-ordonnancemen
de manière efficace et pertinente.

# Chapitre 2

# Etude bibliographique

Je rends compte ici de quelques lectures intéressantes, qui ont inspiré les travaux menés au cours du stage, et pourraient inspirer leur poursuite.

## 2.1 Un peu de combinatoire

Les articles sur le ré-ordonnancement en traduction statistique font souvent référence à la littérature sur les permutations. J'ai donc naturellement été porté à me documenter à ce sujet (le plus souvent, suite à des suggestions de mes directeurs de stage). Ces lectures, si elles n'ont pas toutes donné lieu à des applications concrètes au cours du stage, ont permis une compréhension plus profonde des objets mathématiques que sont les permutations, et pourront aussi être la base de nouvelles idées pour des travaux futurs. Je choisis de rendre compte ici des éléments qui m'ont été les plus utiles, ou qui me paraissent pouvoir l'être plus tard.

### 2.1.1 Encodage de permutations

Une permutation $\sigma$ d'ordre $n \in \mathbb{N}^*$ est une bijection de $\{1, ..., n\}$ dans lui-même. Elle représente un agencement de $n$ objets distincts et convient donc pour modéliser un ordre possible des mots ou segments d'une phrase. L'ensemble des permutations d'ordre $n$ est de cardinal $n!$ et forme un groupe (pour la loi de composition usuelle "$\circ$") appelé groupe symétrique, noté $\mathfrak{S}_n$. On peut représenter une permutation $\sigma$ sur une ligne par ses valeurs $(\sigma(1)\sigma(2)...\sigma(n))$, et ainsi énumérer les éléments de $\mathfrak{S}_3$ ( Table 2.1).

| **Agencements de mots** | **Permutations** |
|---|---|
| la maison bleue | 123 |
| la bleue maison | 132 |
| maison la bleue | 213 |
| maison bleue la | 231 |
| bleue la maison | 312 |
| bleue maison la | 321 |

TABLE 2.1: Eléménts de $\mathfrak{S}_3$ et agencements associés de trois mots

Cette représentation est canonique, mais il en existe bien d'autres. Il convient pour nous d'utiliser celles qui permettront un encodage efficace. Par exemple, l'article [11] étudie quatre autres représen-

tations (très similaires entre elles) basées sur les *inversions* d'une permutation. Une inversion d'une permutation $\sigma$ est un couple $(i, j)$ d'entiers tels que $1 \leq i < j \leq n$ et $\sigma^{-1}(i) > \sigma^{-1}(j)$. Les inversions d'une permutation permettent de la caractériser entièrement. Ainsi peut-on représenter $\sigma$ par sa *table d'inversions* $E$, définie par :

$$\forall i \in \{1, ..., n\} \quad E(i) = \#\left\{j > i \ : \ \sigma^{-1}(i) > \sigma^{-1}(j)\right\}. \tag{2.1}$$

La table d'inversion est l'une des quatre représentations étudiées dans [11]. L'intérêt de cet article est la présentation d'algorithmes efficaces permettant d'encoder et de décoder une permutation en jouant sur ces diverses représentations. C'est une piste pour toute optimisation de nos implémentations.

| $i$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $\sigma(i)$ | 3 | 5 | 2 | 1 | 4 |
| $E(i)$ | 3 | 2 | 0 | 1 | 0 |

TABLE 2.2: Une permutation d'ordre 5 et sa table d'inversions ; les inversions de $\sigma$ sont $(1,2)$ $(1,3)$ $(1,5)$ $(2,3)$ $(2,5)$ $(4,5)$.

### 2.1.2 Sous-ensembles du groupe symétrique

Lors de la recherche d'un ré-ordonnancement de mots ou segments, en phase de traduction, on ne souhaite pas examiner toutes les permutations possibles. En effet, d'après l'équation 1.1, une phrase source peut être associée à plusieurs segmentations et, pour une segmentation et une permutation des segments, plusieurs hypothèses de phrases cible sont à envisager (on cherche un maximum sur les phrases $\mathbf{e}$), à compter avec leurs différentes segmentations ($\tilde{\mathbf{e}}$) ! Une recherche exhaustive sur tout cet espace est donc impossible (il s'agit d'un problème NP-difficile, comme le rappelle [18]), et réduire le nombre de permutations examinées est indispensable.
Or le tableau 2.1 montre qu'un certain nombre de permutations ne valent pas la peine d'être prises en compte, car elle n'ont pas plus d'intérêt linguistique en source qu'en cible. L'idéal serait de n'explorer qu'un sous ensemble de permutations "adaptées" à notre application, soit aux changements d'ordre caractéristiques des langues source et cible considérées.
L'article [17] a justement cherché à délimiter un espace de ré-ordonnancements adaptés à des paires de langues, en définissant les **Inversion Transduction Grammars**, que nous appellerons "grammaires ITG". De façon générale, ces grammaires permettent une analyse syntaxique ("parsing") synchrone de deux langages, d'où leur utilité en traduction.
Dans notre cadre d'étude, nous utilisons une grammaire inspirée des ITG afin de définir un type de permutations, que nous appellerons "permutations ITG".

#### Permutations ITG

Afin de définir ce sous-ensemble de $\mathfrak{S}_n$, considérons la grammaire suivante, pour un certain entier $n$ :

$$\begin{array}{rcll} S & \rightarrow & S_{1,n+1} & \\ S_{i,j} & \rightarrow & S_{i,k}S_{k,j} \mid S_{k,j}S_{i,k} & \forall\, 1 \leq i < k < j \leq n+1 \\ S_{i,i+1} & \rightarrow & i & \forall\, 1 < i \leq n+1 \end{array}$$

Cette grammaire génère des séquences de $n$ symboles $\sigma_1 ... \sigma_n$ tous distincts et à valeurs dans $\{1, ..., n\}$ : elle génère donc des permutations de $\mathfrak{S}_n$. Ce sont les permutations qui respectent cette grammaire que l'on appelle "permutations ITG". La figure 2.1 donne un exemple de dérivation d'une telle permutation. Un intérêt de ce sous ensemble, que nous noterons $\mathbf{ITG_n}$, est d'être significativement plus petit que $\mathfrak{S}_n$

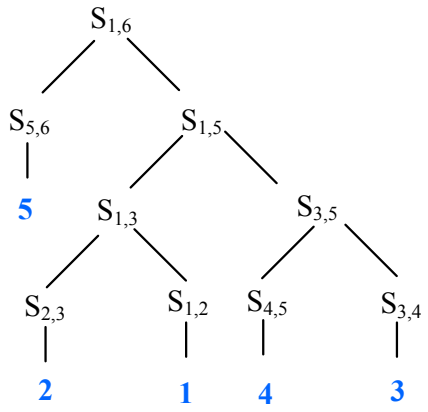


FIGURE 2.1: Une dérivation de la permutation ITG $\sigma = (52143)$

| $n$ | $\lvert\mathbf{ITG_n}\rvert$ | $\lvert\mathfrak{S}_n\rvert = n!$ | $\frac{\lvert\mathbf{ITG_n}\rvert}{\lvert\mathfrak{S}_n\rvert}$ |
|---|---|---|---|
| 1 | 1 | 1 | 1.000 |
| 2 | 2 | 2 | 1.000 |
| 3 | 6 | 6 | 1.000 |
| 4 | 22 | 24 | 0.917 |
| 5 | 90 | 120 | 0.750 |
| 6 | 394 | 720 | 0.547 |
| 7 | 1806 | 5 040 | 0.358 |
| 8 | 8 558 | 40 320 | 0.212 |
| 9 | 41 586 | 362 880 | 0.115 |
| 10 | 206 098 | 3 628 800 | 0.057 |

TABLE 2.3: Comparaison des tailles des ensembles $\mathbf{ITG_n}$ et $\mathfrak{S}_n$

(voir tableau 2.3), mais suffisamment grand pour inclure de nombreux ré-ordonnancements observés en pratique. En effet comme cela est démontré dans [15] le cardinal de $\mathbf{ITG_n}$ est $S_{n-1}$, le grand nombre de Schröder d'ordre $n-1$, qui croit exponentiellement ($S_{n+1}/S_n \approx 5.8$). De plus, comme nous le verrons plus loin, les permutations ITG peuvent être aisément être parsées et représentées par la forêt de leurs arbres de dérivation.

### Permutations OSS

Malgré tous les avantages que comporte le travail dans le sous-ensemble ITG (nous le verrons en particulier au chapitre 3), les algorithmes permettant de manipuler ces permutations restent de complexité assez importante (de l'ordre de $n^3$), ce qui tend à limiter le passage à l'application de tels algorithmes sur de gros corpus (plusieurs millions de phrases). C'est pourquoi tout autre sous ensemble du groupe symétrique, et notamment plus restreint, est susceptible de mériter notre attention.
Un autre sous ensemble de permutations, plus connu [1] , est celui des permutations que l'on peut trier à l'aide d'une pile (en anglais : "One Stack Sortable permutation", d'où l'acronyme OSS). Trier une permutation $\sigma$ à l'aide d'une pile consiste à lire ses symboles $\sigma(1)...\sigma(n)$ (par exemple de la gauche vers la droite), et à effectuer, à la lecture d'un symbole $\sigma(k)$ les actions suivantes :
– dépiler zéro (aucun), un, ou plusieurs symbole(s) du haut de la pile

1. En combinatoire, mais pas en traduction.

– empiler le symbole lu sur la pile

Lorsque tous les symboles ont été lus (et donc empilés une fois), ceux qui sont encore dans la pile sont dépilés jusqu'à vider la pile.

Dépiler un symbole revient à l'écrire en sortie, et on dit qu'une permutation a été triée à l'aide d'une pile si les empilements et dépilements ont permis d'écrire les symboles dans l'ordre croissant (autrement dit, on a écrit la permutation identité).

Une caractérisation des permutations triables à l'aide d'une pile (voir [1] pour des détails sur cette caractérisation) permet de montrer que ce sous ensemble (que nous noterons $\mathbf{OSS_n}$) est inclus dans $\mathbf{ITG_n}$. Nous avons donc un ensemble de permutations encore plus restreint à explorer et à représenter. Il se trouve que ce dernier est significativement plus petit que $\mathbf{ITG_n}$ : en effet le cardinal de $\mathbf{OSS_n}$ est le nombre de Catalan $C_n$ ([1]), et pour $n = 10$ on a déjà $C_{10} = 16796 \approx 0.0046 S_9$.

Les permutations "OSS" ne semblent pas avoir été considérées en traduction automatique. Il pourrait être intéressant de voir sur des données si ce type de ré-ordonnancement est courant ou non. Bien qu'ayant formulé, au cours du stage, un algorithme de test d'appartenance aux permutations OSS, je n'ai pas eu l'occasion de l'implémenter et de réaliser ces tests.

### 2.1.3 Analyse harmonique des permutations

La tâche de ré-ordonnancement consiste en un *choix* parmi un ensemble de permutations, ce qui suppose donc de pouvoir comparer ces permutations entre elles. Dès lors, on peut penser à définir des noyaux sur ces ensembles de permutation, les méthodes à noyaux permettant souvent des approches géométriques de comparaison entre objets.

L'article [10] propose justement des outils d'analyse harmonique (type transformée de Fourier) des permutations, qui permettent de définir des noyaux sur le groupe symétrique. On peut donc espérer utiliser ces notions afin de calculer des distances, des projections etc. entre permutations.

On remarque notamment qu'il est possible de définir un noyau de diffusion à partir d'une relation d'adjacence entre permutations. Or, s'il est classique de parler d'adjacence (invariante à droite) entre deux permutations qui diffèrent d'une seule transposition ($\sigma \sim \sigma'$ ssi il existe une transposition $\tau$ telle que $\sigma' = \tau \circ \sigma$), on peut aussi définir une relation d'adjacence entre des permutations qui diffèrent d'une permutation ITG, l'ensemble ITG étant symétrique ($\pi \in ITG \Leftrightarrow \pi^{-1} \in ITG$).

Ces pistes n'ont toutefois pas vraiment été approfondies au cours du stage, faute de temps.

Les articles présentés dans les sections suivantes ont l'intérêt d'aborder deux aspects du ré-ordonnancement qui correspondent à l'approche que nous avons envisagée : d'une part, le ré-ordonnancement des données d'apprentissage ; d'autre part, le ré-ordonnancement au moment du décodage.

## 2.2 Une approche générale

**"Novel reordering approaches in phrase-based statistical machine translation" : [7]** On lit ici des pistes intéressantes concernant les ré-ordonnancements en apprentissage et dans le décodage. Les auteurs remarquent d'abord l'intérêt d'avoir des alignements monotones (positions des mots source et des mots cible correspondants, classées dans le même ordre) pour l'extraction de segments (il en est de même pour l'extraction de tuples dans N-code). Ils proposent donc de ré-ordonnancer les phrases source ET cible, afin d'obtenir de nouveaux alignements "plus monotones" que les précédents. Pour ce faire, ils procèdent en 4 étapes :

1. A partir des alignements initiaux ils estiment, pour chaque paire de phrases $(f_1^J, e_1^I)$, une matrice $C$ dont chaque coefficient $c_{ij}$ représente le coût d'aligner $e_i$ avec $f_j$.

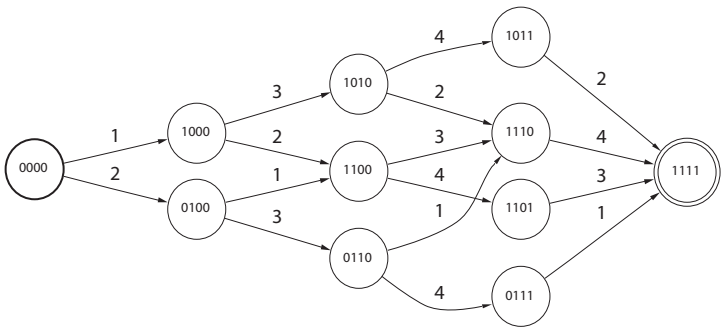


FIGURE 2.2: Automate représentant les permutations des mots source $f_1 f_2 f_3 f_4$ qui respectent les contraintes IBM pour une fenêtre de taille 2, ce qui signifie que l'on peut choisir de traduire, à chaque état, un des 2 premiers mots source parmi ceux non encore traduits. (figure reprise de [7]).

