# Au sujet des algorithmes de recherche des systèmes de reconnaissance de la parole à grands vocabulaires

Roxane Lacouture

School of Computer Science
Université McGill, Montréal

Thèse de doctorat présentée à la faculté des Etudes Supérieures et de la Recherche
en vue de l'obtention du grade de
Philosophiae Doctor



18 septembre1995

*à Caroline et Renaud*
*avec tout mon amour*

*Don't Stop me Now*
*Queen*

# Table des matières

# Liste des figures

# Liste des tableaux

# Sommaire

Dans le contexte de l'approche probabiliste actuellement utilisée en reconnaissance automatique de la parole (RAP), l'étape de recherche, si effectuée de façon exhaustive, devient rapidement irréalisable tant d'un point de vue espace mémoire que temps machine, lorsque le vocabulaire traité dépasse plusieurs milliers de mots. Le recourt à divers heuristiques et autres méthodes de réduction de l'espace mémoire et du temps machine est donc inévitable.

Cette thèse décrit plusieurs de ces méthodes conçues tant pour des systèmes en mots isolés, multi-locuteurs à très grand vocabulaire (>20000 mots) que pour des systèmes en parole continue, multi-locuteurs à grand vocabulaire (entre 3000 et 20000 mots). Ainsi, dans le cadre de systèmes en mots isolés, elle décrit un algorithme efficace de minimisation de l'espace de recherche. Dans le cadre de systèmes en parole continue, elle présente trois nouveaux type de faisceaux d'émondage. Elle décrit aussi une implantation d'un système multi-passes qui fut utilisé, tant de façon traditionnelle au cours d'une évaluation de la tâche CSR de ARPA, que pour le développement d'une nouvelle méthode de rejet et d'une nouvelle méthode d'intégration de techniques d'adaptation.

# Abstract

Given that a probabilistic approach is now broadly used in automatic speech recognition (ASR), it becomes very rapidly impossible from a memory and execution time point of view to perform an exhaustive search, especially for lexica of more than a thousand words. The use of various heuristics or other approaches to reduce the amount of space and/or time needed is thus inevitable.

This thesis describes many such approaches, both for very large lexicon (>20000 words) speaker-independent isolated word systems and for large lexicon (between 3000 and 20000 words) speaker-independent continuous speech systems. For isolated words systems, it describes an efficient data structure minimization algorithm. For continuous speech systems, it presents three new types of beams. A multi-pass algorithm was also incorporated in a system used for an official ARPA CSR evaluation. Finally, this system was used as a test bed for the development of a new rejection algorithm and a new speech adaptation integration method.

## Remerciements

J'aimerais d'abord remercier mes directeur Renato De Mori et co-directeur Yves Normandin pour leur patience et leur précieux conseils techniques.

J'aimerais aussi souligner l'apport financier de Bell Northern Research, du Centre de Recherche Informatique de Montréal (CRIM) et du Conseil de Recherche en Sciences Naturelles et en Génie sans qui ce travail n'aurait pu être possible.

Un gros merci à toute l'équipe de recherche du groupe parole du CRIM au sein duquel la grande majorité de ce travail fut effectuée. Leur professionnalisme, compétence et... camaraderie ont grandement facilité mon travail.

Je ne peut passer sous silence, le support moral de Caroline, Ariane et Roland durant mes moments de doute qui furent relativement nombreux, surtout dans les derniers milles.

Un merci particulier à Roland, pour avoir accepté de lire et commenter cette thèse, ses conseils et remarques ne furent point perdus.

Finalement, le dernier mais non le moindre, merci à Stéphane pour sa confiance inébranlable durant toutes ces années.

# 1 Introduction

Sujet devenu très en vogue vers le début des années 70 avec le projet ARPA (Advanced Research Projects Agency) [NEWE 73][KLAT 90], la reconnaissance automatique de la parole (RAP) n'a cessée depuis, d'être au coeur de nombreuses recherches. Si les conclusions de ce premier projet ARPA soulignaient l'importance d'obtenir de meilleurs résultats au niveau du décodage phonétique, l'utilisation dans le milieu des années soixante-dix de modèles stochastiques basés sur les modèles de Markov cachés [JELI 76] permirent ce progrès. Ceci fit en sorte que des systèmes de RAP expérimentaux mono-locuteur, en mots séparés et petit vocabulaire du début des années 80, la recherche en est aux systèmes expérimentaux pluri-locuteurs, en parole continue et grand vocabulaire.

Or l'un des problèmes majeurs de tels systèmes est l'énormité des ressources en espace mémoire et en temps machine que nécessite l'étape de "fouille" ou de "recherche"[1]. Dans un système de RAP, c'est lors de cette étape qu'est associé à un énoncé acoustique donné, la transcription lexicale la plus probable.

Le but du travail décrit dans cette thèse, fut de développer des méthodes qui permettent cette association, en minimisant la demande en temps machine et/ou en espace mémoire. Deux grands types de systèmes furent étudiés. Les systèmes en mots isolés, multi-locuteurs à très grand vocabulaire (>20000 mots) et les systèmes en parole continue, multi-locuteurs à grand vocabulaire (entre 3000 et 20000 mots). Comme on le verra, les problèmes posés par la reconnaissance en parole continue sont différents mais surtout plus complexes que ceux rencontrés dans des systèmes fonctionnant en mots isolés. Par

1. Le terme de "recherche" est ici employé dans le sens de "search" et non dans le sens de "research". En français, il n'existe malheureusement qu'un seul mot pour traduire ces deux sens. Dans ce document, les synonymes fouille ou, lorsque le sens le permet, reconnaissance sont utilisés sauf dans certaines expressions tel espace ou structure de recherche.

conséquent, des méthodes fort différentes peuvent et doivent être employées pour chacun de ces types de systèmes.

Le chapitre 2 présente les outils de base les plus couramment utilisés actuellement lors de l'étape de reconnaissance. Le chapitre 3 porte sur l'algorithme au coeur de presque tous les systèmes de RAP actuel, l'algorithme de Viterbi. Y est fait mention, entre autres, des contraintes que son emploi fait peser sur la reconnaissance. Contraintes que le reste de cette thèse, divisée en deux parties, tente d'attaquer sur plusieurs fronts.

La première partie porte sur les systèmes en mots isolés. Elle se compose des chapitres 4 et 5. Le chapitre 4 présente le problème de la reconnaissance en mots isolés. Un survol des approches qui ont déjà été suggérées dans la littérature y est fait. Le chapitre 5, présente une approche originale au problème de la représentation de l'espace de recherche.

Pour sa part, la deuxième partie porte sur les systèmes en parole continue. Elle se compose des chapitres 6 à 9. Le chapitre 6 explique en quoi le problème de la reconnaissance en parole continue est différent de celui de la reconnaissance en mots isolés. Le chapitre 7 fait état de l'avancement des connaissances dans le domaine et décrit les systèmes développés par les principales équipes de recherche en RAP. Le chapitre 8 présente le système développé au CRIM et souligne en quoi il diffère de ou s'apparente à ceux présentés au chapitre 7. Deux applications où ce système fut utilisé sont présentées. Il fait aussi état de trois résultats intéressants obtenus dans les domaines de l'adaptation, du rejet et de l'entraînement à partir de composantes de ce système. Finalement, le chapitre 9 fait état de travaux originaux dans le domaine des faisceaux d'émondage. La conclusion suit.

# 2 Bases théoriques

Le but de ce chapitre est d'introduire brièvement le lecteur aux modèles mathématiques à la base des systèmes de RAP actuels. Ainsi la section 2.1 présente l'approche probabiliste d'où, on verra, découle la nécessité d'un modèle acoustique et d'un modèle du langage (ML). Le modèle acoustique utilisé par la quasi totalité des systèmes de RAP est basé sur les modèles de Markov cachés[1]. Ces derniers sont le sujet des sections 2.2 à 2.5. D'autre part, les MLs les plus usités sont le sujet de la section 2.6.

Il est cependant important de souligner que, tant dans le cas des HMMs que dans celui des MLs, il ne s'agit que d'une brève description du concept, l'accent étant plutôt mis sur le comment de leur emploi lors de l'étape de reconnaissance des systèmes de RAP. Il ne s'agit pas d'une description mathématiquement exhaustive, ni surtout d'une discussion sur la pertinence de leur emploi en RAP. Pour ce type d'information, le lecteur peut se référer à [RABI 86], [BAKE 75] et [RABI 89] pour les HMMs et [JELI 90] pour les MLs. Au cours des travaux dont cette thèse fait état, les HMMs et MLs furent considérés comme des paramètres de base, des outils, desquels il fallait tirer meilleur partie lors de l'étape de reconnaissance.

## 2.1 L'approche probabiliste de la RAP

On peut voir la reconnaissance de la parole comme un problème de la théorie des communications [BAHL 83] dans lequel le but de la reconnaissance est de reconstituer un message $m$ à partir d'une séquence d'observations $y$, elle-même obtenue lors de la transformation de $m$ par un canal acoustique. Dans la pratique, le message $m$ subit, en fait, plusieurs transformations avant de prendre la forme $y$. D'onde acoustique, il est d'abord transformé en onde électrique par l'emploi d'un microphone puis est digitalisé. C'est sous

1. Dans le reste de ce document, on utilisera plutôt l'acronyme anglais HMMs (Hidden Markov Models) pour parler des modèles de Markov cachés.

cette forme, qu'on dénote $s$ que le signal est conservé sur support informatique. Puis $s$, est ensuite subdivisé en unités de temps (typiquement de 10ms chacune). De ces unités de temps, on extrait, au moyen de techniques usuelles de traitement du signal (transformée rapide de Fournier, coefficient de prédiction linéaire), un certain nombre de paramètres caractérisant adéquatement les diverses réalisations acoustiques de la voix humaine[1]. Pour plus de détails concernant le type de caractéristiques extraites, le pourquoi et le comment de leur extraction, le lecteur peut se référer à [SHAU87]. On dénote $y_t$ le vecteur de paramètres correspondant à l'unité de temps $t$ et $y = y_1, y_2, \ldots, y_L$ la séquence de vecteurs correspondant à un message $m$, où $L$ est la longueur de ce message en unités de temps.

En pratique, la reconstitution d'un message $m$ inconnu étant donnée une séquence $y$, consiste à retrouver, parmi tous les messages possibles, celui, qui selon toute probabilité, correspond à $y$. Selon le modèle stochastique de la parole ceci se traduit par:

$$m = \underset{m}{\operatorname{argmax}} \; P(m|y) = \underset{m}{\operatorname{argmax}} \; \frac{P(y|m)P(m)}{P(y)} \tag{EQ 2.1}$$

Or le terme du dénominateur étant constant pour tous les messages possibles, on peut l'omettre ce qui donne:

$$m = \underset{m}{\operatorname{argmax}} \; P(y|m)P(m) \tag{EQ 2.2}$$

L'étape de la reconnaissance se résume donc au calcul des deux termes $P(y|m)$ et $P(m)$. Ces termes représentent respectivement la probabilité à posteriori de $y$ étant donné $m$ et la probabilité à priori de $m$. Ils sont aussi connus sous le nom de probabilité du modèle acoustique et probabilité du modèle du langage.

## 2.2 La modélisation acoustique: les modèles de Markov cachés

L'outil le plus communément utilisé pour estimer la probabilité $P(y|m)$ est le HMM. Un HMM est un modèle de Markov caché de premier ordre. Un modèle de Markov est un ensemble de noeuds ou états (dont au moins un noeud initial et un noeud final) et de transitions ou arcs reliant ces états. La figure 2.1 illustre un HMM typique tel qu'utilisé en

---

1. Pour les résultats rapportés dans cette thèse, il s'agit de l'énergie, du delta-énergie, du delta-delta énergie, de 12 cepstraux, 12 delta-cepstraux et 12 delta-delta cepstraux, soit 39 paramètres.

RAP. On remarque que seules sont permises des transitions de type gauche-droite, et ce,

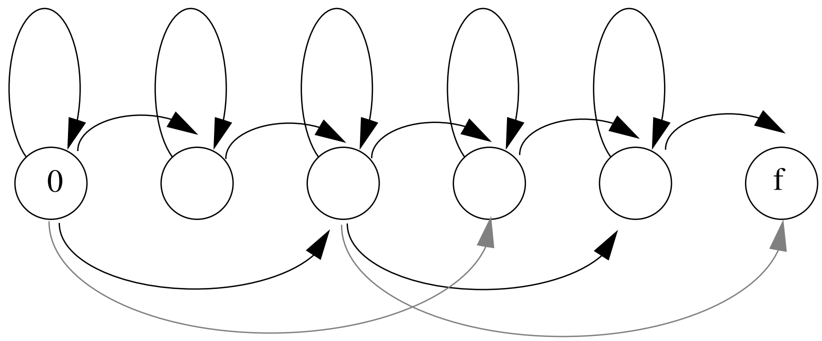


**Figure 2.1 Exemple de HMM**

dans le but de mieux modéliser la contrainte temporelle de la parole.

De plus, on a qu'à chaque transition $tr_{ij}$ reliant un état $i$ à un état $j$, est associée une probabilité de transition $q_{tr}$. La probabilité de transition est la probabilité de choisir la transition $tr_{ij}$ pour accéder à l'état $i$, étant donné un processus à l'état $j$. Dans un HMM du premier ordre, cette probabilité ne dépend que de l'état de départ $i$.

A la plupart des transitions est aussi associée une probabilité d'émission $b_{tr}$ auquelle correspond une distribution d'émission. Cette distribution représente la probabilité que soit émis lors de l'emprunt de la transition, chacun des vecteurs possibles $y_t$. La forme que prend cette distribution détermine le type du HMM. C'est ainsi qu'on parle de HMMs discrets, semi-continus, continus, à livres de codes[1] partagés etc. Les transitions auxquelles aucune distribution n'est associée sont appelées transitions vides ou nulles (en pointillés sur la figure 2.1). Aucun vecteur $y$ n'est émis lors de leur emprunt.

Quel que soit le type de HMMs, l'estimation des paramètres des distributions d'émission et des probabilités de transitions se fait à partir d'un corpus d'entraînement et non selon de quelconques considérations théoriques.

Pour effectuer l'entraînement de HMMs, on fait correspondre à chaque message possible $m$ (i.e. chaque entité devant être reconnue: mots, syllabes, phones etc.) un HMM $HMM_m$ distinct. On assume ensuite que toutes les séquences $y$, obtenues des diverses occurrences acoustiques du message $m$ dans le corpus d'entraînement, correspondent à un chemin dans $HMM_m$ reliant les états 0 et $f$. Puis par diverses méthodes d'optimisation, on fixe les dif-

1. codebooks

férents paramètres des divers HMMs de sorte que $P(y|HMM_m)$ soit élevé lorsque $m$ est obtenu de $HMM_m$. L'étude d'algorithmes d'entraînement efficaces tant au niveau des résultats qu'au niveau de l'efficacité est en lui-même un champs important de recherche. Il existe en fait une méthode principale d'entraînement: le MLE (Maximum Likelihood Estimation) [BAUM 72] qui peut être employée seule ou couplée à des méthodes complémentaires de type discriminatoire tel le MMIE (Maximum Mutual Information Estimation) [NORM 91]. Dans ce type d'entraînement, il est aussi important que $P(y|HMM_m)$ soit élevé lorsque $m$ est obtenu de $HMM_m$, qu'il soit faible autrement, d'où une meilleure “discrimination”.

Lors de l'étape de la reconnaissance, les paramètres des HMMs employés sont fixes. Il ne reste donc plus qu'à calculer $P(y|HMM_m)$ pour chaque HMM. Dans les sections qui suivent, nous verrons comment ce calcul est effectué tout d'abord au niveau de la simple transition pour chaque type de HMM (section 2.3), puis au niveau d'un HMM (section 2.4).

## 2.3 Distribution des probabilités d'émission

On a mentionné à la section précédente que le type de distribution employé pour représenter les probabilités d'émission est l'un des critères permettant de qualifier un HMM. En fait cette classification n'est pas strictement rigoureuse puisque d'un point de vue mathématique, il existe une multitude de formes de distributions possibles.

Dans les sections 2.3.1 et 2.3.2, sont respectivement décrites, les distributions discrètes et les distributions continues, qui représentent, en quelques sortes les distributions extrêmes, puis, à la section 2.3.3, un aperçu des variantes possibles.

### 2.3.1 HMMs discrets

Dans un HMM discret, les paramètres extraits du signal (voir la note en bas de page, page 7) sont répartis en un certains nombres $nbc$ de groupes. Ensuite, l'espace couvert par chaque groupe est vectorisé en un certain nombre d'entrées. Chaque vecteur $y_t$ est alors représenté par $nbc$ constantes $cte_k$ dont chacune correspond à un numéro d'entrée du livre de code $k$ correspondant; $y_t = (cte_1, \ldots, cte_{nbc})$. La probabilité d'émission associée à chaque transition pour une unité de temps donnée, se traduit donc au niveau de la reconnaissance par l'équation 2.3,

$$\log P(y_t | b_{tr_{ij}}) = \sum_{k=1}^{nbc} ldc_{tr_{ij}}[k][cte_k] \qquad \text{(EQ 2.3)}$$

où $ldc_{tr_{ij}}[k][cte_k]$ est la probabilité de l'entrée $cte_k$ du livre de codes $k$ pour la transition $tr_{ij}$[1].

La rapidité de calcul et par conséquent de reconnaissance est le principal atout de ce type de HMMs. En effet, en plus de n'être constitué que par des additions, l'on peut efficacement (d'un point de vue accès mémoire) pré-effectuer ce calcul pour toutes les distributions d'un HMMs.

Par contre, les résultats obtenus sont, en général, moins bons que pour tout autre type de HMMs, à cause d'une moins grande tolérance face à des entrées trop différentes de celles utilisées pour l'entraînement. De plus, l'espace nécessaire à la mémorisation de chaque HMM est très grand soit:

$$\left(\sum_{k=1}^{nbc} \text{nombre d'entrées du répertoire}_k\right) \times \text{nombre de transitions} . \qquad \text{(EQ 2.4)}$$

### 2.3.2 HMMs continus

Dans un HMM continu, la probabilité d'émission $b_{tr}$ est la plupart du temps représentée par une loi normale à $nbp$ paramètres $p$. On a donc que $y_t = (p_1, \ldots, p_{nbp})$ et:

$$P(y_t | b_{tr_{ij}}) = \prod_{l=1}^{nbp} \frac{1}{\sigma_{tr,l}\sqrt{2\pi}} e^{\frac{-(p_l - \mu_{tr,l})^2}{2\sigma^2_{tr,l}}} \qquad \text{(EQ 2.5)}$$

où $\mu_{tr,l}$ et $\sigma_{tr,l}$ sont respectivement la moyenne et la variance du paramètre $l$ pour la transition $tr$. La mémoire nécessaire pour représenter ce type de HMMs est donc en théorie minimale. Par contre, en général, pour obtenir de bons résultats, il faut plutôt utiliser une mixture de plusieurs normales, ce qui augmente d'autant l'espace mémoire requis pour loger les paramètres. C'est ainsi que la probabilité d'émission est plutôt donné par:

1. Les probabilités en jeu lors d'applications de reconnaissance étant très petites, c'est souvent sous forme de logarithmes qu'elles sont utilisées. A moins d'indication contraire, c'est donc cette forme qui sera assumée dans cette thèse.

$$P(y_t | b_{tr_{ij}}) = \sum_{l=1}^{nbmixt} K_l \text{norm}_l(y_t) \qquad \text{(EQ 2.6)}$$

où $nbmixt$ est le nombre de normales $norm$ tel que calculé à l'équation 2.5 composant la mixture. La sommation est non logarithmique (ce qui rend le calcul de ce type de distribution plus complexe), et les différents $K_l$, des constantes déterminées lors de l'entraînement et représentant la contribution de chaque normale (somme à un). En général, les HMMs continus donnent les meilleurs taux de reconnaissance, entre autres, parce qu'ils s'avèrent plus robustes face à des entrées différentes des données d'entraînement. Par contre, le temps de reconnaissance est évidemment beaucoup plus long. Qui plus est, lorsque le nombre de HMMs grandit, le nombre de paramètres à évaluer devient lui aussi très grands. C'est pourquoi le nombre de normales différentes est souvent limité et ces dernières partagées entre plusieurs distributions associées à des HMMs ou partie de HMMs représentant des phénomènes acoustiques similaires.

### 2.3.3 Aperçu des différentes méthodes de partage des distributions

Il existe une multitude d'approches différentes régissant le partage de normales entre distributions. Par exemple, à un extrême, un seul ensemble de normales est partagé entre toutes les distributions possibles, et seules les poids $k$ varient d'une distribution à l'autre. Dans un autre exemple, illustré à la figure 2.2, toutes les distributions des modèles d'un

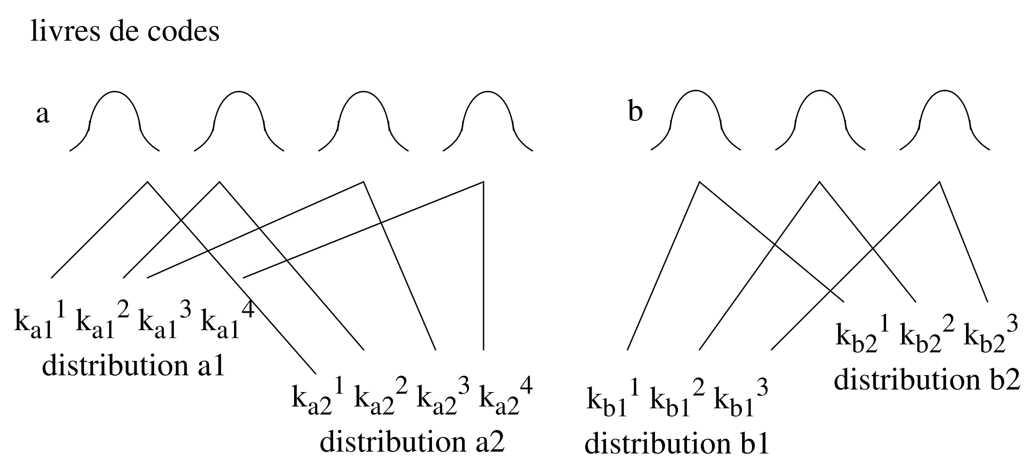


**Figure 2.2 Illustration graphique de distributions à livres de codes**

même phonème (voir section 2.5) partagent les normales d'un même groupe (on parle aussi d'un livre de codes où les codes sont les différentes normales). C'est ainsi que les distributions *a1* et *a2* partagent les normales du livre de codes *a*, et *b1* et *b2*, celles du livre de code *b*, seules les constantes ${k_{a1}}^1$-${k_{a1}}^4$ et ${k_{b1}}^1$-${k_{b1}}^4$ différenciant la distribution *a1* de la distribution *a2*. On référera à ce type de distribution par le terme de distributions à livres de codes partagés. Le fait de choisir de construire un livre de code par phonème est une alternative, d'autres divisions sont possibles.

L'avantage de ces méthodes est évidemment de diminuer le nombre de paramètres devant être évalués, et par conséquent la quantité de données nécessaire à un bon entraînement tout en conservant la flexibilité et la robustesse des distributions continues. Evidemment le choix d'un partage plutôt qu'un autre a un impact direct sur les procédures d'entraînement. Choisir une méthode de partage efficace tant au niveau efficacité des HMMs résultant, que facilité d'entraînement, économie d'espace et de temps est un sujet de recherche en soi, d'ailleurs en pleine ébullition, mais qu'on n'abordera pas davantage dans cette thèse.

## 2.4 Calcul des probabilités: l'algorithme "avant"

Dans les sections précédentes, nous avons vu comment calculer la probabilité d'émission au niveau de la distribution associée à une transition. Voyons maintenant comment ces probabilités sont utilisées de paire avec les probabilités de transitions dans l'algorithme "avant"[1] afin de calculer la probabilité acoustique $P(y|HMM_m)$ du HMM associé au message $m$.

L'algorithme "avant" est un algorithme itératif, et la meilleure façon d'expliquer son fonctionnement est de l'illustrer au moyen d'une matrice $mat[x]\ [y]$ dans laquelle le passage de la $t-1^{ieme}$ à la $t^{ieme}$ colonne, correspond au traitement du vecteur d'observation $y_t$ (pour $t$ variant de 1 à L) et le passage de la $i^{ieme}$ à la $j^{ieme}$ rangée à une transition $tr_{ij}$ du HMM. Selon cette définition, on a qu'un passage est permis entre $mat[i]\ [t-1]$ et $mat[j]\ [t]$ si et seulement si le HMM contient une transition pleine $tr_{ij}$, et un passage est permis entre $mat[i]\ [t]$ et $mat[j]\ [t]$ si et seulement si le HMM contient une transition vide $tr_{ij}$. On se rappelle en effet, qu'aucune probabilité d'émission n'étant associée à une transition vide, aucun traitement du vecteur d'observation n'y est non plus associé et que le passage correspondant dans la matrice doit se faire entre deux cases d'une même colonne. La figure 2.3 illustre la matrice de fouille (incluant l'ensemble des passages permis entre les diverses cases de celle-ci) associée au HMM de la figure 2.1. On peut noter,

1. "forward algorithm"

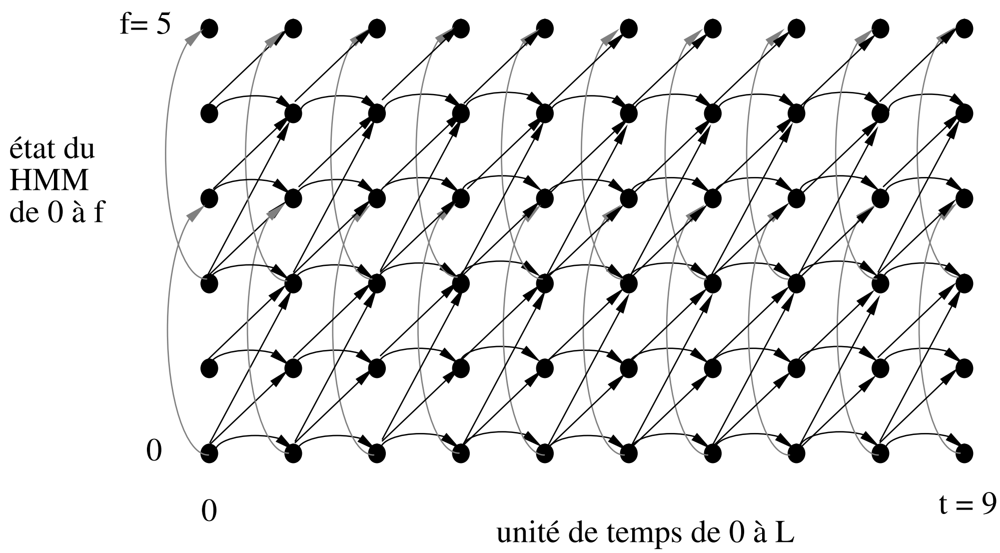


**Figure 2.3 Représentation d'une matrice de fouille**

que les divers passages entre cases définissent un ensemble de chemins possibles dans le temps entre chaque paire de noeuds du HMM.

Par la suite, l'on définit

$$\alpha_t(j) = P(y_1, \ldots, y_t, e_t = j | m) \quad \text{(EQ 2.7)}$$

soit la probabilité que le vecteur d'observation $y_1, \ldots, y_t$ aussi dénoté $y_1^t$, ait été généré au moyen d'un chemin se terminant à l'état $j$, ($e_t$ étant l'état occupé au temps $t$). On constate que $P(y | HMM_m)$ peut s'exprimer sous la forme $\alpha_L(f)$ où $L$ est la longueur totale du message et $f$ l'état final du HMM. Finalement, en mémorisant les différentes valeurs de $\alpha_t(j)$ dans les entrées $[j]$ $[t]$ correspondantes de la matrice *mat*, le calcul itératif suivant nous permet d'obtenir $\alpha_L(f)$ directement de $mat[f]$ $[L]$

$$\alpha_0(0) = 1 \quad \text{(EQ 2.8)}$$

et

$$\alpha_t(j) = \sum_{(tr | tr\text{pleine},\, a_{tr} = j)} \alpha_{t-1}(d_{tr}) q_{tr} b_{tr}(y_t) + \sum_{(tr | tr\text{vide},\, a_{tr} = j)} \alpha_t(d_{tr}) q_{tr} \quad \text{(EQ 2.9)}$$

où $a_{tr}$ est l'état d'arrivée et $d_{tr}$ l'état de départ de la transition $tr$.

### 2.4.1 L'ordonnancement des transitions vides

Dans l'équation 2.9, on note qu'étant donnée une transition vide $tr_{ij}$, il est nécessaire de connaître la valeur de $\alpha_t(i)$ pour effectuer le calcul de $\alpha_t(j)$. Il faut donc, dans ce cas, s'assurer de calculer $\alpha_t(i)$ avant $\alpha_t(j)$. En fait, plus généralement, en présence d'un chemin composé de transitions vides $tr_{ij}$, $tr_{jk}$, …, $tr_{lm}$, il faut toujours s'assurer d'effectuer le traitement des $\alpha_t(x)$ en ordre de $x = i, \ldots, m$. Pour ce faire, il suffit simplement d'appliquer un algorithme d'ordonnancement des états de sorte à ce que les états traversés par un chemin de transitions vides se voient attribués des numéros d'ordre croissant (on suppose que les états sont visités selon cet ordre). De même on s'assurera, lors d'un parcours inversé de la structure de recherche (voir section 7.2), de traiter les états en ordre décroissant. Cette contrainte implique qu'il ne peut y avoir de boucles de transitions vides dans une structure de recherche.

## 2.5 L'approche phonétique

On a déjà mentionné à la section 2.2 qu'un HMM pouvait représenter n'importe quel ensemble d'unités acoustiques: mots, syllabes, phonèmes etc. Or dans le cas de grands vocabulaires, le fait d'associer à chaque mot un HMM distinct pose de graves problèmes au niveau de l'entraînement. En effet, il faudrait pour effectuer cette tâche correctement, que le corpus d'entraînement contienne plusieurs occurrences de chaque mot. Ceci est pratiquement impossible. Qui plus est, à l'ajout d'un nouveau mot au lexique, correspondrait la collecte de nouvelles occurrences acoustiques d'entraînement et un nouvel entraînement, ce qui n'est guère flexible. Dans les faits, la reconnaissance de grands vocabulaires est donc toujours effectuée au moyen de HMMs de mots élaborés à partir de HMMs représentant des sous-unités de mots.

Il est, en effet facile de combiner plusieurs HMMs en un seul. Un exemple de ce type de construction, où un HMM correspondant au mot *petit* est obtenu par la concaténation des HMMs correspondant aux phonèmes *p, e, t,* et *i* est illustré à la figure 2.4. On peut voir en A que pour réaliser une telle construction, on peut faire coïncider le dernier état d'un HMM de phonème avec le premier état du HMM du phonème suivant. En passant, la topologie de HMM choisie pour représenter les modèles unitaires peut ou non faciliter un tel amalgame, comme ici le fait de ne pas permettre de transitions bouclées sur le dernier état, mais un amalgame reste toujours possible en liant au moyen d'une transition vide le dernier état d'un HMM au premier état du HMM suivant. Un HMM un peu plus sophistiqué en B, pour le même mot, illustre l'ajout d'une transition vide permettant l'omission d'un phonème.

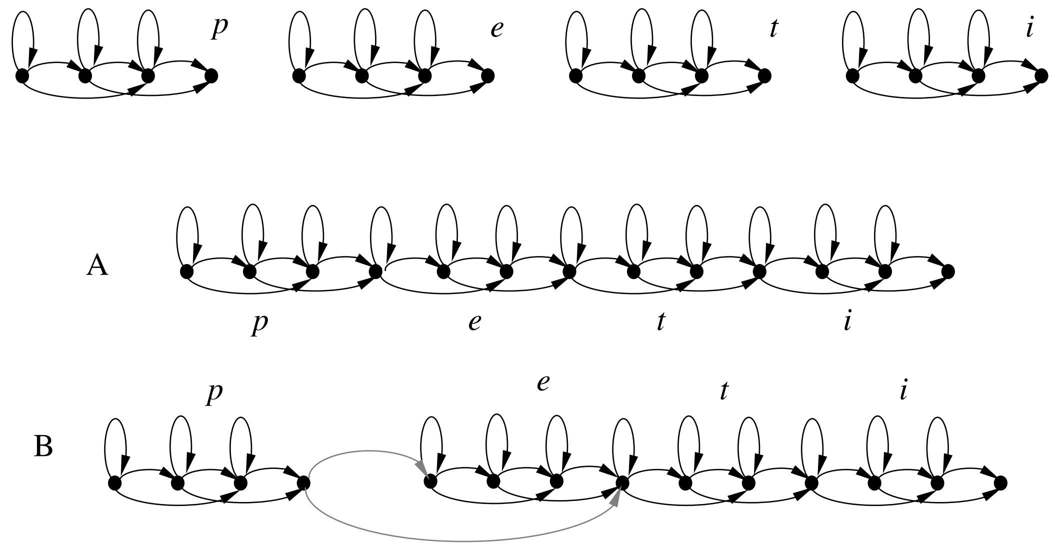


**Figure 2.4 HMM d'un mot obtenu par concaténation de HMMs de phonèmes.**

Les HMMs obtenus par concaténation d'autres HMMs donnent en général des résultats de reconnaissance légèrement inférieurs à ceux obtenus directement au moyen de HMMs entraînés sur les unités qu'ils sont censés représenter. Cependant, l'utilisation de ce type de construction permettant de réduire le nombre de HMMs nécessaire pour une tâche donnée (par exemple pour un vocabulaire de 5000 entrées, 46 HMMs de phonèmes pourraient suffire plutôt que 5000 HMMs de mots), elle permet d'éviter de devoir collecter les énormes corpus d'entraînement qui auraient été nécessaires lors de l'utilisation de HMMs de mots et donc de rendre réalisable la reconnaissance de grands vocabulaires. Qui plus est, l'impact au niveau flexibilité est aussi très important. En effet, l'ajout de nouveaux mots dans le lexique ne nécessite plus aucune collecte ni entraînement supplémentaire, il suffit simplement de déterminer la ou les séquences de sous-unités correspondant aux nouveaux mots et de construire selon ces séquences et à partir des HMMs des sous-unités correspondantes, les HMMs modélisant ces nouveaux mots. En prime, cette approche, qui par ailleurs est connue sous le nom d'approche phonétique du nom de la sous-unités la plus “linguistique” des mots: le phonème, permet une économie au niveau de l'espace mémoire requis tant lors de la reconnaissance que de l'entraînement. En effet, le nombre de distributions devant être conservé en mémoire est de beaucoup inférieur à celui qu'aurait nécessité l'utilisation d'un HMM par mot du lexique.

Finalement, de la même façon qu'on peut construire des HMMs de mots à partir de HMMs de phonèmes, on peut construire des HMMs de phrases à partir de HMMs de mots.

C'est ce qui est illustré à la figure 2.5. On soulignera que "*pause*" n'est pas ici le mot

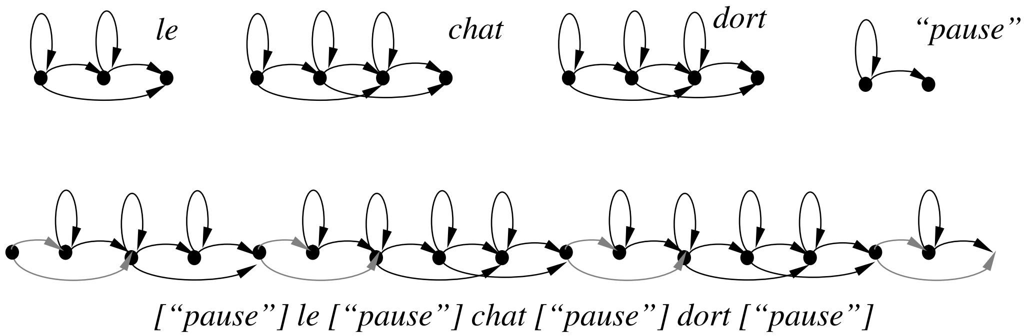


**Figure 2.5 HMM d'une phrase obtenu par concaténation de HMMs de mots**

*pause* mais bien la réalisation acoustique d'une pause entre deux mots. Cette pause pouvant ou non être présente, on a rajouté une transition vide en parallèle avec le HMM correspondant, de la même façon qu'on l'avait fait pour le phonème *e* à la figure 2.4B.

### 2.5.1 Unités de bases

Quoique d'un point de vue linguistique, il soit naturel de définir un mot au moyen d'une séquence (incluant possiblement une série de variantes) de phonèmes, ce dernier type de sous-unités n'est pas celui le plus couramment utilisé afin de construire des HMMs de mots. En effet, bien que clairement définie d'un point de vue linguistique, la réalisation acoustique d'un phonème l'est beaucoup moins. Cette réalisation acoustique est, en effet, fortement influencée par l'identité des phonèmes qui précèdent et suivent immédiatement un phonème donné. C'est ce qu'on appelle le phénomène de co-articulation. C'est pourquoi en RAP, les unités servant à la construction des HMMs de mots, bien qu'en général dérivées de phonèmes, seront plus nombreuses et mieux définies d'un point de vue acoustique que ces derniers. Les sections suivantes en présentent trois types, du plus simple au plus sophistiqué.

#### 2.5.1.1 Les triphones

La solution au problème de co-articulation qui vient immédiatement à l'esprit, est de modéliser séparément autant d'instances d'un phonème qu'il existe de contextes possibles pour ce phonème. Ainsi, si l'on choisit, par exemple, de fixer ce contexte au phonème précédant et suivant immédiatement le phonème à modéliser, on aura alors défini ce qui est connu sous le nom de triphones. C'est ainsi qu'au lieu d'un seul HMM pour le phonème *a*, on aura autant de HMMs qu'il y a d'instances de $x$=$a$=$z$ où $x$ est un phonème pouvant possiblement précéder $a$ et $z$ un phonème pouvant possiblement le suivre. Par exemple, en français, *m*=*a*=*m* (*maman*), *p*=*a*=*p* (*papa*), *m*=*a*=*t* (*tomate*) etc. sont tous des triphones valides de $a$ pour lesquels un HMM devrait être entraîné.

Comme on peut le voir, pour un même vocabulaire, le nombre de HMMs de triphones nécessaires à la construction des HMMs de mots est beaucoup plus grand que le nombre de HMMs de phonèmes nécessaires. Qui plus est, dans le cas de la parole continue, il faut possiblement rajouter aux HMMs modélisant des occurrences de phones à l'intérieur des mots (dont sont les exemples du paragraphe précédent) des HMMs permettant de modéliser des occurrences de phones ne se retrouvant qu'à la frontière de ceux-ci, par exemple, par opposition au triphone de $a$, *m*=*a*=*m* (*maman*) de type intra-mot, on a le triphone *è*=*k*=*z* (*…avec zouave…*) de type inter-mot. C'est ainsi que de façon générale, on passera d'une quarantaine de phonèmes à quelques milliers de triphones intra-mots et plus du double de triphones inter-mots[1]. Ce qui est beaucoup. [YOUN 94] cite même le cas de la tâche WSJ où le nombre potentiel de HMMs représentant des triphones (55 000) est supérieur au nombre de mots du lexique (20 000). En fait, le nombre théorique de triphones possible pour un ensemble de $x$ phonèmes est de $x^3$. Avec l'emploi de triphones, on peut donc se retrouver face à l'un des problèmes que l'emploi de modèles d'unités de base était censé solutionner, soit la nécessité de collecter de grandes quantités de données afin d'entraîner adéquatement un grand nombre de HMMs.

#### 2.5.1.2 Les modèles de phonèmes "généralisés"

Pour contourner le problème posé par un nombre potentiellement trop grand de HMMs de triphones, tout en conservant le potentiel accru de modélisation qu'ils procurent, on peut utiliser ce qu'on choisit d'appeler des HMMs de phonèmes généralisés. On doit le qualificatif de "généralisé" à Kai Fu Lee [LEEH 90] qui le premier l'appliqua aux triphones.

Un triphone généralisé est un modèle de triphone obtenu par regroupement de modèles de triphones "similaires". L'idée à la base de l'algorithme initialement proposé par Lee, est de combiner de façon successive et jusqu'à ce qu'un critère de convergence soit rencontré, les HMMs jugés similaires selon une mesure donnée. Initialement, un HMM pour chaque triphone présent au sein des données d'entraînement est créé. Par la suite, ce nombre diminue progressivement au fur et à mesure que l'algorithme progresse. Lee justifie cette

1. Pour des exemples concrets voir [LEEH 89], en gardant à l'esprit que ces nombres varient selon le lexique et la liste des phonèmes initiaux.

approche par le fait que, certains contextes gauches et/ou droits similaires risquant d'avoir le même effet sur le phonème modifié, il est correct de combiner les triphones représentant ces contextes. Par exemple, la réalisation acoustique du phonème *a* risque d'être similaire lorsque coincé entre l'une des deux fricatives *s* et *z* d'une part et l'une des deux plosives *p* et *b* d'autre part. Un seul HMM pourrait donc être utilisé pour représenter les triphones *s=a=b, s=a=p, z=a=b* et *z=a=p.*

Cette idée, reprise par l'équipe du Speech Recognition Group de IBM [BAHL 91] (et depuis par d'autres dont l'équipe du CRIM [KUHN 95]) et combinée à la théorie des arbres de classification [BREI 84] a donné naissance à des modèles de phonèmes généralisés tant au niveau du type de contexte qu'au niveau de la longueur de ce contexte. En effet, les triphones ne tiennent compte que d'un seul phonème immédiatement à gauche et/ou à droite d'un autre. Or, IBM émit l'hypothèse que la prononciation d'un phonème puisse être influencée par les deux, trois, ou même plus, phonèmes précédant et/ou suivant. Evidemment, en généralisant la longueur possible des contextes, il devient impératif, vu leur nombre possiblement élevé, de trouver une méthode permettant de les combiner.

Le principe est le suivant. D'abord tous les contextes possibles pour un phonème sont regroupés. Puis, l'arbre est construit, de façon à ce qu'à chaque noeud interne corresponde la question qui départage le plus "efficacement" les contextes associés au noeud en deux sous-groupes. Ces sous-groupes sont ensuite associés à deux nouveaux noeuds issus du noeud courant. Ce processus est poursuivi jusqu'à un critère d'arrêt quelconque comme par exemple, jusqu'à ce que le nombre de contextes associés à un noeud soit inférieur à un seuil donné. Le résultat final est un arbre dont chaque feuille représente une classe de contextes équivalents pour lesquels un HMM est entraîné. C'est ce que réalisent les arbres de décision, à raison d'un arbre par phonème. Un exemple d'un tel arbre est illustré à la figure 2.6 pour le phonème *b* où une question du genre "le deuxième phonème de gauche est-il une pause" est traduit sous la forme simplifiée "- 2 = pau".

En plus de théoriquement permettre la prise en compte de contexte très large, cette approche fait en sorte qu'il est très facile d'associer un HMM à un contexte n'ayant jamais été rencontré auparavant. Il suffit de parcourir l'arbre correspondant au phonème pour lequel un HMM est désiré, en répondant aux questions associées aux noeuds et ce, jusqu'à la feuille correspondante. Le HMM désiré est celui associé à cette feuille. Il s'agit d'un avantage indéniable au problème des contextes inconnus traditionnellement résolu en utilisant en lieu et place d'un HMM de triphones, un modèle de diphones (modèle où seul un des deux contextes est connus) ou à défaut de phonèmes. Finalement, détail non négligeable,

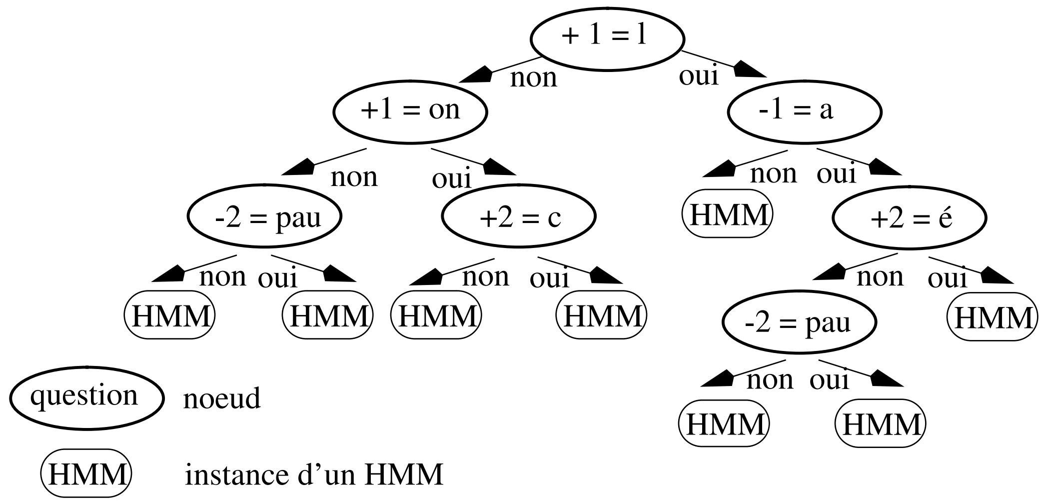


**Figure 2.6 Exemple d'un arbre de classification utilisé dans la conception de HMMs généralisés**

cette approche est plus économique en temps et espace que celle proposée par LEE, car il n'est pas nécessaire d'effectivement construire un HMM pour chaque triphone possible comme c'est le cas dans l'approche de LEE.