2. Ils définissent alors une première fonction d'alignement $A_1 : \{1, ..., J\} \to \{1, ..., I\}$, par

$$A_1(j) \; = \; arg \min_i c_{ij} \tag{2.2}$$

et ré-ordonnent ainsi chaque phrase source en une nouvelle phrase $\breve{f}_1^J$, où $\breve{f}_{j_1}$ précède tout mot $\breve{f}_{j_2}$ tel que $A_1(j_1) < A_1(j_2)$ (si $A_1(j_1) = A_1(j_2)$, l'ordre initial est probablement conservé).

3. Une nouvelle matrice $\breve{C}$ est estimée pour chaque paire de phrases $(\breve{f}_1^J, e_1^I)$, dont la source a été ré-ordonnée [2] .

4. **Soit :** une deuxième fonction d'alignement $A_2 : \{1, ..., I\} \to \{1, ..., J\}$ est définie par

$$A_2(i) \; = \; arg \min_j \breve{c}_{ij} \tag{2.3}$$

puis rendue monotone par une méthode tirée de [3] ;

**Soit**(et c'est le choix fait par les auteurs) : on calcule directement une fonction $A_3 : \{1, ..., I\} \to \{1, ..., J\}$ monotone, en déterminant par programmation dynamique le plus court chemin monotone entre la première et la dernière ligne de $\breve{C}$ (par monotone, les auteurs veulent probablement dire qu'au passage d'une ligne à l'autre, le chemin ne peut passer d'une colonne $j$ à une colonne d'indice strictement inférieur)

**Dans les deux cas**, on obtient un ré-ordonnancement en cible du corpus d'apprentissage.

En ce qui concerne le décodage, les auteurs proposent de représenter les permutations d'un espace de recherche, par un transducteur à états finis appelé *Lazy permutation automaton*. L'allure d'un tel

2. En utilisant par exemple la même méthode que pour estimer $C$, appliquée au nouveau corpus ré-ordonné en source ; les auteurs proposent une alternative, qui est de permuter les colonnes de $C$ suivant l'ordre $A_1$ pour obtenir $\breve{C}$.

automate est déterminée par les contraintes qui définissent l'espace de recherche. Ainsi, cette représentation est adaptée tant à des contraintes IBM ([2]) qu'aux permutations ITG (même si l'article ne détaille pas la construction du transducteur dans ce dernier cas).
De façon plus précise, pour des permutations d'ordre $n$, un état de l'automate est un vecteur de $n$ bits, dont ceux à 1 correspondent aux indices des mots source déjà traduits. Les $n$ bits de l'état initial sont donc tous à 0 et, ceux de l'état final, tous à 1. La transition d'un état à un autre correspond à la traduction d'un nouveau mot, dont l'indice étiquette cette transition.
Cette représentation a l'avantage de permettre la construction d'une permutation à la volée, en n'examinant que les transitions autorisées par les contraintes. De plus, la définition de scores favorisant les ré-ordonnancements monotones, permet de pondérer les arcs entre les états et d'élaguer les parties de l'automate les moins pertinentes.

## 2.3 Quelques modèles hiérarchiques

**"A simple and effective hierarchical phrase reordering model" [6] :** Cet article reprend le principe des modèles de ré-ordonnancement lexicalisé, en proposant une méthode permettant d'envisager des changements d'ordre plus généraux que ceux induits par le "lexicalized reordering" classique. Rappelons que cette approche définit le ré-ordonnancement comme un problème de classification : connaissant le segment source $\tilde{f}_j$ qui a été traduit par le segment cible précédent $\tilde{e}_i$, on associe une étiquette au nouveau segment cible $\tilde{e}_{i+1}$ selon l'orientation du segment $\tilde{f}_{j'}$ qu'il traduit, par rapport à $\tilde{f}_j$. Les étiquettes d'orientation possibles sont, le plus souvent, au nombre de 3 :

- "M" (Monotone), lorsque $j' = j + 1$
- "S" (Swap), lorsque $j' = j - 1$
- "D" (Discontinuous), dans les autres cas

Si l'on déplace les segments source en tenant compte des étiquettes des segments cible qui leurs correspondent, on ne peut faire que des transpositions entre segments adjacents, ce qui correspond à l'étiquette "swap". En effet des déplacements de segments plus généraux ne sont pas possibles, car ils correspondraient aux cas étiquetés "D" - étiquette qui ne donne pas d'information de ré-ordonnancement. Or ces déplacements sont très fréquents, et la classe "Discontinuous" regroupe la majeure partie des segments cible dans le modèle de ré-ordonnancement lexicalisé.
Le passage à un modèle hiérarchique, présenté dans [6], consiste à fusionner des segments extraits initialement pour former des segments plus longs, lorsque cela conduit à une configuration *monotone* ou *swap* (figure 2.3).
Une fois appris ce modèle de ré-ordonnancement sur les phrases d'apprentissage (modèle "*max-entropie*"), il est appliqué au moment du décodage grâce à l'algorithme "*shift-reduce*", qui permet d'effectuer, à la volée, les fusions de segments et l'étiquetage sur des hypothèses de traduction, et ainsi d'évaluer ces hypothèses.
Cette généralisation du modèle lexicalisé fait penser aux ré-ordonnancements ITG : en effet, lorsque les opérations de fusion aboutissent à des segments dont chacun est étiqueté monotone ou swap, on obtient bien une permutation ITG des mots source (figure 2.4).

**"Learning linear ordering problems for better translation" : [16]** L'article ne parle pas directement de modèle hiérarchique, mais le type de ré-ordonnancements étudié consiste bien à hiérarchiser les mouvements de mots source. En effet, il propose une utilisation très intéressante des permutations ITG. Les auteurs définissent une notion de voisinage $\mathcal{N}_{ITG}(\pi)$ d'une permutation $\pi \in \mathfrak{S}_n$, que l'on

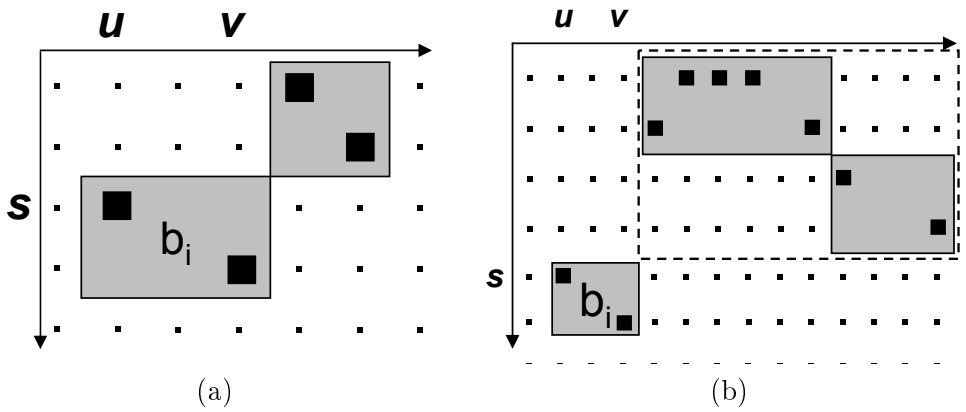


FIGURE 2.3: Illustration du principe de swap ; les ordonnées couvrent les mots source, les abscisses les mots cible ; les gros carrés noirs représentent des alignements entre mots source et cible, et les rectangles gris sont les segments ; dans la configuration (a) le segment $b_i$ est étiqueté "swap" par le modèle lexicalisé classique, alors que dans (b), il n'est vu comme swap que par le modèle hiérarchique, grâce à la fusion des deux segments de droite (figure reprise de [6]).

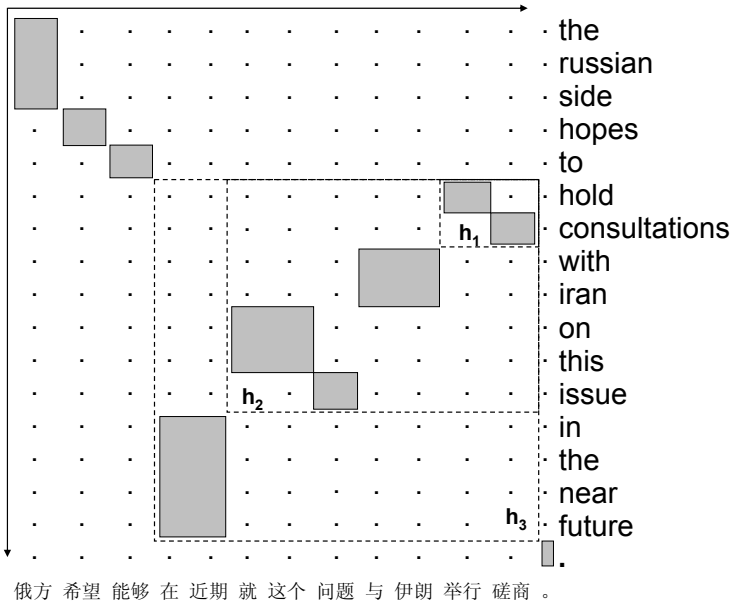


FIGURE 2.4: Ré-ordonnancement d'une phrase source chinoise, par le modèle hiérarchique lexicalisé ; on aboutit à une permutation ITG des 13 caractères chinois : "$1-2-3-11-12-9-10-6-7-8-4-5-13$" (figure reprise de [6]).

peut formuler de la façon suivante :

$$\pi' \in \mathcal{N}_{ITG}(\pi) \Leftrightarrow \pi^{-1} \circ \pi' \in \mathbf{ITG_n} \tag{2.4}$$

On retrouve dans cette formulation l'adjacence au sens ITG, évoquée plus haut.
Une phase d'apprentissage permet de déterminer des scores qui caractérisent, pour une paire de mots donnée, la préférence d'ordre de l'un par rapport à l'autre dans une phrase. On a ainsi une matrice de coefficients $B_{ij}$ indiquant chacun la préférence de trouver $e_i$ avant $e_j$ dans un ré-ordonanncement cohérent avec l'ordre des phrases cible. On peut alors calculer, à partir de cette matrice, un score pour toute permutation.
A partir de ces définitions de score et de voisinage, et d'une permutation initiale $\pi_0$ (l'identité dans ce cas), les auteurs parviennent à trouver, par programmation dynamique, la permutation $\pi^*$ de meilleur score dans le voisinage ITG de $\pi_0$. On peut itérer ainsi sur des optima successifs, jusqu'à atteindre un optimum local.
D'après l'article, cette méthode semble s'appliquer tant comme pré-traitement du corpus d'apprentissage (en ré-ordonnant de façon ITG les phrases sources à partir d'alignements), qu'au moment du décodage.
Cet article a été un point de départ pour les travaux réalisés et détaillés au chapitre 3 .

# Chapitre 3

# Correction du re-ordonnancement

N-code propose, avec l'étape *unfold*, une permutation des mots source de chaque phrase de l'ensemble d'apprentissage. Ces patterns de ré-ordonnancement sont ensuite appris afin d'inférer de nouvelles permutations de la source au moment de la traduction. Or ils ne sont pas toujours satisfaisants (il ne correspondent pas à l'ordre des mots d'une phrase cible naturelle ; voir l'exemple 1 ), et risquent donc de fausser l'apprentissage. De plus la stratégie de recherche de permutations lors de la traduction semble pouvoir être améliorée.

**(1)** (Corpus *Newsco*, Phrase 66) Exemple classique de mauvais positionnement des déterminants :
Français : Pour le peuple pakistanais , en particulier les partisans de la politicienne , ce sont les services des renseignements , seuls ou en collaboration avec les extrémistes , qui auraient finalement décidé de l'éliminer .
Anglais : So , in the eyes of Pakistan's people , and especially of Bhutto's supporters , the intelligence services , either alone or in collaboration with extremists , finally decided to eliminate her .
Français "unfold" : Pour le , les la pakistanais peupe en particulier de politicienne partisans , ce sont les des renseignements services , seuls ou en collaboration avec les extrémistes , qui auraient finalement décidé de l'éliminer .

Les permutations ITG constituent un sous ensemble de permutations intéressant pour la traduction. D'une part, parce que leur définition découle de celle des "Inversion Transduction Grammars", qui semblent être assez pertinentes pour décrire les mouvements de mots dans les langages parlés ; d'autre part, parce que c'est un sous ensemble qui, bien que significativement plus petit que le groupe symétrique tout entier, est assez grand pour ne pas trop perdre en généralité lorsque on l'explore.

Ainsi, on se propose de contraindre les ré-ordonnancements des phrases source de l'ensemble d'apprentissage, à être des permutations ITG. On utilisera ainsi la structure de cet ensemble pour faire un apprentissage pertinent, et pour rechercher efficacement des ré-ordonnancements possibles lors de la traduction.
Pour cela, on cherche les permutations ITG les plus "proches" de celles générées par l'unfolding de N-code. Par "plus proches", on entend les permutations ITG qui respectent au mieux les contraintes d'ordre impliquées par le ré-ordonnancement initial. Par exemple, si l'unfolding appliqué à une phrase de 4 mots donne la permutation en source "2413", qui n'est pas ITG, on peut chercher une permutation ITG qui viole le moins de contraintes parmi la liste suivante, déduite de la permutation initiale :

1 placé après 2 (symbolisé $2 \prec 1$)
$1 \prec 3$
$4 \prec 1$
$2 \prec 3$
$2 \prec 4$

$$4 \prec 3$$

Ces $\frac{n(n-1)}{2}$ contraintes peuvent être considérées comme toutes également importantes, ou avoir des poids différents. Quoi qu'il en soit, on reconnait ici le "Linear Ordering Problem" (LOP), dont on cherche une solution dans le sous ensemble "ITG" du groupe symétrique. Dans le formalisme du LOP, l'importance d'une contrainte est modélisée par le coût associé à la violation de cette contrainte.
En plus de la recherche de la meilleure permutation ITG au sens des contraintes imposées par l'unfolding, on s'est intéressé à celle des $N$ meilleures.

## 3.1 Recherche de la meilleure permutation ITG

On utilise pour cela l'algorithme de parsage tabulaire CYK [1]. La table utilisée est la partie "triangulaire supérieure" d'un tableau de $n \times n$ cases (figure 3.1).
Au cours de l'algorithme on va remplir chacune des cases avec une règle de la grammaire $ITG_n$, et ce de façon *optimale* au sens des contraintes imposées. En effet, une case $(i, i+k)$ devra décrire le meilleur arrangement ITG des symboles $\{i, ..., i+k-1\}$, autrement dit la permutation ITG d'ordre $k-1$ de ces symboles, qui satisfasse au mieux les contraintes imposées. Ainsi, construire la case $(1, n+1)$ de la table reviendra à trouver la permutation d'ordre $n$ recherchée. On sent bien qu'il suffira de procéder par programmation dynamique pour couvrir progressivement un nombre croissant de symboles jusqu'à tous les couvrir, et de façon optimale à chaque étape.
Plutôt que décrire formellement l'algorithme général, nous commencerons par présenter l'exemple d'une démarche naturelle pour résoudre notre problème d'optimisation (section 3.1.1). Nous ferons ensuite une série de 5 remarques permettant de généraliser cette démarche (section 3.1.2), jusqu'au formalisme que nous avons finalement implémenté (section 3.1.3).

### 3.1.1 Un exemple

Prenons l'exemple des contraintes, énoncées plus haut, issues de la permutation non ITG "2413". Celles-ci peuvent se traduire en termes de scores, par exemple probabilistes, c'est à dire en affectant à une contrainte "$i \prec j$" une probabilité $P_{ij} > 0.5$ et $P_{ji} = 1 - P_{ij}$.
Supposons qu'il importe beaucoup que 2 soit en première position, et que 1 soit avant 3, et que le reste importe moins. Voici un jeu de scores cohérent avec les contraintes de "2413" et avec ces priorités d'importance :

$$\begin{aligned} P_{12} &= 0.1 \\ P_{13} &= 0.9 \\ P_{14} &= 0.45 \\ P_{23} &= 0.95 \\ P_{24} &= 0.9 \\ P_{34} &= 0.45 \end{aligned}$$

On peut maintenant commencer à remplir la table CYK. Par définition, on remplit les cases $(i, i+1)$ $(i \in \{1,2,3,4\})$ avec les symboles terminaux $i$ (ou la règle terminale $S_{i,i+1} \rightarrow i$) ; on affecte à chacune de ces cases (ou règle) un score de 1 (ce qui revient à dire $P_{ii} = 1$, ou encore qu'il n'y a pas de choix à faire pour remplir ces cases diagonales car il n'y a qu'une façon d'arranger un symbole tout seul).
Remplissons à présent les 3 cases de la diagonale suivante, de coordonnées $(i, i+2)$ pour $i \in \{1,2,3\}$. La case $(1,3)$ se remplit avec une règle permettant de dériver les règles des cases $(1,2)$ et $(2,3)$. Il n'existe que deux règles qui conviennent : $S_{1,3} \rightarrow S_{1,2}, S_{2,3}$ et $S_{1,3} \rightarrow S_{2,3}, S_{1,2}$, qui correspondent aux

1. Ce nom vient des trois auteurs de l'algorithme : Cocke, Younger et Kasami. Il s'agit d'un algorithme permettant de trouver toutes les dérivations d'une phrase par une grammaire hors contexte.

deux arrangements possibles "12" et "21" pour couvrir les symboles $\{1, 2\}$.
Le choix entre ces deux règles est évident : les contraintes nous indiquent une préférence pour que 2 soit placé avant 1 ; cet ordre correspond à la seconde règle, qui permettra bien d'écrire ces deux symboles dans l'ordre préféré. On remplit donc $(1, 3)$ avec la règle $S_{1,3} \rightarrow S_{2,3}, S_{1,2}$ et le score correspondant $P_{21} = 0.9$.
De même, on met dans $(2, 4)$ la règle $S_{2,4} \rightarrow S_{2,3}, S_{3,4}$, de score $P_{23} = 0.95$, et dans $(3, 5)$ la règle $S_{3,5} \rightarrow S_{4,5}, S_{3,4}$ de score $P_{43} = 0.55$.

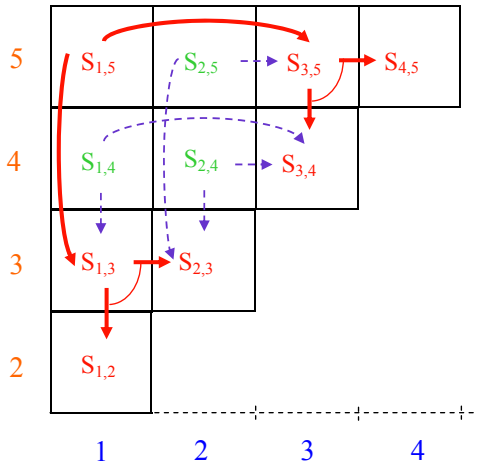


FIGURE 3.1: Exemple d'application de l'algorithme CYK, avec $n = 4$ ; les flèches qui partent de chaque case indiquent la meilleure façon de construire cette case, les arcs entre les flèches indiquant un "swap" ; les cases dont les non terminaux sont en rouge sont celles retenues pour la dérivation finale.

Passons à la diagonale suivante (composée des cases $(1, 4)$ et $(2, 5)$) et intéressons-nous d'abord à $(1, 4)$. Quatre règles sont candidates pour cette case, chacune correspondant à un choix des cases utilisées pour construire la case courante, associé à un ordre dans lequel on les prend (monotone ou non) :

$$S_{1,4} \rightarrow S_{1,2}, S_{2,4}$$

$$S_{1,4} \rightarrow S_{2,4}, S_{1,2}$$

$$S_{1,4} \rightarrow S_{1,3}, S_{3,4}$$

$$S_{1,4} \rightarrow S_{3,4}, S_{1,3}$$

La première de ces 4 règles implique le choix des cases $(1, 2)$ et $(2, 4)$, dans un ordre monotone. Le score de cette règle va donc se calculer à partir de celui de ces deux cases et des scores $P_{12}$ et $P_{13}$ associés aux ordres $1 \prec 2$ et $1 \prec 3$. Il vaut :

$$\begin{aligned} score(S_{1,4} \rightarrow S_{1,2}, S_{2,4}) &= score(1, 2) \times score(2, 4) \times P_{12} \times P_{13} \\ &= 1 \times 0.9 \times 0.1 \times 0.9 \\ &\approx 0.081 \end{aligned} \tag{3.1}$$

De même, on calcule :

$$\begin{aligned} score(S_{1,4} \to S_{2,4}, S_{1,2}) &= score(1,2) \times score(2,4) \times P_{21} \times P_{31} \\ score(S_{1,4} \to S_{1,3}, S_{3,4}) &= score(1,3) \times score(3,4) \times P_{13} \times P_{23} \\ score(S_{1,4} \to S_{3,4}, S_{1,3}) &= score(1,3) \times score(3,4) \times P_{31} \times P_{32} \end{aligned} \tag{3.2}$$

et on trouve comme plus grand score (et donc comme score et règle pour la case $(1,4)$) :

$$score(1,4) = score(S_{1,4} \to S_{1,3}, S_{3,4}) \approx 0.7695 \tag{3.3}$$

Dans notre exemple, la meilleure façon ITG d'écrire les symboles 1 à 3 est donc d'écrire au mieux les symboles 1 à 2 (ici "21"), et de placer à droite le symbole 3 (donc "213").
En procédant de même pour la case $(2,5)$, on trouve :

$$score(2,5) = score(S_{2,5} \to S_{2,3}, S_{3,5}) \approx 0.4703 \tag{3.4}$$

autrement dit, la meilleure permutation ITG des symboles 2 à 4 est d'écrire au mieux les symboles 3 à 4 (ici "43"), et de placer le symbole 2 à gauche (donc "243").

Enfin, parmi les 6 choix possibles (figure 3.2) , le score et la règle pour la dernière case, $(1,5)$, sont les suivants :

$$\begin{aligned} score(1,5) = score(S_{1,5} \to S_{1,3}, S_{3,5}) &= score(1,3) \times score(3,5) \times P_{13} \times P_{14} \times P_{23} \times P_{24} \\ &= 0.9 \times 0.55 \times 0.9 \times 0.45 \times 0.95 \times 0.9 \\ &\approx 0.1714 \end{aligned} \tag{3.5}$$

Ainsi, le meilleur arrangement ITG d'ordre 4, au sens des contraintes fixées, consiste à écrire au mieux les symboles 1 à 2, et d'écrire à droite le meilleur arrangement des symboles 3 à 4, ce qui nous donne "2143" (figure 3.3).
Ce résultat n'est pas surprenant : violer une seule des contraintes suffisait à se ramener de "2413" à une permutation ITG (la seule autre permutation non ITG d'ordre 4 étant "3142", qui viole *toutes* les contraintes), et l'algorithme nous a bien fait enfreindre la règle la plus "faible", à savoir "$4 \prec 1$".

### 3.1.2 Généralisation

Afin de généraliser la démarche vue dans l'exemple précédent, quelques définitions et remarques sont nécessaires.

#### Quelques définitions

**Définition 3.1.1** *On appelle* **règles d'ordre** $k$ *($k \geq 1$) les règles de membre gauche* $S_{i,i+k}$.

**Définition 3.1.2** *On appelle* **règles monotones d'ordre** $k$ *($k \geq 2$) et* **d'indice** $p$ *($1 \leq p < k$) (respectivement* **règles swap d'ordre** $k$ **et d'indice** $p$ *), les règles de type* $S_{i,i+k} \to S_{i,i+p}, S_{i+p,i+k}$ *(respectivement* $S_{i,i+k} \to S_{i+p,i+k}, S_{i,i+p}$*).*

**Définition 3.1.3** *La propriété d'être monotone ou swap pour une règle d'ordre* $k \geq 2$ *est appelée la* **monotonie de la règle**.

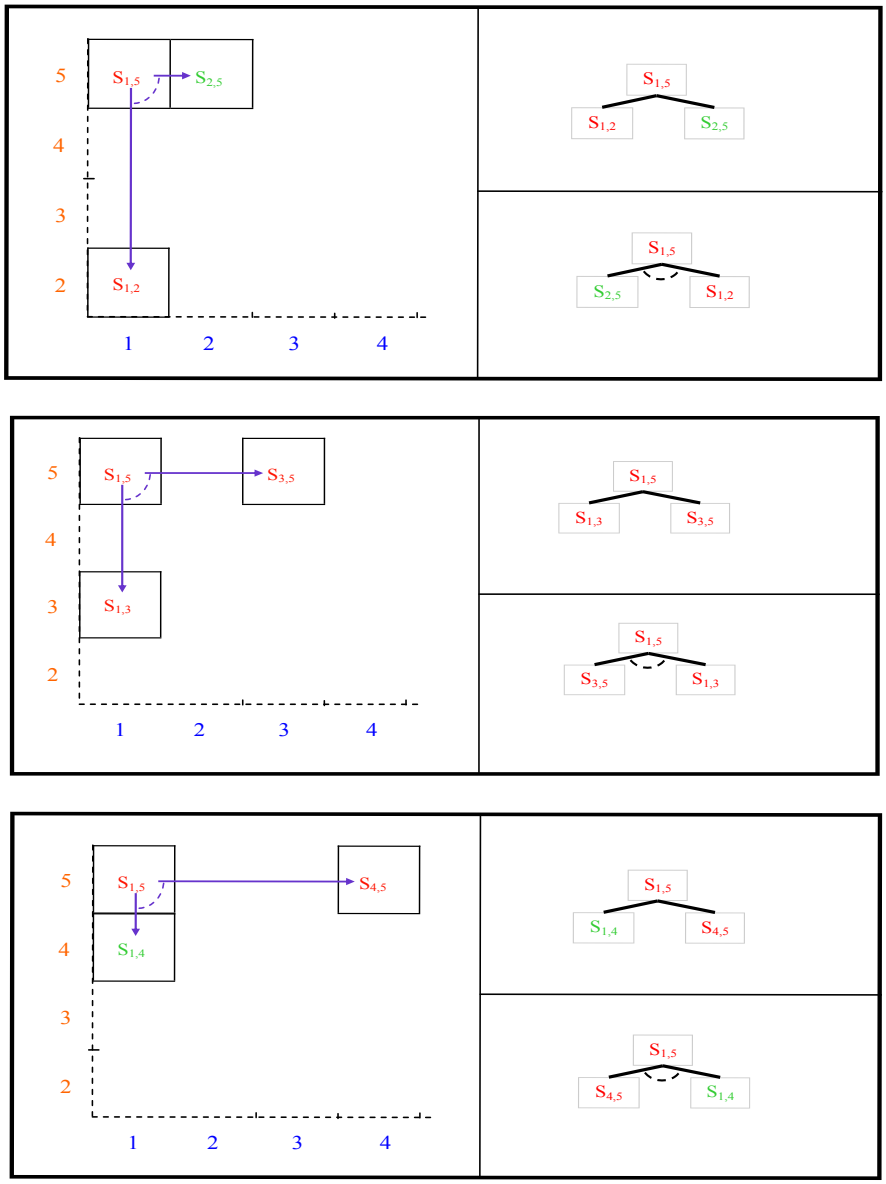


FIGURE 3.2: Les 6 choix de règle possibles pour la case $(1,5)$ (3 paires de non terminaux, chacune envisagée en ordre monotone et swap).

**Définition 3.1.4** *On appelle* **fils droit** *(resp.* **fils gauche** *) le non-terminal $S_{i+p,i+k}$ (resp. $S_{i,i+p}$) d'une règle d'ordre $k$ $(\geq 2)$ et d'indice $p$.*

**Définition 3.1.5** *Une* **règle terminale** *est une règle d'ordre $k = 1$, i.e. du type $S_{i,i+1} \rightarrow i$.*

Pour les définitions qui suivent, on considère un arbre de dérivation :

**Définition 3.1.6** *Un non terminal $S_{i,i+k}$ $(k \geq 2)$ ayant un fils droit et un fils gauche dans la dérivation considérée, est appelé* **noeud**.