2.5.1.3 Les modèles étatiques

Ce qu'il sera convenu d'appeler à défaut d'une meilleure traduction des modèles "étatiques"[1] est un type de HMM correspondant à une position unique de triphones (en fait à un état unique d'un HMM de triphones). La topologie commune de ces HMMs est illustrée à la figure 2.7A, et la façon dont ils se combinent pour former un HMM de triphones à

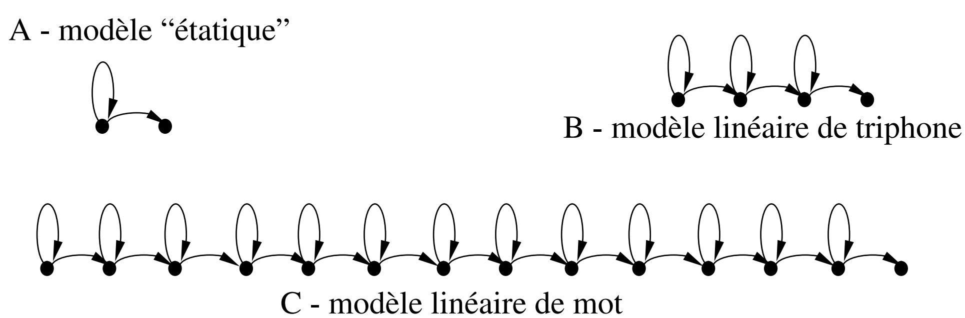


**Figure 2.7 Exemple de modèles "étatiques" et modèles linéaires**

1. Tied tree-based state model [YOUN 94]

la figure 2.7B. Ce type de modèle est appelé linéaire car seules deux transitions par état sont permises: une transition bouclant sur l'état et une transition liant ce dernier à l'état immédiatement à sa droite. Les probabilités d'émission associés à ces deux transitions étant la même, un HMM de mots composé à partir de HMMs étatiques (figure 2.7C) peut efficacement être représenté par un vecteur de $n-1$ distributions et $2(n-1)$ probabilités de transitions.

Mais la véritable révolution des modèles étatiques n'est pas tant au niveau de leur topologie qu'ils partagent d'ailleurs avec les "senones" [HWAN 92] et les "fenones" [BAHL 88b] mais dans ce à quoi ils correspondent et dans la façon dont cette correspondance est établie. Le processus débute par l'entraînement de HMMs linéaires de triphones. Puis les distributions associées au même état des triphones d'un même phone sont regroupées et associées à un arbre de décision. Ensuite, pour chaque arbre de décision (au nombre de un par état par phonème), on procède au regroupement des distributions similaires sur la base de critères acoustiques portant sur les contextes gauches et droits dont sont issus ces distributions. Les distributions associées à chaque feuille de l'arbre sont ensuite mathématiquement combinées pour donner la distribution associée à un HMM étatique. Ce qui veut dire que l'ensemble des HMMs étatiques associé à un arbre de décision correspondent au diverses réalisations acoustiques partagées par une même "partie" d'un même phonème.

## 2.6 Les modèles du langage

Le deuxième terme de l'équation 2.2, $m = \underset{m}{\operatorname{argmax}}\; P(y|m)P(m)$, $P(m)$, est, on se rappelle (voir section 2.1), la probabilité à priori de $m$ aussi appelée la probabilité du modèle du langage. Cette probabilité est censée représenter les probabilités relatives à priori des messages possibles. Par exemple, une façon d'estimer cette probabilité pour une séquence de mots $m = m_1 m_2 \ldots m_{Lm}$ est d'attribuer à chaque mot du lexique $m_i$ dont $m$ est composée une probabilité d'occurrence hors tout contexte[1] et d'obtenir la probabilité de $m$, en multipliant entre eux les probabilités associées à chacun des mots de la séquence soit $P(m) = P(m_1)P(m_2)\ldots P(m_{LM})$.

Ce modèle mathématiquement tout à fait valable, se nomme un unigramme, mais est d'une aide toute relative quand il s'agit de guider le choix de la reconnaissance face à plusieurs possibilités valables d'un point de vue acoustique. La principale raison de cette fai-

1. En pratique, cette probabilité est estimée par la fréquence de chaque mot dans un corpus d'entraînement.

blesse est justement qu'il fait fi de tout contexte, or une langue n'est pas seulement formée d'un ensemble de mots, mais aussi d'une série de règles régissant l'ordre selon lequel ces divers mots peuvent apparaître. C'est ce qu'on appelle la syntaxe. Mais plus encore, les mots eux-mêmes, de par la réalité de ce qu'ils décrivent, leur sens, ont une influence sur la façon dont ils sont utilisés. L'on parle alors de sémantique et de pragmatique. Une bonne valeur de $P(m)$ , et par extension un bon modèle du langage, devra donc, autant que possible, tenir compte de ces diverses contraintes.

Un bref survol des différents modèles du langage existants est fait dans les sections qui suivent. Y sont soulignés les avantages et désavantages de chaque modèle et une attention plus particulière est portée sur leur facilité d'intégration aux algorithmes de reconnaissance et sur l'impact qu'ils ont sur ces derniers tant au niveau vitesse d'exécution que taux de reconnaissance.

### 2.6.1 Description générale

De façon générale, on distingue actuellement trois grandes familles de modèles du langage: deux, basés sur des approches totalement différentes, qui sont les modèles à syntaxe fixe et les modèles probabilistes locaux et une dernière, les modèles à grammaires probabilistes, qui, comme son nom l'indique, est issu du mariage des deux premières.

### 2.6.2 Les modèles du langage à syntaxe fixe

Les modèles du langage à syntaxe fixe sont directement calqués sur la syntaxe propre à une langue. Ils furent très utilisés dans les systèmes du début des années soixante-dix, époque à laquelle l'emploi entre autres des ATN (Augmented Transition Networks) était en pleine expansion [WOOD 70]. Leur emploi en reconnaissance de la parole était alors le reflet de leur emploi dans des tâches de compréhension de la langue écrite où la majorité de l'expertise sur le sujet se trouvait.

Dans un modèle à syntaxe fixe, c'est souvent une série de règles exprimées sous forme de CFG (Context Free Grammar) et souvent traduit sous forme de graphes, qui déterminent quelles sont les séquences de mots valides ou non (voir figure 2.8). En fait, c'est un peu comme si la probabilité $P(m)$ était de 0 ou 1 selon que $m$ est une séquence valide ou non du graphe considéré. Hélas, cette façon de faire devient vite inapplicable pour de très grands vocabulaires et ce, non seulement à cause de l'espace prohibitif qu'occuperait le graphe final mais aussi à cause du fait que les règles à partir desquelles le graphe est construit doivent être élaborées à la main. Or, si cette tâche peut, à la rigueur, être effectuée

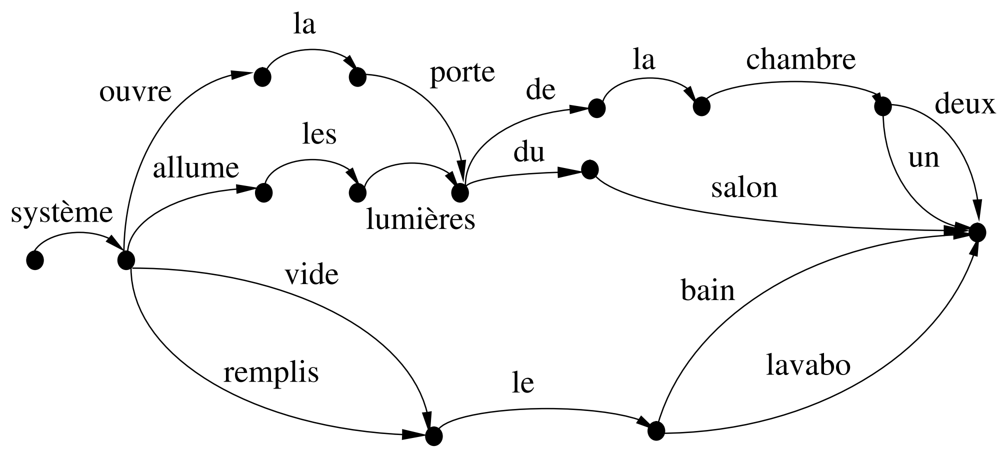


**Figure 2.8 Exemple d'un graphe traduisant un modèle du langage à syntaxe fixe**

sans trop de problème dans des cas d'applications très précises (les linguistes parlent alors de sous-langages [KITT 82a] et [KITT 82b]), cela s'avère impossible pour une langue considérée dans son entier. En fait, il n'y a aucune langue pour laquelle une grammaire tout à fait exhaustive existe [BLAC 93]. De plus, contrairement à la langue écrite où la syntaxe d'une langue est en général assez bien respectée, on retrouve très souvent au niveau de la langue parlée (surtout au niveau de la parole spontanée) des séquences de mots grammaticalement incorrectes, qui, lors de l'emploi de modèles du langage à syntaxe fixe, ne pourraient pas être reconnues.

### 2.6.3 Les modèles du langage probabilistes locaux

Devant la faillite grandissante de l'approche à syntaxe fixe face au problème de RAP pour de grands vocabulaires, une approche alternative fut proposée par Frederick Jelinek [JELI 91], l'approche probabiliste locale. Dans cette approche, les modèles sont basés sur des statistiques d'occurrence de mots appelés n-gram où $n$ représente la longueur en mots d'une séquence. Par exemple "*le petit chien*" est un trigramme et "*petit chien*" un bigramme. Ces statistiques s'obtiennent automatiquement au moyen de textes types servant de corpus d'entraînement. Mathématiquement, la probabilité d'une séquence de mots $m = m_1, \ldots, m_{Lm}$, s'exprime par:

$$P(m) = P(m_1) \prod_{i=2}^{Lm} P(m_i | m_1, \ldots, m_{i-1}) \qquad \text{(EQ 2.10)}$$

En pratique, la longueur de la séquence maximum est fixe, de deux $(n = 2)$ dans le cas d'un modèle de bigramme, équation 2.11, ou de trois $(n = 3)$ dans le cas d'un modèle de trigramme, équation 2.12.

$$P(m_i | m_1, \ldots, m_{i-1}) \cong P(m_i | m_{i-1}) \tag{EQ 2.11}$$

$$P(m_i | m_1, \ldots, m_{i-1}) \cong P(m_i | m_{i-2}, m_{i-1}) \tag{EQ 2.12}$$

En théorie, plus le $n$ choisi est grand, plus le modèle est puissant. C'est-à-dire que mieux il serait à même de prédire la bonne séquence de mots étant donné un corpus d'entraînement infini. Ceci n'étant jamais le cas, le puissance pratique d'un modèle est aussi fonction du corpus d'entraînement. En effet, un plus grand $n$ correspondant à une nombre plus élevé de paramètres à estimer, il est inutile de le choisir élevé si le corpus d'entraînement n'est pas assez large pour permettre une bonne estimation de ces différents paramètres[1]. Autrement dit, mieux vaut moins de paramètres bien estimés que beaucoup de paramètres mal estimés.

La puissance de prédiction pratique d'un modèle est en général estimée au moyen de la perplexité $PP$, qui, pour un modèle de langage statistique donné $P_{ML}$ et un corpus test donné, peut être calculée au moyen de l'équation 2.13:

$$PP = 1 / \left( \prod_{i=1}^{lc} P_{ML}(m_i) \right)^{\frac{1}{lc}} \tag{EQ 2.13}$$

où $lc$ est le nombre de mots total dans le corpus test et où ce dernier est composé de textes différents mais de semblable origine aux textes du corpus d'entraînement.

En général plus la perplexité est haute, moins le modèle est puissant (plus le modèle est perplexe face à un texte) et vice versa. Il faut cependant souligner que la perplexité d'un modèle du langage n'est valable que de façon relative. Ainsi, de grands écarts de perplexité ne se traduisent-ils pas nécessairement par des écarts d'un même ordre au niveau performance de reconnaissance d'un système. Ils indiquent plutôt si, suite à l'emploi d'un modèle plutôt qu'un autre, l'on peut espérer on non une amélioration du taux de reconnaissance. Encore là, on peut trouver des cas où une meilleure perplexité s'est traduite par

1. Lorsqu'une séquence de mots n'apparait jamais ou peu de fois, sa probabilité peut alors être dérivée au moyen de méthodes de lissage ([JELI 90]).

une légère baisse du taux de reconnaissance [CHAS 95]. Finalement, certains comportements propres à la RAP sont ignorés par cette mesure. Par exemple, en RAP en général, les mots courts sont plus difficiles à détecter et distinguer entre eux que les mots plus longs. Or la perplexité ne tenant pas compte de ce type de caractéristiques, un ML qui prédirait avec plus d'exactitude les mots longs, et qui aurait donc une plus basse perplexité pourrait n'offrir aucun avantage face à un autre ML lors d'une tâche de RAP.

Comme on l'a déjà souligné, il existe un compromis à faire au niveau du choix d'un modèle de langage entre une valeur de $n$ élevée et la quantité de données disponible pour l'entraînement. Or lorsque la quantité disponible de données pour l'entraînement est un aspect important à considérer, on aimerait quand même réussir à utiliser un ML ayant un $n$ plus grand que un. Quoiqu'un accès à une quantité de données limitée soit un problème qui puisse se présenter pour toutes les langues et toutes les tâches, une langue comme le français est, à cet égard, désavantagée par rapport à une langue plus compacte comme l'anglais. En effet, le français exige pour une même couverture d'une même tâche, un nombre de mots nettement plus grand que l'anglais[1], entre autres à cause de son caractère déclinable. Ceci, à son tour, se traduit par la nécessité d'un corpus d'entraînement plus grand.

Pour faire face à ce problème, un autre ensemble de modèles probabilistes, appelés n-POS ou ng-gram a été développé par un groupe de français [DERO 84], [DERO 86]. Dans cette variante des n-grams, ce sont les "parties du discours" (en anglais Part of Speech d'où l'acronyme POS) auxquelles appartiennent les différents mots d'une séquence qui sont considérés dans le calcul de la probabilité d'une séquence, la probabilité du mot servant maintenant plutôt à distinguer entre eux les mots d'une même POS.

Ainsi dans le cas de modèles de type n-POS, l'équation générale est:

$$P(m_i | m_1, \ldots, m_{i-1}) \cong P(g(m_i) | g(m_{i-n+1}), \ldots, g(m_{i-1})) \cdot P(m_i | g(m_i)) \quad \text{(EQ 2.14)}$$

et les équations 2.11 et 2.12 se ré-écrivent selon les équations 2.15 (bipos) et 2.16 (tripos):

$$P(m_i | m_1, \ldots, m_{i-1}) \cong P(g(m_i) | g(m_{i-1})) \cdot P(m_i | g(m_i)) \quad \text{(EQ 2.15)}$$

$$P(m_i | m_1, \ldots, m_{i-1}) \cong P(g(m_i) | g(m_{i-2}), g(m_{i-1})) \cdot P(m_i | g(m_i)) \quad \text{(EQ 2.16)}$$

1. 21% de plus pour TransTalk, tâche dont on reparlera aux chapitres 4 et 8.2 et à l'annexe A.

où $g(m_i)$ dénote la POS du mot $m_i$ et $P(m_i | g(m_i))$ la probabilité du mot dans sa POS, probabilité elle-aussi obtenue à partir du corpus d'entraînement.

Les POS peuvent être de nature syntaxique (ce qu'ils étaient originellement): article, pronom possessif, ou, à la limite, de nature plutôt sémantique (dans des articles plus récents): nom d'aéroport, nom de ville. Le nombre, la nature et ce que représente chaque POS sont alors tout à fait arbitraires et dépendent de l'application à laquelle le modèle est destiné.

Le problème des n-POS est que même s'ils demandent de moins grande quantité de textes au niveau de l'entraînement, ces derniers doivent, par contre, être correctement étiquetés. C'est-à-dire qu'à chaque mot doit correspondre sa POS. Or même si cet étiquetage peut, jusqu'à un certain point, être automatisé, il peut souvent s'avérer plus laborieux qu'une collecte de textes supplémentaires non étiquetés. Par exemple, dans le cas de POS d'origine syntaxique, il faudra manuellement distinguer les *la* noms masculins des *la* articles et des *la* pronoms personnels.

D'où l'émergence d'un dernier groupe de modèles, les n-lemma [ELBE 90] qui regroupent les avantages des n-grammes (étiquetage triviale des corpus d'entraînement) et n-POS (corpus d'entraînement à la taille réduite) afin, toujours, de mieux répondre aux besoins des langues à déclinaison comme le français. Dans cette approche, seules quelques POS sont définis, qui correspondent au lemme d'un mot. Par exemple au POS *chanter* correspondra les mots *chantant*, *chantait*, *chante* etc.

Comme on l'a souligné, le $n$ des modèles statistiques est généralement, pour des raisons pratiques (entre autres la taille du corpus nécessaire à leur entraînement), limité aux valeurs 2, 3 parfois 4. Dans ces conditions, on constate qu'ils manquent alors de capacité de prédiction à long terme. Or on peut, par l'ajout de caches, augmenter leur pouvoir de prédiction. On qualifie les modèles ainsi modifiés de dynamiques. Ce processus est dit dynamique car effectué au fur et à mesure de la reconnaissance d'un texte. Il agit en modifiant dynamiquement les probabilités à priori de certains mots. Ces mots peuvent simplement être ceux ayant déjà été rencontrés dans le texte [KUHN 90], ou ils peuvent être liés par le sens à des mots déjà rencontrés. Par exemple, en présence du mot *chien(s)* les mots *puce(s)*, *collier(s)* et *niche(s)* pourraient voir leur probabilité augmentée [ROSE 94b].

De façon générale, qu'il s'agisse de n-grams, ng-grams ou n-lemmes, dynamiques ou pas, les modèles probabilistes possèdent les deux qualités qui manquaient aux modèles à syntaxe fixe. Ils sont faciles à entraîner et sont robustes. En prime, puisque le calcul de $P(m)$ s'effectue essentiellement de manière gauche-droite (seules les mots précédant immédiatement un mot *m* doivent être connus), ils peuvent facilement être incorporés aux algorithmes de fouille, nous y reviendrons à la section 3.1.3 mais surtout au chapitre 7. De fait, pour toutes ces raisons, ce sont actuellement les types de modèles utilisés dans quasi tous les systèmes à grands vocabulaires.

### 2.6.4 Les modèles du langage à grammaire probabiliste

Hélas, même les meilleurs modèles dynamiques sont contraints d'abdiquer devant certains problèmes courants tel les accords à longue portée entre sujets et verbes. Or il est important de pouvoir résoudre ce type d'accord dans certaines applications comme la dictée automatique et ce, spécialement pour des langues comme le français, où le taux d'homonymes par entrée phonétique est très élevé. Par contre, comme de tels accords ne posaient pas de problèmes (ou beaucoup moins) aux modèles à syntaxe fixe, on tente actuellement de combiner les avantages des modèles statistiques et des modèles à syntaxe fixe au moyen de ce qui est connu sous le nom de grammaires probabilistes ([BLAC 93]). L'idée générale de ce genre d'approche est de réussir à dériver de grammaires fixes, des probabilités pouvant être combinées avec celles d'un modèle probabiliste. Les grammaires probabilistes en sont cependant encore à un stade très expérimental. Ils ne sont encore utilisées que dans de très petites applications aux grammaires proches des sous-langages ou alors comme post-traitement applicable, par exemple, sur une liste d'hypothèses produites par la reconnaissance.

# 3 L'algorithme de Viterbi

A la section 2.4, on a présenté l'algorithme "avant" au moyen duquel il est facile d'obtenir la valeur de $P(y|HMM_m)$ . Mais cet algorithme a un inconvénient majeur. En effet, il faut, pour trouver le message associé à un signal inconnu, l'appliquer à tous les candidats possibles. Ceci peut toujours être fait dans le cas où ce nombre n'est pas trop grand. Cependant, cette façon de faire devient rapidement irréaliste. Par exemple, prenons un module de RAP dont la tâche est de reconnaître des numéros de téléphone énoncés chiffres par chiffres et supposons, qu'à priori, aucune séquence ne soit impossible sauf celles débutant par un 0. Cela fait $9 \times 10^6 = 9000000$ séquences possibles. Il est, dans ce cas pourtant fort simple, clairement impossible de construire à tour de rôle le HMM représentant chacune des séquences et d'y appliquer l'algorithme "avant", pour ensuite choisir parmi ces dernières, celles obtenant la meilleure probabilité.

Pour résoudre ce problème, on aura recourt à l'algorithme de Viterbi [VITE 67]. Ce dernier est actuellement à la base, sous une forme ou une autre, de presque tous les systèmes de RAP utilisant comme modèles acoustiques des HMMs. C'est l'algorithme de la situation.

En théorie, l'algorithme de Viterbi est un algorithme de programmation dynamique permettant de trouver parmi tous les chemins possibles d'un graphe celui ayant le "coût" le plus bas. En RAP, on l'utilise pour trouver le chemin ayant le "coût" le plus élevé. Ceci ne présente aucun problème puisque le nombre de transitions pleines de la solution doit être égal à $L$, le nombre d'unités de temps du message[1]. Puisqu'il peut paraître bizarre de vouloir maximiser un "coût", on parlera plus volontier de scores. C'est ainsi que le score d'un chemin menant à un noeud $j$ est simplement obtenu en sommant le score du chemin

1. Dans l'algorithme d'origine, le nombre de transitions de la solution n'étant pas limité et un coût étant défini par une valeur positive, trouver le chemin de coût maximal ferait boucler l'algorithme à l'infini en présence d'une boucle de coût > 0 dans le graphe.

menant à un noeud $i$ et le score du passage du noeud $i$ au noeud $j$. Or, il est naturel, dans un HMM de définir le score du passage d'un noeud $i$ à un noeud $j$ au moyen d'une transition pleine $tr_{ij}$ pour un temps $t$ à $t+1$ donné par $\log(q_{tr_{ij}} b_{tr_{ij}}(y_t))$ et celui du passage d'un noeud $i$ à un noeud $j$ au moyen d'une transition vide $tr_{ij}$ par $\log(q_{tr_{ij}})$ où $q_{tr_{ij}}$ est la probabilité de transition et $b_{tr_{ij}}$ la probabilité d'émission de $tr_{ij}$. Par la suite il suffit des quelques lignes de pseudo-code suivantes pour trouver le chemin ayant le score maximal et correspondant au chemin ayant la plus haute probabilité.

```
/* la fonction cout retourne le score d'une transition tel que définie ci-haut */
int     mat[0..f][0..L+1]       /* colonne = temps, rangée = noeud du HMM */
mat[0][0] = 0
for (col=0; col<L; col++) {
        for (i=0; i<f; i++) {
                for (toutes les transitions tr_ix en provenance du noeud i) {
                        nouv_cout = mat[i][col] + cout(tr_ix)
                        if (tr_ix est vide) {
                                if (nouv_cout > mat[x][col]) {
                                        mat[x][col] = nouv_cout /* 1 */
                                }
                        }
                        if (tr_ix est pleine) {
                                if (nouv_cout > mat[x][col+1]) {
                                        mat[x][col+1] = nouv_cout /* 2 */
                                }
                        }
                }
        }
}
for (i=0; i<f; i++) {
        for (toutes les transitions vides tr_ix en provenance du noeud i) {
                nouv_cout = mat[i][col] + cout(tr_ix)
                if (nouv_cout > mat[x][col]) {
                        mat[x][col] = nouv_cout         /* 3 */
                }
        }
}
```

En fait, lorsqu'on y regarde de plus près, il s'agit simplement d'une ré-écriture des équations 2.8 et 2.9, où la sommation sur tous les chemins menant à une case de la matrice est remplacée par la prise du maximum. Et l'algorithme de Viterbi n'est qu'une méthode alternative pour le calcul des $\alpha_t(i)$, ici aussi utilisé pour remplir une matrice de probabilité. On a donc maintenant que:

$$\alpha_0(0) = 1 \qquad \text{(EQ 3.1)}$$

et

$$\alpha_t(j) = \max(\max1, \max2) \qquad \text{(EQ 3.2)}$$

où $\max1 = \max_{(tr|\ tr\text{pleine},\ a_{tr} = j)} (\alpha_{t-1}(d_{tr})\, q_{tr} b_{tr}(y_t))$

et $\max2 = \max_{(tr|\ tr\text{vide},\ a_{tr} = j)} (\alpha_t(d_{tr})\, q_{tr})$

Graphiquement, la matrice de fouille visitée par l'algorithme de Viterbi est donc la même que celle visitée par l'algorithme "avant" et illustrée à la figure 2.3. En particulier, ici aussi il est important, avant d'effectuer la fouille, de s'assurer de la bonne numérotation des états du HMM, en regard des transitions vides.

L'expérience a prouvé que, pour une même tâche de RAP, le fait de ne tenir compte que du chemin maximal et non de l'ensemble des chemins tel que dans l'algorithme "avant", ne change à peu près pas les taux de reconnaissance et, ce, pour un temps de calcul légèrement moindre. Cependant, là n'est pas le principal avantage de l'algorithme de Viterbi. Son principal avantage est plutôt qu'en plus de donner la probabilité du meilleur chemin on puisse facilement en retracer les étapes. Pour cela, il suffit de modifier légèrement le pseudo-code de la page 28 afin de mémoriser en plus de la probabilité maximale, la transition $tr_{ix}$, d'où cette probabilité maximale a été obtenue. Pour ce faire, il suffit de rajouter suite aux lignes (1), (2) et (3) la ligne suivante:

*mat_chemin*[*x*][*col* (ou *col*+1)] = identité de $tr_{ix}$

où *mat_chemin* est une matrice de dimension [0..f][0..L+1] d'identificateurs de transitions.

On peut visualiser ce à quoi correspond le résultat de cette mémorisation à la figure 3.1, où le meilleur chemin à travers un graphe pour un signal donné, est formée de la séquence des transitions élargies.

Fort bien, mais en quoi cela peut-il nous aider pour notre application de RAP de numéros de téléphone. Rappelons qu'à la section 2.5, on a vu comment à partir de HMMs de phones, construire des HMMs de mots, et à partir de HMMs de mots, construire des HMMs de phrases. On a aussi vu à la section 2.6.2, comment une syntaxe fixe pouvait se traduire sous forme de graphe. Or l'ensemble des numéros de téléphones correspond bien à une syntaxe fixe. Si chaque mot de ce graphe est remplacé par le HMM correspondant, il

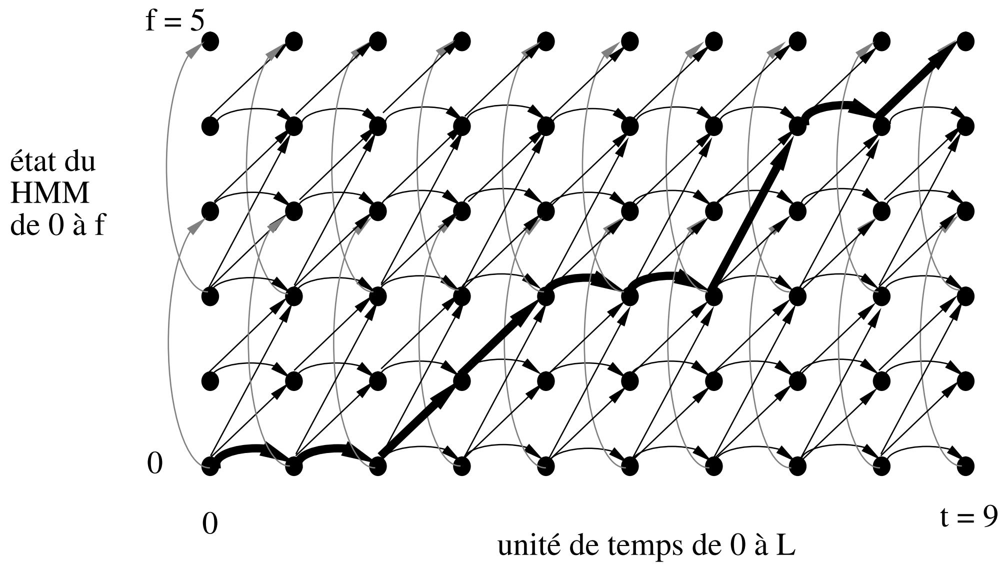


**Figure 3.1  Mémorisation du meilleur chemin dans une matrice de fouille**

est alors possible d'appliquer l'algorithme de Viterbi sur le graphe résultant (voir figure 3.2) et ainsi d'obtenir la séquence de transitions pour laquelle $P(y|HMM_m)$ est maxi-

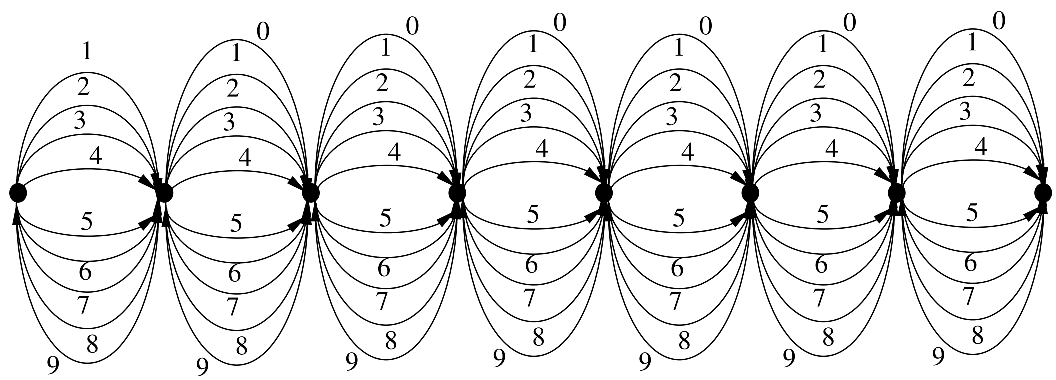

**Figure 3.2  Syntaxe définissant un numéro de téléphone**

male. Comme il est très facile d'associer aux transitions du HMM final, une référence à son HMM d'origine, la séquence de transitions permet de retrouver la séquence de chiffres prononcée .

En prime, l'unité de temps à laquelle chaque transition a été empruntée pouvant, elle aussi, être facilement retrouvée (dans notre exemple de pseudo-code, il s'agit simplement du premier indice de la matrice *mat*), Viterbi permet d'obtenir la segmentation d'un signal, c'est-à-dire, le nombre et l'identité des portions d'un signal attribuées à chaque HMM. Ce type de renseignement sera éventuellement fort utile à l'implantation d'algorithmes plus complexes. Nous en reparlerons entre autres à la section 3.2 et au chapitre 8.

## 3.1 Considérations d'implantation

L'emploi de l'algorithme de Viterbi comme base de la reconnaissance n'a pas que des avantages. Dans les sections qui suivent, nous soulevons quelques-uns des problèmes et des contraintes pratiques que pose son emploi. Des éléments de solutions sont présentés avec, le cas échéant, une référence à des chapitres ultérieurs, les travaux rapportés dans cette thèse ayant justement pour but de solutionner certains de ces problèmes.

### 3.1.1 Construction d'une structure de recherche complexe

Comme on l'a vu à la section suivante, le principe de base de l'algorithme de Viterbi est de trouver au travers d'un graphe représentant l'espace de recherche, le chemin de moindre coût qui correspond à la solution recherchée. Un des premiers problèmes concrets de cette approche réside donc dans le choix de ce graphe ou plutôt de cette structure de recherche comme on préférera l'appeler.

Qu'on se rappelle, que l'algorithme de Viterbi utilise, lors de la reconnaissance, une matrice de grandeur $f \times L$, d'où une demande en mémoire de cet ordre, et que le nombre de calculs effectués lors de la reconnaissance est aussi, entre autres[1], proportionnel à ce nombre. Or, puisque $L$, la longueur de l'énoncé n'est pas vraiment sous contrôle du système de RAP, il ne reste que $f$, le nombre d'états de la structure de recherche qu'on puisse tenter de minimiser, si l'on veut réduire tant l'espace mémoire que le temps d'exécution requis par la reconnaissance[2]. En fait, on touche ici au problème essentiel de l'étape de reconnaissance, soit de réussir à réduire la demande en temps et en mémoire effectivement (par opposition à théoriquement) exigée par l'algorithme de Viterbi, tout en maintenant un taux de reconnaissance acceptable.

1. Il est aussi fonction du nombre de transitions par état, mais ce nombre est au minimum de 1 sauf pour le dernier état.

2. Dans les faits d'un point de vue implantation, seul est nécessaire un vecteur de grandeur $f$ pour effectuer les calculs et un nombre de cases mémoire proportionnel à $L \times M$ pour mémoriser les chemins.

Par exemple, revenons à notre tâche de RAP de numéros de téléphone et compliquons-lui un peu la tâche. Les numéros reconnus ne sont plus nécessairement des séquences de 7 chiffres, mais peuvent aussi être des séquences plus courtes (le 0, le 411, le 911 etc.) ou plus longues (des appels interurbains). On peut évidemment rajouter explicitement ces séquences à la structure illustrée à la figure 3.2, mais cette façon de faire augmente d'autant la valeur de $f$, et par conséquent la mémoire et le temps requis pour effectuer la reconnaissance.

Une meilleure solution pourrait être d'utiliser la structure illustrée à la figure 3.3. On cons-

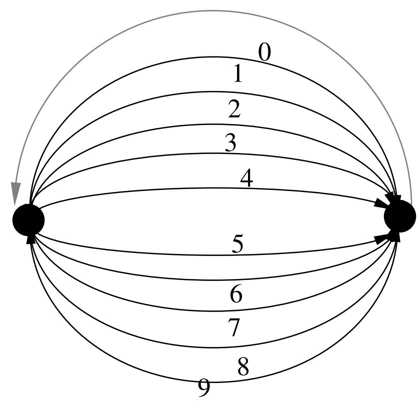


**Figure 3.3 Structure de recherche permettant la reconnaissance de n'importe quelles séquences de chiffres non vides**

tate que cette dernière est nettement plus compacte que celle illustrée à la figure 3.2. On constate aussi qu'elle permet la reconnaissance de plusieurs séquences de chiffres non valides en regard à ce qui a été demandé. On dit d'un telle structure qu'elle est un surgénérateur. Ceci n'est pas nécessairement une mauvaise chose comme on le verra entre autres à la section 3.1.5.

En théorie, c'est-à-dire étant donné des modèles parfaits, le fait qu'une structure soit un surgénérateur, ne devrait avoir aucun impact sur la capacité de l'algorithme de Viterbi à trouver la séquence correspondant à une solution, puisque selon les principes d'entraînement des HMMs, c'est à la solution que devrait correspondre le chemin de coût minimum. Malheureusement, la pratique n'étant pas encore totalement à la hauteur des principes, le choix de la structure de recherche idéale sera toujours un compromis entre une surgénération minimale de l'espace de recherche d'une part, et une occupation mémoire et un temps de fouille minimal d'autre part. Nous reviendrons sur cet important aspect aux chapitres 5 et 7.

### 3.1.2 Utilisation d'un faisceau d'émondage

Une autre façon de réduire le nombre d'opérations effectuées par l'algorithme de Viterbi est l'utilisation de faisceaux d'émondage [LOWE 90]. En effet, puisque l'algorithme de Viterbi ne conserve pour chaque noeud et unité de temps que la meilleure des hypothèses, on gagne lors de son emploi à utiliser une méthode permettant d'éliminer rapidement les hypothèses les moins prometteuses. C'est que ces dernières risquent fort, éventuellement, d'entrer en compétition avec une autre hypothèse plus prometteuse et donc d'être abandonnées à son profit. C'est ce phénomène qu'illustre la figure 3.4. On peut y voir, l'hypo-

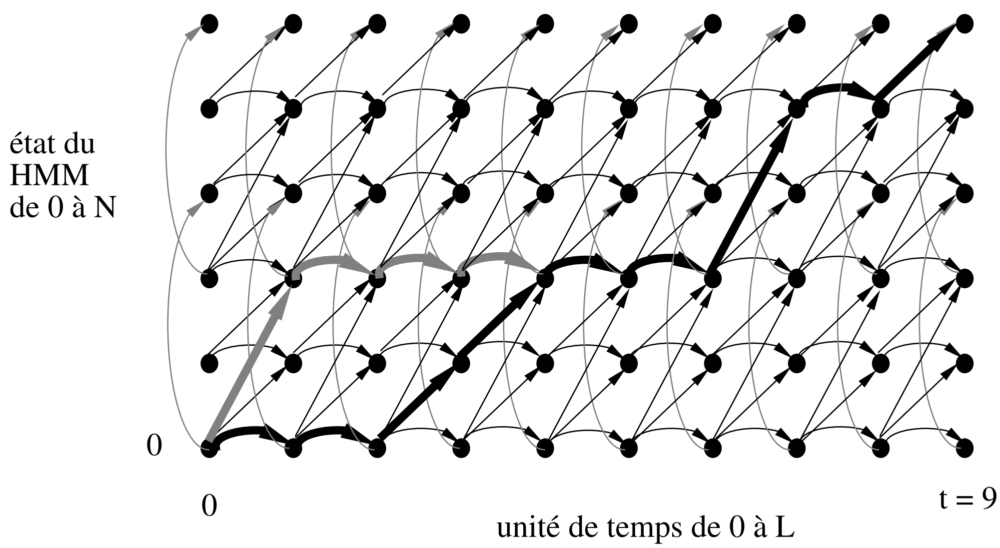


**Figure 3.4 Phénomène d'abandon d'une hypothèse**

thèse correspondant au chemin en gris, entrer en compétition au noeud 3 et au temps 4 avec l'hypothèse correspondant au chemin en noir, et être abandonnée au profit de cette dernière. Tous les calculs associés à l'hypothèse en gris auraient donc pu être évités, si l'on aurait pu prédire cette éventualité.

L'algorithme de Viterbi étant de nature synchrone, ce qui veut dire que toutes les hypothèses actives le sont à une même unité de temps (ou colonne de matrice), la décision d'abandonner une hypothèse ne peut se faire que sur la base de sa probabilité cumulative ou score courant. C'est ainsi que lors de l'utilisation d'un faisceau d'émondage, on coupe, à chaque unité de temps, toutes les hypothèses dont le score est inférieur à un seuil donné. Ce seuil, $se$, est calculé à chaque unité de temps (ou colonne) selon l'équation 3.3. Ainsi pour l'unité de temps $t$ on aura

$$se = \max S_t - \Delta \qquad \text{(EQ 3.3)}$$

où $\Delta$ est choisi expérimentalement et $\max S_t$ est le score maximum des noeuds actifs pour l'unité de temps $t$. A ce temps $t$, seules les hypothèses dont le score cumulatif $S_{hyp}$ est supérieur à $se$, c'est-à-dire dont le score est à l'intérieur du faisceau, seront poursuivies. Lorsqu'incorporé dans le pseudo-code de la page 28, ceci donne (rappel, $t$ = col):

```
int     mat[0..f][0..L+1]       /* colonne = temps, rangée = noeud du HMM */
mat[0][0] = 0
for (col=0; col<L; col++) {
        /* initialiser max S_col à la plus petite valeur possible */

        for (i=0; i<f; i++) {
                if (mat[i][col] > max S_col) {

                        max S_col = mat[i][col]

        }
        for (i=0; i<f; i++) {
                for (toutes les transitions tr_ix en provenance du noeud i) {
                        if (mat[i][col] > max S_col − Δ) {

                                nouv_cout = mat[i][col] + cout(tr_ix)
                                if (tr_ix est vide) {
                                        if (nouv_cout > mat[x][col]) {
                                                mat[x][col] = nouv_cout
                                        }
                                }
                                if (tr_ix est pleine) {
                                        if (nouv_cout > mat[x][col+1]) {
                                                mat[x][col+1] = nouv_cout
                                        }
                                }
                        }
                }
        }
}
/* initialiser max S_col à la plus petite valeur possible */
for (i=0; i<f; i++) {
        if (mat[i][col] > max S_col) {

                max S_col = mat[i][col]

}
for (i=0; i<f; i++) {
        for (toutes les transitions vides tr_ix en provenance du noeud i) {
```

**if** ($mat[i][col] > \max S_{col} - \Delta$) {

$nouv\_cout = mat[i][col] + cout(tr_{ix})$

**if** ($nouv\_cout > mat[x][col]$) {

$mat[x][col] = nouv\_cout$

}

}

}

}

On peut facilement constater que plus $\Delta$ est petit, plus le nombre d'hypothèses conservé sera petit et plus grande sera la réduction du temps de calcul. Malheureusement, rien n'assure que la meilleure solution globale soit toujours la meilleure hypothèse locale. Un plus petit faisceau peut donc entraîner l'abandon de la meilleure solution et ainsi faire diminuer le taux de reconnaissance[1] d'un système. En général, un compromis doit donc être trouvé entre la réduction du temps de calcul et la diminution du taux de reconnaissance. Ceci se fait traditionnellement par essai et erreur sur un corpus d'entraînement.

Il faut souligner que d'un point de vue implantation informatique, il peut être avantageux de regrouper à priori certains noeuds d'un graphe (c'est-à-dire certaines lignes de la matrice de fouille) et d'affecter à ces noeuds, une variable indiquant le statut de l'ensemble. Ceci revient à regrouper en un seul test plusieurs comparaisons avec $\max S_t$. Par exemple, il est intéressant de procéder ainsi pour les noeuds 34 à 40 de la figure 3.5, car

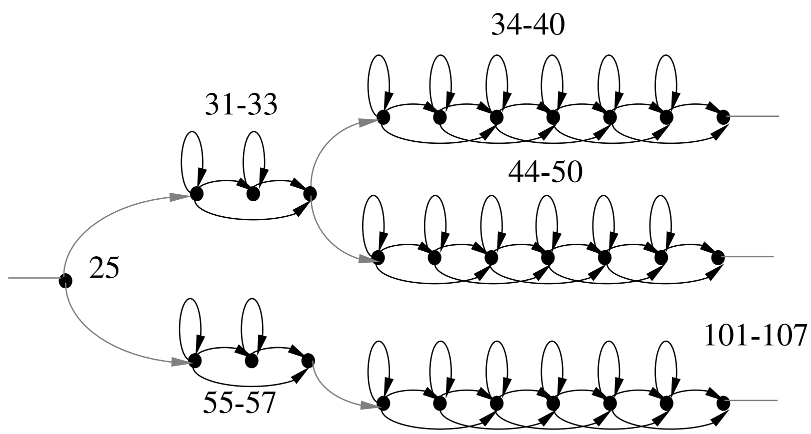


**Figure 3.5 Extrait d'une structure de recherche**

1. Pour plus de détails sur le calcul du taux de reconnaissance d'un système, voir l'annexe B.

seule une hypothèse active en provenance du noeud 33 peut réactiver le noeud 34, lui-même le seul à pouvoir ré-activer les autres noeuds de l'ensemble. On peut évidemment regrouper des regroupements et affecter une autre variable de statut à ce super-regroupement comme par exemple l'ensemble des ensembles de noeuds 31 à 33, 34 à 40, 44 à 50 et 55 à 57, 101 à 107 car seule une hypothèse active en provenance du noeud 25 peut réactiver les premiers noeuds de chaque regroupement soit 55 et 31 et donc ré-activer le super-regroupement. Cette façon de procéder augmente l'efficacité du faisceau sans dommage pour le taux de reconnaissance. La facilité à faire de tels regroupements dépendant de la topologie de la structure de recherche, il est important d'en tenir compte lors du choix de cette dernière.

### 3.1.3 Prise en compte d'un modèle du langage

Comme on l'a souligné, le principal avantage de l'algorithme de Viterbi sur l'algorithme "avant", est, qu'en ayant la possibilité de mémoriser le chemin emprunté, il permette une plus ou moins grande compression de l'espace de recherche. Fort bien, mais on notera que, dans l'équation 2.2, tant la probabilité acoustique $P(y|m)$ que la probabilité du ML, $P(m)$, contribuent au choix de la meilleure solution. Faut-il donc en conclure qu'en pratique l'algorithme de Viterbi ne peut être utilisé que dans les cas où $P(m)$ est constant pour tous les $m$ possibles, comme c'était le cas, par exemple, dans notre petite tâche hypothétique de RAP de numéros de téléphone.

Heureusement la réponse à cette question est non et ce pour deux raisons. La première est l'existence de variantes de l'algorithme de Viterbi qui permettent d'obtenir non seulement le chemin de probabilité minimum, mais un ensemble de chemins de probabilité minimum[1], qu'il suffit alors de reclasser à l'aide du ML. C'est une approche heuristique, rien ne garantie que le $m$ pour lequel $P(y|m)P(m)$ est maximal sera parmi l'ensemble de solutions proposées, mais elle est nettement préférable en terme de temps et espace de calcul à une approche exhaustive, surtout lorsque de grands vocabulaires sont considérés. La deuxième raison pour laquelle l'algorithme de Viterbi est aussi utilisé en présence de MLs, est qu'il est parfois possible, et d'ailleurs toujours souhaitable, d'intégrer la capacité de prédiction de ces derniers au coeur même de l'algorithme.

---

1. Un survol de ces dernières est fait à la section 3.2, et une description plus détaillée de l'une d'elle à la section 8.3.3.

Cette intégration dépend de deux facteurs principaux qui sont d'abord et avant tout le type de ML choisi mais aussi le type d'application de RAP (parole continue ou mot isolé). Dans cette section, nous verrons brièvement le type d'impact que les divers MLs présentés à la section 2.6 peuvent avoir sur la topologie de la structure de recherche, laissant au section 4.2 et chapitre 7, le soin d'en préciser les détails dans le cas respectivement de la parole en mots isolés et continue.