**Définition 3.1.7** *Une* **feuille** *est un non terminal n'ayant pas de fils dans la dérivation considérée.*

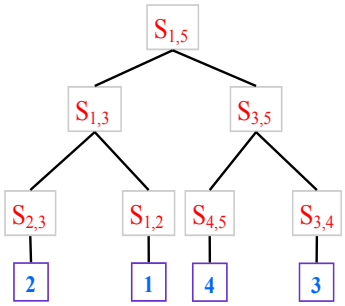


FIGURE 3.3: Dérivation donnant la meilleure permutation ITG au sens des contraintes de l'exemple 3.1.1

Dans une dérivation complète, seuls les membres gauche des règles non terminales sont des feuilles.

**Définition 3.1.8** *La* **monotonie** *et le* **fils droit d'un noeud** *sont ceux de la règle dont ce noeud est le membre gauche, dans la dérivation considérée.*

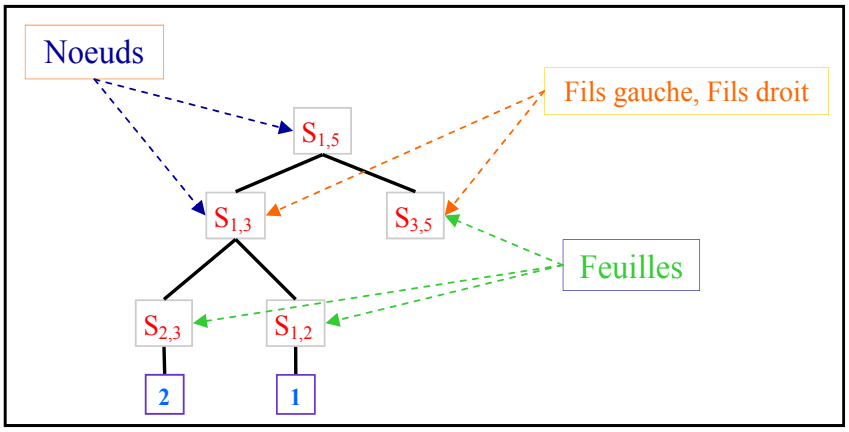


FIGURE 3.4: Illustration des définitions précédentes, sur une dérivation partielle.

### Quelques remarques

**Remarque 1** On montre aisément (et le vérifier sur l'exemple précédent est assez convaincant) que si les poids $P_{ij}$ sont tous strictement positifs, le problème ne dépend que des rapports $\frac{P_{ij}}{P_{ji}}$. On peut donc, par exemple, remplacer les $P_{ij}$ par de nouveaux poids $R_{ij}$ définis par

$$R_{ij} = \begin{cases} \frac{P_{ji}}{P_{ij}} & si \quad i > j \\ 1 & sinon \end{cases} \tag{3.6}$$

sans modifier le problème : le classement des permutations par rapport à leur adéquation aux contraintes, est inchangé car $\frac{R_{ij}}{R_{ji}} = \frac{P_{ij}}{P_{ji}}$ (voir Annexe A pour plus de détails).

Nous avons donc systématiquement travaillé avec des matrices de poids contenant toute l'information dans $\frac{n(n-1)}{2}$ coefficients (un coefficient par paire de symboles), typiquement les coefficients triangulaires supérieures ou inférieurs.

**Remarque 2** Si les phrases à traiter dépassent une certaine longueur, i.e. si $n$ devient grand, les scores des permutations risquent de devenir très petits, ce qui peut poser des problèmes numériques. Nous avons donc naturellement pris les logarithmes des poids, ce qui revient à changer les produits en sommes dans l'algorithme expliqué plus haut, et (tenant compte de la remarque précédente) à utiliser une matrice de poids $Q$ vérifiant par exemple :

$$Q_{ij} = \log R_{ij} = \begin{cases} \log(\frac{P_{ji}}{P_{ij}}) & si \quad i > j \\ 0 & sinon \end{cases} \tag{3.7}$$

En appliquant la remarque 1., on constate que le problème est désormais entièrement déterminé par les écarts $Q_{ij} - Q_{ji}$.

**Remarque 3** De façon analogue à ce qui a été fait dans l'exemple précédent, et en tenant compte des remarques ci-dessus, le score d'une règle ITG se calcule de la façon suivante :

$$score(S_{i,i+k} \to S_{i,i+p}, S_{i+p,i+k}) = score(i, i+p) + score(i+p, i+k) + \sum_{l=i}^{i+p-1} \sum_{m=i+p}^{i+k-1} Q_{lm} \tag{3.8}$$

pour une règle de type monotone,

$$score(S_{i,i+k} \to S_{i+p,i+k}, S_{i,i+p}) = score(i, i+p) + score(i+p, i+k) + \sum_{l=i}^{i+p-1} \sum_{m=i+p}^{i+k-1} Q_{ml} \tag{3.9}$$

pour une règle de type swap,

$$score(S_{i,i+1} \to S_{i,i+p}, S_{i+p,i+k}) = score(i, i+1) = 0 \tag{3.10}$$

pour une règle terminale.

Ces définitions reprennent ce qui a été fait intuitivement dans l'exemple initial : le score d'une règle d'ordre $k \geq 2$ (i.e. une règle candidate pour une case $(i, i+k)$), permettant de dériver les symboles $\{i, ..., i+k-1\}$, dépend

- du choix de la décomposition de ce bloc de symboles en sous blocs $\{i, ..., i+p-1\}$ et $\{i+p, ..., i+k-1\}$, et
- du choix de l'ordre relatif entre les blocs (monotone ou swap).

Etant donnée une décomposition (c'est à dire un entier $p \in \{1, ..., k-1\}$) et un ordre relatif, le score de la règle se calcule à partir de la meilleure façon de couvrir les sous blocs (qui donne les scores des cases $(i, i+p)$ et $(i+p, i+k)$), et du gain à placer tous les symboles du premier bloc à gauche (pour un choix d'ordre monotone) ou à droite (pour une règle swap) du second bloc (c'est la double somme des poids $Q_{lm}$ ou $Q_{ml}$).

La règle de plus grand score parmi les candidates est alors celle choisie pour remplir la case $(i, i+k)$, et le score de la règle devient celui de la case.
Ce qui change dans les formules (3.8 - 3.10) par rapport à l'exemple initial, est la manipulation de scores logarithmiques (que l'on note $Q$ au lieu de $P$), d'où la présence de sommes au lieu de produits et l'initialisation des scores à 0 au lieu de 1.

**Remarque 4** Si l'on regarde de plus près les équations (3.1) , (3.2) et (3.5), on remarque que le calcul des scores tel que nous l'avons fait dans l'exemple est loin d'être optimal.
Tout d'abord, pour des règles "concurrentes" (i.e. une règle monotone et son homologue swap), on voit apparaitre, par exemple, $P_{12} \times P_{13}$ pour une règle et $P_{21} \times P_{31}$ pour l'autre (éq. (3.1) et (3.2) ). Ceci est redondant et, en appliquant la remarque 1, on s'affranchit de cette redondance : avec des scores de type (3.6), il suffit de calculer $R_{21} \times R_{31}$ puisque nous savons que l'autre produit $(R_{12} \times R_{13})$ vaut 1 (ou respectivement, avec les notations additives, on se borne à calculer $Q_{21} + Q_{31}$, sachant que $Q_{12} + Q_{13}$ vaut 0).
Ensuite, le calcul du score de la case $(1, 5)$ (eq. 3.5), fait à nouveau intervenir $P_{13} \times P_{23}$, déjà calculé en (3.2) (tout comme $P_{14} \times P_{24}$, au cours de calculs omis dans l'exposé).

Afin d'éviter cela, on définit (à la manière de [16] la grandeur $\Delta(i, k, p)$ comme le gain à choisir une règle de type swap par rapport à sa correspondante monotone, c'est à dire le gain à placer les symboles $\{i+p, ..., i+k-1\}$ à gauche des symboles $\{i, ..., i+p-1\}$, donc en ordre non monotone.

Formellement :

$$\begin{array}{rcl} \Delta(i,k,p) & = & score(S_{i,i+k} \to S_{i+p,i+k}, S_{i,i+p}) - score(S_{i,i+k} \to S_{i,i+p}, S_{i+p,i+k}) \\ & = & \sum_{l=i}^{i+p-1} \sum_{m=i+p}^{i+k-1} (Q_{ml} - Q_{lm}) \end{array} \tag{3.11}$$

Ainsi avec des poids définis comme dans (3.7), on peut récrire

$$\left\{ \begin{array}{rcl} score(S_{i,i+k} \to S_{i,i+p}, S_{i+p,i+k}) & = & score(i, i+p) + score(i+p, i+k) \\ score(S_{i,i+k} \to S_{i+p,i+k}, S_{i,i+p}) & = & score(i, i+p) + score(i+p, i+k) \quad + \ \Delta(i,k,p) \end{array} \right. \tag{3.12}$$

ce qui est intéressant à deux titres : d'une part, on retrouve dans cette écriture le fait que le calcul de score de deux règles concurrentes ne fait intervenir qu'une seule fois les coefficients $Q_{lm}$ ; d'autre part, [16] donne une relation de récurrence entre les $\Delta$ d'ordres successifs, ce qui permet de ne jamais calculer deux fois la même somme :

$\forall\, k \in \{1, \ldots, n\}$ , $\forall\, i \in \{1, \ldots, n-k+1\}$

$$\left\{ \begin{array}{rcl} \Delta(i,k,0<p<k) & = & \Delta(i,k-1,p) \quad + \quad \Delta(i+1,k-1,p-1) \quad - \quad \Delta(i+1,k-2,p-1) \\ & & \qquad\qquad + \qquad Q_{i+k-1,i} \qquad\quad - \qquad Q_{i,i+k-1} \\ \\ \\ \\ \Delta(i,k,p=0) & = & \Delta(i,k,p=k) \ = \ 0 \end{array} \right. \tag{3.13}$$

**Remarque 5** Tous les éléments sont réunis pour écrire proprement l'algorithme de résolution du "Linear Ordering Problem" restreint aux permutations ITG.
Auparavant, faisons une dernière remarque qui a son importance dans l'implémentation que nous avons faite. Si la démarche précédente assure de trouver la meilleure permutation ITG au sens de certaines contraintes, la dérivation n'est pas forcément unique. En effet, la grammaire ITG est ambigüe et, bien que la permutation "2143" de l'exemple précédent n'a qu'une dérivation possible, ce n'est pas le cas en général (il suffit de considérer l'identité "1234" pour s'en convaincre).

Considérons par exemple l'ordre non ITG "52413", avec un poids plus faible sur $1 \prec 3$ que sur les autres contraintes. En procédant comme vu plus haut, on trouvera pour meilleure permutation ITG "52431". Or cette permutation présente deux dérivations possibles (voir figure(3.5)).
Ceci comporte au moins deux inconvénients :

a) Si on ne force pas les permutations ITG à n'avoir qu'une dérivation possible, l'algorithme n'est pas optimal : en effet, il se peut que l'on ait à comparer et choisir entre deux options qui donnent le même résultat. Dans l'exemple de "52413" qui devient "52431", parmi les options possibles pour remplir la dernière case, $(1, 6)$, se trouvent :
- "couvrir les symboles $\{1, ..., 4\}$ dans l'ordre "2431" et placer $\{5\}$ à gauche",
et :
- " couvrir les symboles $\{2, ..., 5\}$ dans l'ordre "5243", et placer $\{1\}$ à droite".
Ces deux dérivations aboutissent au même résultat avec un même score. On préfèrerait donc ne pas avoir à les envisager toutes les deux et ne comparer, au cours de l'algorithme, que des options strictement distinctes (avec éventuellement des scores égaux).

b) Notre objectif est de générer de nouvelles données d'apprentissage avec des ré-ordonnancements exclusivement ITG, dans le but d'apprendre à ré-ordonnancer de nouvelles phrases selon la grammaire ITG. Dans cette optique, il est important que deux phrases source ré-ordonnancées de la même façon contribuent aux statistiques d'un même schéma de ré-ordonnancement (ou pattern). Or un pattern étant construit à partir des règles ITG, nous devons garantir une unique dérivation pour une permutation donnée.

Heureusement, des formes normales ont déjà été proposées pour les dérivations ITG. L'une d'elles, que nous avons retenue pour sa simplicité, s'appelle "forme normale à droite".