Les trois principaux types de MLs sont, on se rappelle, les modèles à syntaxe fixe, les modèles probabilistes locaux et les modèles à grammaires probabilistes. Les modèles les plus facile à intégrer dans une structure de recherche sont, sans conteste, les modèles à syntaxe fixe. En effet, il s'agit tout simplement, tel que décrit à la section 3.1.1, de construire le HMM correspondant au graphe syntaxique. Evidemment, cette façon de faire peut engendrer des HMMs assez volumineux, raison de plus de restreindre ce type de MLs à des tâches où la syntaxe et le vocabulaire sont restreints.

L'intégration des modèles probabilistes locaux est plus ou moins complexe selon la valeur de $n$ choisi. Par exemple, dans le cas d'un unigramme, il suffit d'ajouter une transition pour laquelle $q_{tr} = P(m_i)$ menant au mot $m_i$ pour qu'automatiquement soit calculé de façon incrémentielle (c'est-à-dire ici mot par mot) la probabilité $P(y|m)P(m)$ d'un message $m$. Par contre dans le cas d'un bigramme, les choses se compliquent un peu. En effet, dans ce cas la probabilité d'un mot n'est plus fonction du mot seulement, mais aussi de celui qui le précède. Or, il faut se rappeler que, dans l'algorithme de Viterbi, lorsque deux hypothèses passent par un noeud commun, seule celle ayant le meilleur score est conservée (propriété de Markov du premier ordre). Ainsi à la figure 3.6, il est impossible d'inté-

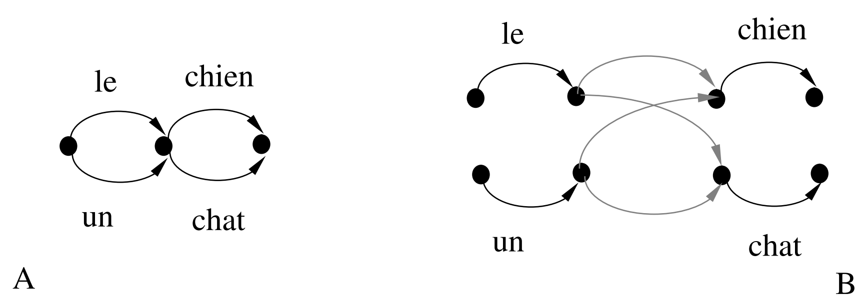


**Figure 3.6 Intégration d'un modèle de bigramme à une structure de recherche**

grer un bigramme à la topologie illustrée en A, mais très facile de le faire pour celle illustrée en B. C'est que dans la topologie illustrée en B, les hypothèses en provenance des deux mots *le* et *un* sont conservées séparées jusqu'au moment où l'application du bigramme est possible. On peut extrapoler à partir de cet exemple, et en conclure que l'intégration d'un trigramme sera d'autant plus complexe que son application ne peut se faire que lorsque les deux mots précédents sont connus et ainsi de suite pour des MLs probabilistes locaux à valeur de $n$ plus grand que trois.

Par contre, bien qu'ils posent certains problèmes d'intégration, les modèles probabilistes locaux ont au moins l'avantage d'être appliqués de façon locale et strictement gauche-droite, deux des caractéristiques importantes de l'algorithme de Viterbi, d'où l'existence de solutions relativement acceptables. Ce n'est malheureusement pas le cas des modèles à grammaire probabilistes, qu'on doit donc se contenter le plus souvent, d'appliquer en post-traitement.

### 3.1.4 Utilisation des triphones en contexte entre les mots

L'intégration de MLs probabilistes locaux n'est pas le seul endroit où la propriété de Markov du premier ordre pose un problème pratique d'implantation. C'est aussi le cas au niveau acoustique lors de l'emploi de modèles de triphones inter-mots (et autres HMMs basés sur le contexte). En effet, s'il est très facile, à priori, de déterminer la séquence de triphones associés aux phonèmes internes d'un mot, il en va tout autrement pour les phonèmes situés aux extrémités des mots. Revenons, par exemple, à la structure de recherche en boucle illustrée à la figure 3.3, puisque tous les mots sont connectés entre eux, le contexte droit (gauche) des phonèmes en bout de mots inclut tous les phonèmes débutant (terminant) tous les mots.

Ainsi, la figure 3.7 illustre la topologie que doivent avoir les HMMs complexes correspondant à des mots de 1, 2 ou plus de trois phonèmes, par exemple: *un* - /*un*/, *deux* - /*d eu*/ et *trois* - /*t r oi*/ dans une telle structure bouclée. Evidemment, cet exemple présente un cas extrême, et il est probable qu'en réalité certains contextes puissent être regroupés, par exemple, le *t* et le *d* et le *eu* et le *un.* Il n'en demeure pas moins, que l'emploi de modèles inter-mots a comme résultat d'accroître considérablement le volume de la structure de recherche.

Qui plus est, pour maximiser leur capacité de modélisation acoustique, seuls doivent être connectés entre eux, les HMMs correspondant à des triphones compatibles, c'est-à-dire obtenus dans un contexte de continuité. Par exemple, le HMM associé au triphone *oi=un=d* ne peut être connecté à droite qu'aux HMMs associés aux triphones *un=d=**x*** où

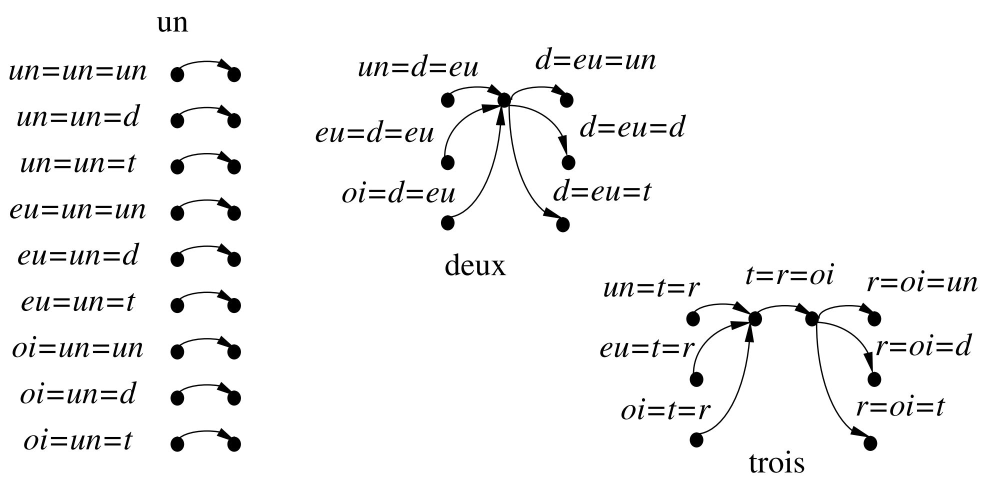


**Figure 3.7  Topologie type de HMMs de mots composés à partir de triphones**

***x*** est un phone quelconque (ici *un=d=eu*) et non à ceux associés à d'autres triphones ***z****=d=****x*** ou *un=****y****=****x*** où ***z*** est différent de *un* (par exemple *oi=d=eu*) et ***y*** différent de *d* (par exemple *un=t=r*). De même, à gauche *oi=un=d* ne peut être connecté qu'aux triphones du genre ***x****=oi=un* (ici *r=oi=un*) et non aux triphones du genre ***x****=oi=****z*** ou ***x****=****w****=un* où **w** est différent de *oi* (par exemple *d=eu=un*). On imagine assez facilement la lourdeur et la complexité que ce genre de restrictions apportent à la topologie des structures de recherche incluant des modèles inter-mots. C'est pourquoi on restreint habituellement l'emploi de ce type de modèles à des structures de recherche de type graphe, où le facteur de branchement (le nombre de transitions arrivant/partant d'un noeud) est beaucoup plus faible que dans une structure bouclée.

Néanmoins, même pour de telles structures, l'emploi de modèles en contexte présente certaines difficultés. Par exemple, la présence de transitions vides signifie qu'on doive tenir compte dans certains cas, de contextes associés non seulement aux transitions partant directement d'un noeud *x* donné, mais aussi de ceux associés aux transitions partant des noeuds liés au noeud *x* par une chaîne de transitions vides. C'est ainsi que dans l'exemple illustré à la figure 3.8, le phone *d* de *dans* - */d an/* doit en plus du phone *r* de *lumières* - */l u m i è r/*, tenir compte du contexte que lui impose les phones *t* de *vite* - */v i t/* et *m* de *ferme /f è r m/*.

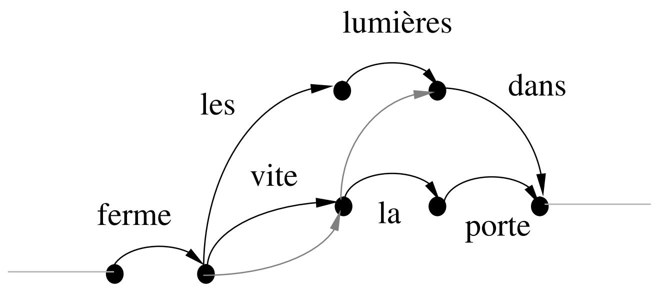


**Figure 3.8 Exemple de contextes éloignés**

### 3.1.5 Le problème du rejet

Un algorithme de reconnaissance capable de rejet est un algorithme capable d'affirmer au sujet d'un énoncé quelconque qu'il ne correspond pas, en tout ou en partie, aux diverses alternatives possibles pour une tâche donnée. Le problème du rejet n'est pas un problème propre à l'algorithme de Viterbi mais plutôt à l'approche stochastique en général [RAHI 95]. En effet, si en théorie il est facile d'affirmer que la reconnaissance se résume au calcul des deux termes $P(y|m)$ et $P(m)$ d'un énoncé, dans les faits, cela n'est pas suffisant. La raison en est non pas la difficulté d'énumérer tous les $m$ possibles mais plutôt la difficulté de contraindre un usager (les humains étant de grands bavards pas toujours disciplinés) à n'émettre que des messages valides. Ainsi si lors de notre tâche de reconnaissance de numéros de téléphone, il est facile de lister tous les numéros possibles, il est plus difficile de prévoir si un usager, étant donné l'existence d'un interface chargé, par exemple, de déterminer le numéro que ce dernier désire composer, ne décidera pas tout-à-coup de questionner le système sur tout ou rien...Même dans le cas d'usagers très disciplinés, ces derniers peuvent devoir s'éclaircir la gorge, éternuer, avoir oublié la bonne syntaxe, se reprendre, remplacer un mot correct par un synonyme valide, échapper leur micro, bref, émettre un son inconnu et imprévu pour et par le système.

C'est la tâche du rejet d'identifier les énoncés où de tels événements se sont produits, soit en déclarant l'énoncé incompréhensible ou mieux, lorsque c'est possible, en récupérant au travers de ces “bruits” un énoncé valide. Une méthode originale de rejet global sera présentée à la section 8.3.3.

## 3.2 Variantes sur un thème

Comme on l'a souligné à la section 3.1.3, il peut parfois s'avérer nécessaire pour, par exemple, pouvoir appliquer certains types de MLs plus sophistiqués, que l'algorithme de

Viterbi puisse générer plus d'une solution. Ces dernières années, plusieurs variantes de l'algorithme de Viterbi ont donc vu le jour, qui permettent cette génération. On réfère à ces derniers par le terme générique d'algorithmes "N-meilleurs[1]", car ils permettent de générer les $N$ "meilleures" solutions pour un énoncé donné. Ces algorithmes se divisent en deux grandes catégories selon l'interprétation qu'ils font du mot "meilleures". On dira d'un algorithme N-meilleurs qu'il est exact, si théoriquement, les solutions qu'il permet d'obtenir sont bien celles correspondant aux $N$ chemins de moindre coût d'une structure de recherche, et on dira qu'il est non-exact ou heuristique, si même théoriquement, il n'est pas garantie que les solutions obtenues correspondent bien aux $N$ chemins de moindre coût d'une structure de recherche.

L'algorithme N-meilleurs exact le plus connu est le "tree-trellis N-best" par Soong et Huang [SOON 90]. Il est la référence dans le domaine, car il demande relativement beaucoup moins de temps d'exécution que les autres algorithmes N-meilleurs exacts décrits dans la littérature, par exemple [SCHW 90]. Les algorithmes de type heuristiques sont plus nombreux, deux des plus connus sont décrits dans [SCHW 91]. Il s'agit du "lattice N-best", un excellent exemple d'algorithme "N-meilleurs" heuristique où l'heuristique choisi est trop simplificateur et les résultats obtenus plutôt loin de la réalité et du "word-dependent N-best", l'archétype de l'algorithme "N-meilleurs" heuristique. D'ailleurs, il est très souvent fait référence à l'hypothèse de base de cet heuristique qui est que pour une solution donnée $m = m_1, \ldots, m_L$, la frontière entre $m_{i-1}$ et $m_i$, n'est influencée que par l'identité de $m_{i-2}$ et non par les autres mots précédant $m_{i-2}$. [STEI 91] présente un bon tableau résumé des algorithmes "N-meilleurs" à cette date.

Depuis plusieurs autres algorithmes "N-meilleurs" ont vu le jour, tous plus ou moins des variantes de ceux mentionnés plus hauts. Ils sont cependant de plus en plus utilisés lors de fouille, au moyen d'un algorithme A*, de treillis produits par des systèmes multi-passes. Une version personnelle d'un tel algorithme sera présenté en détail au chapitre 8, des références plus récentes sur les nouvelles versions d'algorithmes "N-meilleurs" au chapitre 7.

1. N-best algorithms

# Première partie

Les travaux rapportés dans cette partie datent du début des années 90. Ce qui peut sembler hier dans certains domaines de recherche est quasi de l'histoire ancienne dans le domaine de la RAP. En effet, cette dernière a eu, dans les dernières années, une progression foudroyante, en particulier grâce aux récentes tâches ARPA (nous en reparlerons au chapitre 8.2) et aussi, il faut l'avouer, l'avènement de machines toujours plus puissantes. C'est ainsi que la recherche sur la parole en mots isolés est maintenant devenue marginale (ou alors plus secrète car effectuée dans le but d'améliorer des systèmes commerciaux), toute la place étant occupée par des systèmes à parole continue à moyens ou grands vocabulaires.

Il ne faut donc pas s'étonner du peu de références récentes dans ce chapitre et comprendre que l'approche originale présentée au chapitre suivant, l'est en regard des systèmes en mots isolés de l'époque. Depuis, les efforts portent plutôt sur la parole continue, qui constitue le sujet de la deuxième partie de cette thèse.

## 4 La reconnaissance en mots isolés

Lors d'une tâche de reconnaissance en mots isolés, on demande au locuteur de faire une petite pause entre chaque mot. Le signal se présente alors sous la forme illustrée à la figure 4.1. Ce signal correspond à la phrase "Les anges mangent leur poulet avec les ailes". Or puisqu'il existe des algorithmes relativement efficaces qui permettent de distinguer silence et parole, le problème de la reconnaissance en mots isolés, consiste simplement à faire correspondre un mot du vocabulaire à chaque îlot de parole. Néanmoins, pour de très grands vocabulaires, il n'est pas réaliste d'accomplir cette tâche de façon exhaustive, c'est-à-dire en appliquant les algorithmes "avant" ou de Viterbi à chaque mot du vocabulaire, d'où une variété d'approches qui tentent de réduire l'espace de recherche et, par conséquent, le temps d'exécution de la reconnaissance.

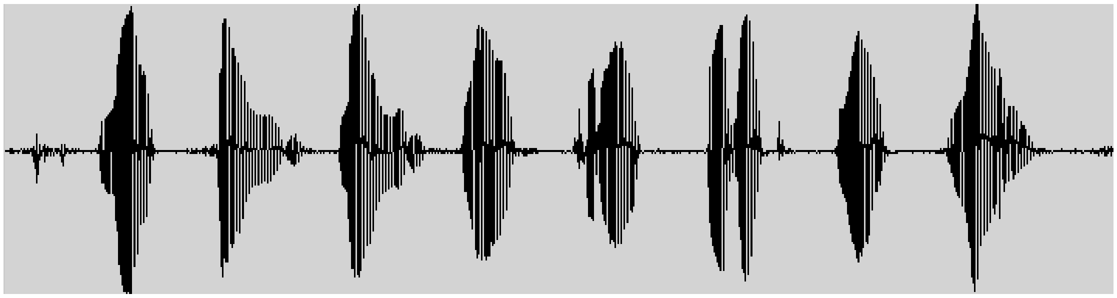

**Figure 4.1 Exemple d'un signal acoustique résultant d'une dictée en mots isolés.**

## 4.1 Description de systèmes de reconnaissance en mots isolés et grand vocabulaire

Dans les prochaines sections, nous examinerons en détails les approches utilisées dans plusieurs systèmes de reconnaissance en mots isolés et grands vocabulaires. Comme nous l'avons souligné la plupart de ces systèmes ne font plus l'objet de recherches actives. Par contre, les idées qu'ils ont permis de développer sont à la base de certaines applications expérimentales ou commerciales. On peut entre autres citer Dragon Dictate (voir section 4.1.5) et StockTalk [LENN 92]). La grande majorité ont été développés pour l'anglais.

### 4.1.1 Système du CSELT

Le système du CSELT (Centro Studi e Laboratori Telecomunicazioni) a été développé pour l'italien ([LAFA 87], [LAFA 88], [FISS 88] et surtout [FISS 89]). Son lexique est d'environ 8000 entrées. La reconnaissance se compose de deux étapes: un pairage rapide[1] et un pairage exact[2]. Lors de la première étape, le signal est étiqueté en terme de traits acoustiques généraux: silence, fricative, voyelle haute etc. Par la suite, lors de la deuxième étape, l'algorithme de Viterbi n'est appliqué qu'aux seuls mots dont la séquence phonétique correspond (plus ou moins exactement c'est-à-dire selon un algorithme d'alignement dynamique) à la séquence de traits attribuée au signal. Ces ensembles de mots auxquels correspond une même séquence de traits sont connus sous le nom de cohortes. Puisque les traits acoustiques généraux sont fixes et connus d'avance, ces cohortes peuvent être précalculées, ce qui rend très rapide la première partie de l'algorithme.

1. fast match
2. exact match

Par contre, il est aussi très important que le moins d'erreurs possibles ne soient introduit lors de cette première étape, c'est-à-dire que le moins de mots possible auxquels l'algorithme de Viterbi aurait attribué le meilleur score, ne soient omis de la liste de mots présélectionnés. Or, si plus les traits acoustiques sont précis, plus les cohortes extraites sont petites, plus malheureusement on peut confondre deux traits et extraire une mauvaise cohorte, d'où la nécessité de bien choisir ces traits (voir à ce sujet [HUTT 84], [LAGG 85] et [VERN 89]). Par exemple, si l'on choisit de distinguer les plosives voisées que sont *b*, *d*, et *g* des plosives correspondantes non voisées *p*, *t*, *k*, on peut alors distinguer les paires de mots *bedon/peton*, *dans/tant* et *gué/quai*, qui si cette distinction n'est pas faite auraient été incluses dans une même cohorte. Le voisement étant un trait relativement facile à détecter de façon fiable, cette distinction est un bon choix. Par contre, la distinction entre elles des diverses plosives voisées étant d'un point de vue acoustique plus ardue, il est préférable de ne pas faire cette distinction, ce qui aura conséquence d'inclure des mots tels *beau, dos* et *go* dans une même cohorte.

On doit souligner que selon [CARL 86] et [ADDA 86], cette approche n'est pas aussi efficace pour le français qu'elle l'est pour d'autres langues tel l'anglais. C'est-à-dire que pour un même nombre de traits, le nombre de cohortes distinctes est plus petit et donc le nombre de mots par cohorte plus grand.

Le groupe de recherche du CSELT est l'un de ceux qui poursuit activement des recherches sur la parole en mots isolés. Ainsi, le dernier article paru porte sur une nouvelle méthode de pairage rapide potentiellement plus facile à adapter à la parole continue [LAFA 95].

### 4.1.2 TANGORA

Le système TANGORA a été élaboré par un groupe le Speech Recognition Group du IBM Thomas J. Watson Research Center ([AVER 87], [BAHL 88a] et [BAHL 89]). Son vocabulaire se compose d'environ 20 000 formes anglaises. Une forme est une instance phonétique distincte d'un mot. Cela veut dire qu'un mot ayant deux prononciations distinctes compte pour deux entrées. Tout comme dans le système du CSELT, la reconnaissance dans TANGORA comporte deux étapes: un pairage rapide et un pairage exact [AVER 86].

La fonction du pairage rapide est ici encore de fournir au pairage exact un sous-ensemble des mots les plus prometteurs. La technique employée est cependant totalement différente. Il s'agit d'une méthode dans laquelle un vote ou valeur est attribué à chacun des mots du lexique par l'ensemble des vecteurs d'observation $y = y_1, y_2, \ldots, y_L$ d'un signal acoustique $m$ [BAHL 88a]. Ce vote est calculé par une équation de la forme:

$$S_m = \sum_{i=1}^{L} v(y_i, m) + cte_m \qquad \text{(EQ 4.1)}$$

où $L$ représente le nombre d'unités de temps de l'énoncé, $cte_m$ une valeur d'initialisation pour chaque mot et $v$ la fonction de votation. Il est à noter que la fonction $v$ n'étant pas fonction du temps mais seulement des valeurs des paramètres des vecteurs $y_t$, la valeur octroyée à deux mots formés des mêmes phonèmes mais pas nécessairement dans le même ordre doit être la même. Cela revient à dire que c'est l'identité des phonèmes d'un mot et non leur ordre qui est surtout importante dans l'identification de ce mot. C'est d'ailleurs aussi à cette conclusion qu'en arrive pour l'anglais [MARC 85]. Pour ma part, un rapide calcul m'a permise de constater une moyenne de 1.68 mots par groupe de phonèmes pour le lexique LF1 de 17 000 entrées décrit à l'annexe A.

Les mots ayant obtenu un vote supérieur à une valeur donnée se qualifient pour l'étape du pairage exacte. Typiquement on fixe cette valeur de façon à ce que entre 20 et 25 mots soient retenus. Par la suite, on rajoute à cette liste de candidats au pairage exacte, un ensemble de mots “similaires” d'un point de vue acoustique aux mots initialement sélectionnées (un type de cohorte d'en moyenne 100 mots) [BAHL 90]. C'est cette liste allongée (typiquement d'environ 2000 mots) qui est considérée lors du pairage exacte consistant en l'algorithme “avant”.

Les articles récents par IBM portent plutôt sur des systèmes à moyens et grands vocabulaires.

### 4.1.3 Système de l'INRS

Le système développé par l'INRS-Télécommunication (Institut national de recherche scientifique, Québec, Canada) pour l'anglais, est basé sur un vocabulaire de 75 000 entrées [GUPT 88]. Il se compose, non pas de deux, mais de trois étapes dont une première étape d'estimation du nombre de syllabes, une deuxième étape de production de scores heuristiques et une troisième étape consistant en un pairage exact (algorithme “avant”) effectué non plus de manière synchrone mais plutôt de manière asynchrone à l'aide d'un algorithme de type A* [NILS 82].

C'est Jelinek ([JELI 69] et [JELI 76]) qui le premier a appliqué l'algorithme A*, une technique d'intelligence artificielle, en reconnaissance de la parole. Dans l'algorithme A*, les hypothèses les plus prometteuses sont conservées dans une pile. Le degré de promesse d'une hypothèse est représenté par une valeur numérique appelée score total. Cette valeur

$E$ est égale à la somme du score courant de l'hypothèse et d'une valeur heuristique représentant un estimé du score associé au meilleur chemin possible entre l'état courant et un état final. Si pour toutes les hypothèses partielles $sol$ on a $E(sol) \geq S(sol)$ et que pour toutes les solutions $sol$, on a $E(sol) = S(sol)$ où $S(sol)$ est le score de la solution, c'est-à-dire que l'heuristique sur-estime toujours le potentiel d'un chemin, l'algorithme A* est admissible et donnera toujours la meilleure solution.

Le résultat de la deuxième étape du système de l'INRS ne consiste donc pas en une liste de mots possibles comme c'était le cas pour les systèmes précédents mais plutôt en des scores heuristiques admissibles devant servir de guide à la fouille exacte. Ces scores sont obtenus en parcourant un graphe syllabique au moyen de l'algorithme de Viterbi. Ce graphe ne contenant pas de boucle, il en existe un pour les mots de une syllabe, un pour ceux de deux syllabes, etc. Chaque graphe est constitué par l'ensemble des syllabes occupant cette position dans un mot du lexique. Par exemple, le graphe pour les mots de trois syllabes est composé de trois sous-graphes, l'un incluant toutes les premières syllabes de tous les mots du lexiques, un autre incluant toutes les deuxièmes syllabes de tous les mots du lexique et un dernier incluant toutes les troisièmes syllabes de tous les mots du lexique.

Le choix du graphe est fonction du nombre de syllabes de l'énoncé, nombre estimé, lors de la première étape, par l'application de l'algorithme de Viterbi sur un structure composée de HMMs génériques capables de détecter les voyelles des consonnes. Les scores heuristiques obtenus au moyen des graphes syllabiques sont ensuite utilisés pour guider l'application de l'algorithme “avant” à une structure d'arbre représentant de façon compacte tous les mots du lexique (voir l'exemple de la figure 5.1).

Plus récemment, les travaux de l'INRS ont porté sur les systèmes de RAP en parole continue (voir section 7.3.11) et de concert avec Bell Northern Research (BNR) sur le développement de produits [LENN 92].

### 4.1.4 PARSYFAL

PARSYFAL ([DERO 86], [DERO 89] et [MERI 87]) était, à l'époque, l'un des rares systèmes de RAP pour le français[1]. Avant de le présenter en détails, soulignons d'abord quelques particularités du français qui rendent sa reconnaissance plus complexe à traiter et,

---

1. Depuis, certains systèmes originalement développés pour l'anglais on été adaptés au français, c'est le cas de Dragon Dictate [BROU 92]. De plus, de nouveaux résultats portant sur la parole continue ont été publiés (voir section 7.3.6).

dans le cadre de cette partie du document, la dictée en mots isolés moins naturelle que pour l'anglais. Il y a d'abord l'existence de nombreuses élisions, soit la suppression de la voyelle finale de certains mots devant un mot débutant par une voyelle ou un h muet, et de liaisons, soit la prononciation d'une consonne normalement muette à la fin d'un mot devant un mot débutant par une voyelle ou un h muet. Ces deux phénomènes tendent, en effet, à lier dans une même prononciation deux mots qu'un système de RAP en mots isolés voudrait séparer. Ensuite, il y a comme on l'a déjà mentionné à la section 2.6.3, le fait que, pour obtenir une même couverture de texte, un lexique français doive contenir un nombre appréciable d'entrées de plus qu'un lexique anglais. Ceci est dû, entre autre, au fait qu'en français, la plupart des mots, en particulier les verbes, ont un nombre d'inflexions phonétiques beaucoup plus grand qu'en anglais. Par exemple, à la phonie anglaise */t a k/* (*talk*) correspond les phonies françaises */p a r l/* (*parle*, *parles*), */p a r l é/* (*parlez*, *parler*), */p a r l on/* (*parlons*), l'ajout de la marque anglaise du future */w i l/* (*will*), force l'ajout des phonies */p a r l e r é/* (*parlerai, parlerez*), */p a r l e r a/* (*parlera*), */p a r l e r on/* (*parlerons*, *parleront*) et ainsi de suite.

Pour contourner ces problèmes, la dictée dans PARSYFAL se fait syllabe par syllabe. Ce faisant, le nombre d'entrées effectivement distinctes, n'est plus de 200 000 mots mais bien de 6400 syllabes liaisons incluses. En effet, avec cette approche, le problème des liaisons et élisions est réglé car les syllabes supplémentaires dues à la présence de ces dernières seront déjà incluses dans l'ensembles des syllabes étant probablement présentes à l'intérieur des mots. Ainsi, la syllabe *pa* que la liaison dans *trop ami* nous forcerait à rajouter est déjà présente à cause de son occurrence dans le mot *patate*. Cela n'aurait pas été le cas, si c'est le mot *pami* qui aurait dû être rajouté.

Malheureusement, cette approche fait que PARSYFAL n'est qu'à moitié un système de reconnaissance à grand vocabulaire. Il serait plutôt un système à moyen vocabulaire (avec en prime des "mots" très courts), qui de plus, oblige les énoncés à être présentées sous la forme de syllabes isolés, ce qui tant qu'à moi est encore moins naturelle que la parole isolée.

Devant l'adaptation réussie pour le français de "vrais" systèmes de RAP à grand vocabulaire, la recherche sur la parole en syllabes isolées a été abandonnée et de ce fait, les algorithmes et structures qui y étaient dédiés.

### 4.1.5 Dragon Dictate

Dragon Dictate est actuellement le plus connu des systèmes de dictée à grand vocabulaire commercial [BAMB 91]. C'est un produit de Dragon Systems Inc. compagnie basée aux Etats-Unis. Pour des raisons assez évidentes, son fonctionnement n'est pas divulgué en détails. Il est basé sur un algorithme de reconnaissance à deux passes, la première un pairage rapide fournissant une liste de mots, à la seconde un pairage exact [BAKE 74]. Caractéristique dû au fait que l'algorithme est utilisé au sein d'une application de dictée, le pairage exact ne produit pas une seule solution mais plutôt une liste ordonnée des meilleures solutions parmi lesquelles l'utilisateur peut, en cas d'erreur, choisir une alternative.

Il faut souligner cependant que Dragon Dictate n'est pas un système indépendant du locuteur. C'est un système de type adaptatif, c'est-à-dire qu'un ré-entraînement partiel des HMMs a lieu de temps en temps à partir d'énoncés tirés d'un locuteur particulier. C'est là un compromis effectué dans le but d'obtenir des taux de reconnaissance acceptable en un temps acceptable. En effet, le temps est un facteur très important pour une application commerciale. Or, les variations pour un même énoncé et un même locuteur étant en général moindre qu'au travers plusieurs locuteurs, le pouvoir discriminant des HMMs obtenus par ré-entraînement sera plus grand. Ce faisant, le pairage sera plus exact et certaines méthodes d'accélération de la fouille, comme l'emploi de plus petites cohortes ou de plus petits faisceaux d'émondage pourront être utilisés sans conséquence au niveau du taux de reconnaissance.

Depuis le début des années 90, c'est la parole continue qui est le sujet de publications chez Dragon (voir section 7.3.8).

## 4.2 Modèle du langage

Dans tous les systèmes décrits plus haut, tout comme dans l'approche originale qui sera décrite au chapitre suivant, l'accent est mis sur la reconnaissance phonétique, c'est-à-dire, si l'on se réfère à l'équation 2.2, à l'évaluation du terme $P(y|m)$ . Peu, ou aucune mention n'est faite de l'évaluation de $P(m)$ , la probabilité à priori du mot, aussi appelée modèle du langage. C'est que dans le cas de la reconnaissance en mots isolés, on peut plus facilement se permettre de dissocier l'application des modèles acoustiques et du modèle du langage. Ainsi par exemple, le modèle acoustique peut proposer une série de mots pour chaque énoncé, d'où est tirée la meilleure séquence selon le modèle du langage. Ou inversement, le modèle du langage peut être utilisé de façon à restreindre, à priori, la liste des mots parmi lesquels effectuer la fouille acoustique. Il est néanmoins apparu dans des articles plus récents [FISS 93], qu'un couplage plus serré, du type pratiqué pour les systèmes en parole continue est aussi souhaitable pour les systèmes en mots isolés.

# 5 Représentation du Lexique - Graphe compressé

Comme nous l'avons vu au chapitre précédent, les algorithmes utilisés pour la reconnaissance en mots isolés se divisent généralement en deux étapes: la première étape ayant comme tâche de restreindre l'espace de recherche sur lequel est effectuée, lors de la deuxième étape, une fouille plus exhaustive. Or, dans tous les systèmes décrits, cette restriction se fait en extrayant du lexique original un sous-ensemble des mots les plus prometteurs.

Il existe cependant une autre façon de restreindre l'espace de recherche. Au lieu de chercher à diminuer le nombre de mots faisant l'effet d'une fouille exhaustive, on peut chercher à réduire la représentation de l'ensemble de ces mots. C'est ce que fait, jusqu'à un certain point, l'équipe de l'INRS avec leur graphe syllabique et leur lexique sous forme d'arbre. Néanmoins, aucune de ces deux structures n'exploite cette idée de façon optimale.

Ainsi, les graphes syllabiques employés par l'INRS sont définitivement une représentation compacte du lexique original. En fait, c'est une représentation plus que compacte, car il existe la possibilité de générer des mots hors vocabulaire, de même en fait, que des mots tout-à-fait illégaux puisqu'aucune mesure n'est prise au niveau du graphe pour s'assurer de la validité des séquences. Une syllabe peut être la première syllabe valide d'un mot et la syllabe qui suit, la deuxième syllable valide d'un autre mot sans que leur juxtaposition forme un mot valide. Il faut donc que soient conservés, sous une forme quelconque, les mots faisant effectivement partie du lexique (ce qui est fait sous forme d'un arbre) de sorte à valider les séquences de syllabes produites. Ceci introduit un délai qui contrecarre, jusqu'à un certain point, le gain de temps réalisé lors de la fouille initiale. C'est ce compromis entre compacité et sur-génération qui nous fait dire que la représentation sous forme de graphes syllabiques n'est pas maximale.

De même, l'arbre utilisé par l'INRS dans sa phase de vérification n'est pas non plus la structure de recherche la plus optimale. Elle l'est dans le sens qu'elle n'entraîne aucune sur-génération, par contre, ce n'est pas la structure la plus compacte possible pouvant être utilisée pour représenter un lexique. En effet, bien que la représentation sous forme d'arbre permette l'amalgame des parties communes en début de mots, les parties communes en fin de mots sont répétées aussi souvent que lorsque les mots sont considérés isolément (voir le tableau 5.1 et la figure 5.1). Ceci entraîne à la fois un gaspillage de mémoire et un

**Tableau 5.1 Liste des mots d'un mini lexique**

| **#** | **graphie** | **phonétique** | **#** | **graphie** | **phonétique** |
|---|---|---|---|---|---|
| 1 | papa | /p a p a/ | 6 | rampa | /r an p a/ |
| 2 | patate | /p a t a t e/ | 7 | rampant | /r an p an/ |
| 3 | patte | /p a t e/ | 8 | rang | /r an/ |
| 4 | parent | /p a r an/ | 9 | paon | /p an/ |
| 5 | parra | /p a r a/ | | | |

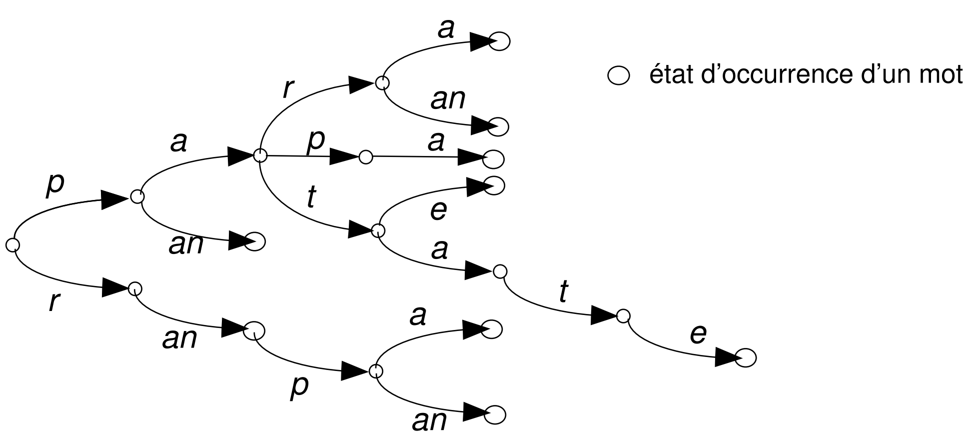


**Figure 5.1 Lexique de la table 5.1 structuré sous forme d'arbre**

gaspillage de temps passé à explorer ces états supplémentaires. Or, on peut supposer que les parties communes en fin de mots sont bien aussi nombreuses que les parties communes en début de mots. En fait, pour certaines langues à haut taux de déclinaisons, comme le français, ces parties peuvent même être plus nombreuses.

Il semblait donc prometteur de tenter de tirer partie de ces suffixes communs, le but poursuivi étant d'obtenir une réduction en espace comparable à celle obtenue lors de l'emploi d'un arbre à la place d'une liste. Réduction, qui à son tour, devait permette un gain similaire au niveau du temps de reconnaissance de sorte à éventuellement rendre caduque l'étape de pairage rapide ou l'utilisation d'un graphe surgénérateur. C'est cette approche originale qu'explore le reste de ce chapitre.

On constatera à la lecture de ce dernier que les espoirs de réduction se sont avérés à moitié bien fondés[1]. En fait, les résultats obtenus au niveau compression sont similaires à ceux obtenus lors de l'application de cette méthode à des domaines connexes [APPE 88][BROW 90] et le gain en temps a rendu possible l'emploi d'un graphe compressé au sein d'applications devant rouler en temps réel [DYME 94]. Finalement, avantage supplémentaire, le graphe représentant l'ensemble de l'espace de recherche, on peut y appliquer directement l'algorithme de Viterbi, et donc utiliser directement des méthodes d'accélération tels des faisceaux d'émondage soit classiques soit plus sophistiqués et performants (voir la section 3.1.2 et le chapitre 9).

## 5.1 Algorithme de compression

L'algorithme de compression utilisé comporte deux étapes. La première consiste à amalgamer au maximum les sous-arbres communs d'un arbre, arbre obtenu à partir des mots du lexique et la deuxième, à minimiser le nombre de transitions pleines de ce graphe. Le résultat de la première étape est le graphe minimal pour le lexique initial, c'est-à-dire le graphe possédant le moins de transitions possibles et dans lequel chaque chemin possible correspond à un mot du lexique. Le résultat de la deuxième étape est un graphe équivalent au premier (contenant exactement les mêmes chemins) mais où le nombre de transitions pleines est réduit au profit des transitions vides.

### 5.1.1 Première étape

Lors de la première étape de l'algorithme de compression, on construit d'abord un arbre à partir de la liste des prononciations phonétiques des mots du lexique. On procède ensuite à la fusion des sous-arbres similaires de l'arbre principal, des plus profonds aux moins profonds, pour obtenir le graphe du lexique. Cette première étape peut s'exprimer sous forme de pseudo-code par:

```
int     poss_compress[0..M][0..M]   /* initialisé à INCONNU */
construire_un_arbre_phonetique()
faire_une_liste_des_feuilles()
for (i=profondeur; i>0; i--) {
```

1. voir aussi [LACO 91]

```
for (toutes les paires de feuilles (j,k)) {
        if (poss_compress[j][k] != INCONNU) {
                if (poss_compress[j][k] == i) {
                        compresser_arbre()
                }
        }
        else {
                comparer_arbre (i, j, k, prof)
                if (prof == i) {
                        compresser_arbre()
                }
                else {
                        poss_compress[j][k] = prof
                }
        }
}
}
```

où la fonction *compresser_arbre* a comme conséquence de réduire la liste des feuilles, et *comparer_arbre* compare les sous-arbres auxquels appartiennent *j* et *k*, et ce jusqu'au minimum des profondeurs *i* (arbre similaire) ou *prof* (arbre similaire jusqu'au *prof* ancêtre).

Ainsi, lorsqu'appliquée à l'arbre de la figure 5.1, la première étape de l'algorithme produit le graphe illustré à la figure 5.2.

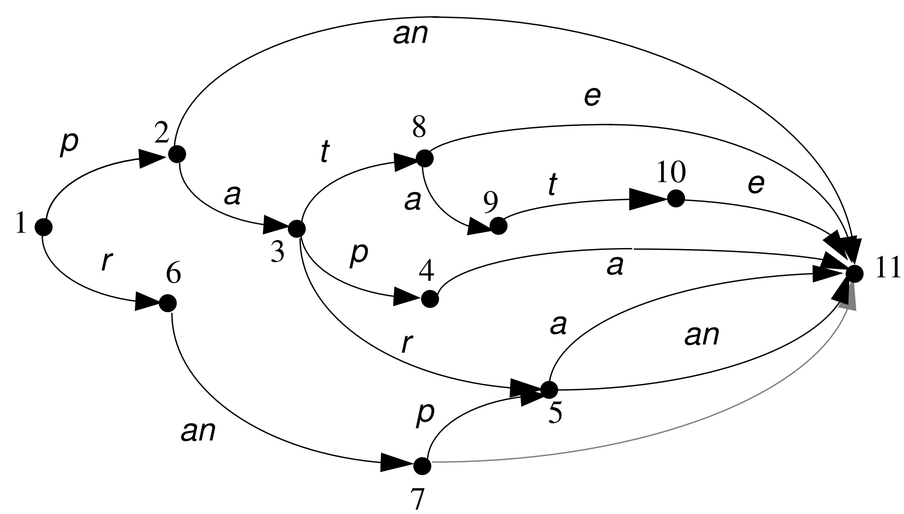


**Figure 5.2 Graphe résultant de la première étape de compression**

### 5.1.2 Deuxième étape

Après une première étape plutôt globale, la deuxième étape de l'algorithme de compression, chargée de réduire le nombre de transitions pleines au profit des transitions vides, travaille plutôt au niveau local. Cette étape est justifiée par le fait que le HMM qui correspond au graphe minimal obtenu suite à la première étape ne serait, lui, pas minimal. En effet, la première étape de l'algorithme ne tient pas compte de l'explosion subséquente subit par le graphe lors de la construction du HMM correspondant. C'est que, lors de cette construction, chaque transition étiquetée d'une unité (transitions pleines) est remplacée par le HMM correspondant à cette unité (voir section 2.5.1). Or chaque HMM est lui-même composé de plusieurs états et transitions, ce qui décuple d'autant l'espace nécessaire à la mémorisation des scores et des chemins qui y sont associés. Au contraire, les transitions vides se retrouvant tel quel dans le HMM final, il n'y a aucune multiplication d'espace. Qu'on se rappelle aussi, que lors de l'emprunt d'une transition vide, seule est calculée la probabilité de transition, comparativement aux calculs de plusieurs probabilités de distributions et transitions pour un seul HMM. D'où, tant d'un point de vue temps de calcul qu'espace mémoire, l'intérêt de maximiser les transitions vides par rapport aux transitions pleines et ce, même si le nombre total de noeuds et/ou transitions dans le graphe final en est augmenté. En fait, en posant comme hypothèse très conservatrice qu'une transition pleine de graphe une fois transposée sous forme de HMM occupe environ 5 fois plus d'espace qu'un noeud ou transition vide, on peut assumer que le nombre supplémentaire de transitions pour un graphe donné, peut être de 5 fois le nombre de transitions pleines retranchées par cet ajout.

Le processus appliqué à cette étape, consiste à identifier un à un les noeuds vers lesquels convergent plus d'une transition étiquetée par une même unité. Cette deuxième étape peut s'exprimer sous forme de pseudo-code par:

```
for (i = chaque noeud de graphe) {
        for (j = chaque phonème) {
                if (plus de deux transitions vers i avec le même phonème j) {
                        appliquer_la_réduction_locale()
                }
        }
}
```

Une telle configuration, tirée du graphe de la figure 5.2, est illustrée à la figure 5.3. On y constate que le noeud 11 est le noeud terminal de deux transitions étiquetées par le phonème *a* et de deux transitions étiquetées par le phonème *an*. L'algorithme transforme localement cette configuration en celle illustrée à la figure 5.4.

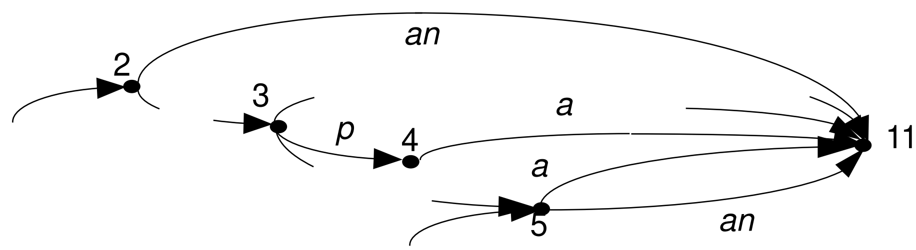


**Figure 5.3 Etat du graphe avant l'élimination de transitions pleines**

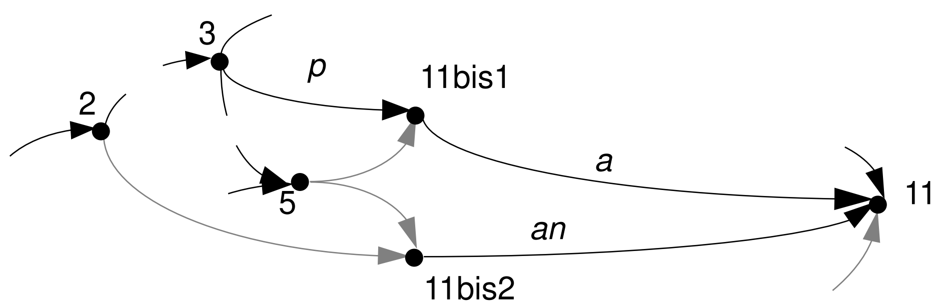


**Figure 5.4 Etat du graphe après l'élimination de transitions pleines**

On constate que deux nouveaux noeuds sont apparus, 11bis1 et 11bis2, mais qu'un autre noeud, 4, a disparu. Cette disparition s'explique par le fait qu'aucune autre transition ne partant du noeud 4, on a pu lier directement le noeud 3 aux noeuds nouvellement créés. Cette réduction supplémentaire, ne peut, par contre, être effectuée que dans un cas, soit lorsque suite à l'ajout des noeuds supplémentaires, un noeud se trouve lié à un seul de ces derniers. Ainsi, cette réduction n'a pu être appliquée au noeud 5, ce noeud se trouvant lié à plus d'un noeud nouvellement créés, en l'occurrence 11bis1 et 11bis2.