**Définition 3.1.9** *Une dérivation ITG est dite* **de forme normale à droite** *si, pour tout noeud de l'arbre de dérivation, son fils droit est soit une feuille, soit de monotonie opposée à celle du noeud père (i.e. monotone si le noeud père est swap, et swap si le noeud père est monotone).*

Ainsi, la dérivation représentée sur la première ligne de la figure(3.5) est normale à droite, mais pas celle de la ligne du bas.
Un raisonnement par induction sur $k \geq 2$ permet de se convaincre de l'*existence* d'une dérivation de forme normale à droite, pour toute permutation ITG des symboles $\{i, ..., i + k\}$. On peut ensuite montrer par l'absurde l'unicité de la dérivation sous forme normale à droite.
Pour intégrer cette normalisation dans la recherche de meilleure permutation ITG il suffit alors, au cours du remplissage d'une case, de ne considérer que les règles dont la monotonie est opposée à celle

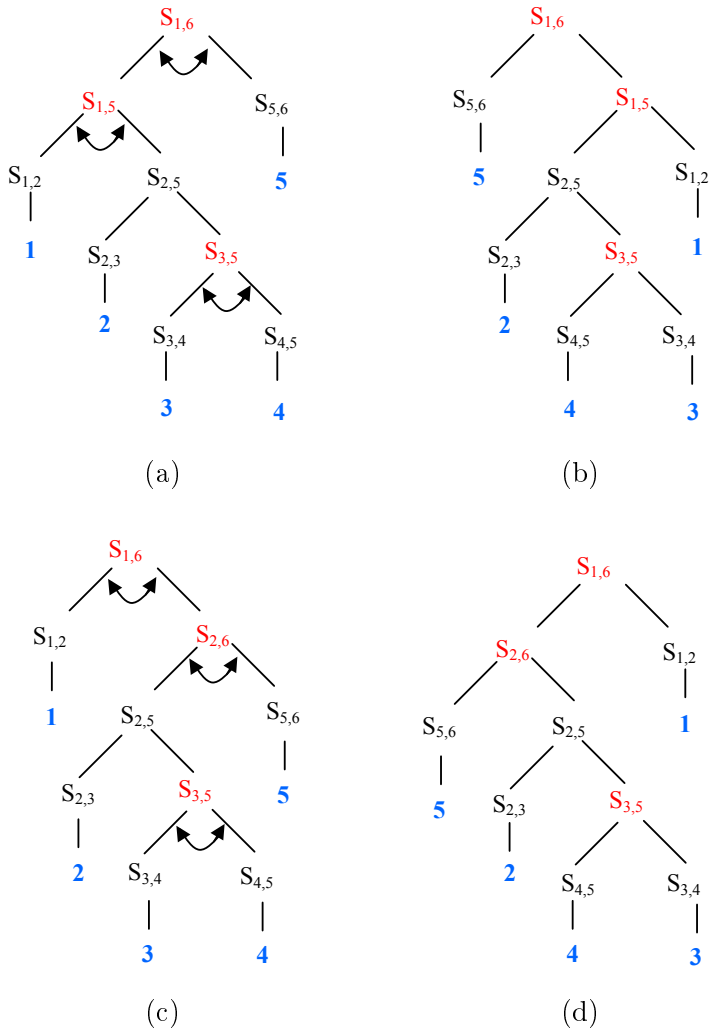


FIGURE 3.5: Deux dérivations pour une même permutation ; dérivation de forme "normale à droite" en haut et non normalisée en bas ; les représentations de gauche permettent de mieux visualiser les fils droits (laissés dans les branches droites) et l'alternance de noeuds swap (en rouge) et monotones (en noir) ; dans les arbres de droite, les swaps ont été "appliqués", i.e. les fils droits sont représentés à gauche des fils gauches lorsqu'il y a un swap.

de leur fils droit (déterminée lors d'une étape antérieure de l'algorithme).

### 3.1.3 Algorithme

Des cinq remarques précédentes découle l'algorithme 1 détaillé dans l'Annexe B, et que nous avons entièrement implémenté en C++. Il permet de remplir toutes les cases de la table CYK, et on reconstitue la dérivation de la meilleure permutation en lisant les règles choisies, à partir de la case $(1, n+1)$. Pour une case d'ordre $k$ $(i, i+k)$ donnée, $case(i, i+k).p$ est l'indice de la règle à appliquer, et indique que la case courante se construit à partir des cases $(i, i+p)$ et $(i+p, i+k)$ ; $case(i, i+k).m$ est la monotonie de la règle, 1 pour monotone et 0 pour swap.
Ainsi, l'algorithme 1 appliqué à l'exemple initial, nous donne pour la case $(1, 5)$ :

$$\begin{aligned} case(1,5).p &= 2 \\ case(1,5).m &= 1 \end{aligned}$$

ce qui signifie que $(1, 5)$ se construit à partir des cases $(1, 3)$ et $(3, 5)$ avec la monotonie 1 (c'est à dire en ordre monotone) ; autrement dit, la dérivation de la meilleure permutation commence par l'application de la règle : $S_{1,5} \rightarrow S_{1,3}, S_{3,5}$.
Ceci nous amène à regarder les cases pointées $(1, 3)$ et $(3, 5)$ :

$$\begin{aligned} case(1,3).p &= 1 \quad et \quad case(1,5).m = 0 \\ case(3,5).p &= 1 \quad et \quad case(3,5).m = 0 \end{aligned}$$

Dans chacune des deux cases la règle choisie est de type swap (la monotonie "$m$" vaut 0), et l'indice $p$ vaut 1 (les nouvelles cases pointées sont $(1, 2)$ et $(2, 3)$ d'une part, $(3, 4)$ et $(4, 5)$ d'autre part ) ; la dérivation se poursuit donc par :

$$S_{1,3} \rightarrow S_{2,3}, S_{1,2}$$

$$S_{3,5} \rightarrow S_{4,5}, S_{3,4}$$

Enfin les 4 cases pointées, de type $(i, i+1)$, contiennent nécessairement une règle terminale $S_{i,i+1} \rightarrow i$, ce qui achève la dérivation.

En ce qui concerne la complexité, l'algorithme CYK a une complexité en $O(n^3)$.
La méthode que nous venons de décrire en généralisant un exemple particulier, est très similaire à ce qui est fait dans [16]. Toutefois, nous avons légèrement optimisé cette dernière démarche en imposant la forme normalisée des dérivations ITG. Dans la section suivante, nous proposons une extension plus importante de cette solution au Linear Ordering Problem.

## 3.2 Recherche des $N$ meilleures permutations ITG

L'article [13] propose un algorithme donnant les $N$ plus courts chemins entre n'importe quel état et l'ensemble des états finaux, au sein d'un automate à états finis. Plus précisément, il explique comment rendre déterministe un automate à états finis puis applique, aux automates déterministes, un algorithme efficace pour déduire les $N$-plus courts chemins connaissant la distance DU plus court chemin de chaque état à l'ensemble des états finaux.
Pour appliquer cette méthode, il est nécessaire :

– de définir un automate déterministe, ses états, ses transitions et leurs étiquettes

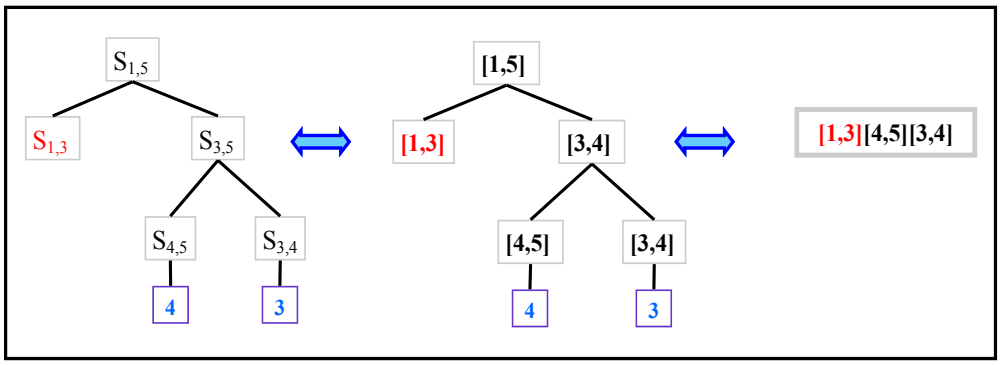


FIGURE 3.6: Trois représentations d'un même état (en rouge : la feuille à développer pour passer à un état suivant).

– de connaitre le meilleur chemin entre chaque état et l'ensemble des états finaux

Suite à la recherche de *la* meilleure permutation ITG, ce cadre nécessaire au calcule des $N$ meilleures se déduit assez naturellement.
En effet, la case $(1, n+1)$ de la table CYK (ou de façon équivalente, le non terminal $S_{1,n+1}$), peut être vue comme l'état initial, depuis lequel on progresse, de choix de règle en choix de règle, vers une dérivation complète d'une permutation, qui constituerait un état final. Les états intermédiaires seraient alors les dérivations partielles (i.e. qui comportent encore des non terminaux à développer), et l'application d'une règle constituerait une transition d'un état à un autre.
Dans ce qui suit nous expliquons plus en détail le formalisme qui, sur la base de ces idées, a abouti à un algorithme de recherche des $N$-meilleures permutations ITG suivant un ensemble de contraintes.

### 3.2.1 Etats et transitions

**Définition 3.2.1** *On définit comme* **état** *une dérivation ITG (partielle ou complète) de forme normale à droite.*

Dans la représentation que l'on fera des états, on "appliquera" les swaps, c'est à dire que le fils droit $S_{i+p,i+k}$ d'une règle d'ordre $k$ et d'indice $p$, apparaîtra dans la branche de droite de la règle monotone $S_{i,i+k} \rightarrow S_{i,i+p}, S_{i+p,i+k}$, mais dans la branche gauche de la règle swap $S_{i,i+k} \rightarrow S_{i+p,i+k}, S_{i,i+p}$.
Grâce à la contrainte de forme normale à droite, un état est entièrement défini par la liste, ordonnée de gauche à droite, de ses feuilles.
Remarquons aussi qu'il y a équivalence entre un non terminal $S_{i,i+k}$ et la case $(i, i+k)$ d'une table CYK (figure 3.6). On fera donc souvent la confusion entre les les deux.

**Définition 3.2.2** *On appelle* **feuille à développer** *d'un état, sa feuille non terminale la plus à droite.*

**Définition 3.2.3** *Un état est dit* **final** *lorsque toutes ses feuilles sont terminales.*

Un état final est donc une dérivation complète d'une permutation, et il n'a pas de feuille à développer.

**Définition 3.2.4** *Soit un état $E_0$ non final et $S_{i,i+k}$ $(k \geq 2)$ sa feuille à développer.*
*L'* **étiquette d'une transition** *pour quitter cet état est alors un couple $(p, m) \in \{1, ..., k-1\} \times \{0, 1\}$.*

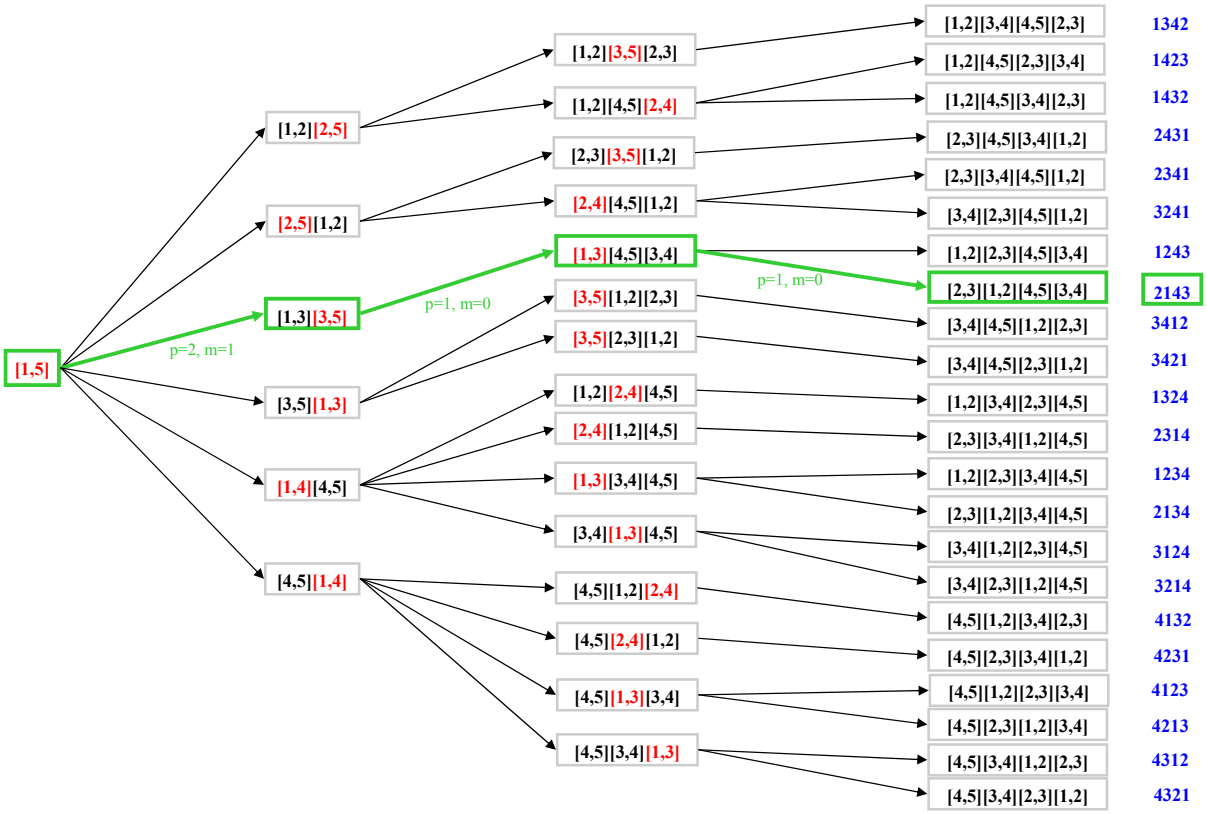


FIGURE 3.7: Automate fini déterministe pou $n = 4$ ; chaque état, encadré en gris, est représenté par la liste ordonnée de ses feuilles (désignées comme des cases $[i, i + k]$) ; en vert le chemin de l'état initial vers l'état final "2143", avec les étiquettes sur les transitions correspondantes ; en rouge, dans chaque état, la feuille à développer.

Ce couple définit la règle d'ordre $k$, d'indice $p$ et de monotonie $m$, qui permet de passer à l'état $E_1$, dont la liste (ordonnée) des feuilles est celle de $E_0$ dans laquelle on a remplacé $S_{i,i+k}$ par $S_{i,i+p}, S_{i+p,i+k}$ si $m = 1$, ou par $S_{i+p,i+k}, S_{i,i+p}$ si $m = 0$. Comme l'illustre la figure (3.7), ces définitions conduisent à la construction d'un automate fini déterministe, avec un unique chemin passant par chaque état.
Le fait de développer systématiquement la feuille non terminale la plus à droite assure le déterminisme de l'automate (car il n'y a pas deux transitions issues d'un même état qui portent la même étiquette), et c'est l'utilisation de la forme normale à droite qui, éliminant certaines transitions, implique l'unicité du chemin passant par un état donné.
Par ailleurs, on retrouve bien toutes les permutations ITG parmi les états finaux, et chaque chemin menant de l'état initial à un état final, correspond bien à la dérivation sous forme normale à droite de la permutation correspondante.