Il existe une autre configuration pour laquelle il est possible de réduire le nombre de transitions vides mais qui ne se produit pas dans le graphe servant d'exemple. Cette dernière est illustrée au moyen d'un extrait hypothétique à la figure 5.5. L'on peut voir au schéma A de cette figure, que les transitions vides destinées aux noeuds e2 et e4 proviennent d'un même sous-ensemble de noeuds soit e1 et e3. Il est donc possible de combiner les noeuds e2 et e4 et ainsi d'éliminer deux transitions vides (accessoirement le noeud e1 a pu être éliminé et les transitions lui étant destinées, être reliées directement au noeud e2, car aucune autre transition ne provenait du noeud e1). En fait, ce type de réduction peut être effectué entre un nombre de noeuds illimité (limité évidemment par le nombre d'unités composant les mots du lexique).

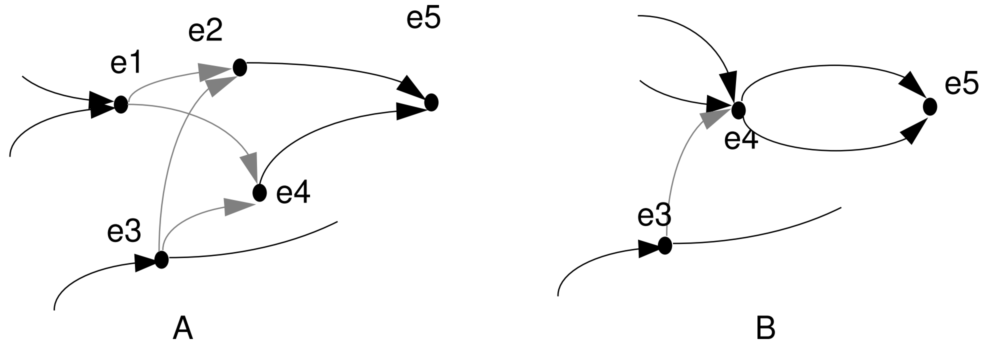


**Figure 5.5 Réduction du nombre de transitions vides.**

Le graphe final est illustré à la figure 5.6. Il est intéressant de noter qu'aucune de ces trans-

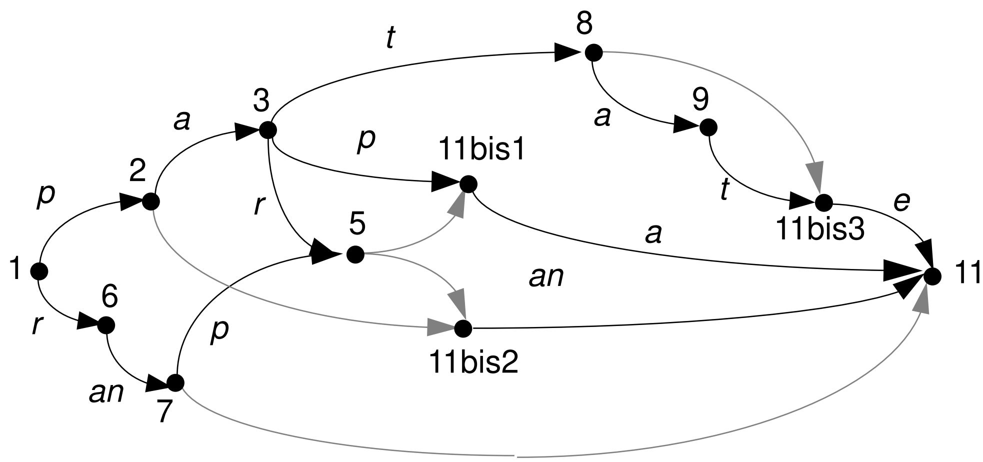


**Figure 5.6 Graphe résultant de la deuxième étape de compression**

formations n'affectent la propriété déterministe du graphe. Tout au plus cela nous oblige-t-il, lors du parcours du chemin associé à un mot, à effectuer un genre de "look-ahead" d'un noeud dans le cas des transitions vides qui elles, peuvent être plusieurs à provenir d'un même noeud. Ce "look-ahead" est limité à un noeud, puisque par construction il est impossible que deux transitions vides se suivent de façon consécutive. Ceci de toute façon, ne serait pas très utile. Cette caractéristique du graphe s'est révélée pratique lors de l'implantation de l'algorithme décrit à la section 5.2.

### 5.1.3 Graphe minimal

Il est intéressant de noter que le graphe obtenu par l'application de l'algorithme décrit aux sections précédentes n'est pas nécessairement le graphe minimal correspondant à un lexique donné. En effet, le degré de compression obtenu est différent selon l'extrémité à partir duquel l'algorithme est appliqué (i.e. de gauche à droite ou de droite à gauche). D'ailleurs ceci s'applique aussi à la représentation sous forme d'arbres, c'est-à-dire que l'arbre obtenu par l'amalgame droite-gauche des mots n'entraîne pas nécessairement le même degré de compression que celui obtenu de gauche à droite. Un exemple théorique illustrant ces dires est données à la figure 5.7. Dans cet exemple, on remarque que la variable **I** est

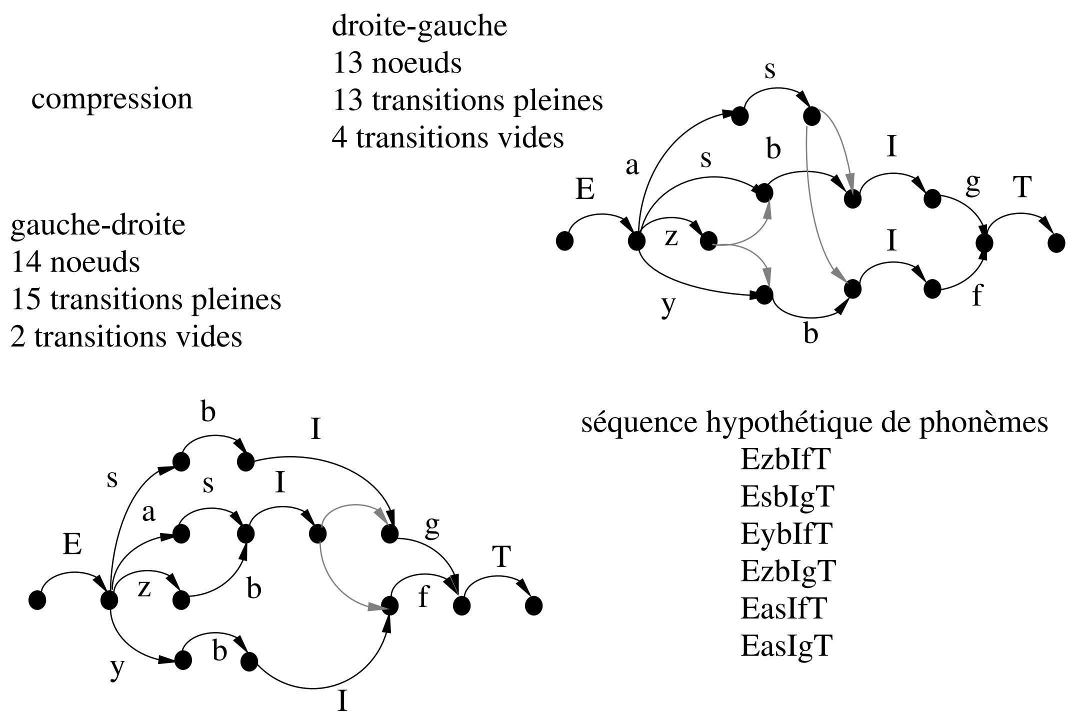


**Figure 5.7 Exemple de compression théorique avantageusement effectuée de droite à gauche**

répétée trois fois dans le graphe obtenu par compression gauche-droite et seulement deux fois dans celui obtenu par compression droite-gauche. Si on remplace cette variable par une séquence de plusieurs phonèmes, une compression droite-gauche deviendrait donc, de plus en plus avantageuse par rapport à une compression gauche-droite. Pour certains lexiques et/ou certaines langues où une majorité de mots présenteraient ce type de patron au niveau de leur transcription phonétique, une compression par la fin plutôt que le début des mots se pourrait se révéler une alternative plus économique en temps de fouille et en espace mémoire, le graphe final obtenu étant plus compact. Dans les faits, il peut être difficile de savoir, à priori, laquelle d'une compression gauche-droite ou droite-gauche donne le graphe le plus compact. Pour obtenir ce dernier, il est donc préférable d'appliquer l'algorithme deux fois et de choisir parmi les deux graphes obtenus, celui pour lequel on observe un taux de compression maximum.

Théoriquement, ces graphes sont les mêmes que ceux obtenus au moyen de l'algorithme plus général de réduction des AFDs (Automates Finis Déterministes) (voir [HOPC 79][1]). En fait, on aurait effectivement pu employer cet algorithme plus général en lieu et place de celui décrit plus haut. Pour ce faire, il suffit de constater qu'un arbre n'est finalement qu'un AFD non minimal duquel on aurait éliminé l'état trappe (voir la figure 5.8)[2]. L'algorithme de compression par la méthode des sous-arbres est cependant nettement moins gourmand en mémoire, de l'ordre de $M^2$ (tableau *poss_compress* du pseudo-code page 52) comparativement à $f^2$ pour l'algorithme général, et en temps, nombre de comparaison de l'ordre de $f^2$ dans le pire cas, comparativement à $f^2$ pour l'algorithme général (routine *poss_compress*), où $f$ est le nombre de noeuds de l'arbre et $M$ le nombre de mots du lexi-

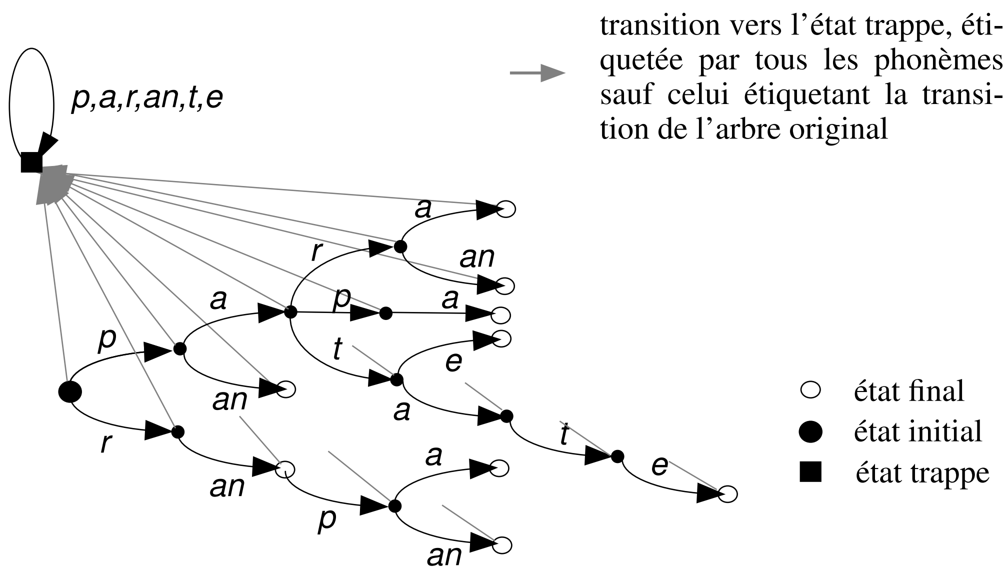


**Figure 5.8 AFD correspondant au lexique de la figure 5.1**

que. Or, on aura toujours (sauf cas totalement dégénéré) que $M \ll f$. De plus, puisque la

1. Les lecteurs non familiers avec cet algorithme peuvent sans crainte omettre de lire le reste de cette section pour passer à la section 5.1.4.

2. L'AFD obtenu à partir de l'arbre inversé (droite-gauche) est différent de celui obtenu de l'arbre gauche-droite, ce qui explique pourquoi le graphe minimal n'est pas le même.

compression de l'arbre est effectuée au fur et à mesure du déroulement de l'algorithme, le nombre de comparaisons sera en général bien inférieur à $f^2$. Dans l'algorithme de compression général, le nombre de comparaisons est toujours égal à $f^2$. C'est qu'en fait, la partie de l'algorithme décrit à la section 5.1.1 n'est qu'une implantation d'un cas particulier de l'algorithme général, algorithme particulier pouvant être utilisé à cause de la topologie particulière des AFDs obtenus à partir d'arbres lexicaux.

On peut assez aisément se convaincre de cette affirmation en appliquant l'algorithme général à la main à n'importe quel AFD obtenu d'un arbre lexical. On constate alors que des deux sous-groupes distincts à l'origine, soit celui regroupant les noeuds acceptants ou feuilles (groupe qui, on note, ne peut être scindé n'étant l'origine d'aucune transition), et celui regroupant les noeuds non acceptants (tous les autres noeuds y compris l'état trappe), le processus qu'on applique naturellement est de procéder en regroupant entre eux les noeuds menant aux feuilles par une transition étiquetée d'une même unité. Ces sous-groupes sont, en effet, distinguables entre eux puisqu'alors $\delta(e_i,x) = \text{trappe}$, soit un noeud non acceptant, et $\delta(e_j,x) = \text{feuille}$, soit un noeud acceptant, où $\delta(e_i,x) = e_k$ est appelée fonction de transition d'un AFD, et indique qu'une transition existe entre les états $e_i$ et $e_k$ pour une unité $x$. Or à leur tour, les nouveaux sous-groupes ainsi formés ne peuvent être scindés, leur unique transition menant à un groupe dont on a prouvé qu'il ne pourra être scindé. Par la suite, il suffit de répéter ce processus d'étage de noeuds en étage de noeuds et ce jusqu'à la racine. Un étage de noeuds est défini par l'ensemble des noeuds à une même distance d'une feuille. Ce processus, on le constate, n'est autre que la fusion de sous-arbres similaires d'où l'équivalence, dans ce cas bien particulier, des deux algorithmes.

### 5.1.4 Résultats et discussion

Cette section présente des résultats de compression obtenus en faisant varier divers paramètres. Ces résultats sont exprimés en termes de transitions pleines (P), transitions vides (V) et noeuds (N) des diverses structures de recherche sous lesquelles peuvent être représentés les mots d'un lexique. Les paramètres considérés sont le sens de compression utilisé ainsi que l'origine et la taille des lexiques, lexiques qui font d'ailleurs l'objet d'une description plus détaillée à l'Appendice B. Les diverses structures considérées sont la liste, l'arbre et les graphes produits après la première et la deuxième étape de compression. Ces résultats nous permettent de constater plusieurs points intéressants.

D'abord dans tous les cas, l'emploi d'un graphe minimal permet, tel qu'on peut le constater au tableau 5.2, de réaliser une économie d'espace non négligeable. En fait, ce gain en

**Tableau 5.2 Résultats de la compression gauche-droite de divers lexiques**

| **lexique** | | **LA2** | **LA1** | **LF1** | **LF2** | **LF3** | |
|---|---|---|---|---|---|---|---|
| nb. d'unités | | 45 | 41 | 37 | 37 | 37 | |
| nb. de phonies | | 6 145 | 47 443 | 17 280 | 180 205 | 44 722 | référence |
| liste | P | 36 120 | 341 022 | 118 929 | 1 451 460 | 333 593 | a |
| arbre | P | 18 409 | 158 589 | 31 996 | 356 221 | 136 861 | b |
| (1 - (b / a)) * 100 | | 49.03 | 53.50 | 73.10 | 75.46 | 58.97 | |
| graphe intermédiaire | P | 9 546 | 66 809 | 6 117 | 69 517 | 54 901 | |
| | V | 1 139 | 6 531 | 1 050 | 12 443 | 3 739 | |
| | N | 4 974 | 28 891 | 2 277 | 23 945 | 22 568 | |
| graphe final | P | 6 781 | 38 175 | 3 032 | 29 702 | 29 255 | c |
| | V | 3 643 | 34 748 | 3 677 | 44 992 | 29 309 | |
| | N | 5 110 | 30 179 | 2 441 | 25 350 | 23 714 | |
| (1 - (c / b)) * 100 | | 63.16 | 75.93 | 90.52 | 91.66 | 78.62 | |

pourcentage, est même supérieur au gain réalisé lors du passage d'une liste à un arbre. Ainsi, pour les lexiques français explosés, c'est-à-dire contenant toutes les formes déclinées ou conjuguées d'un mot, soit LF1 et LF2, le taux moyen de compression lors du passage d'une liste à un arbre est de 74% comparé à un taux moyen de 91% lors du passage d'un arbre à un graphe (97.5% si l'on compare directement liste et graphe). De même ces taux pour l'anglais, bien que légèrement moindres, sont respectivement de 51% lors du passage d'une liste à un arbre et de 74% lors du passage d'un arbre à un graphe (85% si l'on compare directement liste et graphe). En moyenne donc, on obtient un gain supplémentaire de 17% et 23% pour le français et l'anglais respectivement. On constate finalement, que la grosseur du lexique semble favoriser un plus haut taux de compression, ce qui est intuitivement logique.

En passant, on peut noter que le nombre de transitions vides rajoutées est toujours grandement inférieur à 5 fois le nombre de transitions pleines éliminées, seuil de rentabilité espace-temps qui avait été signalé à la section 5.1. En fait dans certains cas, ce nombre est même inférieur aux nombres de transitions pleines éliminées. Il est légèrement supérieur dans les autres cas, mais toujours inférieur au double du nombre de transitions pleines éliminées.

On note aussi que pour un lexique, LF3, formé exclusivement des formes de base du lexique LF2 et donc plus similaire en taille et en composition au lexique anglais LA1, on obtient un taux de compression semblable, bien qu'encore légèrement supérieur, au lexique anglais. On peut donc affirmer que la représentation sous forme de graphe avantage effectivement les langues tel le français, pour lesquelles on a un nombre élevé de formes dérivées pour une même forme de base, ce à quoi l'on s'attendait. Mais ceci n'exclut pas le fait que l'anglais puisse aussi bénéficier de cette forme de représentation.

On a aussi constaté que plus le nombre de phonèmes utilisés augmente, plus le taux de compression diminue. C'est ainsi qu'on obtient respectivement des taux de compression de 92%, 76% et 63% lors de l'emploi de 37, 41 et 45 phonèmes. A cet égard, il faut par contre avouer que notre échantillonnage de lexiques n'était relativement pas très grand.

Le taux de compression est aussi réduit lors d'un changement de type d'unités, triphones et phonèmes dans notre exemple, mais beaucoup moins qu'on aurait pu se l'imaginer compte tenue de la proportion beaucoup plus grande du nombre de triphones par rapport au nombre de phonèmes. C'est ce que l'on constate au niveau des résultats rapportés au tableau 5.3, où deux lexiques anglais et français sont compressés selon un codage phonéti-

**Tableau 5.3 Impact du type d'unités sur le taux de compression**

<table>
<tr><th colspan="2">lexique</th><th colspan="2">LA2</th><th colspan="2">LF1</th><th></th></tr>
<tr><td colspan="2">nb. de phonies</td><td colspan="2">6 145</td><td colspan="2">17 280</td><td></td></tr>
<tr><td colspan="2">nb et type d'unités</td><td>45 phonèmes</td><td>7 366 triphones</td><td>37 phonèmes</td><td>4 272 triphones</td><td>référence</td></tr>
<tr><td>liste</td><td>P</td><td>36 120</td><td>36 120</td><td>118 929</td><td>118 929</td><td>a</td></tr>
<tr><td>arbre</td><td>P</td><td>18 409</td><td>23 151</td><td>31 996</td><td>46 162</td><td>b</td></tr>
<tr><td colspan="2">(1 - (b / a)) * 100</td><td>49.03</td><td>35.91</td><td>73.10</td><td>61.19</td><td></td></tr>
<tr><td rowspan="3">graphe intermédiaire</td><td>P</td><td>9 546</td><td>15 287</td><td>6 117</td><td>10 336</td><td></td></tr>
<tr><td>V</td><td>1 139</td><td>0</td><td>1 050</td><td>0</td><td></td></tr>
<tr><td>N</td><td>4 974</td><td>9 358</td><td>2 277</td><td>4 299</td><td></td></tr>
<tr><td rowspan="3">graphe final</td><td>P</td><td>6 781</td><td>12 975</td><td>3 032</td><td>7 358</td><td>c</td></tr>
<tr><td>V</td><td>3 643</td><td>2 571</td><td>3 677</td><td>3 959</td><td></td></tr>
<tr><td>N</td><td>5 110</td><td>9 797</td><td>2 441</td><td>5 280</td><td></td></tr>
<tr><td colspan="2">(1 - (c / b)) * 100</td><td>63.16</td><td>43.95</td><td>90.52</td><td>84.06</td><td></td></tr>
</table>

que en phonèmes et en triphones. Ceci est une bonne chose étant donné que des HMMs de triphones donnent généralement de meilleurs résultats que des HMMs de phones. C'est ainsi que malgré un nombre de triphones 110 fois supérieur au nombre de phones pour le français et 160 fois supérieur pour l'anglais, le taux de compression obtenu n'est inférieur que d'environ 6% pour le français et 20% pour l'anglais. Ceci s'explique par le fait que l'utilisation de triphones n'ajoute pas vraiment de diversité dans les mots du lexique, par opposition à un nombre supérieur de phonèmes. En effet, c'est le nombre de phonèmes d'une langue qui, en partie du moins, contribue à la diversité de ses mots. Par contre, l'utilisation de triphones n'a comme seul impact sur le lexique en place, que de réduire de une unité la partie commune et donc sujette à fusion entre deux mots. Par exemple, la compression des mots *marchait - /m a r ch è/ - /ma mar arch rchè chè/* et *marchons - /m a r ch on/ - /ma mar arch rchon chon/* ne se trouve réduite que de une "unité", passant de 4 à 3. Et encore faut-il souligner, que ces résultats se rapportent au cas théorique où tous les triphones sont modélisés (un HMM différent par triphone), ce qui en pratique n'est jamais le cas comme on l'a souligné à la section 2.5.1.

Finalement, lorsqu'on compare les résultats obtenus selon la direction de compression (tableau 5.4), surprise, il semble clairement indiquer qu'une direction gauche-droite est

**Tableau 5.4 Impact de la direction de compression sur le taux de compression**

| **lexique** | | **LA2** | | **LF1** | | |
|---|---|---|---|---|---|---|
| nb. de phonies | | 6 145 | | 17 280 | | |
| nb. d'unités | | 45 | | 37 | | |
| direction | | direct | inverse | direct | inverse | référence |
| liste | P | 36 120 | 36 120 | 118 929 | 118 929 | a |
| arbre | P | 18 409 | 18 549 | 31 996 | 44 013 | b |
| (1 - (b / a)) * 100 | | 49.03 | 48.65 | 73.10 | 62.99 | |
| graphe intermédiaire | P | 9 546 | 10 243 | 6 117 | 9 006 | |
| | V | 1 139 | 687 | 1 050 | 1 083 | |
| | N | 4 974 | 4 854 | 2 277 | 2 728 | |
| graphe final | P | 6 781 | 6 652 | 3 032 | 3 499 | c |
| | V | 3 643 | 4 318 | 3 677 | 6 811 | |
| | N | 5 110 | 5 075 | 2 441 | 3 458 | |
| (1 - (c / b)) * 100 | | 63.16 | 64.14 | 90.52 | 92.05 | |

préférable pour le français, mais peut-être pas si avantageuse pour l'anglais. En effet, dans ce dernier cas, une approche droite-gauche pourraient s'avérer légèrement avantageuse au niveau de l'espace occupé par certaines structures. Du point de vue temps de reconnaissance cela est moins évident. On verra, en effet, à la section 5.4 que l'emploi d'un faisceau d'émondage peut rendre préférable l'utilisation de graphes plus compressés au niveau des premiers phonèmes des mots qu'au niveau des derniers phonèmes. A ce niveau, des tests de reconnaissance supplémentaires auraient pu être utiles. Cependant, étant donné l'unique et encore relativement petit gain constaté, cette voie de recherche ne fut pas poursuivie. Au moins en conclurons-nous, qu'il peut exister des cas où il ne serait pas complètement inutile d'effectuer la compression dans les deux directions.

## 5.2 Correspondance segments de graphe - entrée du dictionnaire

Lorsqu'un lexique est représenté sous forme de graphe, le résultat brut d'une reconnaissance consiste en une série de phonèmes plutôt qu'en un numéro d'entrée lexicale. Il faut donc qu'il existe un mécanisme permettant de faire rapidement un lien entre les deux. Ce mécanisme peut consister en une fouille phonème par phonème au travers d'un arbre, ou à un accès à une table de hachage. Deux solutions qui sont quasi instantanées face au temps de calcul pris par l'étape de reconnaissance proprement dite mais qui, néanmoins représentent un fardeau supplémentaire comparé à l'accès direct obtenu par l'emploi d'arbres (où à chaque feuille correspond un mot).

Il est donc intéressant de savoir si un accès plus rapide au mot associé à un chemin peut être trouvé, en utilisant, par exemple, des transitions n'appartenant qu'à un seul chemin. En effet, il serait alors facile d'associer une étiquette d'identification à de telles transitions, étiquette qu'on pourrait alors propager à la place du pointeur arrière dans l'algorithme de Viterbi. Un petit algorithme fut donc élaboré pour tenter de savoir si une telle transition existait pour tous les mots du graphe, ou à défaut, quel était le sous-ensemble minimum de transitions permettant d'identifier un mot (le sous-ensemble maximum étant bien entendu, l'ensemble des transitions d'un chemin).

On trouve au tableau 5.5 les résultats obtenus pour certains des lexiques déjà étudiés. Pre-

**Tableau 5.5 Attribution des transitions pour divers lexiques**

| | **mots pouvant être uniquement identifiés** | | |
|---|---|---|---|
| **nombre de transitions** | **1** | **2** | **3** |
| LF1 | 2 203 | 11 987 | 3 090 |
| LF3 | 25 190 | 19 480 | 52 |
| LA1 | 36 598 | 10 845 | - |
| LA2 | 4 294 | 1 851 | - |

mière constatation, plus le taux de compression obtenu est élevé, plus le nombre de mots exigeant un nombre de transitions élevé pour leur identification est grand. On s'y attendait. Bonne nouvelle cependant, pour tous les lexiques étudiés, un nombre maximum de trois transitions (deux pour l'anglais) est nécessaire à l'identification unique[1] d'une entrée lexicale. Or bien que nécessitant un peu plus d'effort, les cas où plus d'une transition est nécessaire à l'identification d'un mot ne pose pas un problème insurmontable. En fait, dans tous les cas, l'algorithme qui suit permet d'identifier, au fur et à mesure de la fouille, le mot reconnu, et ce, sans avoir à conserver en parallèle un arbre d'accès lexical.

Il suffit d'associer l'identificateur du mot (par exemple un numéro d'entrée lexicale) aux transitions servant à son identification, de même que le nombre de transitions nécessaire à cette identification. Par la suite, lors de la fouille, on associe à chaque hypothèse une liste des mots tirés des listes de mots des transitions empruntées, en prenant soin de leur adjoindre un compteur servant à indiquer le nombre de transitions empruntées identifiant chaque mot. Lorsque pour un mot ce nombre devient égal au nombre de transitions nécessaire à son identification, le mot peut être associé à l'hypothèse. Le processus d'identification est terminé. La figure 5.9 illustre ce que ce type d'association donnerait pour le mini-lexique

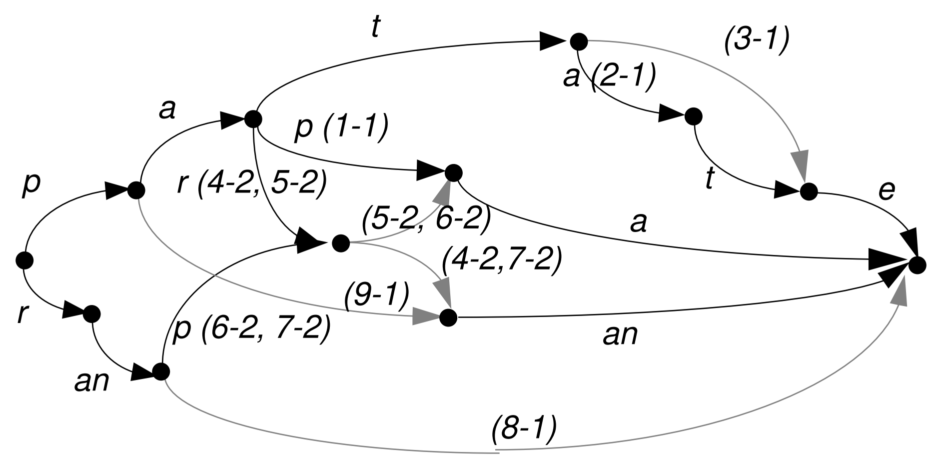


**Figure 5.9 Exemple d'association entre identificateurs et transitions**

du tableau 5.1. Le graphe illustré est le même que celui de la figure 5.6, les numéros de noeud en moins pour plus de clarté. Un identificateur consiste en une paire (numéro de

1. Bien entendu, toutes les entrées homonymes sont regroupées sous une même entrée lexicale, l'identification d'un homonyme particulier devant se faire par la suite à l'aide d'un autre moyen (non acoustique).

mot-compteur). On y retrouve 5 mots pouvant être identifiés par une seule transition et 4 mots ayant besoin de deux transitions.

Le fait que la correspondance entre un chemin du graphe et une entrée lexicale puisse se faire au moyen d'un nombre minimum de transitions peut aussi servir à inhiber un sous-ensemble du lexique de façon relativement rapide. Cette propriété peut s'avérer utile dans le cas, par exemple, où un modèle du langage est utilisé à priori pour activer ou désactiver un ensemble de mots.

## 5.3 Autres avantages de la représentation en graphe

Finalement, le fait de représenter le lexique sous forme de graphe présente un autre intérêt. Ceci permet, en effet, d'utiliser facilement tous les algorithmes et heuristiques basés sur l'algorithme de Viterbi. C'est le cas entre autres des faisceaux d'émondage, qui peuvent être employés sous leurs formes les plus simples (section 2.3.1) ou sous des formes plus élaborées (section 9).

C'est le cas aussi des algorithmes N-meilleurs qui peuvent facilement être employés de façon à obtenir la liste des meilleures entrées lexicales. Ceci permet de compenser pour l'un des inconvénients de l'approche du graphe compressé qui est, dans sa forme la plus simple, de ne donner que la meilleure solution et ce contrairement aux méthodes où un pairage exact est effectué mot par mot (voir section 4.1.1 et 4.1.2).

## 5.4 Applications.

Les résultats rapportés ici ont été obtenus lors de l'emploi de l'algorithme de compression dans le cadre du projet TransTalk. TransTalk est un projet de traduction assisté par ordinateur dans lequel un traducteur est appelé à dicter dans une langue de destination la traduction d'un texte d'origine. Le système, fort d'un module de reconnaissance pour la langue de destination, du texte d'origine et d'un modèle de traduction, se doit de prendre en dictée correctement le texte traduit. Lors d'une première phase [DYME 94], dans laquelle il était prévu que le texte soit dicté sous forme de mots isolés, il nous est apparu idéal d'utiliser l'algorithme de compression décrit plus haut. Le tableau 5.3 donne le taux de com-

**Tableau 5.6 Résultats de compression du lexique de TransTalk**

| type d'unités | phonèmes | |
|---|---|---|
| nombre d'unités | 47 | |
| nombre de mots | 14731 | référence |

**Tableau 5.6 Résultats de compression du lexique de TransTalk**

| | | | |
|---|---|---|---|
| liste | P | 134 869 | a |
| arbre | P | 42 372 | b |
| (1 - (b / a)) * 100 | | 68.58% | |
| graphe intermédiaire | P | 20 164 | |
| | V | 1 415 | |
| | N | 9 118 | |
| graphe final | P | 11 696 | c |
| | V | 9 966 | |
| | N | 9 569 | |
| (1 - (c / b)) * 100 | | 72.40% | |

pression obtenu pour le vocabulaire de TransTalk. Au tableau 5.8, sont rapportés les temps de reconnaissance obtenus à partir de différentes structures.

**Tableau 5.7 Temps de reconnaissance pour le lexique TransTalk**

| **représentation du lexique** | **temps (Min.)** | **gain en temps** | **gain en espace** |
|---|---|---|---|
| liste de mots | 46:24 | | |
| arbre de mots | 7:47 | 83.85% | 68.58% |
| graphe de mots | 5:54 | 25.84% | 72.40% |

Dans tous les cas, un faisceau d'émondage fut employé de manière à obtenir un taux de reconnaissance le plus similaire possible. Surprise, le gain en temps n'est pas proportionnel au gain d'espace entre liste et arbre et graphe et arbre. En fait, ce surprenant résultat est le fruit de l'impact plus ou moins important qu'a le faisceau d'émondage sur chaque type de structure. C'est ce que tente d'illustrer la figure 5.10. On peut y voir que l'emploi du faisceau, dans le cas de listes de mots, ne permet rien de plus que de couper un mot et ce même si ce mot partage un même préfixe et/ou suffixe avec plusieurs autres car chaque mot doit être désactivé individuellement. Dans le cas de l'arbre, tous les mots ayant un même préfixe sont coupés d'un seul coup, coupant du même coup tout les sous-arbres A et B et ne laissant actif que le sous-arbre C. La même chose se produit dans le cas du graphe mais ici, les sous-arbres A, B et C ayant été combinés lors de la compression, et le préfixe C demeurant actif, le reste du graphe demeurera actif. C'est en fait en quelque sorte comme si le faisceau d'émondage compressait l'arbre en éliminant des sous-arbres inutiles. Néanmoins, puisqu'il y a définitivement gain en espace mémoire avec la structure de graphe, cette dernière demeure un bon choix lorsque les contraintes de mémoires sont importantes et/ou lorsque le nombre de suffixes est impressionnant.

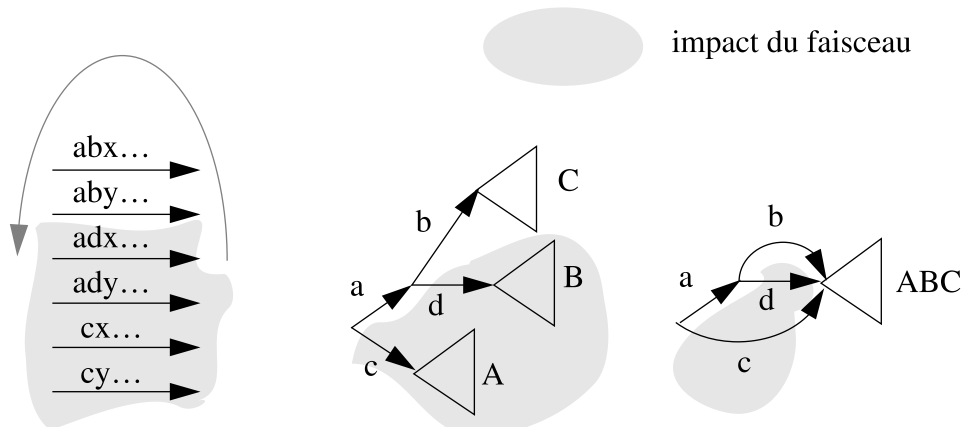


**Figure 5.10 Impact d'un faisceau d'émondage classique sur différentes structures de recherche**

De plus, on comprend maintenant mieux pourquoi il n'est peut-être pas judicieux de choisir un graphe obtenu par compression droite-gauche qui ne soit que légèrement plus petit que celui obtenu de gauche à droite. En effet, plus un faisceau peut être appliqué tôt et de façon efficace pendant la fouille, plus le gain de temps risque d'être élevé, d'où l'avantage d'utiliser des graphes ou des arbres plus comprimés à gauche qu'à droite. Evidemment, ici encore, si la compression droite-gauche est nettement plus avantageuse du point de vue espace et que cette contrainte soit prépondérante, alors il serait souhaitable d'utiliser le plus petit graphe.

### 5.4.1 Traitement des liaisons

Comme on l'a déjà mentionné à la section 4.1.4, les liaisons et élisions sont des particularités propres au français. Or, tant les liaisons que les élisions sont directement influencées par le contexte. Ainsi dans le cas des liaisons, le *s* final de *petits* est muet sauf si le mot suivant est un nom débutant par une voyelle phonétique comme *amis,* auquel cas, il se prononce *z*. De même dans le cas des élisions, le *e* final de *le* ou *la* est escamoté devant un nom débutant par une voyelle. Or même en mots isolés, il est plus naturel, du moins pour des sujets dont le français est la langue maternelle, de faire la liaison (tant qu'à l'élision, il n'est même pas question de l'omettre). La consonne de liaison, tel qu'on nomme le phonème ajouté, ou le mot élidé, comme *le* ou *la*, se retrouve alors, non pas seul ou lié au mot où il apparait graphiquement, mais bien lié au mot suivant. Par exemple, la séquence, "*les petits amis*" se prononce en mots isolés "*lè peti zami*" et la séquence "*l' arbre tombe*", "*larbr tomb*".

Or le graphe compressé permet de résoudre ces problèmes de façon fort économique tant du point de vue temps de calcul que du point de vue espace mémoire. Mentionnons d'abord qu'il est assez facile d'établir un processus semi-automatique qui permet de préfixer par les consonnes de liaisons pertinentes tous les mots débutant par une voyelle phonétique ou un mot élidé. Notons en passant qu'il ne serait pas faux, (dans le sens que certaines séquences phonétiques plutôt rares ou absentes chez un locuteur de langue maternelle française serait alors permises comme "*sarbr*" pour "*ce arbre*" qui, est en français correct, impossible) de préfixer tous les mots débutant par une voyelle phonétique par toutes les consonnes de liaison possibles et tous les mots élidés. Cela n'entraînerait pas de coût supplémentaire au niveau temps et mémoire (bien au contraire, comme on le verra, cela augmenterait le taux de compression), mais il s'agirait là d'une surgénération. Une fois le processus de préfixation effectué, il s'agit simplement de compresser le dictionnaire augmenté ainsi obtenu de la façon habituelle pour obtenir un graphe à peine plus grand que le graphe obtenu suite à la compression du même vocabulaire sans consonne de liaisons ou mots élidés ajoutées (voir le tableau 5.8). Dans cet exemple, seules les liaisons

**Tableau 5.8 Différent taux de compression pour le lexique TransTalk**

| | **consonnes de liaisons** | | **% d' augmentation** |
|---|---|---|---|
| | **présentes** | **absentes** | |
| nombre de formes phonétiques | 14 731 | 14 629 | 0.7 |
| arbre - P | 42 372 | 42 009 | 0.86 |
| graphe - P | 11 696 | 11 598 | 0.84 |

portant sur les consonnes de liaisons les plus fréquentes *z, t* et *n* étaient traitées. On constate, en effet, que le rapport entre le graphe avec et sans liaisons est plus petit que celui entre l'arbre avec et sans liaisons.

Cela s'explique par le fait que sous forme de graphe, une série de formes phonétiques dérivées d'un même mot, se présente tel qu'illustrée à la figure 5.11A (pour le mot *ami*, on aura *zami* ("*les amis*"), *nami* ("*bon ami*"), *pami* ("*trop ami*") et *tami* ("*petit ami*"). Qui plus est, n'importe quel autre mot pour lequel les mêmes consonnes de liaisons sont permises, par exemple *aimable*, partagera avec *ami* les mêmes instances de ces dernières (voir la figure 5.11B)[1]. L'emploi d'un arbre n'aurait pu permettre un tel degré de compres-

1. Ne pas oublier qu'il peut parfois être avantageux de procéder à la compression à partir de la fin des mots.

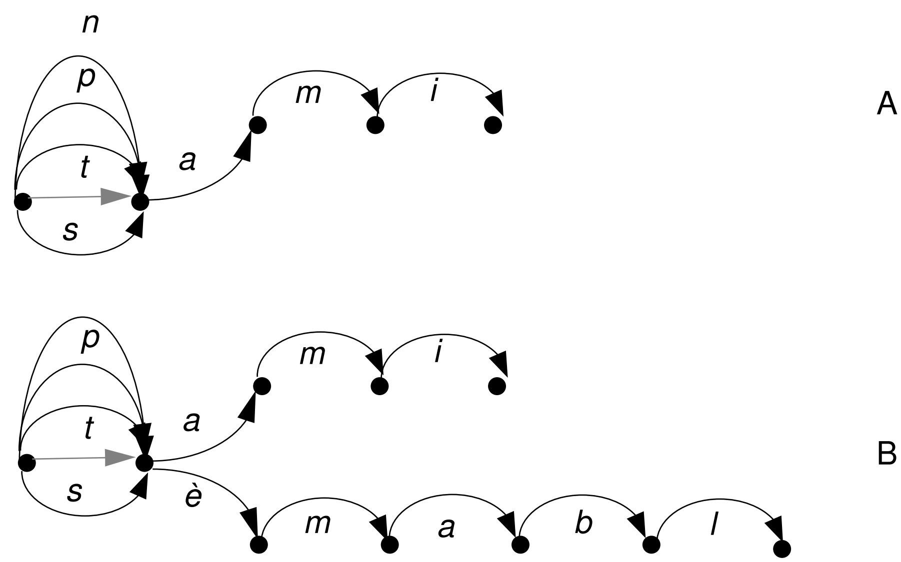


**Figure 5.11  Exemple de compression de séries phonétiques dérivées de mêmes mots**

sion puisque les suffixes communs générés par le traitement des liaisons n'auraient pu être amalgamés. C'est d'ailleurs un problème que soulève [GAUV 94c].

# Deuxième partie

Les travaux rapportées dans cette deuxième partie sont on ne peut plus actuels, la RAP continue étant encore au coeur de nombreuses recherches.

## 6 La reconnaissance en parole continue

La parole continue est celle normalement utilisée pour la communication entre humains. On peut comparer le signal obtenu pour la phrase “Les anges mangent leur poulets avec les ailes” prononcée en continue, donc normalement, et illustré à la figure 6.1, à celui

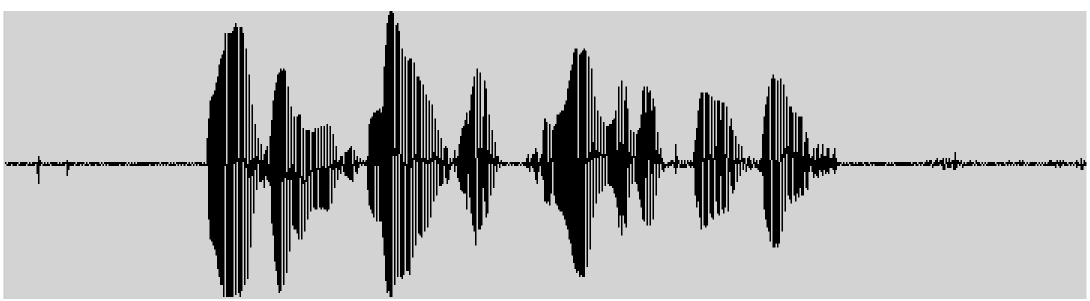

**Figure 6.1 Exemple d’un signal acoustique résultant d’une dictée en parole continue.**

obtenu pour la même phrase prononcée en mots isolés et illustré à la figure 4.1.

On constate immédiatement que, si dans la parole en mots isolés, des silences délimiteurs permettent de bien situer les mots dans le temps, ce n’est pas le cas en parole continue. Au contraire, leur présence est tout à fait imprévisible, et si l’on peut assumer sans trop d’erreur qu’un silence assez long (et ici on est toujours un peu dans l’arbitraire) marque bel et bien la fin et le début de deux groupes de mots, rien ne permet d’affirmer combien de mots composent ces groupes.

Bref, on se retrouve avec un espace de recherche beaucoup plus grand, car le nombre de messages pouvant être émis par un locuteur n'est plus égal à $M$, le nombre de mots du vocabulaire, mais bien à $\sum_{i=1}^{maxnbm} M^i$ où $maxnbm$ est le nombre maximum de mots dans une phrase. Or, même en fixant, disons à 15, ce nombre de mots, le nombre de solutions théoriquement possibles qu'on obtient pour un lexique de disons 10000 entrées est d'environs de $1 \times 10^{60}$. Ceci représente un nombre de solutions qu'il est bien évidemment impossible d'énumérer séparément et encore moins d'évaluer de la sorte.

En fait en mots isolés, les silences délimiteurs permettent de naturellement subdiviser la reconnaissance d'une phrase en un nombre fini de sous-problèmes beaucoup plus restreints. En perdant ces délimiteurs, l'on se retrouve face à un problème d'un ordre de grandeur tout à fait différent. A tel point en fait, que, comme on le verra, les approches utilisées pour la reconnaissance en parole continue de grands vocabulaires ne sont que de lointains dérivés des approches utilisées en reconnaissance de mots isolés.

## 6.1 Limites des représentations utilisées en mots isolés

Pour illustrer cette dernière affirmation, nous voyons d'abord comment, si cela est possible, l'on peut utiliser les graphes compressés pour la reconnaissance de la parole continue. Au chapitre suivant, on retrouvera une description des diverses approches réellement utilisées pour la reconnaissance de grands vocabulaires en parole continue. Ces approches seront illustrées par la présentation des systèmes de RAP actuellement développés par les principales équipes de recherche dans le domaine. Lorsqu'applicable, référence sera faite aux systèmes développés par les mêmes équipes pour la RAP en mots isolés. L'on mentionnera alors si oui ou non, l'un est l'ancêtre de l'autre et si oui sous quels aspects.