### 3.2.2 Plus court chemin

Comme nous l'avons dit au début de la section, l'algorithme de recherche des $N$ meilleures permutations que nous avons implémenté, s'appuie sur la connaissance du plus court chemin[2] entre chaque état et l'ensemble des états finaux, dans l'automate défini précédemment.

2. On parlera plutôt du *meilleur* chemin car, depuis le début, nous raisonnons en termes de scores à maximiser, et non de distances à minimiser.

Il est clair que l'algorithme 1 nous donne le meilleur chemin entre l'état initial $[1, n+1]$ et l'ensemble des états finaux, ainsi que le score associé à ce chemin optimal. Qu'en est-il des autres états ?
Ce qui sépare un état non final des états finaux, est le développement (ou la dérivation) de ses feuilles non terminales. Il existe plusieurs façons de les développer, qui sont autant de chemins vers des états finaux.
Ainsi, le meilleur chemin entre un état et l'ensemble des états finaux est la meilleure dérivation des feuilles non terminales, et le score associé est la somme des scores de ces feuilles non terminales (vues comme des cases de table CYK [3]).
En réalité, il faut quelque peu nuancer cette définition : la contrainte de forme normale à droite suppose que nous soyons attentifs à deux points :

- lorsque nous parlons de meilleure dérivation d'une feuille, il s'agit en fait de la meilleure dérivation de forme normale à droite ; mais nous avons déjà pris ceci en compte dans l'algorithme 1, où nous construisons une dérivation de manière ascendante en imposant une opposition de monotonie entre un noeud et son fils droit .

- lorsque nous considérons un état, c'est une dérivation partielle de forme normale à droite que l'on regarde ; ainsi, si l'une des feuilles non terminales est un fils droit, il existe une contrainte "ascendante" sur la monotonie de la règle à choisir pour développer cette feuille. De plus, il se peut que la case correspondante soit également une feuille d'un autre état, au sein duquel elle peut être contrainte à une autre monotonie. Dans l'exemple où $n = 4$, la case $[2, 4]$ est un fils gauche pour l'état $[2, 4]\,[4, 5]\,[1, 2]$ (4e état de la 3e colonne dans la figure (3.7)), mais un fils droit à dériver en "swap" dans l'état $[1, 2]\,[2, 4]\,[4, 5]$, et un fils droit à dériver de façon monotone dans l'état $[2, 4]\,[1, 2]\,[4, 5]$ (8e et 9e états de la 3e colonne).
  Il faut donc connaître, pour chaque case, non pas *la* meilleure dérivation et *le* meilleur score associé, mais deux dérivations optimales avec leurs scores, une pour chaque cas de monotonie (swap et monotone). L'information retenue par l'algorithme 1 est donc insuffisante (sauf en ce qui concerne l'état initial $[1, n+1]$, dont l'unique feuille n'apparaît que dans cet état !).

Par ailleurs, si l'on veut trouver les $N$ meilleurs chemins, il faut (et suffit) que de chaque état non final $E$ partent $M_E = \min(N, \mathcal{N}_E)$ transitions au moins [4], où $\mathcal{N}_E$ est le nombre maximal de transitions issues de l'état $E$ [5]. Or les transitions issues d'un état non final sont données par les dérivations possibles de la feuille à développer de cet état. Par conséquent, nous avons besoin de connaitre, pour chaque case non terminale $[i, i+k]$, les $\mathcal{N}_k = \min(N, 2(k-1))$ meilleures dérivations normales à droite de cette case, avec leurs scores (qui se calculent toujours avec les équations (3.13) et (3.12) ).
En appelant alors *scoreMono* et *scoreSwap* les scores associés respectivement aux meilleures dérivation monotone et swap (et en gardant la notation *score* pour la meilleure dérivation en absolu, i.e. $score = \max(scoreMono, scoreSwap)$ ), on peut déduire le score $\Phi$ du meilleur chemin entre un état non final $E$ et l'ensemble des états finaux :

$$\Phi(E) = \sum_{f \in \mathcal{F}(E) \setminus \mathcal{F}_d(E)} score(f) \; + \sum_{f_d \in \mathcal{F}_d(E)} \{\, f_d.m \times scoreSwap(f_d) \; + \; (1 - f_d.m) \times scoreMono(f_d) \,\} \tag{3.14}$$

3. On peut même définir le score d'un état comme la somme des scores de *toutes* ses feuilles, les feuilles terminales ayant un score nul.

4. En effet on peut se convaincre facilement (on n'en donnera pas de démonstration ici) qu'en gardant moins de $M_E$ transitions sortantes, pour chaque état $E$, on élimine des chemins candidats potentiels à être parmi les $N$ meilleurs chemins ; réciproquement, tous les $N$ meilleurs chemins sont inclus dans l'ensemble des chemins dessinés en gardant $M_E$ transitions sortantes par état $E$.

5. Si $[i, i+k]$ est la feuille à développer de l'état $E$, alors $\mathcal{N}_E = \begin{cases} k-1 & si\ [i, i+k]\ est\ un\ fils\ droit \\ 2(k-1) & sinon \end{cases}$ .

où

- $\mathcal{F}(E)$ est l'ensemble des feuilles de $E$
- $\mathcal{F}_d(E)$ est l'ensemble des feuilles de $E$ qui sont des fils droits
- pour $f_d \in \mathcal{F}_d(E)$, $f_d.m$ est la monotonie que doit respecter la dérivation de cette feuille : $f_d.m = 1$ si $f_d$ doit être dérivée de façon monotone (i.e. si le noeud père a été dérivé en swap) et $f_d.m = 0$ dans le cas contraire.

Notons que, pour éviter de calculer plusieurs fois les mêmes sommes, on peut tirer avantage du fait qu'au passage d'un état $E_j$ à l'un de ses suivants $E_{j+1}$, on ne remplace que la feuille à développer de $E_j$ par ses deux fils dans la dérivation qui entraîne la transition.

### 3.2.3 Algorithme

A partir des réflexions précédentes, on déduit la structure de l'algorithme de recherche de $N$ meilleures permutations ITG : une première étape consiste à construire une table CYK de façon similaire à ce qui est fait dans l'algorithme 1 (sans ajout de complexité), mais en conservant davantage d'informations, comme expliqué plus haut ; la seconde étape est l'application d'un algorithme proche de celui évoqué dans [13].

Dans la première étape, dont le pseudo-code est donné en Annexe C, chaque case possède :

- Une liste de dérivations, ordonnées par ordre décroissant de score, et de longueur
- Un score "*score*" , qui est celui de la meilleure dérivation
- Un score "*scoreMono*", qui est celui de la meilleure dérivation monotone
- Un score "*ScoreSwap*", qui est celui de la meilleure dérivation swap

Dans la deuxième étape, chaque état $E$ possède :

- Une liste de feuilles, chaque feuille ayant les mêmes propriétés que les cases, avec en plus une contrainte de monotonie : $m = 1$ si la feuille *doit* être dérivée de façon monotone, $m = 0$ si elle *doit* être dérivée en swap, $m = -1$ si la monotonie de sa dérivation est libre (il s'agit d'un fils gauche ou de la seule feuille $[1, n + 1]$ de l'état initial)
- Un score qui est la somme de $\Phi$, qui caractérise le meilleur chemin vers l'ensemble des états finaux, et de $c$, qui évalue le chemin qui sépare l'état $E$ de l'état initial.

Nous sommes donc arrivés, au terme de ce chapitre, à un algorithme qui étend la résolution du Linear Reordering Problem à une liste des N-meilleures permutations. Nous avons implémenté la totalité de cet algorithme en C++, et l'avons utilisé lors des expérimentations décrites dans la section suivante.

# Chapitre 4

# Expérimentation

## 4.1 Expériences menées

Les expérimentations ont été conduites sur le corpus d'apprentissage "*Newsco*", constitué de 115 043 paires de phrases français/anglais tirées d'articles de journaux et de leurs traductions. Le but était d'utiliser les algorithmes présentés dans les sections précédentes afin d'améliorer la qualité du ré-ordonnancement en source (ici, le français) avant apprentissage.
Si $F$ désigne l'ensemble des phrases source $f_1^J$ du corpus d'apprentissage, "ré-ordonner en source avant apprentissage" consiste à générer un ensemble $F'$ où chaque phrase est dans un nouvel ordre, idéalement plus cohérent avec l'ordre des mots de la phrase cible correspondante. Le système **n**-code construit déjà un tel ensemble, par l'étape d'"*unfolding*", et nous espérons améliorer ce ré-ordonnancement en source.

### 4.1.1 Analyse des données.

Nous avons d'abord testé, pour chaque paire de phrase du corpus, l'appartenance de la permutation induite par l'*unfolding* à l'ensemble des permutations ITG. Le tri entre phrases "ITG" et "non ITG" avait pour but de déterminer s'il existe un lien entre "bons" ré-ordonnancements et ré-ordonnancements ITG. En effet, avant de "corriger" des ré-ordonnancements en les forçant à être ITG, il fallait vérifier que la non appartenance à l'ensemble ITG n'était pas un cas fréquent parmi les ré-ordonnancements corrects.
La longueur des phrases a également été analysée au sein des ré-ordonnancements ITG et non ITG.

### 4.1.2 Correction du ré-ordonnancement.

Afin d'améliorer le ré-ordonnancement en source avant apprentissage, notre stratégie a été de déduire du ré-ordonnancement produit par **n**-code, des contraintes d'ordre (telles que celles décrites au chapitre 3 ), et de calculer les $N$-meilleures permutations ITG au sens de ces contraintes.
Pour plus de clarté, prenons l'exemple de la paire de phrases suivante, tirée de *Newsco* :

Français : "Il a tenté de faire passer cette action - ingénieusement , quoique sans vergogne - pour une mesure visant à amener la stabilité et à renforcer la guerre contre la terreur. "

Anglais : "Artfully , though shamelessly , he has tried to sell this action as an effort to bring about stability and help fight the war on terror more efficiently. "

Le procédé d' *unfolding* donne la segmentation de la figure 4.1, et la permutation suivante des mots source : $\pi_{unfold} = 4-9-10-11-12-13-14-15-1-2-3-5-6-7-8-id(16...32)$ où $id(i...i+k)$ indique qu'aucun mot source d'indice compris entre $i$ et $i+k$ n'a été déplacé. Cette permutation n'est

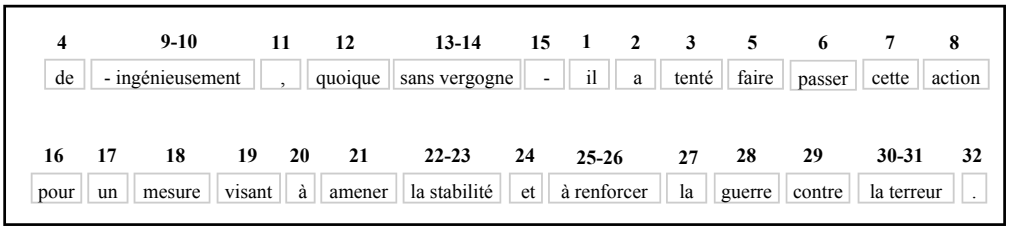


FIGURE 4.1: Segmentation et permutation d'une phrase source de *Newsco*.

pas ITG, le 4 en tête de permutation ne pouvant être associé à aucun bloc adjacent. A partir de cette permutation, plusieurs façons d'extraire des contraintes d'ordre (autrement dit : de remplir la matrice $Q$ définie au chapitre 3 ) ont été testées.

### Contraintes uniformes

La première a été d'affecter à toute paire d'indices $(i, j)$ avec $1 \leq i < j \leq n$ (ici $n = 32$), une probabilité

$$P_{ij} = \begin{cases} 1 - \epsilon & si\ i \prec j \quad dans\ \pi_{unfold} \\ \\ \epsilon & sinon \end{cases}$$

($\epsilon < 0.5$) et $P_{ji} = 1 - P_{ij}$. On définit alors la matrice $Q$, triangulaire supérieure, comme suit [1] :

$$Q_{ij} = \begin{cases} \log(\frac{P_{ij}}{P_{ji}}) & si \quad i < j \\ \\ 0 & sinon \end{cases}$$

Dans ce premier calcul de contraintes, la valeur exacte de $\epsilon \in [0, 1/2]$ importait peu (nous avons choisi $\epsilon = 0.1$). Ce qui compte dans cette expérience est le fait que toutes les contraintes sont d'égale importance, ce qui ne traduit pas la réalité en général.

### Relâchement de contraintes

Sur l'exemple de la figure 4.1, il serait judicieux de ne pas tenir compte des contraintes d'ordre que l'*unfolding* impose sur le mot "de" d'indicie 4. En effet, le fait qu'il se trouve en première position est dû à un travers souvent observé dans les ré-ordonnancements en source de ce corpus par **n**-code : ici le mots français "de" n'a été aligné avec aucun mot anglais (ce qui arrive souvent aux articles partitifs français dans la traduction vers l'anglais), et affecté à aucun tuple. Ne sachant pas à quel tuple rattacher l'article, *unfold* a choisie de le mettre en début de phrase.
Afin de ne pas tenir compte de ce genre de choix arbitraires, nous avons annulé toute contrainte sur les paires de mots impliquant le mot 4 "de". Ceci correspond à affecter des probabilités

$$\forall i < 4\ , \forall\ j > 4 \quad P_{i4} = P_{4j} = 0.5$$

ce qui donne :

$$\forall i\ ,\ \forall j \quad Q_{i4} = Q_{4j} = 0$$

De façon plus générale, nous avons fait cela avec tout mot français qui n'était aligné avec aucun mot anglais.

1. Cette définition, qui a effectivement été implémentée, diffère de celle de l'équation 3.7, qui en est l'opposée transposée ; mais comme $Q_{ij} - Q_{ji}$ ne change pas entre les deux définitions, celles-ci sont équivalentes pour la recherche des $N$-meilleures permutations au sens des contraintes d'ordre (voir A en remarquant que $Q_{ij} - Q_{ji} = \log(R_{ij})$).

#### Contraintes monotones

Toujours d'après l'observation d'exemples tels que le précédent, nous avons remarqué que les mots non alignés étaient souvent des déterminants (articles, articles partitifs...), et qu'au moment du ré-ordonnancement un tel mot gagnerait à rester juste après le mot qui le précède ou juste avant celui qui le suit, dans l'ordre original.
Nous avons donc modifié les contraintes sur les mots concernés, de façon à favoriser une position "monotone" pour ces derniers, c'est à dire après les mots d'indices plus petits dans la phrase originale, et avant ceux d'indices plus grands.
Dans notre exemple, cela revient à modifier $Q$ de la façon suivante :

$$\forall\, i < 4, \quad \forall\, j > 4, \quad Q_{i4} = Q_{4j} = \log(\tfrac{1-\epsilon}{\epsilon})$$

## 4.2 Résultats

### 4.2.1 Analyse des données

#### Statistiques

Sur les 115 043 phrases que compte le corpus *Newsco*, 22 360 sont ré-ordonnancées par une permutation non ITG des mots source, soit environ 19,44% des cas. Cette proportion de phrases "non ITG", voisine des 20%, est pratiquement constante sur tout le corpus, ce qui montre que ces phrases sont uniformément réparties.
En ce qui concerne la longueur des phrases, les statistiques sont assez similaires au sein des deux ensembles (ITG et non ITG) avec toute fois, comme attendu, un mode atteint pour une longueur supérieure dans le cas des phrases non ITG (voir figure 4.2). En effet le cardinal de $ITG_n$ implique que la probabilité d'avoir une phrase ITG décroit avec $n$, c'est pourquoi on pouvait s'attendre à trouver des phrases plus longues dans l'ensemble non ITG.

#### Patrons récurrents

**Parmi les phrases** dites **"non ITG"**, nous observons plusieurs patrons récurrents qui ne correspondent pas, en général, à des ré-ordonnancements corrects.
Notons que la qualité des ré-ordonnancements en apprentissage a été estimée "à l'oeil", sur un nombre assez limité de phrase, mais probablement représentatif vue la récurrence des mêmes problèmes.

**Articles mal positionnés :** la phrase de la figure 4.1 en est un exemple. Il est fréquent que des articles tels que "de", "le/la/l' ", soient regroupés en début de phrase ou avant un tuple, lorsqu'ils ne sont alignés avec aucun mot de la phrase cible.
Voici quelques autres exemples similaires :

**(2)** (Phrase 30)
Français : Des avocats, des défenseurs des droites de l'homme et des dirigeants politiques ont depuis lors été arrêtés.
Anglais : Lawyers, human rights activists, and political leaders have since been arrested.
Français "unfold" : Des avocats , **de des des** l'homme droits défenseurs et des politiques dirigeants ont depuis lors été arrêtés.
**(3)** (Phrase 52)
Français : Le Lieutenant Général Hamid Gul, ancien chef de la Direction des renseignements inter-services ( Inter-Services Intelligence ) ,a confirmé avoir soutenu une alliance de partis politiques de droite pour l'empêcher d'obtenir la majorité parlementaire.
Anglais : Lt. Gen. Hamid Gul , the former Inter-Services Intelligence (ISI) chief, confirmed that he sponsored an alliance of right-wing political parties to stop her from getting a parliamentary majority .
Français "unfold" : Le Lieutenant Général Hamid Gul, **de la** ancien Direction des iner-services ( Inter-Services Intelligence ) chef , a confirmé avoir soutenu une alliance de de droite politiques partis pour l'empêcher d'obtenir la parlementaire majorité .

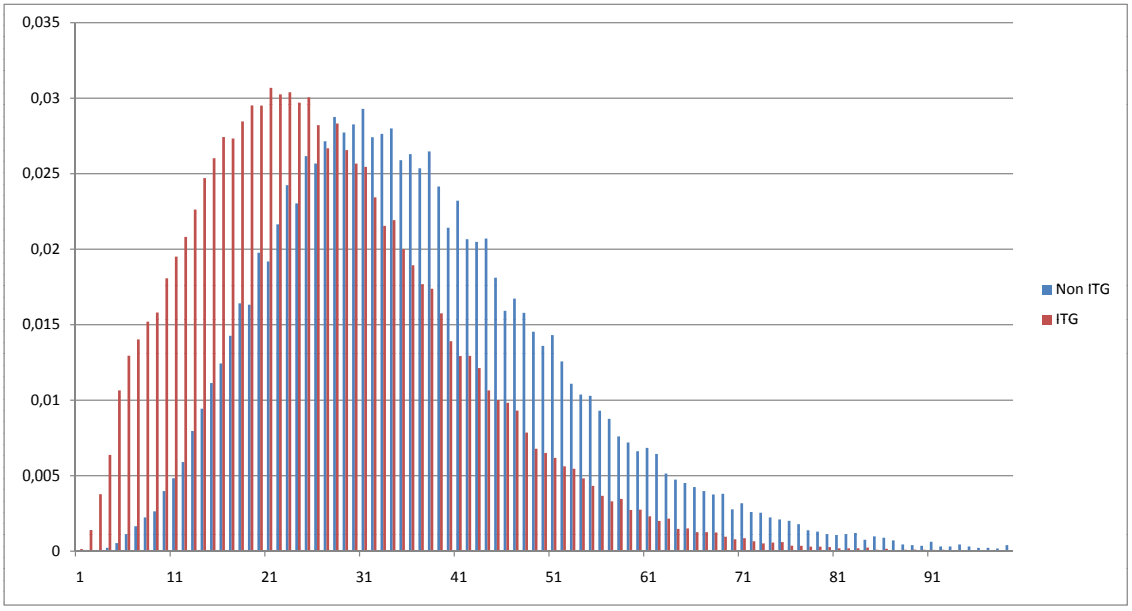


FIGURE 4.2: Histogrammes des longueurs des phrases source de *Newsco*, selon la nature de leur ré-ordonnancement par *unfold* ; en abscisses : la longueur des phrases (en nombre de mots) ; en ordonnées : la proportion de phrases au sein du sous ensemble de phrases (ITG en rouge, non ITG en bleu).

**(4)** (Phrase 68, première partie)
Français : L'avenir du Pakistan étant en jeu , l'aide et le soutien de l'Occident joueront un rôle crucial .
Anglais : With Pakistan's future in the balance , the West's help and support will be crucial.
Français "unfold" : l' du Pakistan avenir étant en jeu , **l' le de l'** Occident aide et soutien **un** joueront rôle crucial .

**Reformulation totale** Un autre cas récurrent parmi les permutations *unfold* non ITG, est un mauvais ré-ordonnancement causé par une reformulation, dans la traduction, trop éloignée de la structure de la phrase source. Dans ces cas là, le ré-ordonnancement n'a pas vraiment de sens. Voici quelques exemples :

**(5)** (Phrase 10, deuxième partie)
Français : [...] alors qu'il décidait de soutenir les Etats-Unis dans la guerre contre la terreur , plutôt que les Talibans .
Anglais : [...] as he agreed to ditch the Taliban and support the United States-led war on terror .
Français "unfold" : [...] alors qu'il décidait de dans, plutôt les Talibans que soutenir les Etats-Unis la guerre contre la terreur .

**(6)** (Phrase 31)
Français : Ces mesures font l'objet d'un ressentiment généralisé parmi les populations .
Anglais : There is widespread public resentment in response to these moves .
Français "unfold" : font l'objet d'généralisé population un ressentiment parmi la ces mesures .

**Parmi les phrases "ITG"**, au contraire, très peu de ces cas pathologiques ont été observés. Toutes les phrases de l'ensemble ITG que nous avons examinées présentent un ré-ordonnancement satisfaisant par *unfold*, au sens où l'ordre des mots source correspond bien à celui des mots cible, et qu'une traduction monotone de ces phrases source ré-ordonnancées donnerait des phrases anglaises naturelles d'un point de vue structurel.

Cette simple analyse met en avant deux phénomènes remarquables [2] :

1. Le critère ITG semble être assez pertinent pour discriminer les "bons" ré-ordonnancements des mauvais, puisque les échecs observés d'*unfold* se concentrent dans les permutations non ITG .

2. Les permutations ITG représentent près de 80% des ré-ordonnancements produits par *unfold* sur ce corpus. Si toutes les permutations de mots étaient équiprobables, avec une longueur de phrase moyenne située entre 20 et 30 mots, les permutations ITG représenteraient bien moins de $0,1$ % des cas (à partir de $n = 16$ mots, le rapport $\frac{|\mathbf{ITG_n}|}{|\mathfrak{S}_n|}$ passe en dessous de $0,001$).
Si de plus, tenant compte de la remarque précédente, on considère que les permutations ITG coïncident avec de bons ré-ordonnancements, et les non ITG avec des cas d'échec, on mesure à quel point ce sous-ensemble du groupe symétrique est représentatif des permutations de mots en traduction français/anglais.

### 4.2.2 Correction du ré-ordonnancement

Les résultats suivants doivent nous indiquer s'il est possible d'arriver à des ré-ordonnancements satisfaisants à partir de ré-ordonnancements non ITG, en appliquant les méthodes de corrections expliquées plus haut.

#### Contraintes uniformes

En général, les patrons pathologiques observés dans les ré-ordonnancements non ITG, ne sont pas corrigés par cette méthode. Toutes les contraintes d'ordre étant de même importance, l'algorithme se contente de déplacer le moins de mots possibles vers les premières positions qui permettent à la permutation de devenir ITG. Ainsi, pour la phrase de la figure 4.1, le mot "de", d'indice 4, est déplacé de façon à être adjacent au bloc "$1-2-3-$" pour donner la nouvelle permutation $\pi_{ITG} = 9-10-11-12-13-14-15-4-1-2-3-5-6-7-8-id(16...32)$, soit la phrase :

*-ingénieusement, quoique sans vergogne -* ***de*** *il a tenté faire passer cette action pour une mesure visant à amener la stabilité et à renforcer la guerre contre la terreur .*

alors que l'on souhaiterait que 4 fût en 11e position, entre "tenté", d'indice 3 et "faire", d'indice 5, toutes choses égales par ailleurs.
Or une telle permutation ne figure pas parmi les 1000 meilleures ITG au sens des contraintes uniformes ! Et pour cause : avec de telles contraintes, on constate énormément de cas d'ex-aequo (les permutations classées 2 à 29 ont toutes le même score, puis les permutations 30 à 430 sont à égalité, de même pour les permutations 431 à 1000...).

#### Relâchement de contraintes

Cette modification n'a pas eu les effets escomptés : le principal problème observé est qu'avec cette configuration, la meilleure permutation est à égalité avec ses suivantes (jusqu'au rang 100 et au delà). On ne voit donc pas, en général, remonter le ré-ordonnancement espéré en haut du classement. En ce qui concerne, par exemple, la phrase de la section précédente, l'article partitif "de" est correctement positionné dans la permutation la mieux classée, mais d'autres mouvements de mots, indésirables, ont lieu.

2. Ceci, dans le cadre précis de notre expérience, c'est à dire sur un petit corpus anglais/français.

**Contraintes monotones**

Ce calcul de contraintes améliore le ré-ordonnancement pour la phrase de la figure 4.1 : la meilleure ITG est bien celle espérée. La phrase 52 (exemple 3) connait également une amélioration, puisque le meilleur ordre ITG donne "*ancien de la direction des inter-services (Inter-Services Intelligence) chef*", même si l'on n'obtient pas "*une alliance de droite de politiques partis*".
Ce n'est toutefois pas le cas pour les phrases 30 (exemple 2) et 68 (exemple 4 ), et d'autres, qui ne voient pas leurs erreurs corrigées. De plus, on constate toujours de nombreux cas d'égalité.

Il est donc clair que de nouvelles stratégies sont à concevoir en terme de formulation de contraintes, de façon à retrouver, à partir de ré-ordonnancements erronés (hors cas de reformulation totale), des permutations ITG aussi satisfaisantes que celles que produit *unfold* dans 80% des cas.
Déduire des contraintes d'ordre directement à partir des alignements (i.e. sans tenir compte de la segmentation et du premier ré-ordonnancement dû à *unfold*) semble raisonnable. Toutefois, un examen rapide de ces alignements, montre que dans les cas problématiques les alignements mêmes sont souvent approximatifs et donc peu fiables pour établir des contraintes d'ordre.
Une façon de surmonter cette difficulté serait de ne plus calculer de telles contraintes *localement*, c'est à dire pour chaque parie de phrases, mais de façon globale sur tout le corpus, comme cela est fait dans [7] et [16]. De cette manière, on peut espérer "lisser" les irrégularités observées dans les cas non ITG (l'apprentissage des scores de contraintes pourrait d'ailleurs être fait uniquement sur les phrases ré-ordonnées de façon ITG par *unfold*, puisqu'elles semblent fiables).

# Chapitre 5

# Conclusions et perspectives

Au terme de ce stage, je retiens d'abord l'enrichissement que celui-ci m'a apporté. Mes lectures en traduction statistique en général, sur le problème de ré-ordonnancement en particulier, mais aussi en mathématiques (statistiques, algèbre, combinatoire), qui ont occupé une partie importante de mon temps, m'ont permis d'assimiler de nouvelles notions en traduction, en statistiques, en théorie des langages, en algorithmique.
Cette étude bibliographique et les conseils de mes tuteurs, ont particulièrement orienté mon attention vers les permutations ITG. La compréhension de leur structure, de leur caractère représentatif des ré-ordonnancements en traduction, de la possibilité qu'elles offrent de résoudre des problèmes d'optimisation (tel que le "Linear Ordeing Problem") NP-difficiles sur le groupe symétrique tout entier, a été à l'origine de la partie pratique de mon stage.
Celle-ci a d'abord consisté en la conception et l'implémentation en C++ de deux algorithmes d'optimisation : l'un résolvant le Linear Ordering Problem au sein des permutations ITG, l'autre son extension "N-best". Ceci a aussi été l'occasion de me perfectionner en programmation.
Ensuite, nous avons procédé à l'expérimentation de ces méthodes sur un corpus français-anglais de petite taille (la complexité des algorithmes implémentés rend difficile un passage à des échelles plus importantes pour le moment). Nous nous sommes appuyés sur les ré-ordonnancements, en source et en apprentissage, produits par le système de traduction du LIMSI, **n**-code, pour les améliorer en les rendant ITG, et ce phrase par phrase. Nous avons constaté l'intérêt du critère ITG pour détecter de mauvais ré-ordonnancements, et l'aptitude des permutations ITG à représenter une majorité des ré-ordonnancements en source sur notre corpus. Les résultats de ces expériences, mais aussi mes lectures préalables qui avaient ouvert des pistes restées inexplorées, permettent maintenant d'entrevoir les perspectives de poursuite de ce travail de stage.