Hélas, bien que la représentation d'un lexique sous forme de graphe compressé soit une structure de recherche pratique et intéressante pour la reconnaissance en mots isolés, son utilisation pour la parole continue ne donne pas de très bon résultats. En effet, toujours en se rappelant de la faculté qu'a l'algorithme de Viterbi de mémoriser les chemins, on peut lier le dernier état d'un graphe au premier état du même graphe au moyen d'une transition vide. L'on obtient alors ce qu'il est convenu d'appeler une structure bouclée[1], laquelle en théorie devrait permettre de retrouver la séquence de mots correspondant à une entrée.

1. looped model

Ce n'est malheureusement pas le cas. Puisque plus rien ne permet de facilement détecter les jonctions entre mots, une fouille guidée par l'acoustique seule aura tendance à trouver comme meilleure solution, une phrase comportant un très grand nombre de petits mots plutôt que la bonne séquence. En effet, cette dernière inclut normalement des mots longs qui souvent correspondent moins bien au signal acoustique, qu'une séquence de mots courts, car plus grandes sont les chances que certains phonèmes soit omis ou mal prononcés. Au contraire, un choix adéquat de petits mots[1] offre souvent un meilleur pairage acoustique, puisqu'ils permettent, jusqu'à un certain point, de mimer un pairage phonème par phonème.

C'est ainsi que la meilleure solution associée par un graphe bouclé à la phrase tirée des données du projet TransTalk:

> "Ils espèrent que le parlement VIRGULE y compris les sénateurs libéraux VIRGULE feront l'impossible pour rejeter ce projet de taxe POINT_FINAL."

est:

> "T heap Hull espère fer là parlement Harvey gus aile p compris Rae Sam lenteur les bureau VIRGULE ferons auraient possède pourra jette US reprochait de taxe POINT_FINAL".

Or, même en faisant appel à des algorithmes N-meilleurs ou en introduisant un facteur de pénalité au niveau du score pour chaque nouveau mot (l'équivalent d'un unigramme équiprobable), il est fort probable qu'il faille générer un nombre prohibitif de phrases avant d'en obtenir une faisant un tant soit peu de sens, c'est-à-dire ayant un contenu sémantique valide et à peu près construite selon les règles de syntaxe de la langue traitée.

Le fait est, qu'en perdant les silences délimiteurs de mots, le rôle de la syntaxe et de la sémantique devient encore plus important qu'en mots isolés, ou plus exactement, son application doit se faire de façon plus précoce durant le processus de reconnaissance. Plus concrètement, ce sont ces derniers, qui au travers de MLs, doivent maintenant et en autant que possible, guider la fouille en réduisant le nombre de mots possiblement actifs à un moment donné et pour un contexte donné. En parole continue, il n'est plus possible de choisir d'appliquer le modèle du langage sur les résultats du pairage acoustique comme cela peut être le cas dans la parole en mots isolés. Dans le cas de cette dernière, il peut, en effet, être tout à fait suffisant de calculer le deuxième terme $P(m)$ de l'équation 2.2,

1. Par exemple, en français on a *l, m, s, y, on, en, a, un, et, ou*.

page 7 ($m = \underset{m}{\operatorname{argmax}} \; P(y|m)P(m)$), après avoir établi une liste des mots les plus probables correspondant à chaque îlot de parole.

Il faut donc que la structure utilisée lors de la reconnaissance permette ce guidage de façon efficace, guidage qui comme on l'a entrevu à la section 3.1.3, dépend totalement du modèle utilisé pour estimer $P(m)$. Malheureusement, il est très difficile, voir impossible d'intégrer de façon efficace quelques MLs que ce soit à la structure de recherche sous forme de graphe compressé présentée au chapitre 5. En effet, même l'intégration à une structure d'un des plus simples ML probabilistes qui soit, l'unigramme, exige qu'on puisse associer un mot ou groupe de mots à une partie de la structure. Or comme on l'a vu à la section 5.2, ceci n'est pas aussi facile avec le graphe compressé qu'avec, par exemple, un arbre (où à tout le moins, à chaque mot correspond une feuille de l'arbre). L'utilisation de la structure de graphe compressé est donc limitée à la parole en mots isolés.

### 6.1.1 Un cas particulier, la parole spontanée

Au début de ce chapitre, il a été mentionné que par opposition à la parole en mots isolés, la parole continue est celle normalement utilisée par l'être humain pour communiquer. Il est en fait plus juste de distinguer deux types de parole continue: la parole dictée (ou lue car c'est souvent le cas) et la parole spontanée. Ainsi, dite de façon spontanée, notre phrase exemple pourrait être “Hum les anges mangent leur poulets avec leurs eee non les ailes.”, énoncé auquel correspond le signal illustré à la figure 6.2.

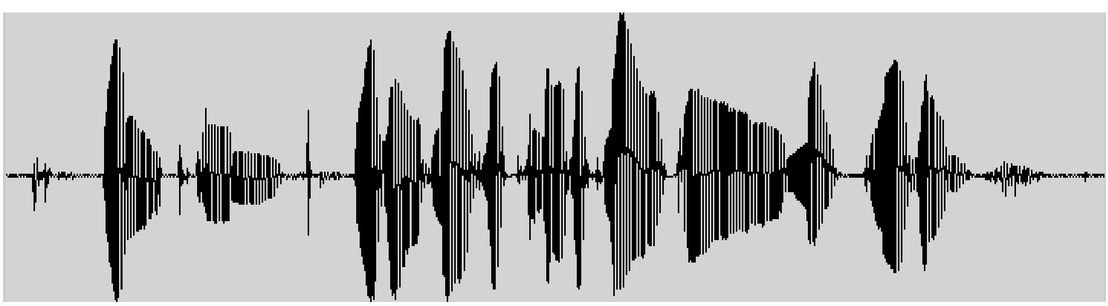

**Figure 6.2 Exemple d'un signal acoustique résultant de la parole spontanée.**

Comme on peut le constater, ce signal n'est pas aussi différent de celui de la figure 6.1 que ce dernier l'était de la figure 4.1. Autrement dit, les distinctions entre parole spontanée et parole lue ne se trouvent pas tant au niveau acoustique qu'aux niveaux syntaxique et sémantique. La parole spontanée est réellement celle normalement utilisée par l'être humain pour communiquer. Elle se caractérise par des hésitations, des reprises et un taux de constructions asyntaxiques assez élevé. De son côté, la parole dictée (ou lue) est, comme son nom l'indique, plus près de la langue écrite. C'est celle des récitations, des nouvelles télévisées et des discours.

Cependant, dans l'état actuel des choses, les mêmes techniques et approches peuvent et sont utilisées pour des tests de reconnaissance en parole spontanée et en parole dictée. Ceci n'empêche pas que, si éventuellement une approche complètement différente, au niveau du modèle du langage par exemple, s'avérait plus efficace pour un type de parole plutôt que l'autre, il faille alors aussi utiliser des approches différentes pour la reconnaissance. Par exemple, la parole spontanée pourrait possiblement tirer partie d'un éventuel modèle de dialogue, un champs de recherche qui n'en est encore qu'à ses débuts. Dans ce cas, les algorithmes de reconnaissance devraient éventuellement être adaptés pour satisfaire ces MLs nouveaux genres.

Néanmoins, en ce qui nous concerne, et comme il a déjà été mentionné, le type de modèle de langage employé est considéré comme une constante lors de ce travail. Or pour l'heure, ce type est le même pour la parole continue spontanée et dictée. Le même type d'algorithme de reconnaissance est donc utilisé dans les deux cas.

# 7 Description des approches de RAP pour les très grands vocabulaires

Dans ce chapitre est fait un survol des principaux systèmes de reconnaissance en parole continue et grand vocabulaire existant actuellement. Tous ceux décrits sont expérimentaux bien que depuis peu, au moins un produit commercial, peut-être plus, soit disponible. Ce dernier, par Philips, n'est cependant pas encore très répandu. Beaucoup des systèmes décrits ont participé à l'évaluation associée à la tâche ARPA du Wall Street Journal de novembre 1994 (voir section 8.2.2). Les résultats qu'ils y ont obtenus y sont rapportés. Tous sont à base de HMMs et donc utilisent un algorithme de type Viterbi lors de la reconnaissance. Tous aussi utilisent des MLs de type statistique pour guider leur fouille.

## 7.1 Structure de recherche

Comme on l'a déjà mentionné au chapitre 5 et à la section 3.1.1, le choix de la structure de recherche est aussi important pour l'étape de reconnaissance que le choix de l'algorithme utilisé, sinon plus important. Ceci est encore plus vrai dans le cas de la parole continue. En effet, il faut se rappeler que la structure choisie doit en plus de représenter l'ensemble de l'espace de recherche, permettre une bonne intégration du ML choisi. Or comme on l'a souligné au chapitre précédent, cet aspect est essentiel au bon fonctionnement d'un système RAP en parole continue et grand vocabulaire.

On peut grossièrement classer les structures de recherche utilisées par les systèmes actuels en trois groupes: les structures bouclées, les structures en arbre et les treillis.

### 7.1.1 Les structures bouclées

Typiquement dans une structure bouclée, chaque mot du lexique n'apparait qu'une fois (figure 7.1A). L'intégration d'un modèle d'unigramme est alors très simple (voir section 3.1.3). Il suffit que $q_{tr} = P(m_i)$ pour la transition $tr$ associée au mot $m_i$. Par contre, ce type de modèle n'est pas très puissant. La structure illustrée en B est supérieure. En effet,

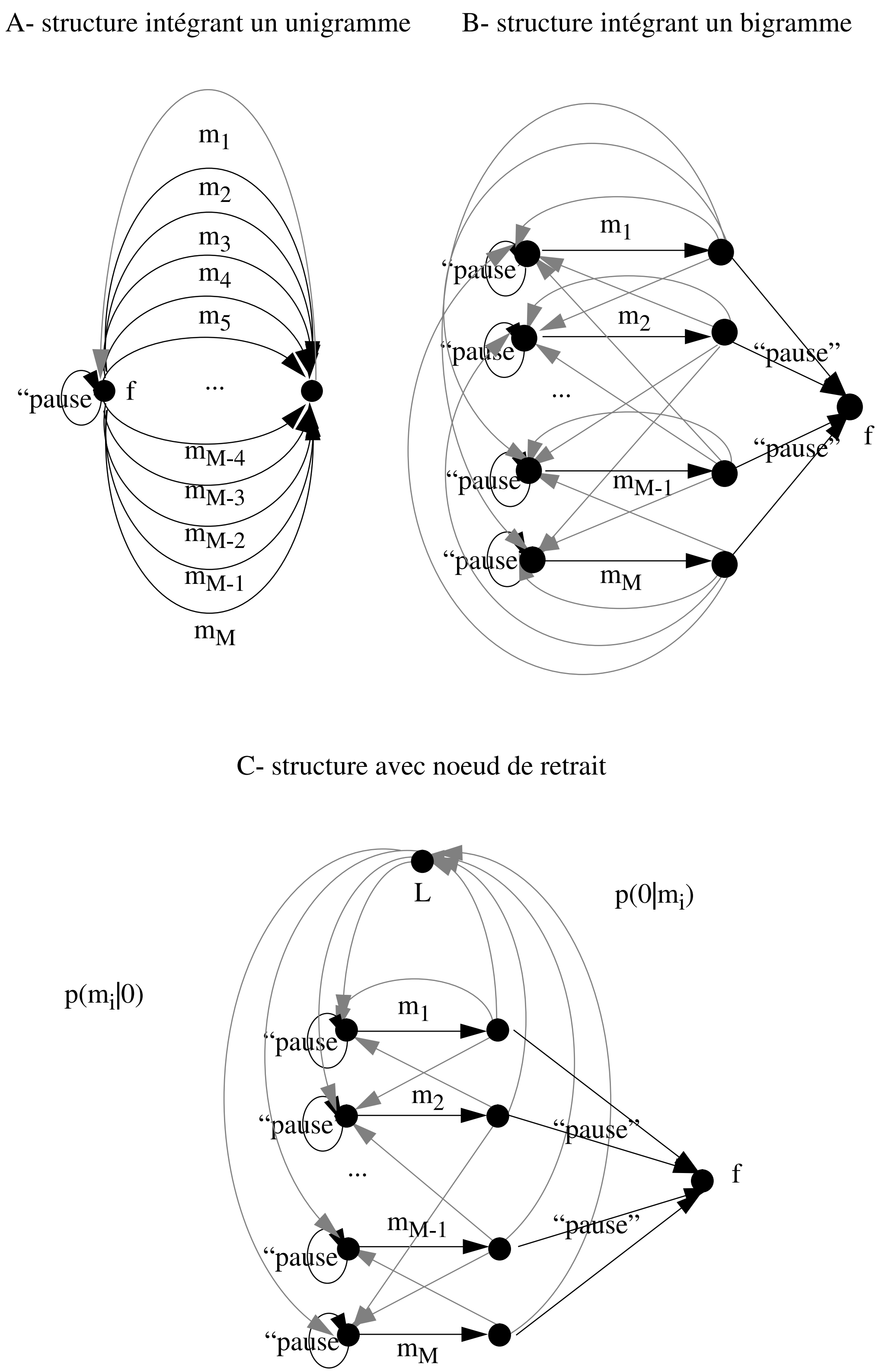


**Figure 7.1 Différents types de structures bouclées**

tous les mots étant inter-reliés de façon séparée, il est alors facile, cette fois, d'y intégrer un modèle de bigramme. Pour ce faire, il suffit que $q_{tr} = P(m_j | m_i)$ pour toutes les transitions vide $tr$ reliant deux mots $m_i$ et $m_j$. Une autre différence marquante entre cette structure et celle illustrée en A est la multiplication des occurrences de pauses.

En effet, la pause est l'une de ces réalisations sonores dont les modèles de langage actuels ne tiennent pas compte. On assume que leur apparition au sein d'un signal de parole continue est imprévisible. Des bruits de toux, des hésitations sonores "*eee*", des bruits de respirations sont d'autres exemples de réalisations tout aussi imprévisibles. Leur présence doit être possible en tout temps et n'avoir aucun impact sur le modèle du langage, c'est-à-dire que $P(m_j | pau, m_i) = P(m_j | m_i)$. En des termes plus graphiques, la présence d'une pause ne doit pas perturber le choix des chemins possibles menant à un mot. C'est ainsi que, si toutes les transitions vides du graphe B passaient par une même occurrence de pause, on obtiendrait un graphe très similaire au graphe A, où tous les mots du lexique partageraient, pour une même unité de temps $t$, une même hypothèse sélectionnée par l'algorithme de Viterbi au noeud f (qui par ailleurs dénote aussi le noeud final de la structure de recherche). La seule solution à ce problème consiste donc à introduire autant d'occurrences de pauses qu'il y a de mots dans le lexique. D'ailleurs, si la détection d'autres réalisations acoustiques intermittentes est souhaitée, le même type de processus doit être répété.

Hélas, bien qu'une structure bouclée de type B améliore énormément les taux de reconnaissance, elle n'est pas efficace du point de vue temps de calcul et ce dû au nombre de transitions vides élevé (de l'ordre du nombre de mots au carré $O(M^2)$).

Pour contrer ce problème, un groupe de BBN ([AUST 90], [PLAC 93]) a développé une méthode de lissage un peu particulière. Selon cette méthode, par ailleurs inspirée de celle développée par [KATZ 87], la probabilité d'un bigramme absent ou peu fréquent $P(m_i | m_{i-1})$ est égal au produit de

$$P(0 | m_{i-1}) = \text{nb\_cont}_{i-1} / (\text{nb\_cont}_{i-1} + \text{nb\_occ}_{i-1}) \qquad \text{(EQ 7.1)}$$

$$\text{et } P(m_i | 0) = \text{nb\_occ}_i / \text{nb\_total} \qquad \text{(EQ 7.2)}$$

où $\text{nb\_total}$ est le nombre total de mots du corpus d'entraînement, $\text{nb\_occ}_i$ le nombre d'occurrence de $m_i$ et $\text{nb\_cont}_{i-1}$ le nombre de contextes $x$ différents $m_{i-1}x$ présent dans le corpus.

On nomme ce bigramme, bigramme de retrait[1] par opposition aux bigrammes dit “réels” qui correspondent aux pairs de mots pour lesquels un nombre suffisant d’occurrences a été observé à l’entraînement. L’avantage de cette méthode réside surtout dans la façon dont elle est intégrée à la structure de recherche proposée par [PLAC 93]. En effet, lorsqu’une telle méthode de lissage est employée, on peut utiliser la structure de recherche illustrée à la figure 7.1C. On constate que cette dernière compte beaucoup moins de transitions vides que son homologue en B, puisque seules les transitions correspondant aux bigrammes réels sont effectivement présentes. Les bigrammes calculés au moyen de la méthode de lissage font maintenant utilisation d’un noeud commun L, appelé noeud de retrait[2].

En prime, on peut augmenter, parfois évidemment au détriment de la perplexité du modèle, le nombre d’occurrences minimum au-dessous duquel un bigramme est calculé par lissage. Ceci a alors comme conséquence, de réduire d’autant le nombre de transitions vides du graphe, et par conséquent, le temps nécessaire à la fouille.

Finalement, récemment Murveit [MURV 94] proposa la structure illustrée à la figure 7.2, structure d’ailleurs maintenant utilisée par plusieurs systèmes basés sur une structure bouclée. La représentation graphique de cette structure étant assez complexe, elle est illustrée au moyen d’un lexique jouet où les seuls bigrammes “réels” possibles sont ceux présents dans la phrase “la lapine a lapé le lait” */la lapin a lapé le lè/*. De plus, les noeuds correspondant au début de chaque mot et la racine de l’arbre de retrait sont dédoublés et les boucles correspondant aux pauses facultatives sont omises.

On peut y voir que les transitions qui auparavant se dirigeaient vers le noeud de retrait, se dirigent maintenant vers une représentation en arbre du lexique, qu’on appelle arbre de retrait (partie grise du graphique). Dans cette topologie un mot $m_i$ ne peut être activé que par un mot $m_{i-1}$ pour lequel un bigramme réel existe. Par exemple *lapine* peut activer *a* directement et *a* activer *lapé* directement. Dans tous les autres cas, ce sont uniquement les premiers phonèmes de l’arbre de retrait qui sont activés. Par exemple *lapine* ne peut activer *lapé* et *a* activer *lapine* qu’en passant par l’arbre de retrait F. Ceci fait que l’emploi d’une structure avec arbre de retrait est plus efficace d’un point de vue temps de calcul par rapport à une structure avec noeud de retrait. En effet dans cette dernière, le nombre de premiers phonèmes activé aurait été beaucoup plus grand, les différentes occurences de ces derniers n’étant pas compressées en une seule par la représentation en arbre.

1. “back-off probability”

2. “back-off node”

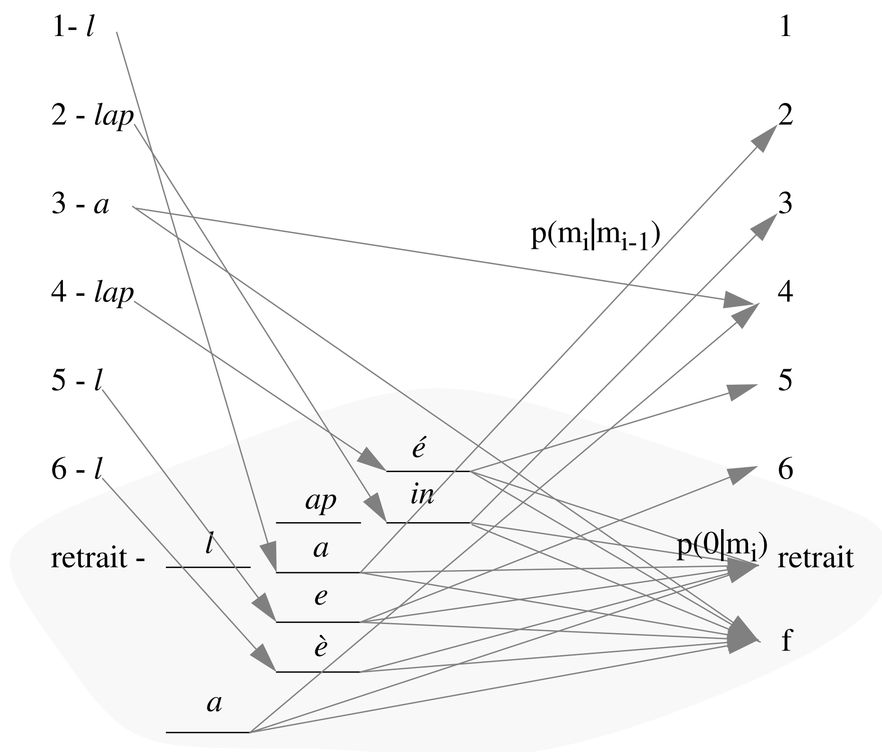


**Figure 7.2 Structure de recherche bouclée avec arbre de retrait**

De plus, $P(m_i|0)$ (équation 7.2) est inclus dans l'arbre de façon incrémentielle, en ajoutant à un noeud la probabilité maximale des mots du sous-arbre défini à partir de ce noeud. La fouille ne se trouve donc pas handicapée par une application reportée du modèle du langage, ce qui aurait été le cas, si le deuxième terme n'avait été inclus dans la fouille que lorsqu'un mot est identifié de façon unique (soit au début du segment menant à la feuille associée au mot).

D'un autre côté, l'utilisation d'un noeud de retrait a comme conséquence que chaque mot doit être présent deux fois, une fois sous une forme qu'on pourrait qualifier de linéaire et activé par des transitions auxquelles corresponde une valeur de bigramme "réel", et une autre fois sous forme compressée dans l'arbre de retrait. Pour éviter que cette duplication ne coûte trop chère en temps et en mémoire, la forme linéaire est, chaque fois que cela est possible, c'est-à-dire pour tous les mots de plus de un phone, combinée à la fin de sa forme compressée.

On retrouve au tableau 7.1 des résultats comparatifs entre temps de calcul nécessaires pour

**Tableau 7.1 Comparaison du temps de calcul entre deux structures bouclées**

| type de structure | grosseur du lexique | nombre de bigrammes réels | temps de calcul | gain |
|---|---|---|---|---|
| noeud de retrait | 2K | 13 910 | 36:35 | |
| arbre de retrait | | | 33:17 | 8.33% |
| noeud de retrait | 5K | 826 072 | 1:48:33 | |
| arbre de retrait | | | 1:18:22 | 27.78% |
| noeud de retrait | 5K | 111 357 | 1:54:29 | |
| arbre de retrait | | | 0:51:16 | 55.26% |
| noeud de retrait | 20K | 4 987 350 | 6:18:47 | |
| arbre de retrait | | | 4:15:22 | 32.54% |

une même tâche lors de l'emploi d'une structure bouclée avec noeud de retrait et structure bouclée avec arbre de retrait. Ces résultats ont été obtenus à partir de 101 phrases des données tests de l'évaluation ATIS de décembre 93 (lexique de 2K), de 53 phrases des données de développement de l'évaluation WSJ de novembre 93 (lexique de 5K), et de 53 phrases des données de développement de l'évaluation WSJ de novembre 94 (lexique de 20K).

Notons que le gain en temps est d'autant plus grand que le lexique est grand (de 8.33% à 32.54%) ce, à cause d'un degré de compression plus élevé au niveau de l'arbre (on y évite un plus grand nombre de duplications), et que le nombre de bigrammes de retrait est grand (27.78% comparé à 55.26%), ce, à cause d'une plus grande utilisation de l'arbre. Dans les faits, lors d'applications réelles, ces deux variables sont liées. En général, on aura qu'un plus grand lexique se traduisant en un plus grand nombre de bigrammes, on aura plus de difficulté à les entraîner, et l'on devra avoir recours au noeud de retrait pour un plus grand nombre d'entres eux.

### 7.1.2 Les structures en arbre

A la structure bouclée, s'oppose la structure en arbre. Lorsqu'utilisée sous sa forme la plus pure (voir l'exemple A de la figure 7.3), chaque branche de l'arbre représente une hypothèse active, et une copie complète du lexique doit être créée à la fin de chaque mot. Pour réduire l'espace mémoire potentiellement énorme que nécessite ce genre de représentation (on le constate d'ailleurs sur l'illustration), le lexique est toujours représenté sous forme

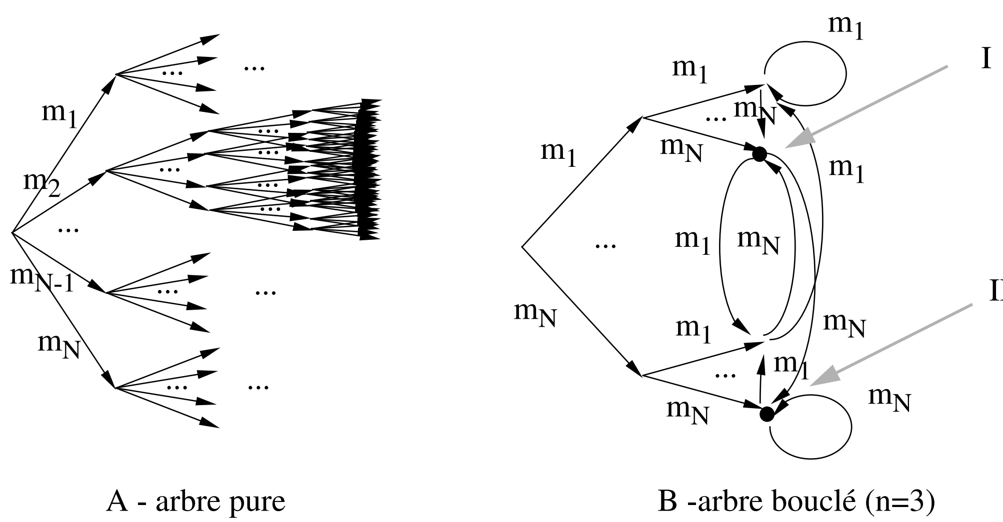


**Figure 7.3 Exemple de structures en arbre**

d'arbre compressé (du genre illustré à la figure 5.1, page 51) de sorte que la croissance de la structure s'effectue phonème par phonème.

L'avantage le plus évident de ce type de structure est qu'elle permet d'incorporer très facilement un modèle probabiliste et ce pour n'importe quelle valeur de $n$, la longueur du contexte utilisé dans le calcul d'un tel ML. On se rappelle en effet, que dans ce genre de modèles, le calcul de $P(m)$ s'effectue essentiellement de manière gauche-droite et que seuls les mots précédant immédiatement un mot $m$ doivent être connus.

Bien qu'elle en permette une intégration facile, il est rare que la structure en arbre pure soit utilisée avec des modèles du langage plus complexes que des trigrammes. C'est que, qu'on se rappelle (voir section 2.6.3), plus le $n$ est grand, plus grande est la quantité de données nécessaire à un bon entraînement. Comme il peut être difficile de trouver et toujours long de traiter de grandes quantités de données, en pratique le $n$ choisi est fixe et petit. Or, lorsqu'on fixe le $n$ maximum pouvant être utilisé, l'on peut employer une structure de recherche en arbre bouclé. Dans ce type de structure, toutes les hypothèses possédant un même passé en regard du $n$ fixé, convergent vers un même noeud. Un exemple d'une telle structure est illustrée (en partie) pour $n = 3$ à la figure 7.3B. On y constate, entre autres, que seuls conduisent au noeud I (flèche grise), des chemins dont les deux derniers mots sont $m_1, m_N$ et au noeud II, des chemins dont les deux derniers mots sont $m_N, m_N$. Ce type de boucle est ce que [ODEL 94] nomme recombinaison comparativement à ce qu'il nomme dominance dans un arbre pur. La structure bouclée illustrée à la figure 7.1B n'est autre qu'une structure en arbre bouclée où $n$ est égal à 2.

Une des différences essentielles entre un modèle bouclé et un modèle en arbre (plus ou moins pure) est qu'il est très facile de prédire l'espace maximale occupé par une structure en boucle et plus difficile de le prédire pour un modèle en arbre. Le choix de l'une ou l'autre des différentes structures de recherche dépend donc principalement des deux facteurs ayant un impact direct sur l'accroissement potentiel de l'espace de recherche: l'efficacité des faisceaux d'émondage utilisés et l'excellence des HMMs utilisés. En effet, meilleurs sont les faisceaux et les HMMs, plus précise sera la fouille et moins grande sera la mémoire nécessaire à une structure d'arbre pur. Par contre, moins bons seront les faisceaux et les HMMs, moins précise sera la fouille et plus grande la mémoire nécessaire à une structure d'arbre pur, rendant alors, plus intéressant l'emploi d'une structure bouclée.

### 7.1.3 Les treillis[1]

La structure d'un treillis (voir figure 7.4) est tout à fait semblable à celle d'un graphe (voir

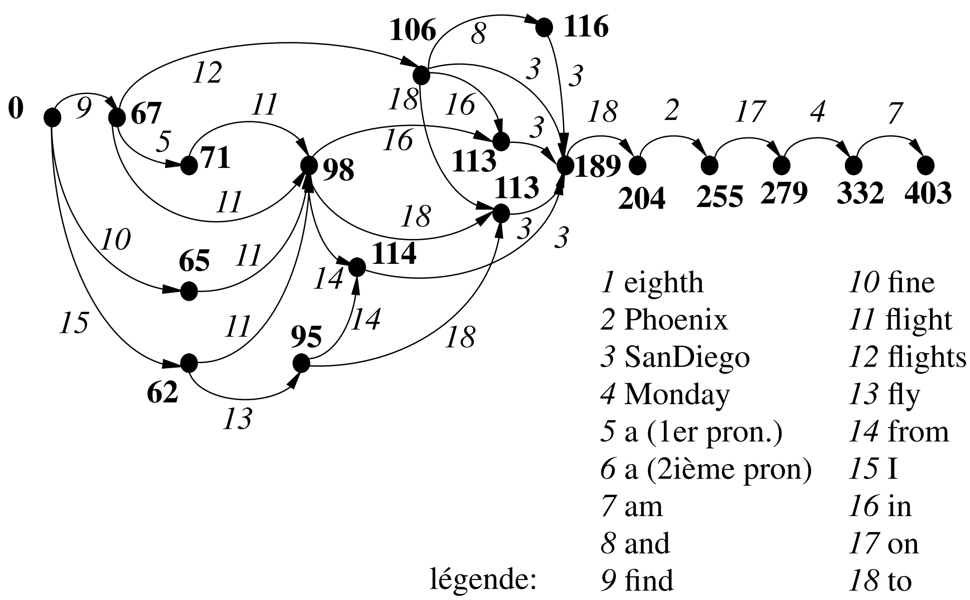


**Figure 7.4 Exemple de treillis**

la figure 2.8, page 22). Ce qui distingue un treillis d'un graphe est le processus ayant mené à sa définition et le fait qu'il puisse explicitement contenir des indications temporelles ou des scores. Ainsi à chaque noeud de celui illustré à la figure 7.4, est indiqué l'unité auxquels les mots correspondant aux transitions arrivant à ce noeud, cessent d'être actifs.

1. "lattices"

Ainsi, si les graphes sont le reflet direct de la définition d'un CFG, les treillis sont plutôt les produit dérivés de reconnaissances initiales effectuées sur des structures en arbre ou bouclées.

Le mécanisme général conduisant à leur définition est le suivant. Une (ou plusieurs) étape(s) de reconnaissance est (sont) effectuée(s) sur une (ou plusieurs) structure(s) représentant l'ensemble de l'espace de recherche. Durant cette (ou ces) étape(s) diverses informations sont amassées et ce pour chaque énoncé. Ces dernières peuvent par exemple prendre la forme de liste de mots ou paires de mots actifs auxquels peuvent être associés ou non différentes indications temporelles et/ou différents scores. Un exemple du type d'information qui peut être associé à un énoncé est donné à la figure 7.5. Il illustre les

```
FIND FLIGHT FROM SAN DIEGO TO PHOENIX ON MONDAY A M
19
0 start_end -1 0 3 (9, 0, 67, 0, 0) (10, -20767, 65, 0, 0) (15, -16549, 62, 0, 0)
1 EIGHT 512 0 1 (7, -12030, 403, 336, 341)
2 PHOENIX 1194 0 1 (17, 0, 279, 252, 258)
3 SANDIEGO 1371 0 1 (18, 0, 204, 185, 192)
4 MONDAY 1027 0 2 (1, -12030, 338, 316, 325) (7, 0, 403, 328, 336)
5 A 11 0 1 (11, -29249, 98, 71, 71)
6 A 11 1 1 (11, -17407, 98, 70, 71)
7 A_M 141 0 1 (0, 0, -1, 399, 403)
8 AND 79 1 1 (3, -23579, 189, 115, 116)
9 FIND 587 0 4 (5, -29249, 71, 68, 68) (6, -17407, 71, 67, 68) (11, 0, 98, 66, 73) (12, -7385,
106, 66, 72)
10 FINE 589 0 1 (11, -20767, 98, 64, 67)
11 FLIGHT 600 0 4 (3, -29906, 189, 113, 113) (14, 0, 114, 96, 103) (16, -18203, 113, 98,
108) (18, -30, 113, 96, 103)
12 FLIGHTS 601 0 6 (3, -24545, 189, 106, 109) (8, -23579, 116, 105, 107) (14, -10820, 114,
100, 102) (16, -7385, 113, 104, 108) (18, -13431, 113, 100, 102) (18, -25491, 113, 105, 107)
13 FLY 604 0 2 (14, -26456, 114, 95, 95) (18, -16549, 113, 93, 96)
14 FROM 632 0 1 (3, 0, 189, 114, 116)
15 I 718 0 2 (11, -26255, 98, 61, 63) (13, -16549, 95, 60, 64)
16 IN 730 0 1 (3, -7385, 189, 112, 115)
17 ON 1122 0 1 (4, 0, 332, 275, 283)
18 TO 1655 0 2 (2, 0, 255, 203, 207) (3, -30, 189, 108, 115)
```

**Figure 7.5 Exemple possible d'un extrait d'informations utilisées pour la construction d'un treillis associé à un énoncé**

informations obtenues pour un énoncé tiré des données de la tâche ATIS. Puis directement à partir de ces informations ou après traitement un treillis est construit. Ce sont d'ailleurs ces dernières, qui utilisées selon certaines règles, ont permis la construction du treillis illustré à la figure 7.4. Une description exhaustive des différents champs ainsi qu'un aperçu des règles utilisées dans la construction du treillis sont donnés à la section 8.3.3.

En conséquence, et contrairement aux graphes qui sont, étant donnée une application, construits à priori, les treillis sont construits lors de la reconnaissance d'un message $m$. Ils correspondent à un sous-ensemble de l'espace de recherche initial, sous-ensemble qui, pour ce message, semble le plus prometteur. Les treillis sont donc le produit d'un genre de pairage rapide pour la parole continue. Ce pairage rapide doit d'ailleurs faire face au même type de compromis temps/efficacité auquel est soumis le pairage rapide en mots isolés, on y reviendra à la section 8.1.2.

Ceci dit, il existe des auteurs pour dire qu'un treillis ne peut contenir de boucle, par opposition aux graphes qui eux peuvent prendre n'importe quelle topologie. Pour d'autres encore, un treillis peut contenir des boucles à condition que des étiquettes de temps soient explicitement associées aux transitions ou aux noeuds de ce dernier. Ne voulant point me mêler à cette mini-polémique[1], j'ai choisi de définir un treillis non pas selon des critères topologiques, très variables selon les références, mais plutôt selon sa méthode de production qui elle, est commune à tous.

Les treillis sont principalement utilisés de deux façons. D'abord comme méthode rapide de test, par exemple, pour essayer de nouveaux algorithmes d'adaptation au bruit ou au locuteur, ou pour déterminer la valeur optimale de certains paramètres numériques tel la largeur d'un faisceau d'émondage ou les poids attribuables aux différents types de modèles utilisés dans un système (modèle acoustique, ML probabiliste, grammaire déterministe, modèle du dialogue) (voir entre autres [RICH 95]). En effet, pour de très grands vocabulaires, chaque tâche de reconnaissance peut demander un temps assez long (jusqu'à plusieurs centaines de fois temps réel pour certains systèmes). Les treillis sont alors un moyen avantageux de tester de nouvelles théories plus rapidement, l'hypothèse étant que si ces théories présentent un quelconque avantage au niveau d'un sous-ensemble de l'espace de recherche, les chances sont qu'elles en présentent aussi au niveau de l'espace de recherche total. Par exemple, si on constate lors de tests de reconnaissance utilisant un treillis, qu'un certain rapport entre un bigramme et un modèle acoustique est idéal, ce même rapport pourra éventuellement être utilisé lors des autres étapes de reconnaissance portant sur l'ensemble de l'espace de recherche comme les passes initiales utilisant des

1. Pour la petite histoire, un des groupes Internet auquel appartiennent les participants au test NAB (section 8.2.3) a vue plusieurs messages sur le sujet, voir aussi [RICH 95].

structures bouclées ou en arbre. Or en venir à la même conclusion en utilisant ces structures, auraient demandé un temps beaucoup plus long.

On peut aussi utiliser les treillis directement lors de l'étape de reconnaissance. Leur petite taille permet alors l'emploi de modèles tant acoustiques, comme des HMMs de triphones inter-mots (section 3.1.4), que du langage, comme des trigrammes (section 3.1.3), qui se seraient avérés trop coûteux en temps et/ou espace mémoire pour être incorporés aux structures initiales.

## 7.2 Algorithmes de fouille

On peut classer les algorithmes employés dans les systèmes de reconnaissance à parole continue grand vocabulaire en deux familles: les algorithmes à une passe et les algorithmes multi-passes. Dans les algorithmes à une passe les données sont traitées en une seule fois de gauche à droite au fur et à mesure qu'elles sont disponibles. Plusieurs des systèmes qui utilisent ce type d'algorithmes sont dérivés de systèmes de reconnaissance en mots isolés obtenus après adaptation des algorithmes de pré-sélection (pairage rapide). C'est le cas des systèmes du CSELT, de IBM et de Dragon.

Dans les algorithmes multi-passes, les données sont traitées plusieurs fois, en général en alternance de gauche à droite et de droite à gauche. Les résultats recueillis à une étape sont alors utilisés par les étapes subséquentes jusqu'à l'obtention, à la dernière étape, des résultats finaux.

Chaque famille d'algorithmes a évidemment ses avantages et ses désavantages. Ainsi, les algorithmes à une passe permettent de traiter des énoncés qu'on pourraient qualifier "d'infinis". En effet, puisqu'aucune information autre que l'historique des chemins ne doit être conservées (information que le mécanisme de retour arrière partiel [SPOH80] permet d'ailleurs de minimiser, en permettant de ne conserver qu'une instance des historiques communes à certaines hypothèses), de très longs énoncés peuvent être traités sans problème de façon ininterrompue. Ceci n'est pas le cas des algorithmes multi-passes, dans lesquels il est en général nécessaire, lors des premières passes, d'amasser de nombreuses informations pour chaque unité de temps. Toujours grâce au mécanisme du retour arrière partiel, il est possible, avec des algorithmes à une passe, de produire des résultats partiaux en cours de reconnaissance. Seul ce type d'algorithme peut prétendre s'inspirer des mécanismes de compréhension de la parole utilisés par les humains. Par contre, puisque pour éviter une explosion de l'espace de recherche, ils doivent utiliser de nombreux heuristiques d'émondage, ils sont plus sujets aux erreurs induites par ces méthodes. Ils ne peuvent récupérer d'un mauvais émondage. Finalement, il est moins facile de générer plus d'une solution à partir de ce type d'algorithme, et si cela est fait, les solutions obtenues ont souvent tendance à être issues d'un passé commun et donc à se ressembler.

Les algorithmes multi-passes ont évidemment les défauts des qualités des méthodes à une passe. Leur utilisation rend plus difficile le traitement de très longs énoncés et l'obtention de résultats partiaux. Ils ne peuvent absolument pas prétendre imiter le comportement humain, mais par contre sont moins sujets aux erreurs d'émondage.

Les algorithmes à une passe favorisent l'utilisation des structures d'arbre pures ou bouclés mais peuvent aussi être utilisés de concert avec des structures bouclées. De leur côté, les algorithmes multi-passes sont en général utilisés de concert avec des structures bouclées bien qu'il existe au moins un exemple où ils sont utilisés avec une structure d'arbre pure (voir section 7.3.4). Eux seuls, par contre, font utilisation de treillis à la reconnaissance.

## 7.3 Description de systèmes

Les sections suivantes décrivent les systèmes développés par les principales équipes de recherche. Ces descriptions permettront de mieux comprendre l'interaction entre structures et algorithmes de fouille et de voir comment chaque type de l'un peut se combiner avec chaque type de l'autre.

### 7.3.1 Système du laboratoire Lincoln au MIT

Le système du MIT Lincoln Laboratory est l'un des représentants des algorithmes à une passe [PAUL 91], [PAUL 92a], [PAUL 94] et [PAUL 95]. C'est un système asynchrone dans lequel des hypothèses partielles sont conservées triées sur une pile. L'hypothèse occupant le haut de la pile est considérée la plus prometteuse (selon son score total, pondéré par sa longueur). A chaque étape de l'algorithme, un mot est rajouté à l'hypothèse sur le haut de la pile. Les mots pouvant être rajoutés à une hypothèse sont obtenus par un pairage rapide à deux étapes effectué localement. Pour un dictionnaire d'environ 20000 mots, une vingtaines de mots en moyenne sont obtenus du pairage rapide.

Un modèle de bigramme est utilisé tant au niveau du pairage rapide que du pairage exact. Les HMMs utilisés lors du pairage rapide sont des diphones gauches (voir section 2.5.1.1, page 16), les HMMs utilisés lors du pairage exact, des triphones. Pour sauver de l'espace sur la pile, les histoires communes à plusieurs hypothèses sont amalgamées. La profondeur moyenne atteint par la pile pour des énoncés standards (une douzaine de secondes) est d'une centaine d'hypothèses.

### 7.3.2 Système de l'université Carnegie Mellon

Le système de l'université Carnegie Mellon (CMU) est connu sous le nom de Sphinx-II. C'est un système à quatre passes. La première passe consiste en une fouille Viterbi gauche-droite. Un modèle de bigramme dynamique et des triphones inter-mots à base de "senones" [HUAN 93b] y sont utilisés. Le résultat de cette première passes consiste en un treillis. Ce dernier se compose d'intervalles d'activation de mots, intervalles caractérisés par un unité de temps du début d'activation et plusieurs unités de temps et scores de fin d'activation d'un mot. La deuxième passe, effectuée de droite à gauche est tout à fait semblable à la première sauf qu'elle est guidée par le treillis construit lors de cette dernière. Le résultat de cette deuxième passe consiste aussi en un treillis mais contenant, cette fois pour chaque activation d'un mot, un unité de temps de fin d'activation et plusieurs unités de temps et scores de début d'activation [ALLE 93].

Ce treillis est ensuite fouillé au moyen d'un algorithme de type A* [HUAN 93a]. La progression d'une hypothèse se fait un mot à la fois. Cette fouille ne nécessite aucun re-scoring acoustique. Une centaine de solutions sont générées qui sont finalement re-scorées par un trigramme, dynamique lui aussi, lors d'une dernière passe. Lors des deux premières passes, trois faisceaux d'émondage sont utilisés: un sur les états de HMMs, l'autre sur les phonèmes et le troisième sur les mots. Finalement, caractéristique un peu spéciale, des HMMs en contexte distincts sont utilisés pour modéliser les acronymes, certaines expressions et les mots composés d'un seul phonème.

### 7.3.3 Système de SRI International

Prénommé Decipher, le système du SRI est basé sur un algorithme 4-passes ([MURV 93] [MURV 94], [DIGA 95]). La première passe est effectuée de gauche à droite, la seconde de droite à gauche. Lors de la première passe, consistant en l'algorithme de Viterbi, les informations suivantes sont mémorisées: soit le score $\alpha_{tm_i}$ de chaque hypothèse active à chaque fin de mot $m_i$ pour chaque unité de temps $t$, et le score total de la meilleure hypothèse $\Psi_{max} = \alpha_{max}$. Lors de la seconde passe, l'algorithme de Viterbi est effectué sur les seules hypothèses pour lesquelles on a

$$H_{t(m_i, m_j)} = \alpha_{tm_i} + \beta_{tm_j} \qquad \text{(EQ 7.3)}$$

compris entre $\Psi_{max}$ et $\Psi_{max} - \Delta_H$, où $\Delta_H$ est la largeur d'un faisceau d'émondage qu'on désignera sous le nom de faisceau heuristique, et $\beta_{tm_j}$ le score au temps $t$ de l'hypothèse active au début du mot $m_j$ lors de la fouille droite-gauche. Lors de cette deuxième passe, l'on obtient donc une liste des périodes de haute probabilité (ou si on préfère d'activation) des mots du lexique. Par la suite, cette liste est utilisée pour construire un treillis de recherche utilisé pour générer au moyen du "word dependent N-best" une liste des N-meilleurs solutions. Ces dernières sont éventuellement re-scorées au moyen de modèles plus performants dont des triphones inter-mots et un trigramme.

Des HMMs sexués[1] sont utilisés lors de toutes les passes. Les deux premières passes, ainsi que la génération des N-meilleures hypothèses est fait en utilisant des triphones intra-mots et un bigramme. La structure utilisée lors des deux premières passes est, bien évidemment, une structure bouclée avec arbre de retrait.

### 7.3.4 Système de l'université Cambridge

Le système développé en Angleterre à l'université Cambridge est basé sur un algorithme à deux passes, où la première passe se distingue par le fait qu'elle est effectuée de droite à gauche et ce, de façon à pouvoir être omise. Le cas échéant la deuxième passe devient alors l'unique passe gauche-droite du système (voir [ODEL 94], [VALT 94] et [WOOD 95]). La passe facultative droite-gauche utilise des HMMs simples et aucun modèle de langage. Elle est effectuée sur une structure bouclée simple. Elle permet de mémoriser, tout comme dans l'algorithme du SRI, les scores des derniers mots actifs des hypothèses non optimales pour chaque unité de temps.

Ces informations peuvent alors être utilisées, tout comme lors de la deuxième passe du SRI, pour guider la passe gauche-droite du système. Cette passe consistant en un Viterbi classique, s'effectue cependant sur une structure d'arbre pure. Un modèle de 4-gramme est donc facilement incorporé à la fouille. Par contre, pour éviter l'explosion liée à ce type de fouille, trois méthodes d'émondage sont employées: un faisceau d'émondage traditionnel (section 3.1.2), un faisceau de fin de mot (section 9.1.2) et, tout comme dans le système de Philips, une borne sur le nombre de noeuds possiblement actifs à chaque unité de temps (section 9.2).

1. Un HMM sexué est un HMM exclusivement entraîné sur données de locuteurs de même sexe.

### 7.3.5 Système de BBN Systems and Technologies

Cela fait déjà une dizaine d'années que BBN travaille au développement de systèmes RAP en parole continue. L'actuel système, du nom de Byblos, se compose de cinq passes ([BATE 93] et [NGUY 94]). La première passe est un pairage rapide servant à mémoriser des scores de fin de mots. La seconde, une passe droite-gauche, guidée par les scores de la première passe, est utilisée pour mémoriser les scores et unités de temps du début des périodes d'activation de tous les mots actifs au cours de la reconnaissance. Elle est suivie par une passe identique mais effectuée de gauche à droite et servant à mémoriser les scores et unités de temps de la fin des périodes d'activation de tous les mots actifs au cours de la reconnaissance. Au cours de ces deux dernières passes, un bigramme et des triphones inter-mots sont utilisés. C'est une structure bouclée avec noeud de retrait qui est utilisée lors des trois premières passes.

Puis, à partir des informations mémorisées lors des deuxième et troisième passes, un treillis est construit auquel est incorporé un trigramme et des HMMs de triphones inter-mots. La quatrième passe utilise le "word-dependent N-best" pour générer, à partir des treillis, une liste des N-meilleures solutions. Finalement, chacune des solutions générées est re-scorée en utilisant des modèles acoustiques et de langage plus puissants.

### 7.3.6 Système du LIMSI

Le système développé au LIMSI ([GAUV 94a], [GAUV 94b], [GAUV 94c] et [GAUV 94d]) est basé sur un algorithme multi-passes. Le sexe présumé d'un locuteur est déterminé à priori au moyen de quelques unités de temps de parole, de sorte que la reconnaissance s'effectue dès le début au moyen de HMMs sexués.

Lors d'une première passe gauche-droite au sein d'un structure bouclée avec arbre de retrait, les scores et derniers mots actifs des hypothèses non optimales sont mémorisés, tout comme dans le système du SRI. Cependant on y associe aussi l'unité de temps de début du dernier mot. Ces informations sont ensuite directement utilisées pour construire un treillis de mots incorporant un modèle de trigramme. Puis ce treillis est fouillé toujours de gauche à droite pour obtenir le résultat final. Un faisceau d'émondage standard est appliqué lors des deux passes de reconnaissance. Un seuil sur le nombre de segments contenus dans le treillis est aussi utilisé. Ceci veut dire que certaines des hypothèses partielles peuvent ne pas être utilisées lors de la construction des treillis.

Aspect intéressant et original, des règles représentant certains phénomènes phonologiques tant à l'intérieur des mots (comme la présence de phonèmes optionnels) qu'entre les mots (comme la palatisation) sont appliquées lors des deux étapes de reconnaissance. Il faut aussi souligner que le LIMSI développe parallèlement à son système de RAP anglais un système de RAP français. Ce dernier ne diffère cependant du système anglais qu'au niveau du dictionnaire, des phonèmes, des données d'entraînement et bien entendu des règles phonologiques utilisées.

### 7.3.7 Système de Philips

Le système développé depuis déjà quelques années aux laboratoires Philips en collaboration avec l'université de Aachen est basé sur un algorithme multi-passes ([STEI 93], [DUGA 95]). La première passe est effectuée sur une structure de recherche en arbre pure. En plus du traditionnel faisceau d'émondage, plusieurs autres méthodes sont utilisées afin de contraindre l'espace de recherche qui, on le rappelle encore, grossit rapidement dans ce type de structure. Ainsi, une limite est imposée sur le nombre total d'hypothèses actives à chaque unité de temps [STEI 94]. Une méthode de pairage rapide au niveau des phonèmes est aussi appliquée à chaque deux unités de temps. Elle consiste, au temps $t$, à calculer le score de chaque phonème pour les unités de temps $t$ à $t+5$, de sorte à éliminer, à priori, les mots débutant ou se poursuivant par un phonème peu prometteur. Finalement, un unigramme est introduit de façon incrémentielle dans l'arbre (de la même façon que le terme de l'équation 7.2 l'était dans la structure à arbre de retrait, voir la section 7.1.1).

Au fur et à mesure que progresse cette première passe, un treillis est construit [NEYA 94]. Le fait que cette construction ait lieu durant la première passe est une des particularités de cette approche. Ce treillis, qui inclut des prononciations multiples, est finalement fouillé lors d'une deuxième passe. Lors de cette dernière passe, le treillis est dynamiquement explosé pour introduire un modèle de trigramme à la fouille. Cette façon de faire permet, au détriment d'un peu de temps de calcul, de réduire l'espace mémoire nécessaire à la représentation du treillis.

### 7.3.8 Système Dragon

Le système développé pour la parole continue chez Dragon est un des exemples de systèmes de RAP en parole continue ayant été conçu à partir d'un système de RAP en mots isolés. C'est un système synchrone à base de triphones en contexte (inter et intra-mots) [ROTH 95]. Dans les articles disponibles décrivant le système, aucune mention n'est fait de la structure de recherche utilisée mais tout porte à croire qu'il s'agit d'un arbre soit pure, soit bouclé. Dragon est, il ne faut pas l'oublier, une entreprise privée dont les produits sont essentiellement constitués de systèmes de RAP. Un trigramme est incorporé à la structure de recherche. La fouille est effectuée en une passe et un algorithme de pairage rapide est utilisé, version améliorée de celui développée pour leur système en mots isolés [GILL 92]. Détail intéressant, l'entraînement des HMMs s'effectue à partir d'unités de temps de 10 millisecondes (longueur quasi standard) mais la reconnaissance à partir d'unités de temps de 20 millisecondes, ce qui permet de couper de moitié le temps de reconnaissance. On se rappelle on effet, que le nombre d'unités de temps est l'une des variables auquelle le temps d'exécution de l'algorithme de Viterbi est directement proportionnel.

### 7.3.9 Système de IBM

Le système d'IBM est un autre exemple de systèmes de RAP en parole continue ayant été conçu à partir d'un système de RAP en mots isolés. C'est un système utilisant un algorithme de fouille asynchrone et une structure de recherche en arbre bouclé. L'algorithme de fouille est appelé "envelope search" [GOPA 95]. La progression des hypothèses se fait un mot à la fois. Les mots candidats pouvant être utilisés pour cette progression, à un instant donné, sont obtenus au moyen d'un pairage rapide similaire à celui utilisé dans la version du système en mots isolés. Comme aucun ML n'est appliqué lors de ce pairage rapide, le dictionnaire utilisé par ce dernier peut être totalement différent de celui utilisé lors de l'entraînement du ML. C'est ainsi qu'on peut y ajouter facilement de nouveaux mots, en acceptant bien entendu, que la valeur de ML de ces derniers, soit fixé de façon plus ou moins arbitraire [BAHL 95].

### 7.3.10 Système de AT&T

Le système de AT&T est un système à deux passes. La deuxième, un Viterbi classique, consiste à re-scorer au moyen de triphones inter-mots et d'un 5-gram un treillis obtenu de la première passe [LJOL 95]. La première est un peu spécial [PERE 94]. Elle se compose de quatre étapes en cascade. Une première reconnaissance acoustique utilisant des HMMs très simples, en fait des modèles de phonèmes, permet de construire un treillis de syllabes utilisé à son tour pour construire un treillis de mots, re-scoré à l'aide d'un bigramme pour donner le treillis final traité par la passe suivante. La grosseur de ce dernier treillis est contrôlé au moyen d'un faisceau d'émondage.

### 7.3.11 Autres systèmes

Il existe encore sûrement bien d'autres systèmes de RAP grand vocabulaires qui, ne participant pas aux évaluations associées aux tâches ARPA jouissent d'une moins grande publicité que les autres. Il faut dire que n'étant pas évalués à partir de tests standardisés, ils est plus difficile d'estimer leur valeur. Malheureusement c'est le cas de la majorité des systèmes conçu pour la RAP de langues autres que l'anglais, à moins que leurs concepteurs prennent le temps d'adapter leur système à l'anglais et participe aux évaluations associées aux tâches ARPA. C'est le cas de Philips, du LIMSI et du CRIM. On peut, cependant, déplorer que de tels évaluation n'existent que pour l'anglais.

Entre autres systèmes intéressants mais ne participant pas aux évaluations associées aux tâches ARPA, on peut mentionner celui de l'INRS, que caractérise la capacité de traiter des énoncés de plusieurs heures ([KENN 94], [LIKE 95]); celui du CSELT, pour l'italien, un exemple d'adaptation réussie d'un système conçu pour la parole en mots isolés à la RAP en continue (pour des vocabulaires d'environs 1000 mots) ([LAFA 95]) et d'autres encore qui traitant des langues non indo-européennes emploient parfois des approches particulières. C'est le cas par exemple, des systèmes RAP pour le japonais et le chinois.

# 8 Description du système de RAP grand vocabulaire du CRIM

Dans le cadre de ce travail et afin d'étudier les divers aspects et problèmes reliés à la reconnaissance de la parole continue à grand vocabulaire, un système de type multi-passes de reconnaissance a été développé au CRIM. Bien qu'en partie constitué de structures de recherche ou algorithmes de fouille empruntés de travaux effectués par d'autres équipes de recherche, il constitue, par l'assemblage qu'il en a été fait, en un système orignal se distinguant en particulier par sa simplicité et sa robustesse. Ce système a été utilisé lors de l'évaluation de novembre 94 associée à la tâche ARPA du WSJ. Il a aussi servi de base à une étape préliminaire de la deuxième phase du projet TransTalk.

De plus, plusieurs des mécanismes élaborés lors de son développement, se sont avérés utiles dans d'autres situations. En particulier, certaines informations amassées lors d'une des étapes intermédiaires ont été utilisées de façon originale pour l'entraînement MMIE (section 8.3.2), une méthode originale d'intégration d'algorithmes d'adaptation (section 8.3.1) et une méthode originale de rejet globale (section 8.3.3.1).

## 8.1 Description du système multi-passes

Le système est basé sur un algorithme multi-passes comportant un total de 4 étapes distinctes, dont 3 passes de reconnaissance et une étape de compression/expansion. La dernière passe de reconnaissance peut éventuellement prendre une autre forme, de sorte à générer une liste de solutions plutôt qu'une solution unique. Les structures et algorithmes de recherche utilisés à chaque étape sont différents et adaptés au type de reconnaissance effectuée. Les deux premières passes de reconnaissance sont similaires à celles décrites dans [MURV 93], mais les données collectées sont utilisées de façon fort différente. En fait, la façon dont ces données sont utilisées est l'un des aspect qui distingue le système du CRIM des systèmes décrits au chapitre précédent. Ce dernier est conçu, à l'inverse des systèmes une passe, dans le but d'offrir la plus grande robustesse possible à de mauvais modèles acoustiques.

### 8.1.1 Première passe de reconnaissance

La première passe de reconnaissance est effectuée sur une structure bouclée avec arbre de retrait tel qu'illustrée à la figure 7.2. A noter cependant que la compression au niveau de l'arbre est basée sur les distributions et non les phonèmes ou les triphones. Ceci a un impact particulièrement important lorsque les différents HMMs partagent certaines distributions ou lorsque des HMMs étatiques sont utilisés (voir section 2.5.1.3). On peut en effet constater, au tableau 8.1, qu'une compression basée sur les distributions résulte en

**Tableau 8.1 Comparaison du taux de compression[a]**

| type de compression | grosseur du lexique | nombre de cellules | temps de calcul | gain espace |
|---|---|---|---|---|
| distributions | 2K | 23289 | 0:19:24 | |
| triphones | | 25467 | 0:33:17 | 8.55% |

a. tests effectués sur 101 énoncés de l'évaluation ATIS de décembre 93 - HMMs discrets

une réduction de 8.55% du nombre de cellules nécessaires pour représenter l'arbre de retrait. Plus dramatique est la réduction obtenue au niveau du temps de calcul, soit une réduction de près de 50%. C'est que, on se rappelle, c'est au début des mots qu'un plus haut degré de compression a le plus d'impact sur le temps de fouille lorsqu'on utilise un faisceau d'émondage (voir section 7.1.1). Or, c'est justement à cet endroit que l'impact dû à une compression au niveau des distributions a le plus de chance de se faire sentir, car ce sont les distributions des extrémités que les HMMs de triphones risquent le plus de partager. Par exemple, il est fort possible que les triphones *b=a=i* et *b=a=l*, ayant un même contexte gauche, partagent leurs premières distributions et puissent donc être, en partie, amalgamés, ce qui n'aurait pas été le cas lors d'une compression basée sur les triphones.

La structure bouclée inclut tous les mots du vocabulaire et est parcourue lors de la première passe au moyen de l'algorithme de Viterbi. Trois types de faisceau sont alors utilisés: un faisceau standard (section 3.1.2), un faisceau de début de bigramme (section 9.1.1) et un faisceau de fin de mot (section 9.1.2). A chaque unité de temps $t$, le score $\alpha_{tm}$ de chaque hypothèse active à chaque fin de mot $m$ est mémorisé. Seuls des HMMs intra-mots sont utilisés lors de cette passe.

### 8.1.2 Deuxième passe de reconnaissance

La deuxième passe droite-gauche de reconnaissance consiste en l'algorithme de Viterbi guidé par les résultats obtenus lors de la première passe. Ces informations sont utilisées d'une façon fort similaire à celle employée dans le système de Murveit [MURV 93]. Ce qui distingue la passe arrière du système du CRIM de celle du système du SRI, sont les informations amassées lors celle-ci. Ces dernières ne consistent qu'en la liste des paires de mots actives à un moment donné de la fouille. A chaque paire $(m_i, m_j)$ est associée la valeur minimum $\Delta(m_i, m_j) = H_{t(m_i, m_j)} - \Psi_{max}$ obtenue sur l'ensemble des unités de temps pour lesquelles cette paire de mots donnée est active. On se rappelle que $H_{t(m_i, m_j)}$ est égal à la somme des scores $\alpha_{tm_i}$ et $\beta_{tm_j}$, et $\Psi_{max}$ est le score de la solution obtenue lors de la passe gauche-droite (voir section 7.3.3). Par contre, contrairement à l'algorithme du SRI, aucune information relative au temps d'activation des paires de mots n'est mémorisée[1]. La fouille lors de cette deuxième passe est effectuée sur une structure bouclée du type illustrée à la figure 7.1B mais où les seules transitions actives entre deux mots, pour une unité de temps donnée, sont celles liant les deux mots des paires de mots actives à cette unité de temps. Ces dernières étant en général peu nombreuses, on évite l'explosion du temps de calcul généralement associée à ce type de structure. La figure 8.1

```
19
0 start_end -1 0 2 (16, -5378) (17, 0)
1 ANOTHER 227 0 3 (8, -1858) (9, 0) (13, -3955)
2 BANK 397 0 1 (10, 0)
3 COMPANIES 870 0 1 (15, -3136)
4 COMPANY 872 0 1 (15, 0)
5 COMPANY'S 873 0 1 (15, -3323)
6 DOLLAR 1307 0 1 (14, 0)
7 DOLLARS 1309 0 1 (14, -5048)
8 FIFTEEN 1697 1 1 (11, -1858)
9 FIFTY 1701 0 1 (11, 0)
10 HOLDING 2080 0 3 (3, -3136) (4, 0) (5, -3323)
11 MILLION 2769 0 2 (6, 0) (7, -5048)
12 NEXT 2921 0 1 (18, 0)
13 OF 2997 0 2 (8, -4920) (9, -3955)
14 SALE 3871 0 1 (12, 0)
15 SCHEDULED 3899 0 1 (1, 0)
16 THE 4497 0 1 (2, -5378)
17 THE 4497 1 1 (2, 0)
18 TUESDAY 4643 0 1 (0, 0)
```

**Figure 8.1 Exemple de l'information mémorisée lors de la deuxième passe du système multi-passes du CRIM**

illustre le type d'informations mémorisées lors de cette deuxième passe, pour une phrase tirée des données de la tâche WSJ: "*The bank holding company scheduled another fifty million dollar sale next tuesday*". *19* est le nombre de listes produits pour la phrase. Cha-

1. Du moins dans la version utilisée dans l'algorithme multi-passes.

que liste contient comme premier champs un numéro de référence, par exemple *10* à la douzième ligne, l'orthographe du mot: *HOLDING*, son numéro dans un dictionnaire de référence: *2080*, son numéro de prononciation: *0* et le nombre de mots avec lequel il a pu être connecté: *3*. Pour chacun de ces mots, on retrouve entre parenthèses un numéro de référence et la valeur (en probabilité logarithmique) de $\Delta(m_i, m_j)$. Ainsi les trois mots auxquels *HOLDING* a pu être connecté sont *(3 (COMPANIES), -3136), (4 (COMPANY), 0)* et *(5 (COMPANY'S), -3323)*. "*start_end*" est un indicateur spécial représentant le début (lorsque membre gauche d'une paire) et la fin (lorsque membre droit d'une paire) de l'énoncé.

La valeur de $\Delta(m_i, m_j)$ est mémorisée pour chaque paire de mots, car elle est un indicateur du degré de "certitude" d'une paire. En effet, plus cette valeur est élevée, plus l'hypothèse auquelle appartient la paire est prometteuse, et moins cette valeur est élevée, moins l'hypothèse auquelle appartient cette paire est prometteuse. $\Delta(m_i, m_j)$ permet donc de classer entre elles les diverses paires, ce qui peut être utile. Ainsi, on peut choisir de ne générer qu'une fois, en utilisant de larges faisceaux, une grande quantité de paires, lesquelles on pourra individuellement décider d'utiliser ou non lors de traitements ultérieurs.

En effet, le nombre de paires obtenues lors de la deuxième passe dépend de la largeur du faisceau heuristique $\Delta_H$. Or, il est important, lors d'approches multi-passes, que les informations propagées d'une passe à l'autre permettent une bonne réduction de l'espace de recherche tout en incluant la bonne solution. Il s'agit donc de choisir $\Delta_H$ de sorte à minimiser le nombre de paires de mots possibles tout en essayant qu'y soient inclus les paires de mots de la solution.

Dans le but de bien choisir $\Delta_H$, deux indicateurs furent utilisées. Ils permettent d'obtenir une estimation du taux maximum de reconnaissance potentiel des treillis qui seront éventuellement construit à partir des listes de paires de mots. L'un d'eux s'appliquant aux treillis eux-mêmes, il ne sera décrit qu'à la section 5.1.3. L'autre consiste, en fait, en une combinaison de plusieurs compteurs. Ces derniers sont le nombre de mots et de paires de mots à la fois présents dans la solution et dans les listes, et le nombre total de mots présents dans les listes. En effet, il n'est pas suffisant de ne tenir compte que du nombre de mots présents à la fois dans la solution et dans les listes, car ces mots peuvent ne pas avoir comme successeur possible celui des mots qui les suit dans la solution. Il est aussi important d'estimer la grandeur des listes pour lesquelles on obtient ces valeurs. A la limite, une liste contenant la moitié des mots du dictionnaire, et où chacun peut être suivis de tous, liste pour laquelle le taux d'inclusion serait certainement assez haut, ne correspond pas à une réduction adéquate de l'espace de recherche. Dans notre cas, l'importance de cette réduction est estimée à l'aide du nombre moyen de mots par phrase. Une mesure alternative à celle-ci, et employée par Philips, est le nombre moyen de mots par mots de solution ou WGD (Word Graph Density) [DUGA 95].

Le tableau 8.2 donne les valeurs de ces différents compteurs pour différents $\Delta_H$ et pour

**Tableau 8.2 Comparaison de différentes approches pour la deuxième passe**

| type de HMM | $\Delta_H$ | % nb. de mots inclus | % nb de paires inclus | nb de mots par mot (WGD) | nb de mots par solution |
|---|---|---|---|---|---|
| à livres de codes partagés | 60000 | 99.13 | 93.90 | 15.27 | 364.45 |
| discrets | 20000 | 84.39 | 71.15 | 2.54 | 60.66 |
| | 40000 | 95.38 | 86.27 | 6.28 | 149.97 |
| | 60000 | 99.13 | 93.07 | 20.66 | 493.07 |
| | 80000 | 99.57 | 97.23 | 62.29 | 1486.38 |
| | 100000 | 99.71 | 97.36 | 169.84 | 4052.62 |

deux différents types de HMMs obtenues sur 29 phrases du test de développement de l'évaluation de novembre 94 de la tâche WSJ. On constate que plus $\Delta_H$ est grand, plus la liste de paires de mots obtenue est complète et meilleur sera le taux de reconnaissance potentiel. Par contre, le nombre de mots par solution ou mot de solution augmente aussi rapidement et de façon non linéaire. Il en va de même du temps nécessaire pour la génération des listes qui va du simple au double lorsque $\Delta_H$ passe de 20000 à 100000[1]. On constate aussi, que meilleurs sont les HMMs utilisés, ici des HMMs à livres de codes partagés, plus le nombre de mots par solution est petit, et ce pour un même taux de reconnaissance potentiel. De son côté, pour une même valeur de $\Delta_H$, le temps de calcul requis pour l'obtention des listes en utilisant les HMMs à livres de codes partagés est le même que celui requis pour l'obtention des listes en utilisant des HMMs discrets, et ce même si le temps nécessaire au calcul des distributions des HMMs à livres de codes partagés est plus long que celui nécessaire au calcul des distributions des HMMs discrets. Ceci s'explique de deux façons. D'abord il y a le fait que l'emploi de meilleurs modèles acoustiques se traduit toujours par une efficacité accrue des faisceaux d'émondage. Puis il y a le fait que les valeurs des distributions utilisées pour chaque unité de temps sont tout simplement mémorisés dans un vecteur lors de la passe gauche-droite ce qui évite de les re-calculer

1. Mesure relative non consignée au tableau.

lors de la passe droite-gauche. En effet, dans le cas de distributions à livres de codes partagés, le temps de calcul épargné vaut amplement l'espace mémoire que nécessite le vecteur supplémentaire.

Généralement, un faisceau d'émondage traditionnel très large est aussi utilisé lors de cette deuxième passe. On le choisit très large pour ne pas que son action interfère avec le faisceau heuristique. La seule raison d'être de ce faisceau traditionnel est de pouvoir désactiver les mots qui sinon une fois activés, resteraient actifs jusqu'à la fin de la reconnaissance, souvent, sans raison valable.

### 8.1.3 Phase de compression/expansion

Cette phase consiste à construire un treillis à partir des informations amassées lors de la deuxième passe de l'algorithme en incluant à même ce dernier, les probabilités d'un modèle de trigramme. Ceci est fait en attribuant la bonne valeur de trigramme à la probabilité de transition $q_{tr}$ de chaque transition correspondant à une instance de mot et ce, pour chacun des contextes possibles de ce mot. Autrement dit, le treillis obtenu est une structure de recherche en arbre bouclé (voir figure 7.3B) où $n = 3$, où les mots ne sont pas amalgamés au niveau des phonèmes et où les valeurs de probabilités de transitions sont des valeurs de trigrammes. En effet, les paires de mots utilisées pour la construction du treillis ne contenant aucune information temporelle, il est normal que ce dernier contienne des boucles, par opposition à celui illustré à la figure 7.4, page 82 et construit à partir d'informations contenant des indications temporelles. Par contre cette approche a l'avantage de permettre d'éviter certaines erreurs éventuellement introduites par l'emploi d'algorithme de construction de treillis incorporant des indications temporelles. Un exemple d'erreur est donné à la page 111.

C'est à partir de ce treillis qu'est calculé l'autre indicateur utilisé pour évaluer le taux de reconnaissance potentiel découlant des informations amassées lors de la deuxième passe de reconnaissance. Cet indicateur est tout simplement le taux d'erreur de reconnaissance et le pourcentage de mots corrects potentiels du treillis. Il est calculé à partir des chemins représentant le meilleur pairage, en terme de taux d'erreur, par rapport à chaque retranscription d'énoncé. C'est une méthode semblable à celle utilisée par d'autres équipes, dont Cambridge. Néanmoins, nos treillis comportant des boucles et donc pouvant générer une infinité de chemins différents, l'identification du meilleur chemin ne peut se faire de façon exhaustive. Une approche A* est donc utilisée, qui favorise l'expansion du plus long chemin ayant obtenu, lors d'un pairage avec un nombre de mots $y$ de la solution, un nombre minimal d'erreurs. La fouille se termine lorsque la valeur de $y$ pour lequel un chemin obtient un nombre minimal d'erreurs est égale au nombre total de mots de la solution. Pour accélérer davantage la fouille, on fixe le nombre de chemins partiaux pouvant prendre place sur la pile, et les chemins de même longueur obtenant un même taux d'erreur pour une même valeur de $y$ sont "amalgamés" sans égard aux chemins eux-mêmes qui peuvent donc être différents. Le taux de reconnaissance estimé est donc une borne inférieure du taux de reconnaissance potentiel réel. Le taux d'erreur estimé pour les treillis générés lors de l'évaluation de novembre 94 associée à la tâche du WSJ fut de 7.64% et le pourcentage de mots corrects de 93.0%.

### 8.1.4 Troisième passe de reconnaissance

La troisième passe de reconnaissance est un Viterbi gauche-droite effectué sur le treillis construit à la section précédente. Un treillis n'est essentiellement, on se rappelle, qu'un type de graphe. On peut donc lui associer, par le mécanisme décrit à la section 2.5, un HMM complexe sur lequel directement appliquer une fouille de type Viterbi. Cette construction pouvant, surtout pour des graphes de bonnes dimensions, demander une quantité de mémoire non négligeable, elle n'est effectuée, dans la version implantée dans le système du CRIM, que pour les transitions effectivement activées lors de la fouille. Autrement dit, seules les cases de la matrices de fouille (voir figure 2.3, page 13) associées aux transitions actives pour un faisceau d'émondage donné sont allouées lors du traitement d'un énoncé.

Le cas échéant, des pauses optionnelles sont automatiquement ajoutées entre chaque paire de mots. Ces dernières n'ont donc pas à être explicitement incluses dans le treillis, ce qui permet encore de réduire l'espace occupé par ce dernier. En effet, lorsque les pauses sont explicitement incluses dans le treillis, une quantité non négligeable d'information est répétée inutilement, ne serait-ce seulement, par exemple, qu'un pointeur vers le HMM devant être utilisé et la valeur de $q_{tr}$.

Des triphones inter-mots peuvent être utilisés lors de cette passe de reconnaissance. Dans ce cas cependant, les pauses doivent explicitement être incluses dans le treillis. En effet, les pauses constituant un contexte valable et différent au même titre que les autres phonèmes, la partie de l'algorithme chargé d'identifier les HMMs nécessaires à la représentation des extrémités de mots et d'établir la liste des connections possibles, doit en tenir compte. Or, ce traitement n'étant, pour le moment, pas implanté dans le cas des pauses implicites, ces dernières doivent être explicitement incluses dans le treillis. Cela constitue un léger handicap dans l'emploi des modèles en contextes, l'utilisation de pauses explicites étant plus exigeant en temps (26% de plus en moyenne pour les treillis utilisés lors de l'évaluation de novembre 94 de la tâche WSJ) et en mémoire. Un léger handicape seulement, car l'emploi de HMMs inter-mots est, de toute façon, beaucoup plus exigeant en temps et en mémoire que l'emploi de HMMs intra-mots.

### 8.1.5 Autre version de la troisième passe de reconnaissance

Les treillis obtenus à partir des données amassées lors de la deuxième passe, peuvent aussi être utilisés pour la génération d'une liste des N-meilleures solutions. Ces N-meilleures solutions peuvent être utilisées pour fixer les coefficients des différents modèles incorporés dans le treillis (soit dans notre cas un bigramme, un trigramme et une pénalité d'insertion). Une pénalité d'insertion est simplement une valeur fixe ajoutée au score d'une hypothèse à chaque fois qu'un nouveau mot est ajouté à cette hypothèse. Le rôle de cette pénalité est de diminuer l'insertion de petits mots en faveur des longs mots. En effet, ces derniers, parce qu'ils doivent correspondre à une plus grande partie du signal, sont souvent désavantagés par rapport aux mots courts qui peuvent plus facilement correspondre erronément à une petite partie de signal.

Les N-meilleurs solutions peuvent aussi être re-scorées par des MLs de plus haut niveau (n>3, ou grammaires probabilistes) ou d'autres types de modèles comme, dans l'application TransTalk, un modèle de traduction (voir section 8.2.3).

## 8.2 Applications

Le système décrit aux sections précédentes a été utilisé en tout ou en parties dans plusieurs applications dont deux applications à très grand vocabulaire et une application à moyen vocabulaire. Deux de ces applications sont des tâches ARPA, la troisième est le deuxième volet du projet TransTalk.

### 8.2.1 Les tâches ARPA

Comme il fut souligné en introduction, c'est suite à un projet ARPA que les recherches en RAP on connu un véritable essor. Or, aujourd'hui encore ARPA est un acteur important dans le domaine de la recherche en RAP. Il chapeaute, en effet, l'acquisition de bases de données et leur utilisation dans le cadre d'applications bien définies. Ces applications sont aussi souvent désignées par le terme de tâche, car à chacune d'elles est aussi rattachée une série d'évaluations auxquelles sont encouragés à participer, les différents groupes de recherche utilisant les données amassées pour cette application. L'avantage de telles évaluations est, sans contredit, de pouvoir directement comparer divers systèmes de reconnaissance afin, non seulement de déterminer le meilleur système de façon absolue, mais aussi, ce qui, d'un point de vue recherche, est plus intéressant, de découvrir les points forts et faibles des diverses approches utilisées. C'est d'ailleurs dans ce sens que chaque évaluation est suivie d'un séminaire de travail, où chaque équipe de recherche est invitée à venir présenter son système en détails.

Le CRIM a participé aux évaluations de novembre 92 et 93 de la tâche ATIS et novembre 94 de la tâche WSJ. La tâche ATIS (Air Travelling Information Service) est une tâche d'interrogation vocale anglaise d'une base de données. Son vocabulaire est de grandeur moyenne (quelques milliers de mots). Les systèmes du CRIM utilisés pour les évaluations associées à cette tâche, sont en partie les ancêtres du système multi-passes décrit dans les premières sections de ce chapitre. Ces derniers ne présentant plus vraiment d'intérêt, ils ne seront donc pas décrits plus en détails ici. Néanmoins, plusieurs expériences rapportées dans ce travail ont été faites à partir des bases de données de cette tâche. Le lexique de cette dernière sera donc décrit à l'annexe C. Par ailleurs, on peut retrouver une description des systèmes utilisés lors de l'évaluations de novembre 92 dans [CARD 92] et lors de l'évaluation de novembre 93 dans [NORM 94a]. Finalement, les résultats obtenus lors de ces deux évaluations sont très brièvement résumés à l'annexe D.

### 8.2.2 La tâche du Wall Street Journal

La tâche du Wall Street Journal (WSJ) existe depuis 1986. Elle vise à évaluer la performance de systèmes de RAP pour une tâche multi-locuteur grand vocabulaire. Elle consiste en la dictée de passages de divers journaux financiers principalement le Wall Street Journal mais aussi, plus récemment, d'autres journaux financiers américains, d'où l'acronyme de NAB (North American Business) aussi donné à cette tâche (voir [PAUL 92b]).

La grande majorité des systèmes décrits aux sections 7.3.1 à 7.3.11 participent régulièrement aux évaluations rattachées à cette tâche. En 1994, pour la première fois, le CRIM était aussi au nombre des participants.

De façon générale, les évaluations NAB se composent d'une multitudes de sous-évaluations regroupées en Hub (moyeu) et Spoke (rayon) par analogie à une roue de bicyclette. Le but des "Hub" est d'évaluer les performances des systèmes dans un environnement similaire à celui dans lequel les données d'entraînement ont été recueillies, celui des "Spoke", d'évaluer et comparer les performances des systèmes, lorsque certaines conditions présentes initialement sont modifiées (origine des locuteurs, micro utilisé, bruits ambiants etc.).

#### 8.2.2.1 Le test H1-C1

En 94, les évaluations “Hub” étaient au nombre de deux H1 et H2, H1 étant lui-même subdivisé en P0, C1, et C2. A l’origine seul C1 étant obligatoire[1], c’est la seule sous-évaluation auquelle le système multi-passes du CRIM a été soumis. L’abréviation C1 tire son nom du fait que, pour cette évaluation, la majorité des données nécessaires tant à l’entraînement qu’à la reconnaissance sont fixes (“controlled”). C’est ainsi que les entrées du dictionnaire, le ML et le corpus d’entraînement utilisables sont tous pré-définis. Peuvent donc varier d’un système à l’autre, le type d’entraînement, de HMMs, de distributions, et d’unités utilisés, en particulier ceci implique que, bien que les entrées du dictionnaire soient fixes, le nombre et la forme des transcriptions phonétiques de chaque entrées peut varier. Finalement, bien sûre, le type d’algorithmes et de structures de recherche employés lors de la reconnaissance sont complètement libres.

#### 8.2.2.2 Sommaire des résultats

Le sommaire des résultats tel qu’obtenus par les différentes équipes ayant participé à l’évaluation C1 se trouve à l’annexe E. La description exacte des procédures d’entraînement, des unités et du traitement de signal utilisé se trouve dans [NORM 95]. Le système de reconnaissance utilisé est exactement celui décrit dans les premières sections de ce chapitre. Trois passes furent utilisées lors de l’évaluation proprement dite. La dernière passe fut effectuée trois fois en utilisant des HMMs mixtes, mâles et femelles (sexués) et en ne conservant, pour chaque locuteur, que la séquence ayant donné les meilleurs résultats. La quatrième passe facultative (voir section 8.1.5) fut néanmoins utilisée avec le test de développement afin déterminer les poids octroyés à chacun des modèles du système.

Il est intéressant de comparer les résultats obtenus lors de cette première participation à l’évaluation de la tâche du WSJ, à ceux obtenus lors de notre première et même seconde

1. Eventuellement, P0 fut plus ou moins transformée en évaluation obligatoire elle aussi, du moins pour les sites dont une partie du financement provient de l’agence ARPA. Comme ce n’est pas le cas du CRIM, nous n’étions pas tenu de et n’avons pas participé à l’évaluation P0.

participation à l'évaluation de la tâche ATIS (voir le tableau 8.3). On y constate l'amélio-

**Tableau 8.3 Comparaison des taux d'erreur**

| évaluation | meilleur résultat | résultat du CRIM |
|---|---|---|
| ATIS 92 | 4.3 | 14.1 |
| ATIS 93 | 3.3 | 6.6 |
| WSJ 94 | 10.5 | 20.2 |

ration croissante du système du CRIM par rapport au meilleur système en lice. En fait, compte tenu de la difficulté nettement plus grande d'un point de vue reconnaissance de la tâche WSJ par rapport à la tâche ATIS, et de l'adaptation toujours nécessaire lors d'une première participation à l'évaluation associée à une tâche, on peut assez facilement affirmer que le système de reconnaissance développé au CRIM est du même calibre que les meilleurs systèmes au monde (la seule autre équipe dont c'était le première évaluation a obtenu un taux d'erreur de 22.8).

En fait, moyennant de meilleurs HMMs (la faiblesse de notre système lors de cette évaluation était au niveau entraînement, en particulier au niveau du lissage des distributions), il est fort probable que nos résultats se seraient situés dans la moyenne. En fait, en utilisant les mêmes algorithmes de reconnaissance mais des méthodes d'entraînement et des unités plus performantes (seule différence, seuls des HMMs mixtes furent utilisés lors de la troisième passe), notre plus récent taux d'erreur pour le test de novembre 94 est de 17.17%.

### 8.2.3 TransTalk

L'algorithme multi-passes a aussi été utilisé lors de la deuxième phase du projet TransTalk ([BROU 95], et voir aussi la section 5.4)[1]. Le but ultime de cette deuxième phase était la conception d'un prototype temps réel d'aide à la traduction. Cette deuxième phase était composée de deux étapes. La première étape consistait en le développement d'un système de dictée grand vocabulaire à parole continue pour le français, capable de générer les N-meilleures solutions correspondant à chaque énoncé, de sorte à pouvoir évaluer l'impact d'un re-scoring de ces dernières au moyen d'un modèle de traduction. La deuxième consistait en l'élaboration effective du prototype combinant les systèmes de traduction et de

1. L'incorporation du modèle de traduction au système de reconnaissance n'ayant pas pris la forme escomptée, l'utilisation d'un système multi-passes fut inutile lors de la seconde étape. C'est une version accélérée de la première passe qui fut utilisée.

reconnaissance le plus efficacement possible selon les résultats obtenus lors de la première phase. Le système multi-passes fut utilisé lors de la première étape.

Les N-meilleures solutions correspondant à un énoncé furent donc générées au moyen du système multi-passes en utilisant la deuxième variante de la troisième passe. C'est-à-dire qu'une liste de paires de mots fut générée pour chaque énoncé, à partir desquelles furent construits des treillis utilisés pour la génération des listes des N-meilleures solutions. Puis ces listes furent re-triées, lors d'un post-traitement, par le module de traduction.

Cependant, les algorithmes de reconnaissance et de construction des graphes durent être légèrement modifiés pour tenir compte d'une des particularités du français déjà présentée à la section 5.4.1: les liaisons. En effet, dans un système à parole continue, les élisions ne présentent, pour leur part, aucun problème particulier. Il suffit simplement d'inclure sous forme d'une entrée distincte dans le dictionnaire, les mots élidées comme *l* (*l'arbre*), *s* (*s'est*), *m* (*m'arrive*) et d'entraîner le ML en conséquence. Les liaisons n'étant pas marquées orthographiquement et pouvant ou non être réalisées acoustiquement, une approche différente dut être développée.

En fait, ce sont trois approches qui furent développées et testées. Le tableau 8.4 donne les

**Tableau 8.4 Résultats comparatifs de divers traitements de la liaison**

| type d'entraînement | type de reconnaissance | taux d'erreur | % correct |
|---|---|---|---|
| sans liaison | sans liaison | 22.15 | 81.07 |
| avec liaisons | avec liaisons | 21.33 | 81.17 |
| avec liaisons restreintes | avec liaisons restreintes | 21.28 | 81.31 |

résultats obtenus au moyen des trois approches de traitement des liaisons étudiées sur un test composé de 200 phrases du Hansard[1]. Les consonnes de liaisons traitées sont *z* (*deux autres*), *t* (*grand ami*), *n* (*on a dit*), *r* (*dernier étage*), *p* (*trop aimable*) et *v* (*neuf heures*).

La première approche consiste simplement à ignorer la présence des liaisons. Au niveau de la reconnaissance, cela signifie que dans la structure de recherche aucun HMM n'est

1. Le Hansard est composé des transcriptions des débats parlementaires canadiens en français et en anglais, ce qui en fait un outil précieux dans le domaine de la traduction automatique.

prévu qui corresponde à la réalisation acoustique d'une liaison. Ainsi, la seule réalisation acoustique possible entre, disons les mots *petits* et *amis*, est la pause. Dans la deuxième approche, des HMMs représentant la réalisation acoustique des consonnes de liaison sont ajoutés devant les mots débutants par une voyelle phonétique mais sans égard au mot précédent. Ainsi, un HMM représentant le son *z*, mais aussi un représentant les sons *t*, *n*, *r*, *p* et *v* sont insérés entre les mots *petits* et *amis*, mais aussi entre les mots *crocodile* et *amis*! Finalement, dans la troisième approche seuls certains mots peuvent être suivis par certaines consonnes de liaisons lorsqu'eux-mêmes précèdent un mot débutant par une voyelle phonétique. Ainsi, seul un HMM représentant le son *z* est inséré entre les mots *petits* et *amis* et aucun entre les mots *crocodile* et *amis*.

Tout comme lors de la reconnaissance en mots isolés, la réalisation acoustique d'une consonne de liaison ne peut que suivre la pause optionnelle permise entre chaque mot. De plus, tout comme les pauses, les consonnes de liaisons n'étant par véritablement porteuses d'une signification syntaxique, elles doivent n'avoir aucun impact sur le modèle du langage et on doit donc multiplier leurs occurrences. Finalement, elle peuvent, comme les pauses, être présentes ou non. Il faut enfin souligner que puisqu'elles sont traitées comme des pauses, les consonnes de liaisons ne sont pas présentes dans les listes de paires de mots produites lors de la deuxième passe de l'algorithme multi-passes. Elle doivent donc être ré-introduites dans les treillis construits à partir de ces listes, entre les mots le permettant.

Les résultats du tableau 8.4 semblent indiquer que la meilleure approche consiste à traiter les consonnes de liaison de façon restreinte.

## 8.3 Autres applications à partir de modules du système multi-passes

Certains modules développés pour le système multi-passes ont été utilisés soit tel quel, soit légèrement modifiés et utilisés à d'autre fins que la RAP de grands vocabulaires.

C'est ainsi que la deuxième passe, légèrement modifiée, est utilisée lors de l'implantation de processus d'adaptation en temps réel (section 8.3.1); les listes de paires de mots amassées lors de la passe arrière, utilisées en entraînement MMIE (section 8.3.2); et les listes de paires de mots, modifiées pour contenir des indications temporelles, lors de l'application d'un algorithme N-meilleurs rapide (section 8.3.3), lui-même utilisé dans l'élaboration d'une méthode de rejet (section 8.3.3.1).

### 8.3.1 Implantation de processus d'adaptation en temps réel[1]

Lors d'applications de reconnaissance de la parole, il arrive très souvent que les conditions acoustiques présentes au moment où les données utilisées à l'entraînement furent recueillies ne soient plus les mêmes au moment de la reconnaissance. Les causes de ce changement sont multiples: microphones, pièces, lignes téléphoniques différents. L'exploration d'algorithmes permettant d'atténuer l'impact de tels changements est un champs de recherche très important.

Certains de ces algorithmes, dont la soustraction cepstrale ([ATAL 74] et [ROSE 94a]) et la normalisation de l'énergie, ne sont applicables que lorsque toutes les unités de temps d'un signal ont été reçues, introduisant un délai, qui lors d'applications temps réel peut parfois être inacceptable. Une autre façon possible de procéder serait d'exécuter une première passe gauche-droite semblable à celle décrite à la section 8.1.1, en appliquant, le cas échéant, une partie de l'algorithme de normalisation (par exemple le calcul d'une moyenne ou d'un maximum), puis de procéder à une passe droite-gauche où est appliquée la normalisation comme tel. Bien entendu, cette façon de faire entraîne, elle aussi, un délai. Ce qui est espéré est que ce dernier soit plus petit que le délai encouru lorsque la fin de l'énoncé est attendue avant de débuter la reconnaissance normalisée.

Cependant, en procédant à l'application de la normalisation en deux passes, les valeurs de $\alpha_{tm}$ et $\beta_{tm}$ ne sont plus directement comparables puisque les paramètres utilisées par la reconnaissance ne sont plus les mêmes lors des passes gauche-droite et droite-gauche. La façon d'utiliser les scores des hypothèses partielles diffèrent donc légèrement de celle décrites à la section 8.1.2.

La plage des valeurs possibles de $\alpha_{tm}$ et $\beta_{tm}$ sera probablement fort différentes, chacune ayant sa propre moyenne et propre variance. Pour contrer ces différences, deux mécanismes sont utilisés. La différence de variance est compensée en multipliant par une valeur, $\phi$, la valeur de $\beta_{tm}$ dans l'équation 7.3, page 87. Cette valeur devrait théoriquement être égale au rapport des écart-types des valeurs possibles de $\alpha_{tm}$ et $\beta_{tm}$ mais $\phi$ est aussi utilisé de façon à donner, dans l'équation 7.3, plus de poids à $\beta_{tm}$ qu'à $\alpha_{tm}$. En effet, les valeurs $\beta_{tm}$ étant calculées à partir des données normalisées, il serait normal qu'elles aient plus de poids sur la décision de poursuivre ou non une hypothèse. En conséquent, la valeur de $\phi$ est plutôt choisie de façon expérimentale.

1. voir aussi [LACO 95]

De son côté, la différence entre les moyennes des valeurs possibles de $\alpha_{tm}$ et $\beta_{tm}$, est compensée en choisissant de faire varier $\Psi_{max}$. En effet, au lieu de garder $\Psi_{max} = \alpha_{max}$ tout au long de la passe droite-gauche, sa valeur, pour une unité de temps $t$ sera égale à la valeur maximale de $H_{t(m_i, m_j)}$ pour l'ensemble des paires de mots $(m_i, m_j)$ actives au temps $t$. Dans les faits, lorsque $\Delta_H$ est petit, il arrive qu'aucune hypothèse ne soit active. Dans ce cas, la valeur de $\Psi_{max}$ reste inchangée. Cette façon de faire explique qu'on puisse obtenir des résultats intéressants même pour de petites valeurs de $\Delta_H$.