Parmi les idées de futures études, deux me semblent peu urgentes mais dignes d'être gardées à l'esprit : la première serait d'appliquer le même genre de méthodes que celles déjà implémentées, mais avec d'autres sous ensembles du groupe symétrique (par exemple OSS) ; la deuxième, de s'intéresser aux méthodes à noyaux appliquées aux permutations : elles pourraient fournir des outils plus fins de comparaison et de recherche de permutations.
De façon plus immédiate, il semble utile de réfléchir à présent à une méthode *globale* d'extraction de contraintes sur les ordres de mots, à partir des alignements de mots de tous le corpus, en s'inspirant par exemple de méthodes évoquées dans certains articles. Par ailleurs, ce qui a manqué au stage pour approcher (sinon atteindre) tous les objectifs fixés, est une vraie réflexion sur le modèle d'apprentissage des patrons de ré-ordonnancements ITG. Un modèle "max-entropie" a été envisagé, mais les questions des features à utiliser et de l'implémentation n'ont pas vraiment été traitées. Ce sera un point incontournable si l'on veut exploiter les travaux débutés au cours de ce stage.

# Annexe A

# Sur les contraintes d'ordre.

Dans l'exemple de la section (3.1), le score final de la permutation trouvée est le produit des $P_{ij}$ tels que $i \prec j$ dans cette permutation. Autrement dit

$$score(\text{"2143"}) = P_{21} \times P_{14} \times P_{13} \times P_{23} \times P_{24} \times P_{43} \tag{A.1}$$

et de façon générale le score d'une permutation $A$ d'ordre $n$ s'écrit

$$score(A) = \prod_{1 \leq i < j \leq n} a_{ij} \tag{A.2}$$

avec

$$a_{ij} = \begin{cases} P_{ij} & si \quad i \prec j \quad dans \ A \\ P_{ji} & sinon \end{cases}$$

Pour deux permutations $ITG_n$ $A$ et $B$, $A$ est meilleure que $B$, au sens des contraintes traduites par les poids $P_{ij}$ (que l'on considère tous strictement positifs), si et seulement si :

$$\begin{aligned} & \frac{score(A)}{score(B)} > 1 \\ \Leftrightarrow \quad & \prod_{1 \leq i < j \leq n} \frac{a_{ij}}{b_{ij}} > 1 \\ \Leftrightarrow \quad & \prod_{1 \leq i < j \leq n, a_{ij} \neq b_{ij}} \frac{a_{ij}}{b_{ij}} > 1 \end{aligned} \tag{A.3}$$

Or, en notant $R_{ij}$ le rapport $\frac{P_{ij}}{P_{ji}}$, on remarque que pour $a_{ij} \neq b_{ij}$, on a $\frac{a_{ij}}{b_{ij}} = R_{ij}$ ou $\frac{a_{ij}}{b_{ij}} = \frac{1}{R_{ij}}$. Donc toute transformation des poids $P_{ij}$ qui laissent inchangés les rapports $R_{ij}$ ne modifie pas le rapport de score $\frac{score(A)}{score(B)}$, et définit un problème équivalent au problème initial.

# Annexe B

# Algorithme 1

**Algorithm 1** Calcul de la meilleure permutation ITG pour des contraintes $Q$

**Input:** matrice de contraintes $Q$, de taille $n \times n$, où $n$ est l'ordre de la permutation
**Output:** table CYK remplie permettant de reconstituer la meilleure permutation

{Initialisation $(k = 1)$}
**for** $i = 1 \rightarrow n$ **do**
  $case(i, i+1).score \leftarrow 0$
  $\Delta(i, 1, 0) \leftarrow 0$
  $\Delta(i, 1, 1) \leftarrow 0$
**end for**
{Remplissage des autres cases, diagonale $(k)$ par diagonale}
**for** $k = 2 \rightarrow n$ **do**
  **for** $i = 1 \rightarrow n - k + 1$ **do**
    {Initialisation $(k \geq 2)$}
    $case(i, i+k).score \leftarrow -\infty$
    $\Delta(i, k, 0) \leftarrow 0$
    $\Delta(i, k, k) \leftarrow 0$
    **for** $p = 1 \rightarrow k - 1$ **do**
      $\Delta(i, k, p) \leftarrow \Delta(i, k-1, p) + \Delta(i+1, k-1, p-1) - \Delta(i+1, k-2, p-1) + Q_{i+k-1,i} - Q_{i,i+k-1}$
      **if** $p < k - 1$ **then**
        $ScoreCourant \leftarrow case(i, i+p).score + case(i+p, i+k).score + case(i+p, i+k).m \times \Delta(i, k, p)$
      **else**
        $ScoreCourant \leftarrow case(i, i+p).score + case(i+p, i+k).score + \max(0, \Delta(i, k, p))$
      **end if**
      **if** $ScoreCourant > case(i, i+k).score$ **then**
        $case(i, i+k).score \leftarrow ScoreCourant$
        $case(i, i+k).indice \leftarrow p$
        **if** $p < k - 1$ **then**
          {la case courante prend la monotonie opposée de celle de son fils droit}
          $case(i, i+k).m \leftarrow 1 - case(i+p, i+k).m$
        **else**
          $case(i, i+k).m \leftarrow \mathbb{1}(\Delta(i, k, p) \leq 0)$
        **end if**
      **end if**
    **end for**
  **end for**
**end for**

# Annexe C

# Algorithme 2

**Algorithm 2** Calcul *des* meilleures dérivations de forme normale à droite, pour chaque case CYK

**Input:** matrice de contraintes $Q$, de taille $n \times n$, nombre $N$ de meilleures permutations souhaitées
**Output:** table CYK remplie permettant de reconstituer les $N$ meilleures permutations

**for** $i = 1 \rightarrow n$ **do**
  $case(i, i+1).score \leftarrow 0$
  $case(i, i+1).scoreMono \leftarrow 0$
  $case(i, i+1).scoreSwap \leftarrow 0$
  $\Delta(i, 1, 0) \leftarrow 0$
  $\Delta(i, 1, 1) \leftarrow 0$
**end for**
**for** $k = 2 \rightarrow n$ **do**
  **for** $i = 1 \rightarrow n - k + 1$ **do**
    {Initialisation ($k \geq 2$)}
    $case(i, i+k).score \leftarrow -\infty$
    $case(i, i+k).scoreMin \leftarrow -\infty$
    $case(i, i+k).scoreMono \leftarrow -\infty$
    $case(i, i+k).scoreSwap \leftarrow -\infty$
    $\Delta(i, k, 0) \leftarrow 0$
    $\Delta(i, k, k) \leftarrow 0$
    **for** $p = 1 \rightarrow k - 1$ **do**
      $\Delta(i, k, p) \leftarrow \Delta(i, k-1, p) + \Delta(i+1, k-1, p-1) - \Delta(i+1, k-2, p-1) + Q_{i+k-1,i} - Q_{i,i+k-1}$
      $score1 \leftarrow case(i, i+p).score + case(i+p, i+k).scoreSwap$
      $score0 \leftarrow case(i, i+p).score + case(i+p, i+k).scoreMono + \Delta(i, k, p)$
      **if** $\#(case(i, i+k).derivations) < N \ \vee \ score1 > case(i, i+k).scoreMin$ **then**
        Update($case(i, i+k).derivations$ , $case(i, i+p)$ , $case(i+p, i+k)$ , $score1$)
      **end if**
      **if** $\#(case(i, i+k).derivations) < N \ \vee \ score0 > case(i, i+k).scoreMin$ **then**
        Update($case(i, i+k).derivations$ , $case(i+p, i+k)$ , $case(i, i+p)$ , $score0$)
      **end if**
    **end for**
  **end for**
**end for**

La fonction "Update($case(i, i+k).derivations$ , $case_g$ , $case_d$, , $scoreDerivation$ )" ajoute la dérivation $case(i, i+k) \rightarrow case_g$ , $case_d$ de score $scoreDerivation$ à la liste $case(i, i+k).derivations$, de

façon à ce que cette liste reste triée par ordre décroissant de score, et les valeurs $case(i, i+k).score$, $case(i, i+1).scoreMono$, $case(i, i+1).scoreSwap$, $case(i, i+1).scoreMin$ sont mises à jour avec, respectivement, le plus grand score de la liste, le plus grand score parmi les dérivations monotones, le plus grand score parmi les dérivations swap, le plus petit score de la liste.

# Annexe D

# Algorithme 3

**Algorithm 3** Caclcul des $N$ meilleures permutations ITG

**Input:** Table CYK de taille $n$ remplie par l'algorithme 2 avec le paramètre $N$

**Output:** la liste des $N$ meilleures permutations ITG au sens des contraintes données par $Q$ à l'algorithme 2

$case0 \leftarrow case(1, n+1)$
$case0.m \leftarrow -1$
$Etat0.feuilles \leftarrow case0$
$Etat0.\Phi \leftarrow case(1, n+1).score$
$Etat0.c \leftarrow 0$
$EtatsFinaux \leftarrow \emptyset$
$S \leftarrow Etat0$
**while** $S \neq \emptyset \wedge \#(EtatsFinaux) < N$ **do**
  $EtatCourant \leftarrow head(S)$ ; DEQUEUE($S$)
  **if** isFinal($EtatCourant$) **then**
    $EtatsFinaux \leftarrow EtatsFinaux \cup EtatCourant$
  **else**
    **for all** $E \in$ EtatsSuivants($EtatCourant$) **do**
      $E.\Phi \leftarrow \Phi(E)$ (Eq. 3.14 )
      $E.c \leftarrow EtatCourant.c +$ GainTransition($EtatCourant, E$)
      $E.score \leftarrow E.\Phi + E.c$
      ENQUEUE($S$ , $E$)
    **end for**
  **end if**
**end while**

Dans cet algorithme, $S$ est une file de priorité, dont les éléments (qui sont des états) sont classés, au moment de leur ajout par la fonction "ENQUEUE", par ordre décroissant de score. Autrement dit, l'élément de tête de $S$, que l'on récupère par "$head(S)$", est l'état de plus grand score dans la file.
L'opération "$\cup$" correspond ici à une concaténation à droite, si bien qu'à la fin de l'algorithme, les états finaux sont classés en ordre décroissant de score.
Pour un état dont la feuille à développer est $[i, i+k]$, le gain de transition est nul pour un label monotone, i.e. une dérivation $[i, i+k] \rightarrow [i, i+p][i+p, i+k]$ de la feuille à développer, et vaut $\Delta(i, k, p)$ (calculé dans l'algorithme 2) pour une transition de label swap, i.e. pour une dérivation $[i, i+k] \rightarrow [i+p, i+k][i, i+p]$.

# Bibliographie


[1] Miklos Bona. *A Walk Through Combinatorics : An Introduction to Enumeration and Graph Theory (Second Edition)*. World Scientific Publishing Company, 2 edition, October 2006.

[2] Peter F. Brown, John Cocke, Stephen Della Pietra, Vincent J. Della Pietra, Frederick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2) :79–85, 1990.

[3] Francisco Casacuberta and Enrique Vidal. Machine Translation with Inferred Stochastic Finite-State Transducers. *Comput. Linguist.*, 30(2) :205–225, 2004.

[4] David Chiang. Hierarchical phrase-based translation. *Comput. Linguist.*, 33(2) :201–228, 2007.

[5] Josep Maria Crego and José B. Mari no. Improving statistical mt by coupling reordering and decoding. *Machine Translation*, 20(3) :199–215, 2006.

[6] Michel Galley and Christopher D. Manning. A simple and effective hierarchical phrase reordering model. In *In Proceedings of EMNLP 2008*, 2008.

[7] Stephan Kanthak, David Vilar, Evgeny Matusov, Richard Zens, and Hermann Ney. Novel reordering approaches in phrase-based statistical machine translation. In *Proceedings of the ACL Workshop on Building and Using Parallel Texts*, ParaText '05, pages 167–174, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[8] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[9] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. Moses : Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[10] Risi Imre Kondor and Marconi S. Barbosa. Ranking with kernels in fourier space. In *COLT'10*, pages 451–463, 2010.

[11] K. R Leino. Computing permutation encodings. Technical report, Pasadena, CA, USA, 1994.

[12] José B. Marinò, Rafael E. Banchs, Josep M. Crego, Adrià de Gispert, Patrik Lambert, José A. R. Fonollosa, and Marta R. Costa-jussà. N-gram-based machine translation. *Comput. Linguist.*, 32(4) :527–549, 2006.

[13] Mehryar Mohri and Michael Riley. An efficient algorithm for the n-best-strings problem. In *INTERSPEECH*, 2002.

[14] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Comput. Linguist.*, 29(1) :19–51, 2003.

[15] Louis W. Shapiro and A. B. Stephens. Bootstrap percolation, the schröder numbers, and the -kings problem. *SIAM J. Discrete Math.*, 4(2) :275–280, 1991.

[16] Roy Tromble and Jason Eisner. Learning linear ordering problems for better translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing : Volume 2 - Volume 2*, EMNLP '09, pages 1007–1016, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[17] Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Comput. Linguist.*, 23 :377–403, September 1997.

[18] Richard Zens and Hermann Ney. A comparative study on reordering constraints in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 144–151, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.