Le tableau 8.5 présente les résultats obtenus lors de l'application de l'algorithme d'adapta-

**Tableau 8.5 Utilisation d'une approche multi-passes pour des processus d'adaptation par soustraction cepstrale**

| $\phi$ | $\Delta_H$ | taux d'erreur | % correct | temps (hres) |
|---|---|---|---|---|
| 1 | 50 | 31.62 | 71.97 | 13:07 |
| | 120 | 31.62 | 71.97 | 13:13 |
| 0.8 | 50 | 31.59 | 71.92 | 13:09 |
| | 120 | 31.59 | 71.92 | 13:14 |
| - | - | 31.98 | 71.46 | 13.12 |

tion par soustraction cepstrale selon l'approche multi-passes et selon l'approche différée. La durée réelle des énoncés traités, énoncés tirés du test de l'évaluation WSJ de novembre 93, est de 46 minutes (276083 unité de temps * 10 millisecondes/unité / 60000). On constate qu'en effet $\Delta_H$ peut être choisi petit et que ce faisant l'approche multi-passes semble aussi rapide (les écarts de temps de calcul ne sont pas significatifs compte tenu des outils utilisés pour estimer le temps de calcul) et ce, pour un taux de reconnaissance semblable. C'est donc dire que cette approche pourrait être tout à fait viable et même peut-être préférable à l'approche différée dans le cas d'une application devant être effectuée en temps réel.

On doit souligner que dans cet exemple particulier, la première passe est effectuée sur des données normalisées par les moyennes calculées lors de la passe droite-gauche de l'énoncé précédent. Cette façon de faire nous a donné de meilleurs résultats que lorsque seule la passe droite-gauche est calculées à partir des données normalisées et n'entraîne aucun inconvénient pour l'approche multi-passes.

### 8.3.2 Entraînement MMIE à partir d'un sous-espace de recherche[1]

Un entraînement par la méthode MMIE (Maximum Mutual Information Estimation) est, pour des raisons qui sont au-deçà du sujet de cette thèse, beaucoup trop long pour être utilisé dans un système à parole continue grand vocabulaire. L'un des problèmes de l'approche MMIE, est qu'il faille théoriquement considérer lors de l'entraînement, l'ensemble des solutions possibles, soit dans un système à parole continue grand vocabulaire, l'ensemble des phrases possibles. Ceci est, évidemment, tout à fait irréaliste.

Or pour contourner ce problème, on peut n'utiliser qu'un sous-ensemble des meilleures phrases possibles. L'une des façons d'obtenir ce type de sous-ensemble est par l'utilisation d'un algorithme N-meilleur (section 5.3). Cependant les solutions générées de cette façon sont souvent très similaires à moins de choisir un N assez grand, ce qui a comme conséquence d'augmenter le temps de calcul nécessaire à la génération des différentes solutions. Pour un système de RAP en parole continue et grand vocabulaire, l'approche des N-meilleures solutions n'est donc pas une approche très pratique pour générer le sous-ensemble de solutions prometteuses nécessaires à l'entraînement MMIE.

C'est qu'en fait, ce n'est pas tant des N-meilleures solutions dont on a besoin lors d'un entraînement MMIE, mais plutôt d'une structure représentant un sous-ensemble de l'espace de recherche parmi laquelle se retrouvent des hypothèses similaires à la solution recherchée. Or, cette structure peut facilement être construite à partir des listes de paires de mots obtenues lors de la deuxième passe de l'algorithme multi-passes. La structure construite pour un entraînement MMIE pouvant se contenter de n'intégrer que les probabilités d'un bigramme, un modèle bouclé où les seules transitions possibles sont celles liant les mots d'une paire et où on rajoute, si manquants, les mots et/ou transitions entre les mots de la solution, est amplement suffisant. En prime on le rappelle, la grosseur de ces structures d'entraînement peut être modifiée selon les besoins en choisissant un seuil plus ou moins grand et en n'excluant de la structure, toutes les paires pour lesquelles la valeur $\Delta H_{tm}$ est inférieure au seuil choisi. C'est donc dire qu'on peut, pour un corpus d'entraînement donné, produire une fois pour toutes des listes de paires de mots en utilisant un $\Delta_H$ assez large pour par la suite et selon les besoins, choisir d'effectuer l'entraînement MMIE à partir de l'ensemble des paires produites ou seulement un sous-ensemble de paires sélectionnées selon leur valeur de $\Delta H_{tm}$.

1. Voir aussi à ce sujet [NORM 94b]

### 8.3.3 Algorithme de génération rapide des N-meilleures solutions

Lors de la deuxième passe de l'algorithme multi-passes, il est relativement facile de mémoriser la période d'activation d'une paire de mots, en plus des autres informations déjà amassées. Si cela n'était pas nécessaire pour les applications décrites jusqu'à maintenant, il en va autrement pour l'algorithme de génération des N-meilleurs décrit ici. Cet algorithme fut conçu comme une alternative au "tree trellis N-best" habituellement utilisé au CRIM lorsque la génération de solutions multiples est nécessaire. C'est un algorithme original, bien qu'il s'apparente à certaines méthodes de fouille de treillis… et de construction de treillis. En effet, bien qu'il s'agisse d'un algorithme de fouille, il est facile de s'en inspirer pour générer un algorithme de construction de treillis. Entre autres, il se distingue de l'algorithme de construction de treillis du SRI [MURV 93] par une plus grande rigidité au niveau des connections possibles et par un processus de collectes d'informations intermédiaires plus compacte au niveau de la mémoire.

La période d'activation d'une paire de mots $(m_i,m_j)$ est représentée au moyen de trois valeurs $t_{startleft}$, $t_{endleft}$ et $t_{bestright}$ représentant respectivement le début et la fin de la période d'activation entre $m_i$ et $m_j$, et l'unité de temps correspondant à l'activation du mot $m_j$ soit par un mot $m_x$ quelconque, soit au tout début de la passe arrière, et pour laquelle la valeur de $\Delta(m_i, m_j)$ (voir section 8.1.2, page 94) maximum est atteinte. Ces valeurs sont calculées ainsi. Soit une paire de mots $(m_i,m_j)$ active au temps $t$, si $(m_i,m_j)$ est déjà dans la liste d'activation et que $t$ soit égal à $t_{startleft} - 1$, $t_{startleft}$ prend comme valeur $t$ et, si nécessaire, c'est-à-dire si la nouvelle valeur de $\Delta(m_i, m_j)$ est supérieure à l'ancienne, les valeurs de $\Delta(m_i, m_j)$ et $t_{bestright}$ sont modifiées. Par contre, si $t$ est plus petit que $t_{startleft} - 1$, il s'agit d'une nouvelle période d'activation pour la paire $(m_i,m_j)$ et une nouvelle entrée dans la liste de paires est créée. $t_{startleft}$ et $t_{endleft}$ prennent alors tous deux la valeur de $t$ et les valeurs courantes de $\Delta(m_i, m_j)$ et $t_{bestright}$ sont mémorisées. Les valeurs de $t_{bestright}$ sont obtenues à partir de l'historique associé à une hypothèse (à condition bien entendu, que l'unité de temps d'activation $t$ de chaque mot ait été conservée dans les cases d'histoire, voir section 3, page 27).

C'est donc dire que lorsqu'une période d'activation est associée à une paire de mots $(m_i,m_j)$, ces derniers peuvent apparaître plus d'une fois dans la liste des paires de mots actives. Pour chacune de ces occurrences, la paire $(m_i,m_j)$ est active durant toute la période comprise entre $t_{startleft}$ et $t_{endleft}$. C'est d'ailleurs un exemple des informations produites pour la génération des N-meilleures hypothèses qui est illustré à la figure 7.5, page 83.

C'est à partir de ces "nouvelles" listes de paires de mots qu'est effectué l'algorithme N-meilleurs. L'algorithme proposé est de type A* et n'effectue l'évaluation d'aucun HMM. Une hypothèse se compose d'une liste de paires de mots $(m_1, m_2)\ (m_2, m_3) \ldots (m_i, m_j) \ldots$ à laquelle est associée la valeur minimum $S = \Delta(m_i, m_j)$ de toutes les paires de mots de l'hypothèse. A chaque étape, on crée à partir de l'hypothèse sur le dessus de la pile (soit celle pour laquelle $S$ est maximum), de nouvelles hypothèses. Soit une hypothèse se terminant par la paire de mots $(m_i, m_j)$, de nouvelles hypothèses pourront être créées par l'ajout d'un mot $m_{j+1}$, à condition que $t_{bestright}$ de $\Delta(m_i, m_j)$ soit compris dans l'intervalle d'activation d'une paire $(m_j, m_{j+1})$, autrement dit à condition que

$$t_{startleft}(m_j, m_{j+1}) < t_{bestright}(m_i, m_j) < t_{endleft}(m_j, m_{j+1}) \qquad \text{(EQ 8.1)}$$

Le cas échéant, les valeurs $S$ de ces nouvelles hypothèses sont mise-à-jour. Une hypothèse est une solution valide lorsque les seules paires répondant au critère de continuation sont de la forme $(m_j, x)$ où $x$ est un indicateur que le mot $m_j$ à été parmi les mots initialement activés lors du début de la reconnaissance droite-gauche.

Des résultats comparatifs entre notre algorithme de génération des N-meilleurs et le "tree trellis N-best" (voir section 3.2) sont données aux tableau 8.6. Ces résultats ont été obte-

**Tableau 8.6 Comparaison de deux algorithmes de génération de solutions multiples**

| algorithme | faisceaux D-G | | taux d'erreur | % correct | temps (Min.) | nb de sol. |
|---|---|---|---|---|---|---|
| | $\Delta_H$ | 2em | | | | |
| rapide | 50000 | 150000 | 39.06 | 71.96 | 9:08 | 10 |
| | 80000 | 150000 | 37.76 | 72.61 | 9:24 | 25 |
| "Tree trellis" | - | | 39.38 | 71.31 | 16:31 | 10 |
| | - | | 37.11 | 72.93 | 23:43 | 25 |

nus à partir de treillis de recherche (environ 7000 transitions en moyenne) obtenus de données Transtalk au moyen de l'algorithme multi-passes. Les valeurs des faisceaux d'émondage gauche-droit étaient de 80000 et 50000 respectivement pour toutes les expériences. Les taux de reconnaissance sont des taux potentiels, c'est-à-dire que ce sont les taux maximales qu'un algorithme de re-scoring parfait aurait pu obtenir à partir de la liste des N-meilleures solutions générées en sélectionnant à tout coup la meilleure solution en termes de nombre d'erreurs. On constate que notre algorithme de génération des N-meilleurs est aussi performant d'un point de vue taux de reconnaissance potentiel, du moins pour de petits N, mais beaucoup plus rapide que le “tree-trelllis N-best”. Par contre bien que la première solution générée par chaque algorithme soit la même, notre algorithme n’est pas un algorithme exact, l’ordre et les séquences possibles étant fonction tant des faisceaux de la passe gauche-droite que des faisceaux de la passe droite-gauche[1].

Afin de savoir si les différences entre les deux algorithmes étaient dues à l’algorithme rapide des N-meilleurs ou à l’effet des faisceaux, quelques expériences supplémentaires furent effectuées. Les résultats de ces expériences sont rapportées au tableau 8.7. Tout

**Tableau 8.7 Impact du choix de l’algorithme et des faisceaux**

| **algorithme** | **structure** | **faisceaux G-D** | **faisceaux D-G** $\Delta_H$ | **faisceaux D-G** **2em** | **taux d’erreur** | **% correct** |
|---|---|---|---|---|---|---|
| rapide | arbre de retrait | 150000 | 30000 | 200000 | 4.57 | 97.17 |
| “tree-trellis” | | | - | - | 3.48 | 97.83 |
| | treillis | 400000 | - | - | 3.91 | 97.61 |

d’abord on a produit pour 43 énoncés tirés de la tâche ATIS, les 50 meilleures hypothèses, au moyen des deux algorithmes N-meilleurs et en utilisant une structure en boucle avec noeud de retrait. Puis, en utilisant les mêmes valeurs de faisceaux, on a produit cette fois, au moyen de la deuxième passe de l’algorithme multi-passes, une liste de paires de mots à partir desquelles un treillis par énoncé fut construit mais en n’y incorporant qu’un modèle de bigramme. Puis, on a de nouveau produit les 50 meilleures hypothèses en utilisant le “tree-trellis N-best” mais cette fois à partir du treillis. D’après les résultats obtenus, il semble qu’un peu moins de la moitié des erreurs additionnelles causées par l’emploi de l’algorithme N-meilleur rapide soit attribuable aux informations mémorisées pendant la passe droite-gauche (et donc, par ricochet, au faisceau heuristique), le reste des erreurs étant dû au design même de l’algorithme N-meilleurs rapide.

Qui plus est on constate que la différence entre le taux potentiel obtenu par les deux algorithmes est plus grand dans le cas d’un modèle bouclé que dans celui d’un graphe. Ceci est dû à l’une des approximations utilisées dans l’algorithme N-meilleur rapide. En effet puisqu’une seule valeur de $t_{bestright}$ est conservée pour chaque paire d’activation, des cas tel celui dont la description suit au prochain paragraphe et tiré de la figure 7.5 peuvent se présenter dans lesquels certaines connections sont perdues.

1. A noter que lors de l’emploi de faisceaux dans la passe gauche-droite, l’exactitude du “tree-trellis N-best” non plus, n’est plus assurée.

Dans l'exemple de la figure 7.5, on retrouve, en effet, le mot *on* (ligne 17) qui peut être suivie par *monday* (ligne 4) avec comme valeur $t_{bestright}$ de *monday* 332, ce qui signifie que seules les mots *x* dont la période d'activation incluent 332 peuvent suivre *monday*. Le mot *a_m* répond à ce critère, mais non le mot *eight* dont la période d'activation 316-325 est cependant très proche de celle de *a_m* 328-336. Ce qui s'est en fait produit est que le mot *monday* à, en fait, été actif durant toute la période 316-336, et que n'ayant pu durant cette période, qu'activer un seul mot, en l'occurrence *on*, une seule valeur maximale de $t_{bestright}$ fut conservée qui correspondait à la période d'activation par *a_m*. Pour résoudre ce problème, il faudrait soit converser plus d'une valeur $t_{bestright}$, ce qui complique un peu l'algorithme de connection des mots, ou alors élargir l'intervalle de l'équation 8.1, ce qui entraîne une sur-génération de solutions.

Ce type de situation se présentant plus souvent dans un modèle bouclé que dans un graphe, c'est ce qui explique la meilleure performance de l'algorithme N-meilleur rapide dans ce deuxième cas. Lorsque le temps n'est pas un facteur important mais que le taux de reconnaissance l'est et que la reconnaissance est effectuée au moyen d'une structure de recherche bouclée, le meilleur choix reste donc le "tree-trellis N-best". C'est d'ailleurs pourquoi c'est lui qui fut utilisé lorsqu'un tel algorithme était nécessaire, lors de la première étape de la deuxième phase du projet TransTalk et lors de l'évaluation de novembre 94 associée à la tâche du WSJ. Par contre, lorsque la vitesse est importante, l'algorithme N-meilleurs proposé est un bon choix.

Il existe une autre différence majeure entre notre algorithme et le "tree-trellis N-Best". Ce dernier est en général toujours capable de générer le nombre de solutions demandées puisque seuls les faisceaux utilisés lors de la passe gauche-droite peuvent le limiter. Or pour obtenir un taux de reconnaissance acceptable, ces derniers ne peuvent jamais être trop petits. Par contre, dans notre algorithme de génération des N-meilleurs, le nombre de solution est aussi limité par la valeur de $\Delta_H$. A la limite, lorsque $\Delta_H = 0$, seules les paires associées à la meilleure solution droite-gauche seront actives, et seule cette solution pourra être re-générée par l'algorithme N-meilleurs. Qui plus est, pour une même valeur de $\Delta_H$, le nombre de solutions généré varie d'un énoncé à l'autre. Cette caractéristique a été exploitée dans la conception d'une méthode de rejet.

### 8.3.3.1 Méthode de Rejet basée sur la présence de multiples solutions[1]

L'on peut faire l'hypothèse que, pour une même valeur de $\Delta_H$, ce sont les énoncés "mieux" reconnus pour lesquels l'algorithme N-meilleurs produit moins d'hypothèses et les énoncés moins bien reconnus pour lesquels l'algorithme N-meilleurs produit plus d'hypothèses. Or si un énoncé est à ce point ambiguë que l'algorithme N-meilleurs en tire un grand nombre d'hypothèses, c'est peut-être parce que ce dernier n'est pas un énoncé valide (par exemple une question lorsqu'on tente de reconnaître un numéro de téléphone), ou encore que c'est un énoncé valide mais corrompu par un bruit ambiant ou prononcé de façon inattendue. Dans toutes ces situations, il peut-être préférable, en particulier dans le cas d'applications interactives, de rejeter l'énoncé quitte à utiliser un autre moyen pour obtenir l'information attendue. C'est le problème du rejet dont on a brièvement parlé à la section 3.1.5.

Nous avons donc choisi d'appliquer un critère de rejet basé sur le nombre d'hypothèses produits par l'algorithme rapide N-meilleurs, critère qui à notre connaissance n'a jamais été utilisé à cette fin. Un énoncé est rejeté lorsque plus de N hypothèses peuvent être générées. Une première série d'expériences effectuées sur des données tirées de la tâche WSJ donnent les résultats rapportés au tableau 8.8. Le % rejet correspond au nombre

**Tableau 8.8 Taux de rejet en fonction du N choisi**

| $\Delta_H$ | N | taux d'erreur | % correct | % rejet | %faux rejet |
|---|---|---|---|---|---|
| 1300 | 1 | 4.44 | 96.67 | 83.53 | 21.13 |
| | 10 | 8.56 | 93.10 | 31.77 | 3.7 |
| | 20 | 10.46 | 91.59 | 23.53 | 5.0 |
| 700 | 1 | 5.92 | 95.64 | 70.59 | 20.0 |
| | 10 | 12.97 | 89.53 | 9.41 | 12.5 |
| | 20 | 13.56 | 89.0 | 4.71 | 25.0 |
| - | | 14.01 | 88.50 | - | - |

d'énoncés rejetés par rapport au nombre total, et le % faux rejet, le nombre d'énoncés parmi ceux rejetés pour lesquelles la reconnaissance étant parfaite auraient dû être conservés. La dernière ligne donne les taux de reconnaissance obtenus lorsqu'aucun rejet n'est appliqué. Pour une tâche comme le WSJ, où le taux d'erreur est assez élevé, il faut choisir

1. voir aussi [LACO 95]

un niveau de rejet (valeur de N) assez élevé, sinon le taux d'erreur est effectivement meilleur mais pour la bonne raison que quasi tous les énoncés sont rejetés.

En fait, le rejet en bloc de tout un énoncé n'est peut-être pas la meilleure approche possible dans le cas de tâches de dictée comme celle du WSJ. Cette approche de rejet serait plus appropriée lorsque les énoncés devant être traités par une tâche sont courts, par exemple, un mot ou un numéro de téléphone.

Une nouvelle série d'expériences fut donc réalisée sur une petite base de données téléphoniques comprenant des énoncés composés de séries de chiffres isolés. Le tableau 8.9

**Tableau 8.9 Taux de rejet en fonction de la largeur du faisceau heuristique**

| **faisceaux D-G** | | **taux d'erreur** | **% correct** | **% rejet** | **% faux rejet** |
|---|---|---|---|---|---|
| $\Delta_H$ | **2em** | | | | |
| 10000 | 200000 | 2.5 | 98.27 | 4.34 | 43.72 |
| 20000 | | 1.67 | 98.88 | 8.73 | 49.00 |
| 30000 | | 1.02 | 99.29 | 13.37 | 56.61 |
| 40000 | | 0.69 | 99.57 | 18.11 | 63.37 |
| 50000 | | 0.48 | 99.72 | 23.60 | 69.59 |
| 60000 | | 0.32 | 99.82 | 29.91 | 74.98 |
| - | | 3.51 | 97.61 | - | - |

donne les résultats obtenus en faisant varier $\Delta_H$. Ici encore, la dernière ligne donne les taux de reconnaissance obtenu lorsqu'aucun rejet n'est appliqué. Dans ce cas, le rejet d'un énoncé complet a du sens. Or on peut voir que le taux d'erreur est diminué de moitié pour un taux de rejet de seulement 8.75%.

# 9 Utilisation de faisceaux d'émondage "intelligents"

Comme on l'a souligné à la section 3.1.2, l'utilisation de faisceaux d'émondage, lors de l'étape de reconnaissance, est la méthode de base d'accélération du temps de calcul. Mais comme on l'a aussi souligné, cette méthode donne lieu à un important compromis entre l'obtention d'un bon taux de reconnaissance et l'obtention d'une réduction importante de ce taux. Ceci est dû au fait que, la largeur du faisceau étant fixe tout au long de la reconnaissance, il faille donc la choisir assez grande pour éviter de couper la meilleure solution globale lorsque le score de cette dernière s'éloigne de la meilleure hypothèse locale. L'idéal serait de pouvoir faire varier cette largeur, de sorte à utiliser un faisceau plus étroit pour les unités de temps où le score de la meilleure solution globale se rapproche du score de la meilleure hypothèse locale, et plus large lorsqu'il s'en éloigne. On réussirait ainsi à minimiser les calculs inutiles tout en évitant de couper la meilleure solution globale. C'est un souhait qui avait déjà été émis dans [LEER 90]. C'est pour tenter de répondre à cette demande que furent entrepris les travaux décrits dans ce chapitre[1]. Deux approches en particulier sont originales, soient celles liées à l'application d'un ML (section 9.1.1).

## 9.1 Analyse du comportement de la meilleure solution globale

Afin de pouvoir adapter la largeur du faisceau d'émondage à la largeur entre le score de la solution globale et celui de la meilleure hypothèse locale, largeur dénotée $\Delta_S$, il nous faut trouver un paramètre $x$ pouvant être corrélé avec $\Delta_S$. De plus, $x$ doit pouvoir être calculé rapidement, de sorte que la réduction en temps de calcul obtenu au moyen d'un faisceau dynamique, ne soit pas annulée par le temps de calcul supplémentaire exigé par ce dynamisme.

Plusieurs $x$ ont été considérés. Ainsi, l'on peut penser qu'un plus grand $\Delta_S$ au temps $t$ se traduit par une plus grande incertitude au niveau des données, et donc par une variance

1. voir aussi [LACO 93]

plus forte au niveau de l'ensemble des scores des solutions actives au temps $t$. Lorsque cette hypothèse est appliquée à un sous-ensemble test de la tâche ATIS (voir 8.2.1), on obtient les résultats illustrés à la figure 9.1. Malheureusement, comme on peut le constater,

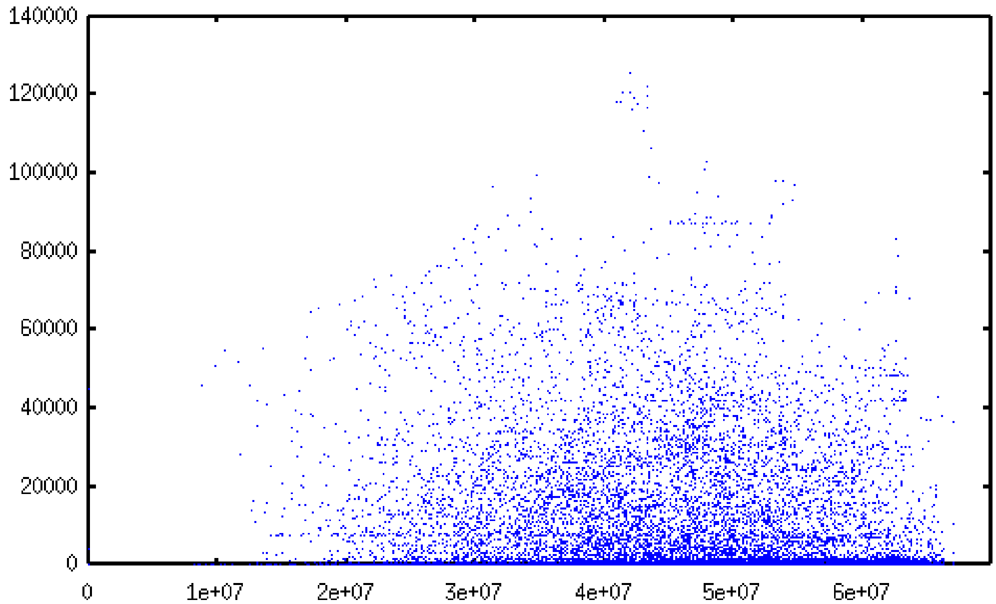


**Figure 9.1 Répartition de $\Delta_S$ en fonction de la variance**

cette hypothèse ne s'avère pas fondée.

En fait, suite à plusieurs autres tentatives tout aussi infructueuses, il semble qu'il faille conclure que le comportement de l'ensemble des scores des hypothèses actives à une unité de temps $t$ donnée n'est pas un très bon indicateur du comportement de $\Delta_S$. Par contre, un autre type d'information pouvant s'avérer pratique nous est donné par la structure de recherche. Il s'agit de l'identité des noeuds en terme de caractéristiques morphologiques et/ou syntaxiques. Par caractéristiques morphologiques et/ou syntaxiques, on entend la position dans un mot, la position dans la phrase, le type syntaxique du mot, sa longueur et ainsi de suite. L'on a donc tenté de savoir s'il existait un lien entre l'une de ces caractéristiques et $\Delta_S$. Comme on peut le voir à la figure 9.2, il semble bien qu'exception faite du silence et de la pause, $\Delta_S$ décroît rapidement au fur et à mesure que l'on progresse dans un mot, pour remonter instantanément dès le début d'un autre mot (les lignes pointillées verticales indiquent la frontière entre deux mots).

La décroissance de $\Delta_S$ dans un mot peut s'expliquer par le fait que plus l'on progresse dans ce dernier, plus l'incertitude acoustique à son sujet décroît. Ce qui donne à penser qu'un faisceau d'émondage devrait logiquement être plus large au début d'un mot et plus

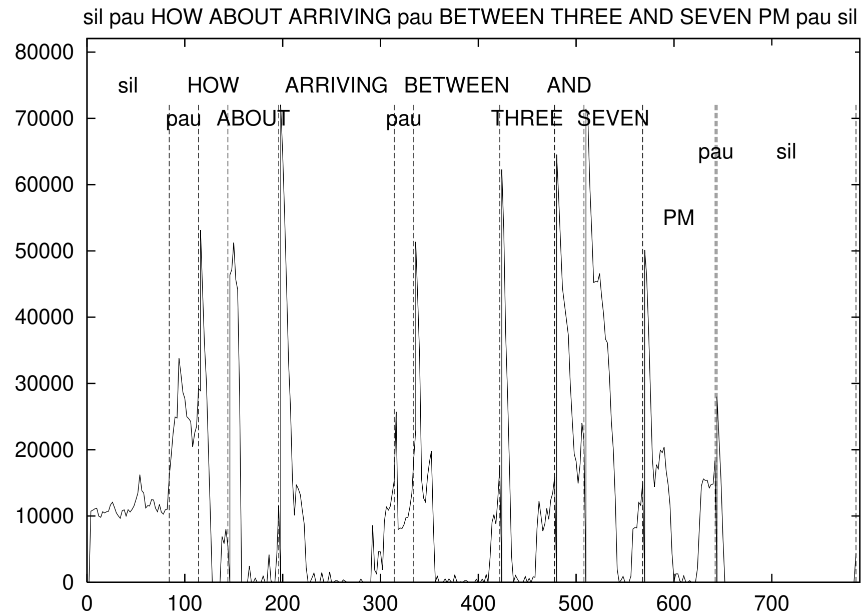


**Figure 9.2 Comportement de $\Delta_S$ dans une structure bouclée**

étroit à la fin. Les résultats obtenus suite à l'exploration de cette hypothèse sont décrits à la section 9.1.2.

Par contre, la remontée de $\Delta_S$, à la sortie d'un mot, s'expliquerait plutôt par l'application, à cet endroit dans la structure de recherche, du modèle de langage. En effet, l'influence de l'application d'un ML n'est pas fonction de l'unité de temps $t$ courante, mais bien de la position occupée par une hypothèse dans la structure de recherche. Ainsi, dans les topologies décrites à la section 7.1.1, leur impact se fait spécialement sentir sur le premier noeud d'un mot. Il est donc fort possible, à cette position, et c'est ce qu'on remarque, que la meilleure solution soit momentanément éclipsée par une autre hypothèse locale potentiellement moins probable acoustiquement mais plus probable selon le ML. Ce comportement, inévitable, n'a toutefois pas que des désavantages. En effet, il serait peut-être avantageux d'appliquer à cet endroit précis un faisceau permettant de couper les hypothèses dont le score serait devenu vraiment trop bas suite à l'application du ML. Les résultats obtenus suite à l'exploration de cette hypothèse sont décrits à la section 9.1.1.

### 9.1.1 Les faisceaux liés à l'application du ML

Le calcul du seuil $se_{ML}$ du faisceau lié au ML se fait de la même façon que le calcul du seuil du faisceau classique, $se$, décrit à la section 3.1.2. On s'assure cependant que le delta, $\Delta_{ML}$, choisi pour le calcul de $se_{ML}$ soit plus grand que celui, $\Delta$, choisi pour le calcul de $se$, de sorte à ce que $se_{ML} < se$ soit toujours vraie. De toutes façons, dans le cas contraire, il serait totalement inutile d'utiliser ce deuxième faisceau, puisque aucune hypothèse ne serait coupée qui ne l'aurait été par l'application du premier faisceau. On rappelle, en effet, que ce dernier est appliqué à toutes les hypothèses de la structure de recherche.

La figure 9.3 illustre la variation du taux de reconnaissance (en pourcentage) en fonction

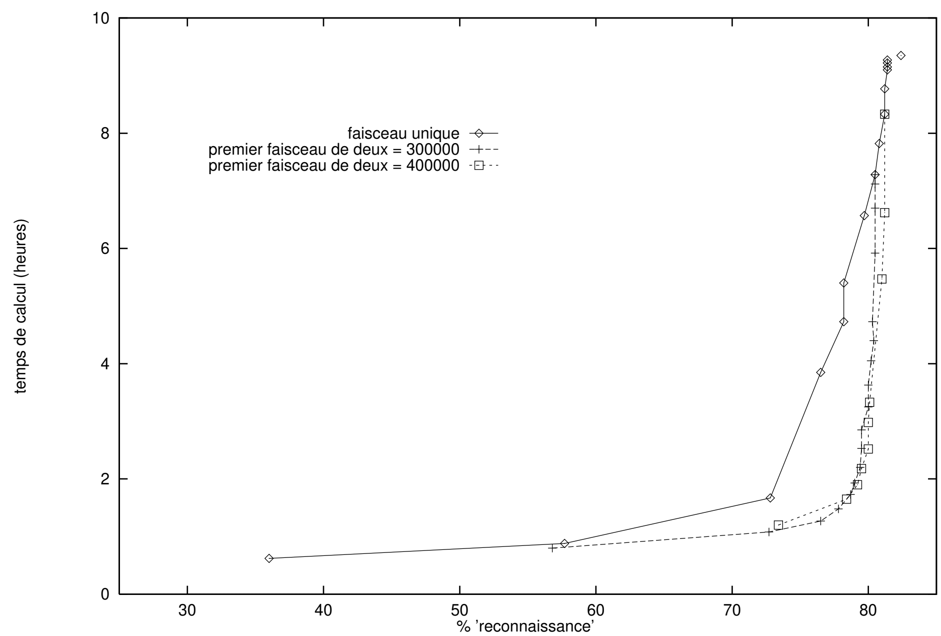


**Figure 9.3 Comparaison des taux de reconnaissance en présence de un et deux faisceaux**

du temps de calcul nécessaire à cette reconnaissance pour diverses valeurs de $\Delta$ (trait plein) et deux valeurs de $\Delta_{ML}$, ($\Delta$ étant alors fixe et ayant comme valeur, traits pointillés, points en forme de croix $\Delta = 300000$, points en forme de carré $\Delta = 400000$). On peut y voir, qu'il est impossible en utilisant un seuil faisceau d'obtenir un aussi bon taux de reconnaissance pour un même temps de calcul que lors de l'emploi de deux faisceaux.

Plus récemment, un autre type de faisceau lié à l'utilisation d'un ML au sein d'une structure bouclée a été testé avec succès [NORM 95]. Il s'agit d'un seuil d'émondage basé sur l'étendue de la plage des probabilités du ML. Son fonctionnement est le suivant: soit $\Delta P(m)$ la différence entre les valeurs maximale et minimale prises par le ML, et soit

$\max S_{fin}$ le score de la meilleure hypothèse parmi celles se terminant à la fin d'un mot, le traitement des transitions vides en partance d'une fin de mot $m_i$ n'est effectué que si on a

$$S_{m_i} > \max S_{fin} - \Delta P(m) \,. \qquad \text{(EQ 9.1)}$$

On remarquera qu'un émondage basé sur ce critère n'introduit aucune erreur supplémentaire. En effet, toutes les fins de mots étant soit directement soit indirectement connectés entre eux, si le score de l'un deux augmenté de la probabilité maximale du ML est inférieur à $\max S_{fin}$ augmenté de la probabilité minimale du ML, il est inutile de poursuivre l'hypothèse reliée à cette fin de mot. Cette hypothèse serait immanquablement abandonnée au profit de celle ayant le score de $\max S_{fin}$. Malheureusement le gain en temps obtenu lors de l'application tel quel de ce faisceau est négligeable.

Par contre moyennant deux approximations, les résultats obtenus se sont révélés très intéressants. Les approximations employées ont toutes deux comme impact de réduire $\Delta P(m)$ avec comme conséquence le non calcul des bigrammes pour un plus grand nombre de mots. L'une de ces approximations consiste à ne pas utiliser les bigrammes du noeud de retrait dans le calcul de $\Delta P(m)$. Ces derniers étant en général plus petits que les bigrammes directs, la borne inférieure servant au calcul de $\Delta P(m)$ est plus haute et $\Delta P(m)$ plus petit. L'autre approximation consiste à faire varier au moyen d'un paramètre, $\rho$, prenant une valeur entre 0.0 et 1.0 la portion de $\Delta P(m)$ utilisée lors du test de l'équation 9.1. La courbe du taux d'erreur en fonction du temps de calcul obtenu pour différentes valeurs de $\rho$ est illustrée à la figure 9.4. On soulignera qu'une valeur de $\rho$, de 0.0, a comme conséquence, dans l'implantation que nous en avons fait, d'éliminer l'emploi du faisceau de prévention. Dans ce cas, seul le faisceau standard est utilisé. On constate que l'emploi de ce faisceau d'émondage supplémentaire permet d'obtenir un gain appréciable au niveau du temps de calcul et ce sans introduire de nouvelles erreurs ($\rho = 0.4$) ou quelques erreurs seulement ($\rho = 0.2$).

### 9.1.2 Le faisceau de fin de mot

Le faisceau proposé ici s'applique à tous les types de structures de recherche et non uniquement aux structures bouclées incorporant un ML comme c'était le cas des deux faisceaux décrits aux sections précédentes.

On se rappelle, que $\Delta_{sol}$ décroît au fur et à mesure que la reconnaissance progresse dans un mot et que logiquement, il remonte à l'entrée d'un mot. Ce comportement étant similaire à celui observé dans les structures incorporant un ML, nous avons d'abord, lors d'une

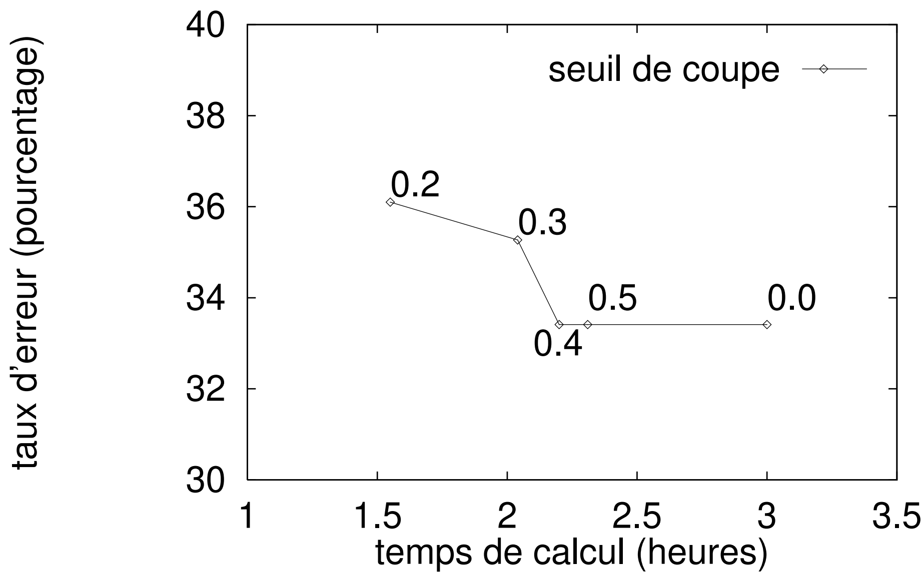


**Figure 9.4 Emploi d'un faisceau de prévention**

première tentative d'implantation d'un faisceau de fin de mot, utilisé le même mécanisme que celui implanté en présence d'un ML. Hélas les résultats obtenus furent mitigés. Cela est probablement dû au fait que dans une structure rigide incorporant un ML, l'effet proéminent est celui du à l'application du ML et que, dans ce cas, c'est à ce niveau qu'a lieu l'impact du faisceau de fin mot.

Dans le cas d'une structure générale il semble que la remontée du score à l'entrée d'un mot soit plus subtile. Pour saisir cette subtilité, nous avons décidé de modifier légèrement notre approche, rejoignant du même coup l'approche décrite dans [ODEL 94]. Ainsi au lieu de calculer $se_{FM}$, le seuil de fin de mot, à partir de $\max S_t$, on le calcule à partir de $\max S_{fin}$. Les résultats obtenus sont alors plus près de ceux attendus, bien que moins spectaculaires que lors de l'emploi d'un deuxième faisceau lié au ML (voir le tableau 9.1).

**Tableau 9.1 Impact d'un deuxième faisceau dans le cas d'un graphe général**

| **seuils (*1000)** | | **%taux d'erreur** | **% correct** | **temps (heures)** |
|---|---|---|---|---|
| **1$^{ier}$** | **2$^{ième}$** | | | |
| 70 | 70 | 47.16 | 65.95 | 8:40 |
| | 40 | 46.84 | 66.61 | 5:47 |
| 60 | 50 | 45.87 | 66.45 | 6:22 |
| | 40 | 45.06 | 66.94 | 5:38 |

**Tableau 9.1 Impact d'un deuxième faisceau dans le cas d'un graphe général**

| seuils (*1000) | | %taux d'erreur | % correct | temps (heures) |
|---|---|---|---|---|
| $1^{ier}$ | $2^{ième}$ | | | |
| 50 | 50 | 44.25 | 66.94 | 5:58 |
| | 40 | 45.06 | 67.10 | 5:34 |
| 40 | 40 | 44.21 | 66.94 | 5:23 |

## 9.2 Autres méthodes d'émondage

Il existe d'autres méthodes d'émondage qui n'ont pas été étudiés directement lors de ce travail mais qui sont couramment utilisées dans d'autres systèmes. Elles ont brièvement été mentionnées aux sections 7.3.1 à 7.3.11. Soulignons ici les deux plus courantes, soit l'imposition d'un seuil sur le nombre d'hypothèses (on parle aussi de noeuds actifs) (système de Philips et Cambridge) et le faisceau de début de phone.

L'imposition d'une limite sur le nombre d'hypothèses actives peut se faire tout simplement en modifiant dynamiquement $\Delta$. Si le nombre d'hypothèses est trop grand, on diminue $\Delta$, et si le nombre d'hypothèses est trop petit on l'augmente ou le laisse fixe. Le degré d'augmentation ou diminution peut être proportionnel à l'écart séparant le nombre d'hypothèses actives du nombre souhaités selon un rapport établi expérimentalement. Il peut aussi être déterminé exactement, ce qui exige alors, que les hypothèses soient triées selon leur score. Ce tri peut se faire de façon approximative, les hypothèses sont alors triées selon des intervalles plus ou moins grands, ou de façon exacte. Par conséquent, plus on désire que le nombre d'hypothèses actives se rapproche du nombre idéal, plus la quantité de calcul nécessaire est grande. Une fois encore, il faut donc faire un compromis entre l'efficacité d'une approche d'émondage et le temps de calcul nécessaire à son application.

Le faisceau de fin de phone est tout à fait semblable tant au niveau de son application qu'au niveau de son calcul au faisceau de fin de mot décrit à la section 9.1.2 sauf qu'il est appliqué et calculé pour chaque phone. Son impact se fait surtout sentir lorsqu'une majorité de segments de la structure de recherche utilisée représentent des phones ou des parties de mots. Comme c'est souvent le cas des structures en arbres pures ou bouclées, c'est dans ce type de systèmes qu'on l'utilise le plus souvent.

Il faut souligner que nous avons tenté d'incorporer le seuil sur le nombre d'hypothèses possiblement actives mais sans grand succès. Il faut dire que le système utilisé lors de cet essai utilisait déjà un seuil de fin de mot lié au ML et un faisceau de prévention. Or, il semble que l'impact des faisceaux employés seuls est plus grand que lorsqu'employé de façon combinée. Il n'est donc pas si surprenant que l'ajout d'une quatrième méthode d'émondage n'ait pas donné grand chose.

# 10 Conclusion

Dans le cadre de l'approche probabiliste actuellement utilisée en RAP, l'étape de la reconnaissance ou de la fouille s'avère être très facile à définir (section 2.1, page 6). Or, en pratique lorsque le nombre de messages possibles dépasse plusieurs milliers, son implantation s'avère beaucoup plus complexe. Le but de cette thèse était d'abord de présenter et bien faire saisir les différentes facettes de cette complexité mais surtout bien entendu d'y apporter des éléments de solutions.

Parce qu'ils diffèrent substantiellement les uns des autres, tant au niveau application, problèmes posés que solutions possibles, les systèmes de RAP en mots isolés et les systèmes de RAP en parole continue sont chacun l'objet d'une partie distincte de ce travail. Et parce qu'ils sont actuellement le principal sujet de recherche en RAP, plus d'emphase a été mis sur les systèmes RAP en parole continue.

## 10.1 Contributions

Les contributions originales de cette thèse au problème de la recherche (dans le sens fouille) en RAP pour de grands vocabulaires sont dans le domaine de la RAP en mots isolés:

1. un algorithme de compression des lexiques permettant de réduire de façon significative la mémoire nécessaire, applicables à de très grands lexiques (>100000) et permettant une solution économique au problème des liaisons en français

et dans le domaine de la RAP en parole continue:

2. une approche de recherche multi-passe robuste ayant permis l'obtention de bon résultats lors de l'évaluation de novembre 94 associée à la tâche WSJ, facilement adaptable au français
3. une famille de faisceaux intelligents applicables en présence d'un ML;
4. une approche de rejet globale basée sur la densité de treillis;
5. une méthode d'application originale de techniques d'adaptation au signal.

## 10.2 Recherche future

Mais rien n'étant parfait, plusieurs des méthodes proposées pourraient être améliorées ou alors présentent des lacunes qui réduisent leurs champs d'application. C'est ainsi que l'approche du graphe compressée ne peut être couplé avec un ML probabiliste que de façon indirecte. Ceci peut être un entrave à son utilisation dans le cadre d'applications où le ML joue un rôle important. Par contre, dans le cas où son impact n'est pas ou peu important par exemple lorsque tous les mots sont équiprobables, l'emploi du graphe compressé représente nettement l'approche la plus économique d'un point de vue mémoire utilisée, spécialement d'ailleurs dans le cas du français, tout en étant aussi efficace d'un point de vue temps de calcul qu'une approche en arbre.

Au fur et à mesure de la disponibilité de modèles du langage et de HMMs plus performants, le système multi-passes décrit au chapitre 8 pourrait être raffiné, par exemple, en créant, à la troisième étape, des treillis sans boucle. C'est ainsi, qu'une version de l'algorithme des N-meilleurs décrits à la section 8.1.3 et potentiellement améliorée pour tenir compte des critiques formulées à son égard, pourrait être utilisée pour construire un treillis contenant un minimum de boucles et dont le re-scoring pourrait se faire plus rapidement.

De même, des méthodes alternatives plus rapides que le calcul des N-meilleurs pourraient être utilisées pour estimer la densité des treillis. En particulier, il serait intéressant de pouvoir déterminer de façon plus locale la densité de ces derniers, afin de pouvoir localiser plus exactement les parties d'énoncés corrompues et/ou mal reconnues. C'est ainsi, par exemple, qu'on peut faire l'hypothèse, en examinant le treillis illustré à la figure 7.4, page 82, que le début de l'énoncé auquel ce dernier correspond, est peut-être moins "claire" acoustiquement que la fin. C'est une avenue de recherche qui vaut la peine d'être poursuivie.

Toujours dans le domaine du rejet, il pourrait être intéressant de voir qu'elle serait l'impact de la méthode décrite à la section 8.3.3 lorsque combinée à d'autres méthodes de rejet plus locales, comme l'emploi d'un modèle générique [BOIT 93]. Un modèle générique, est un HMM dont la caractéristique est de n'être jamais ni trop bon ni trop mauvais, de sorte à ce qu'il puisse modéliser assez bien des événements acoustiques imprévus mais jamais très bien des événements acoustiques prévus et pour lequel des HMMs existent. Des expériences préliminaires semblent indiquer que les deux types d'approches gagnent à être utilisées conjointement.

Finalement, il ne faut pas perdre de vue que ce sont souvent les modèles du langage et acoustiques qui ont indiqué à l'étape de fouille la direction de son développement. Dans cette optique, il pourrait être intéressant de voir comment, par exemple, de futurs modèles du dialogue s'intégreraient à l'étape de la fouille. Bien que minimale, l'intégration du modèle de traduction effectuée lors de la deuxième phase du projet TransTalk est un excellent exemple des possibilités de ce genre d'exercice.

# Glossaire

| | |
|---|---|
| algorithme: | séquence précise d'étapes menant à une solution |
| approche: | ensemble d'algorithmes et de structures de fouille utilisés pour parvenir à résoudre un problème |
| chemin: | séquence de transitions et noeuds dans une structure de recherche |
| corpus: | ensemble d'énoncés ou de textes utilisés pour l'entraînement des HMMs ou des MLs |
| état: | synonyme de noeud pour un HMM, une structure de recherche ou un ATN |
| fouille: | synonyme de recherche qui présente l'avantage de ne pouvoir être interprété que selon le sens de "search" |
| graphe: | type de structure de recherche |
| hypothèse: | solution partielle ou complète générée par un système de RAP, correspond à un chemin dans la structure de recherche |
| lexique: | synonyme de vocabulaire |
| matrice: | représentation graphique du processus de fouille des algorithmes "avant" et de Viterbi |
| noeud: | synonyme de état pour un HMM, une structure de recherche ou un ATN |
| recherche: | processus de reconnaissance |
| | (espace de) ensemble des solutions possibles |
| | (structure de) représentation d'un HMM composite |
| reconnaissance: | terme générique, s'oppose à l'entraînement, synonyme de recherche dans le sens de "search" |
| score: | probabilité cumulative d'une hypothèse |
| solution: | la meilleure hypothèse obtenue à la fin de la reconnaissance |
| système: | (RAP) ensemble de code informatique capable de reconnaissance de la parole |
| transition: | liens entre deux états ou noeuds |
| treillis: | type de structure de recherche semblable à un graphe |
| vocabulaire: | synonyme de lexique, ensemble des mots connus d'un système de RAP |

# Liste des acronymes

| | |
|---|---|
| ACM: | Association Computer Machinery |
| AFD: | Automate Fini Déterministe |
| AI: | Artificial Intelligence |
| ARPA: | Advanced Research Projects Agency |
| ASI: | Advanced Study Institute |
| ASR: | Automatic Speech Recognition |
| ASSP: | Acoustics, Speech and Signal Processing |
| ATIS: | Air Travel Information System |
| ATN: | Augmented Transition Network |
| BBN: | Bolt, Beranek and Newman |
| BNR: | Bell Northern Research |
| CSR | Continuous Speech Recognition |
| CFG: | Context Free Grammar |
| CMU: | Carnegie Mellon University |
| CSELT: | Centro Studi e Laboratori Telecomunicazioni |
| DARPA: | Defense Advanced Research Projects Agency |
| HLT | Human Language Technology |
| HMM: | Hidden Markov Model |
| IBM: | Intelligent Business Machine |
| ICASSP: | International Conference on Acoustic Speech and Signal Processing |
| ICSLP: | International Conference on Spoken Language Processing |
| IEEE: | Institute of Electrical and Electronics Engineers |
| JEP: | Journées d'Etudes sur la Parole |
| INRS: | Institut National de Recherche Scientifique |
| ML: | Modèle du Langage |
| MLE: | Maximum Likelihood Estimation |
| MMIE: | Maximum Mutual Information Estimation |
| NATO: | North Atlantic Treaty Organization |
| PAMI: | Pattern Analysis and Machine Intelligence |
| PC: | Personal Computer |
| RAP: | Reconnaissance Automatique de la Parole |
| SLT: | Spoken Language Systems Technology Workshop |

| | |
|---|---|
| SRI: | Stanford Research Institute |
| TIMIT: | Texas Instrument Massachusetts Institute of Technology |
| WGD: | Word Graph Density |

# Table de nomenclature

| | |
|---|---|
| $b_{tr}$: | probabilité d'émission (et par extension distribution d'émission) |
| $cte$: | constante |
| $e$: | état d'un HMM |
| $f$: | état final d'un HMM |
| $g(m)$ : | classe d'un mot |
| $H$: | score heuristique d'une hypothèse |
| $i, j, k, l$: | indices |
| $lc$: | longueur d'un texte en mots |
| $ldc$: | livre de code d'une transition d'un HMM discret |
| $L$: | longueur d'un message en unités de temps |
| $Lm$ | longueur d'une séquence de mots |
| $m$: | séquence de mots et par extension message émis par un locuteur |
| $\hat{m}$: | message reconnu par le système de RAP |
| $M$: | nombre d'entrées d'un vocabulaire |
| $maxnbm$: | nombre maximum de mots dans une phrase |
| $n$: | contexte d'un modèle du langage probabiliste en nombre de mots |
| $N$: | nombre de solutions trouvées ou désirées |
| $nbc$: | nombre de livres de codes |
| $nbmixt$: | nombre de composantes d'une mixture |
| $nbp$: | nombre de paramètres extraits du signal |
| $O$: | ordre de grandeur du temps d'exécution ou de la mémoire utilisée par un algorithme |
| $p$: | paramètre extrait du signal |
| $P(m)$ : | probabilité du modèle du langage |
| $P(y\| m)$ : | probabilité du modèle acoustique |
| $PP$: | perplexité |
| $q_{tr}$: | probabilité de transition |
| $s$: | message émit par un locuteur sous sa forme digitalisée |
| $se$: | seuil |
| $sol$: | solution |
| $S$: | score d'une hypothèse |
| $t$: | indice dénotant une unité de temps |

| | |
|---|---|
| $tr$: | transition d'un HMM |
| $x$: | variable |
| $y$: | séquence d'observations correspondant à un message |
| $\alpha$: | score obtenu lors d'une fouille gauche-droite |
| $\beta$: | score obtenu lors d'une fouille droite-gauche |
| $\sigma$: | variance d'une loi normale |
| $\mu$: | moyenne d'une loi normale |
| $\Delta$: | largeur d'un faisceau d'émondage |
| $\Delta_H$: | largeur du faisceau heuristique |
| $\Delta_S$: | différence entre le score de la meilleure hypothèse locale et globale |
| $\delta$: | fonction de transition d'un AFD |
| $\Psi$: | score de référence pour la reconnaissance droite-gauche |

# Annexe A Description des lexiques utilisés

## Les lexiques français

Les lexiques français utilisés lors des travaux de recherche rapportés dans cette thèse peuvent être divisés en deux groupes: ceux ayant été utilisés lors de tests de reconnaissance et ceux ayant exclusivement servis aux études de compression rapportées au chapitre 5.

### Général et fondamental

Trois lexiques français ont servis aux études de compression. Tous trois ont été élaborés à partir d'une version informatisée "maison" du dictionnaire Petit Robert [ROBE 86]. Chaque entrée de cette liste comprend entre autres renseignements la transcription graphique du mot, sa classe syntaxique et son code de déclinaison / conjugaison. En effet, il est important de rappeler qu'en français, seule la forme de base des mots a une entrée distincte dans un dictionnaire. Pour obtenir les autres formes, on doit en général s'en remettre à une forme modèle contenue dans la définition du mot. Dans la liste informatisée, cette forme modèle est remplacée par un code précis, qui associé à deux tables de terminaisons permet d'obtenir toutes les formes déclinées / conjuguées associées à une forme de base.

Ces formes furent générées pour deux des lexiques utilisés, LF1 et LF2. Pour LF3 seules les formes de bases furent utilisées. Dans le cas des lexiques utilisées lors des essais de compression, toutes les graphies similaires, qui en général[1] se prononcent de façon identique, ont ensuite été regroupées.

Par la suite, chacune des ces entrées fut phonétisée, c'est-à-dire qu'à chaque mot on a fait correspondre une séquence de phonèmes correspondant à sa prononciation. Cette opération est, en français, relativement facile à automatiser, car la correspondance entre formes

1. Il existe malheureusement des cas qui doivent être traités à la main, par exemple: *couvent - /kuvA/* (nom) et */kuv*/* (verbe), *fils* - /*fil*/ et /*fis*/.

graphiques et formes phonémiques est assez rigoureuse. Par conséquent, plusieurs versions de logiciels effectuant cette tâche existent. Celui employé ici a gracieusement été donné par une équipe de recherche du CNET. Cependant, la prononciation québécoise de certains phonèmes ou séquences de phonèmes étant systématiquement différente de leur prononciation en français de France, quelques modifications aux règles de phonétisation furent apportées [BROU 92]. Puis finalement toutes les phonies similaires étaient regroupées.

Ainsi traitées, des 52 514 formes de base de notre version “maison” du Petit Robert on obtient un dictionnaire de 180 205 entrées phonétiques différentes correspondant à un total de 363 976 graphies différentes (LF2). De la même façon, d’un lexique plus petit, composé à partir des 4000 mots français les plus usuels ([IPN 54] et [IPN 59]), on obtient un dictionnaire comptant 17 280 phonies pour 34 018 graphies différentes (LF1).

Finalement, LF3 élaboré à partir des formes de base seules compte 44 722 phonies pour 47 608 graphies. Le tableau 1 résume les principales caractéristiques des lexiques construits.

**Tableau A.1 Caractéristiques des différents lexiques français utilisés au chapitre 5**

| | LF1 | LF2 | LF3 |
|---|---|---|---|
| formes de bases | 3 925 | 52 514 | 52 514 |
| graphies | 34 018 | 363 976 | 47 608 |
| phonies | 17 280 | 180 205 | 44 722 |

Le nombre de phonèmes utilisés lors de la phonétisation des entrées de LF1, LF2 et LF3 était de 37.

**TransTalk**

Les lexiques utilisées lors des deux phases du projet TransTalk contenaient l’un aux environs de 15000 mots (première phase), l’autre aux environs de 20000 mots (deuxième phase). Ces mots furent obtenus de transcriptions tirées du Hansard. Le Hansard consistent en deux versions des transcriptions des débats du parlement canadien, l’une en français, l’autre en anglais. Cela a fait de cette base de donnée un matériel idéal pour l’élaboration de systèmes de traduction ou d’aide à la traduction automatiques. Les transcriptions utilisées dans l’élaboration du système TransTalk portent sur deux années. Elles comportent aux environs de 10 millions de mots français et 10 millions de mots anglais dont exactement 112976 mots anglais différents et 136177 mots français différents.

Seuls les mots dont la fréquence était supérieure à une certaine valeur furent sélectionnées pour faire partie des lexiques utilisés (aux environs de 100 en isolé et 20 en continu). A ces derniers furent rajoutées toutes les entrées manquantes tirées des transcriptions des phrases utilisées pour l'entraînement des modèles acoustiques.

Les phonèmes utilisés lors de la phonétisation des entrées de TransTalk sont au nombre de 47 (20 voyelles et 27 consonnes) dans le cas du lexique utilisé pour la première phase et de 44 (20 voyelles et 24 consonnes) dans le cas de la deuxième phase. Dans les deux cas, la phonétisation fut effectuée au moyen d'une version complètement remaniée du phonétiseur obtenu du CNET.

### Les lexiques anglais

Les lexiques anglais utilisés pour les tests de compression nécessitaient moins de manipulation que leurs homologues français. Un petit lexique de 6145 mots, (LA2), fut directement obtenu du fichier lexique de la base de données TIMIT, lequel contenait les entrées graphiques et phonétiques correspondantes. Les phonèmes utilisés lors de cette transcription furent ceux utilisés par TIMIT. Ils sont au nombre de 45. TIMIT est l'une des premières bases de données qui fut rendue publiquement accessible dans le cadre des travaux sur la RAP subventionnés par ARPA.

Un autre grand lexique de 47 443 entrées phonétiques (LA1) fut pour sa part obtenu d'une version informatique du Merriam-Webster [WEBS 90] disponible sur les machines NeXT. Essentiellement, on retrouve dans la version informatique de ce dictionnaire, la même information, à la lettre et à la fonte prêt, que ce qui se trouve dans la version papier. Il a donc été relativement facile d'aller extraire le champs phonétique de chaque entrée. En fait, moyennant un peu plus de travail, il aurait aussi été assez aisé d'obtenir aussi les formes dérivées de chaque entrée, mais comme leur nombre est nettement moins grand en anglais, cet exercice ne fut pas effectué. Le nombre de phonèmes utilisés pour ce lexique fut de 41, soit légèrement inférieur au nombre utilisé pour TIMIT. Ceci est essentiellement dû au fait que certains phonèmes différenciés dans la base de données de TIMIT ne le sont pas dans le Webster, en particulier le *e* muet. Le tableau 2 résume les principales caracté-

**Tableau A.2 Caractéristiques des différents lexiques anglais utilisés au chapitre 5**

| | **LF1** | **LF2** |
|---|---|---|
| nb. de phonèmes | 45 | 41 |
| graphies | 53 228 | 6 228 |
| phonies | 47 443 | 6 145 |

ristiques des lexiques construits.

### ATIS

Le lexique de ATIS comporte 1863 entrées, parmi lesquelles 144 possèdent plus d'une forme phonétique. Ces formes phonétiques sont tirées du dictionnaire CMU mis à la disposition des participants aux évaluations chapeautées par ARPA. Ce dictionnaire comporte plus de 100000 entrées phonétisées au moyen de 41 phonèmes.

De son côté, la base de données ATIS comporte plus de 23000 énoncés. Cette base de données est le fruit de plusieurs séances de collectes effectuées sur plusieurs sites américains, selon des méthodes allant de la lecture d'énoncés à l'enregistrement de séances résultant de l'utilisation d'applications réelles sur une période d'au moins 4 ans.

### WSJ

Le lexique du WSJ utilisé lors de l'évaluation de novembre 1994 compte 20000 entrées pour lesquelles la forme phonétique fut directement tirée du dictionnaire CMU. La liste de ces entrées fut directement fournie par ARPA en même temps que les données du test de développement de novembre 1994.

Un autre lexique disponible et sur lequel certaines des expériences rapportées ont été effectuées compte 5000 entrées. Ce lexique fut obtenu de la même façon que le lexique de 20000 entrées mais à partir d'une liste fournie avec le test de développement de novembre 1993.

# Annexe B  Calcul du taux de reconnaissance

Il existe deux mesures reconnues pour estimer le taux de reconnaissance d'un système: le pourcentage de reconnaissance (et son complément, le taux d'erreur qui est celui le plus usité) et le pourcentage de mots corrects ou en bref pourcentage correct. Pour les obtenir, l'on estime pour chaque énoncé, au moyen d'un algorithme d'alignement dynamique classique, le nombre de mots omis $del$, substitués $subs$ et insérés $ins$ de la solution par rapport à la retranscription de l'énoncé (voir la figure 1).

énoncé: **find flight from san diego to phoenix on monday a m**

solution: **I'd like to san diego to phoenix on monday eight a m**

*ins*: 1 *del*: 0 *subs*: 3

énoncé: **the flight number is irrelevant**

solution: **the flight number zero**

*ins*: 0 *del*: 1 *subs*: 1

insertion
substitution
omission

**Figure B.1  Exemples d'alignement entre énoncés et solutions**

Le pourcentage de reconnaissance d'un test comptant un nombre total de $N_{test}$ mots s'obtient au moyen de l'équation suivante:

$$\text{pourc. recog} = \left(\frac{del + subs + ins}{N_{test}}\right) \times 100 \qquad \text{(EQ B.1)}$$

et le pourcentage correct au moyen de:

$$\text{pourc. recog} = \left(\frac{del + subs}{N_{test}}\right) \times 100 \qquad \text{(EQ A.2)}$$

La différence entre le pourcentage de reconnaissance et de mots corrects est fonction du nombre de mots inutilement insérés. Le pourcentage de mots corrects est utile dans les cas, par exemple, où c'est l'obtention du sens général plutôt que la retranscription parfaite d'un énoncé qui est visé.

# Annexe C  Description de la banque de logiciels utilisée

Tous les logiciels dont ils est question dans cette thèse ont été écrits en C. Ils totalisent environ 25000 lignes de code. Le tout est partie constituante d'un ensemble de logiciels plus volumineux (aux environ de 100000 lignes de code) permettant d'effectuer toutes les tâches directement ou indirectement liées à la reconnaissance (entraînement des HMMs, entraînement du modèle du langage, définition de dictionnaires etc).

Toutes les expériences ont été effectuées sur des stations SUN sous environnement UNIX. Lorsqu'il est fait mention d'une comparaison au niveau des temps d'exécution, grand soin a été pris pour que les mêmes conditions prévalent pour tous les tests. Il n'est cependant pas garanti que tous les groupes de tests aient été effectués sous les mêmes conditions.

# Annexe D Résultats ATIS - évaluations novembre 92 et 93

Les résultats de l'évaluation de novembre 92 sont tirés des "handouts" distribués lors du SLT Workshop de 1992, malheureusement non disponible sous forme reliées[1], ceux de l'évaluation de novembre 93 sont tirés de la page 37 des Proceedings du SLT Workshop de 1994.

**Tableau D.1 Résultats officiels du test SPREC des évaluations ATIS de novembre 92 et novembre 93**

| **Evaluation 92** | | **Evaluation 93** | |
|---|---|---|---|
| **équipe** | **taux d'erreur** | **équipe** | **taux d'erreur** |
| BBN | 4.3 | BBN | 3.3 |
| CMU | 4.7 | CMU | 3.3 |
| SRI | 5.7 | Unisys | 3.9 |
| MIT | 8.1 | MIT | 4.5 |
| AT&T | 8.4 | SRI | 4.5 |
| CRIM | 14.1 | CRIM | 6.6 |
| INRS | 100.0[a] | AT&T | 9.0 |

a. Bien qu'étant tenu de soumettre des résultats, aucun ne le fut d'où un taux d'erreur de 100.0

1. On retrouve dans [CARD 92] l'essentiel de la présentation du CRIM au SLT Workshop de 92.

# Annexe E Résultats WSJ - évaluation novembre 94

Les résultats suivants sont tirés de la page 21 des Proceedings du SLT Workshop de 1995.

**Tableau E.1 Résultats officiels du test P0 de d'évaluation WSJ de novembre 94**

| équipe | taux d'erreur | section de référence | équipe | taux d'erreur | section de référence |
|---|---|---|---|---|---|
| Cambridge | 10.5 | 7.3.4 | Dragon | 13.2 | 7.3.8 |
| IBM | 11.1 | 7.3.9 | Phillips | 13.4 | 7.3.8 |
| BBN | 11.9 | 7.3.5 | CMU | 13.7 | 7.3.2 |
| LIMSI | 12.1 | 7.3.6 | Cambridge-2[a] | 14.1 | - |
| SRI | 12.2 | 7.3.3 | MIT | 19.0 | 7.3.1 |
| Boston Univ. | 12.9[b] | - | CRIM | 20.2 | 8 |
| AT&T | 13.0 | 7.3.10 | Karlsruhe Univ. | 22.8 | - |
| New-York Univ. | 13.0[c] | - | | | |

a. système mixte modèles neuronaux et HMMs

b. meilleur de 3 soumissions

c. meilleur de 2 soumissions

Les systèmes du Boston University et du New York University ne sont pas des systèmes complets. Ils ne font que re-scorer une liste des N-meilleures hypothèses fournies par le groupe de recherche de BBN et ne sont donc pas décrits en détails dans cette thèse. Il en est de même pour le système à approche mixte HMMs/modèle neuronaux de Cambridge, ces derniers n'ayant fait le sujet d'aucune recherche pour cette thèse et pour le système du Karlsruhe University, ce dernier en étant, tout comme le système du CRIM, à sa première évaluation WSJ.

# Bibliographie[1]

[ADDA 86] G. Adda, M. Eskenazi, P.E. Stern, ‘‘Reconnaissance de grands vocabulaires: utilisation et évaluation de traits grossiers”, *JEP 86*, pp. 219-222, 1986.

[ALLE 93] F. Alleva, X. Huang, M.-Y. Hwang, “An Improved Search Algorithm Using Incremental Knowledge for Continuous Speech Recognition”, *ICASSP 93*, pp. 307-310, 1993.

[ATAL 74] B. Atal, “Effectiveness of Linear Prediction Characteristics of the Speech Wave for Automatic Speaker Identification and Verification”, *Journal of the Acoustic Society of America*, Vol. 55, pp. 1304-1312, Juin 1974.

[APPE 88] A.W. Appel, G.J. Jacobson, “The World’s Fastest Scrabble Program”, *Communications of the ACM*, Vol.31, No.5, pp. 572-578, 585, May 1988.

[AUST 90] S. Austin, P. Peterson, P. Placeway, R. Schwartz, J. Vandergrift, “Toward a Real-Time Spoken Language System Using Commercial Hardware”, *90 ARPA SLT Workshop*, pp. 72-77, 1990.

[AVER 86] A. Averbuch, L. Bahl, R. Bakis, P. Brown, A. Cole, G. Daggett, S. Das, K. Davies, S. De Gennaro, P. de Souza, E. Epstein, D. Fraleigh, F. Jelinek, S. Katz, B. Lewis, R. Mercer, A. Nadas, D. Nahamoo, M. Picheny, G. Shichman, P. Spinelli, ‘‘An IBM-PC Based Large-vocabulary Isolated-utterance Speech Recognizer”, *ICASSP 86*, pp. 53-56, 1986

[AVER 87] A. Averbuch, L. Bahl, R. Bakis, P. Brown, G. Daggett, S. Das, K. Davies, S. De Gennaro, P. de Souza, E. Epstein, D. Fraleigh, F. Jelinek, B. Lewis, R. Mercer, J. Moorhead, A. Nadas, D. Nahamoo, M. Picheny, G. Shichman, P. Spinelli, D. Van Compernolle, H. Wilkens, ‘‘Experiments with the TANGORA 20 000 Words Speech Recognizer”, *ICASSP 87*, pp. 701-704, 1987.

1. La liste des acronymes inclut ceux cités au niveau des entrées bibliographiques.

[BAHL 83] L.R. Bahl, F. Jelinek, R.L. Mercer, “A Maximum Likelihood Approach to Continuous Speech Recognition”, *IEEE PAMI*, Vol. 5, No. 2, pp. 179-190, March 1983.

[BAHL 88a] L.R. Bahl, R. Bakis, P.V. de Souza, R.L. Mercer, “Obtaining Candidate Words by Polling in a Large Vocabulary Speech Recognition System”, *ICASSP 88*, pp. 489-492, 1988.

[BAHL 88b] L.R. Bahl, P. Brown, P.V. de Souza, R.L. Mercer, “Acoustic Markov Models Used in the Tangora Speech Recognition System”, *ICASSP-88*, pp. 497-500, 1988.

[BAHL 89] L.R. Bahl, R. Bakis, J. Bellegarda, P.F. Brown, D. Burshtein, S.K. Das, P.V. de Souza, P.S. Gopalakrishnan, F. Jelinek, D. Kanevsky, R.L. Mercer, A.J. Nadas, D. Nahamoo, M.A. Picheny, “Large Vocabulary Natural Language Continuous Speech Recognition”, *ICASSP 89*, pp. 465-467, 1989.

[BAHL 90] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Kanevsky, D. Nahamoo, “Constructing Groups of Acoustically Confusable Words”, *ICASSP 90*, pp. 85-88, 1990.

[BAHL 91] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, M.A. Picheny, “Decision Trees for Phonological Rules in Continuous Speech”, *ICASSP-91*, pp. 185-188, 1991.

[BAHL 95] L.R. Bahl, S. Balakrishnan-Aiyer, M. Franz, P.S. Gopalakrishnan, R. Gopinath, M. Novak, M. Padmanabhan, S. Roukos, “The IBM Large Vocabulary Continuous Speech Recognition System for the ARPA NAB News Task”, *95 ARPA SLT Workshop*, pp. 121-126, 1995.

[BAKE 74] J.K. Baker, “The Dragon Automatic Speech Recognition System”, *Speech Communication Seminar*, Stockholm, pp. 1-12, 1974.

[BAKE 75] J.K. Baker, “Stochastic Modeling for Automatic Speech Understanding”, *Speech Recognition*, D.R. Reddy Editeur, Academic Press, New York, pp. 512-542, 1975.

[BAMB 91] P. Bamberg, A. Demedts, J. Elder, C. Huang, C. Ingold, M. Mandel, L. Manganaro, S. van Even, “Phoneme-Based Training for Large-Vocabulary Recognition in Six European Languages”, *Eurospeech 91*, pp. 175-181, 1991.

[BATE 93] M. Bates, R. Bobrow, P. Fung, R. Ingria, F. Kubala, J. Makhoul, L. Nguyen, R. Schwartz, D. Stallard, “The BBN/HARC Spoken Language Understanding System”, *ICASSP 93*, pp. II-111-II-114, 1993.

[BAUM 72] L.E. Baum, "An Inequality and Associated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov Processes", *Inequalities*, 3:1-8, 1972.

[BLAC 93] E. Black, *Statistically-Driven Computer Grammars of English: The IBM/ Lancaster Approach*, 248 pages, Rodopi, Amsterdam - Atlanta, 1993.

[BOIT 93] J.-M. Boite, H. Bourlard, B. D'hoore, M. Haesen, "A New Approach Towards Keyword Spotting", *Eurospeech 93*, pp. 1273-1276, 1993.

[BREI 84] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, *Classification and Regression Trees*, Wadsworth Statistics/Probability Series, Belmont, CA, 1984.

[BROU 92] J. Brousseau, S.A. Fox, "Dialect-Dependent Speech Recognizers for Canadian and European French", *ICSLP 92*, pp.1003-1006, 1992.

[BROU 95] J. Brousseau, C. Drouin, G. Foster, P. Isabelle, R. Kuhn, Y. Normandin, P. Plamondon, "French Speech Recognition in an Automatic Dictation System for Translators: the TransTalk Project", à paraître dans *Eurospeech 95*.

[BROW 90] M.K. Brown, J.G. Wilpon "Automatic Generation of Lexical and Grammatical Constraints for Speech Recognition", *ICASSP 90*, pp. 733-736, 1990.

[CARD 92] R. Cardin, R. De Mori, D. Goupil, R. Lacouture, E. Millien, Y. Normandin, "CRIM's Spontaneous Speech Recognition System for the ATIS Task", *ICSLP 92*, pp. 623-626, 1992.

[CARL 86] R. Carlson, K. Elenius, B. Granstrom, S. Hunnicutt, "Phonetic Properties of the Basic Vocabulary of Five European Languages: Implications for Speech Recognition", *ICASSP 86*, pp. 2763-2766, 1986.

[CHAS 95] L. Chase, R. Rosenfeld, A. Hauptmann, M. Ravishankar, E. Thayer, P. Placeway, R. Weide, C. Lu, "Improvements in Language, Lexical, and Phonetics Modeling in Sphinx-II", *95 ARPA SLT Workshop*, pp. 60-65, 1995.

[DERO 84] A.M. Derouault, B. Merialdo, "Language modelling at the syntactic level", *Proceedings of the. 7th Int. Conf. on Pattern Recognition*, vol II, Montréal, pp. 1373-1375, August 1984.

[DERO 86] A.M. Derouault, B. Merialdo, "Natural language modelling for phone-to-text transcription", *IEEE PAMI*, Vol. 8, No 6, pp. 742-749, Novembre 1986.

[DERO 89] A.M. Derouault, B. Merialdo, "Improving Speech Recognition Accuracy with Contextual Phonemes and MMI Training", *ICASSP 89*, pp. 116-119, 1989.

[DIGA 95] V. Digalakis, M. Weintraub, A. Sankar, H. Franco, L. Neumeyer, H. Murveit, “Continuous Speech Dictation on ARPA’s North American Business News Domain”, *95 ARPA SLT Workshop*, pp. 88-93, 1995.

[DUGA 95] C. Dugast, R. Kneser, X. Aubert, S. Ortmanns, K. Beulen, H. Ney, “Continuous Speech Recognition Tests and Results for the NAB’94 Corpus”, *95 ARPA SLT Workshop*, pp. 156-161, 1995.

[DYME 94] M. Dymetman, J. Brousseau, G. Foster, P. Isabelle, Y. Normandin, P. Plamondon, “Towards an Automatic Dictation System for Translators: the TransTalk Project”, *ICSLP 94*, pp. 691-694, 1994.

[ELBE 90] M. Elbeze, A.M. Derouault, “A Morphological model for large vocabulary speech recognition”, *ICASSP 90*, pp. 577-580, Albuquerque, 1990.

[FISS 88] L. Fissore, P. Laface, G. Micca, R. Pieraccini, “Very Large Vocabulary Isolated Utterance Recognition: a Comparison Between One Pass and Two Pass Strategies”, *ICASSP 88*, pp. 203-206, 1988.

[FISS 89] L. Fissore, P. Laface, G. Micca, R. Pieraccini, “Lexical Access to Large Vocabularies For Speech Recognition”, *IEEE Trans. on ASSP*, Vol. 37, No. 8, pp. 1197-1989, Août 1989.

[FISS 93] L. Fissore, E. Giachin, P. Laface, P. Massafra, “Using Grammars in Forward and Backward Search”, *Eurospeech 93*, pp. 1525-1528, 1993.

[GAUV 94a] J.L. Gauvain, L.F. Lamel, G. Adda, M. Adda-Decker, “The LIMSI Continuous Speech Dictation System”, *94 ARPA HLT Workshop*, pp. 298-303, 1994.

[GAUV 94b] J.L. Gauvain, L.F. Lamel, G. Adda, M. Adda-Decker, “Continuous Speech Dictation in French”, *ICSLP 94*, pp. 2127-2130, 1994.

[GAUV 94c] J.L. Gauvain, L.F. Lamel, G. Adda, J. Mariani, “Speech-to-Text Conversion in French”, I*nternational Journal on Pattern Recognition and A.I.*, Vol. 8, No. 1, 1994.

[GAUV 94d] J.L. Gauvain, L.F. Lamel, G. Adda, M. Adda-Decker, “Speaker-Independent Continuous Speech Dictation”, *Speech Communication*, Vol. 15, pp. 21-37, 1994.

[GILL 92] L. Gillick, B. Peskin, R. Roth, “Rapid Match Training for Large Vocabularies”, *92 ARPA HLT Workshop*, pp. 328-332, 1992.

[GOPA 95] P.S. Gopalakrishnan, L.R. Bahl, R.L. Mercer, “A Tree Search Strategy for Large-Vocabulary Continuous Speech Recognition”, *ICASSP-95*, pp. 572-575, 1995.

[GUPT 88] V.N. Gupta, M. Lennig, P. Mermelstein, ‘‘Fast Search Strategy in a Large Vocabulary Word Recognizer”, *Journal Acoust. Soc. Am.*, Vol. 6, pp. 2007-2017, Décembre 1988.

[HUAN 93a] X. Huang, F. Alleva, M.-Y. Hwang, R. Rosenfeld, “An Overview of the SPHINX-II Speech Recognition System”, *93 ARPA HLT Workshop*, pp. 81-87, 1993.

[HUAN 93b] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, R. Rosenfeld, “The SPHINX-II Speech Recognition System: an Overview”, *Computer Speech and Language,* Vol. 7 No. 2, pp, 137-148, Avril 1993.

[HWAN 92] M.Y. Hwang, X Huang, “Subphonetic Modeling with Markov States”, *ICASSP 92*, pp. I-33-I-36, 1992.

[HOPC 79] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[HUTT 84] D.P. Huttenlocher, V.W. Zue, ‘‘A Model of Lexical Access form Partial Phonetic Information”, *ICASSP 84*, pp. 26.4.1-26.4.4, 1984

[IPN 54] Institut Pédagogique National, *Le français fondamental, 1e degré*, 1954.

[IPN 59] Institut Pédagogique National, *Le français fondamental, 2e degré*, 1959.

[JELI 69] F. Jelinek, “A Fast Sequential Decoding Algorithm Using a Stack”, *IBM J. Research and Development*, Vol. 13, pp. 675-685, Novembre 1969.

[JELI 76] F. Jelinek, ‘‘Continuous Speech Recognition by Statistical Methods”, *Proceedings of the IEEE ASSP*, 64(4), pp. 532-556, Avril 1976.

[JELI 90] F. Jelinek, “Self-Organized Language Modeling for Speech Recognition”, *Reading in Speech Recognition*, édité par Alex Waibel et Kai-Fu Lee, Morgan Kaufmann, pp. 450-506, 1990.

[JELI 91] F. Jelinek, “Up from trigrams! - The Struggle for Improved Language Models”, *Eurospeech 91*, pp. 1037-1040, Genova, 1991.

[KATZ 87] S.M. Katz, “Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer”, *IEEE Transactions on ASSP*, Vol. 35, pp. 400-401, 1987.

[KENN 93] P. Kenny, R. Hollan, V. Gupta, M. Lennig, P. Mermelstein, D. O’Shaughnessy, ‘‘A*-Admissible Heuristics for Rapid Lexical Access”, *IEEE Trans. on Speech and Audio Processing*, Vol. 1, pp. 49-57, Janvier 1993.

[KENN 94] P. Kenny, G. Boulianne, H. Garudadri, S. Trudelle, R. Hollan, V. Gupta, M. Lennig, D. O’Shaughnessy, “Experiments in Continuous Speech Recognition Using Books on Tape", *Speech Communication*, Vol. 14, pp. 49-60, 1994.

[KITT 82a] R. Kittredge, "Sublanguages", *American Journal of Computational Linguistics*, Vol. 8, No. 2, 1982.

[KITT 82b] R. Kittredge, *Sublanguage: Studies of Language in Restricted Semantic Domains*, De Gruyter, 1982.

[KLAT 90] D.H. Klatt, "Review of the ARPA Speech Understanding Project", *Readings in Speech Recognition*, édité par Alex Waibel et Kai-Fu Lee, Morgan Kaufmann, pp. 554-575, 1990.

[KUHN 90] R. Kuhn, R. De Mori, "A Cache-Based Natural Language Model for Speech Recognition", *IEEE PAMI*, Vol. 12, No. 6, pp. 570-583, Juin 1990.

[KUHN 95] R. Kuhn, A. Lazaridès, Y. Normandin, J. Brousseau, "Improved Decision Trees for Phonetic Modeling", *ICASSP 95*, pp. 552-555, 1995.

[LACO 91] R. Lacouture, R. De Mori, "Lexical Tree Compression", *Eurospeech 91*, pp. 581-584, 1991.

[LACO 93] R. Lacouture, Y. Normandin, "Efficient Lexical Access Strategies", *Eurospeech 93*, pp. 1537-1540. 1993.

[LACO 95] R. Lacouture, Y. Normandin, H.M. Cung, "Using Two Pass Recognition for Adaptation or Rejection in Real-time Applications", à paraître dans *IEEE Signal Processing Society 1995 Workshop on ASR*.

[LAFA 87] P. Laface, G. Micca, R. Pieraccini, "Experimental Results on a Large Lexicon Access Task", *ICASSP 87*, pp. 20.4.1-20.4.4, 1987.

[LAFA 88] P. Laface, "Recognition of Words in Very Large Vocabulary", *Recent Advances in Speech Understanding and Dialog Systems*, NATO ASI Series F Volume 46, pp. 235-254, Edited by H. Niemann, M. Lany and G. Sagerer, 1988.

[LAFA 95] P. Laface, C. Vair, L. Fissore, "A Fast Segmental Viterbi Algorithm for Large Vocabulary Recognition", *ICASSP 95*, pp. 560-563, 1995.

[LAGG 85] E. Lagger, A.Waibel, "A Coarse Phonetic Knowledge Source for Template Independent Large Vocabulary Word Recognition", *ICASSP 85*, pp. 2.7.1-2.7.4, 1985.

[LEEH 89] K.F. Lee, H.-W. Hon, M.-Y. Hwang, S. Mahajan, R. Reddy, "The SPHINX Speech Recognition System", *ICASSP 89*, pp. S9.3, 1989.

[LEEH 90] K.F. Lee, H.-W. Hon, R. Reddy, "An Overview of the SPHINX Speech Recognition System", *IEEE Trans. on ASSP*, Vol. 38, No. 1, Janvier 1990.

[LEER 90] C.H. Lee, L.R. Rabiner, R. Pieraccini, J.G. Wilpon, "Acoustic Modeling for Large Vocabulary Speech Recognition", *Computer Speech and Language*, Vol. 4, pp. 127-165, 1990.

[LENN 92] M. Lennig, D. Sharp, P. Kenny, V. Gupta, K. Precoda, "Flexible Vocabulary Recognition of Speech", *ICSLP 92*, pp. 93-96, 1992.

[LIKE 95] Z. Li, P. Kenny, D. O'Shaughnessy, "Searching with a Transcription Graph", *ICASSP 95*, pp. 564-567.

[LJOL 95] A. Ljolje, M. Riley, D. Hindle, F. Pereira, "The AT&T 60,000 Word Speech-To-Text System", *95 ARPA SLT Workshop*, pp. 162-165, 1995.

[LOWE 90] B. Lowerre, R. Reddy, "The Harpy Speech Understanding System", Readings in Speech Recognition, édité par Alex Waibel et Kai-Fu Lee, Morgan Kaufmann, pp. 576-586, 1990.

[MARC 85] S.M. Marcus, ‘‘Associative Models and the Time Course of Speech", *Bibliotheca Phonetica*, No.12, pp. 36-52, Karger, Basel 1985.

[MERI 87] B. Merialdo, ‘‘Speech Recognition with Very Large Size Dictionary", *ICASSP 87*, pp. 364-367, 1987.

[MURV 93] H. Murveit, J. Butzberger, V. Digalakis, M. Weintraub, "Large-Vocabulary Dictation Using SRI's Decipher Speech Recognition System: Progressive Search Techniques", *ICASSP 93*, pp. II-319-II-322, 1993.

[MURV 94] H. Murveit, P. Monaco, V. Digalakis, J. Butzberger, "Techniques to Achieve an Accurate Real-Time Large Vocabulary Speech Recognition System", *94 ARPA HLT Workshop*, pp. 368-373, 1994.

[NEYA 94] H. Ney, X. Aubert, "A Word Graph Algorithm for Large Vocabulary, Continuous Speech Recognition", *ICSLP 94*, pp.1355-1358, 1994.

[NEWE 73] A. Newell, J. Barnett, J.W. Forgie, C.C. Green, D.H. Klatt, J.C.R. Licklider, J. Munson, D.R. Reddy, W.A. Woods, *Speech Understanding Systems: Final Report of a Study Group*, North-Holland/American Elsevier, Amsterdam, 1973.

[NGUY 94] L. Nguyen, R. Schwartz, Y. Zhao, G. Zavaliagkos, "Is N-Best Dead", *94 ARPA HLT Workshop*, pp. 386-389, 1994.

[NILS 82] N. Nilsson, *Principles of Artificial Intelligence*, Tioga Publishing Company, 1982.

[NORM 91] Y. Normandin, *Hidden Markov Models, Maximum Mutual Information Estimation, and the Speech Recognition Problem*, Thèse de doctorat, McGill University, Montréal, Juin 1991.

[NORM 94a] Y. Normandin, D. Bowness, R. Cardin, Y. Cheng, R. De Mori, C. Drouin, D. Goupil, R. Kuhn, A. Lazaridès, E. Millien, "CRIM's December 93 ATIS System", *94 ARPA SLT Workshop*, pp. 59-65, 1994.

[NORM 94b] Y. Normandin, R. Lacouture, R. Cardin, "MMIE Training for Large Vocabulary Speech Recognition", *ICSLP 94*, pp. 1367-1370, 1994.

[NORM 95] Y. Normandin, D. Bowness, R. Cardin, C. Drouin, R. Lacouture, A. Lazaridès, "CRIM's November 94 Continuous Speech Recognition System", *95 ARPA SLT Workshop*, pp. 153-155, 1995.

[ODEL 94] J.J. Odell, V. Valtchev, P.C. Woodland, S.J. Young, "A One Pass Decoder Design For Large Vocabulary Recognition", *94 ARPA HLT Workshop*, pp. 380-385, 1994.

[PAUL 91] D.B. Paul, "Algorithms for an Optimal A* Search and Linearizing the Search in the Stack Decoder", *ICASSP 91*, pp. 693-696, 1991.

[PAUL 92a] D.B. Paul, "An efficient A* Stack Decoder Algorithm for Continuous Speech Recognition with a Stochastic Language Model", *ICASSP 92*, pp. I-25 - I-28, 1992.

[PAUL 92b] D.B. Paul, J.M. Baker, "The Design for the Wall Street Journal-based CSR Corpus", *ICSLP 92*, pp. 899-902, 1992.

[PAUL 94] D.B. Paul, "The Lincoln Large-Vocabulary Stack-Decoder Based HMM CSR", *94 ARPA HLT Workshop*, pp. 374-379, 1994.

[PAUL 95] D.B. Paul, "New Developments in the Lincoln Stack-Decoder Based Large-Vocabulary CSR System", *95 ARPA SLT Workshop*, pp. 143-147, 1995.

[PERE 94] F. Pereira, M. Riley, R. Sproat, "Weighted Rational Transductions and their Application to Human Language Processing", *94 ARPA HLT Workshop*, pp. 262- 267, 1994.

[PLAC 93] P. Placeway, R. Schwartz, P. Fung, L. Nguyen, "The Estimation of Powerful Language Models From Small and Large Corpora", *ICASSP 93*, pp. II-33-II-36, 1993.

[RAHI 95] M.G. Rahim, C.-H. Lee, B.-H. Juang, "Robust Utterance Verification for Connected Digits Recognition", *ICASSP 95*, pp. 285-288, 1995.

[RICH 95] F. Richardson, M. Ostendorf, J.R. Rohlicek, "Lattice-Based Search Strategies for Large Vocabulary Speech Recognition", *ICASSP 95*, pp. 576-579, 1995.

[ROSE 94a] A.E. Rosenberg, C.H. Lee, F.K. Soong, “Cepstral Channel Normalization Techniques for HMM-Based Speaker Verification”, *ICSLP 94*, pp. 1835-1838, Septembre 1994.

[ROSE 94b] R. Rosenfeld, *Adaptative Statistical Language Modeling: A Maximum Entropy Approach*, Thèse de doctorat, Carnegie Mellon University, Pittsburgh, 19 avril 1994.

[RABI 86] L. R. Rabiner, B. H. Juang, “An Introduction to Hidden Markov Models”, *IEEE ASSP*, Janvier 1986.

[RABI 89] L.R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition”, *Proceedings of the IEEE*, Vol. 77, No. 2, Février 1989.

[ROBE 86] P. Robert, *Le Petit Robert 1*, Dictionnaire Le Robert, Paris, 2173 pages, 1986.

[ROTH 95] R. Roth, L. Gillick, J. Orloff, F. Scattone, G. Gao, S. Wegmann, J. Baker, “Dragon Systems’ 1994 Large Vocabulary Continuous Speech Recognizer”, *95 ARPA SLT Workshop*, pp.116-120, 1995.

[SENE 92] S. Seneff, “Robust Parsing for Spoken Language Systems”, *ICASSP 92*, Vol. 1, pp. 189-192, San Francisco, 1992.

[SCHW 90] R. Schwartz, Y.-L. Chow, “The N-Best Algorithm: An efficient and Exact Procedure for Finding the N Most Likely Sentence Hypotheses”, *ICASSP 90*, pp. 81-84, 1990.

[SCHW 91] R. Schwartz, S. Austin, “A Comparison of Several Approximate Algorithms for Finding Multiple (N-BEST) Sentence Hypotheses”, *ICASSP 91*, pp. 701-704, 1991.

[SHAU87] D. O’Shaughnessy, *Speech Communication Human and Machine,* Addison-Wesley, 568 pages, 1987.

[SPOH80] J.C. Spohrer, P.F. Brown, P.H. Hochschild, J.K. Baker, “Partial Backtrace in Continuous Speech Recognition”, *Proc of Int. Conf. on Systems, Man, and Cybernetics*, 1980.

[STEI 91] V. Steinbiss, “A Search Organization for Large-Vocabulary Recognition Based on N-Best Decoding”, *Eurospeech 91*, pp. 1217-1220, 1991.

[STEI 93] V. Steinbiss, H. Ney, R. Haeb-Umbach, B.-H. Tran, U. Essen, R. Kneser, M. Oerder, H.-G. Meier, X. Aubert, C. Dugast, D. Geller, “The Philips Research System for Large-Vocabulary Continuous-Speech Recognition”, *Eurospeech 93*, pp. 2125-2128, 1993.

[SOON 90] F.K. Soong, E.F. Huang, “A Tree-Trellis Based Fast Search for Finding the N Best Sentence Hypotheses in Continuous Speech Recognition”, *90 DARPA SLT Workshop*, pp. 12-19, Juin 1990, aussi *ICASSP 91*, pp. 705-708, 1991.

[STEI 94] V. Steinbiss, B.-H. Tran, H. Ney, “Improvements in Beam Search”, *ICSLP 94*, pp. 2143-2146, 1994.

[VALT 94] V. Valtchev, J.J. Odell, P.C. Woodland, S.J. Young, “A Dynamic Network Decoder Design for Large Vocabulary Speech Recognition”, *ICSLP 94*, pp. 1351-1354, 1994.

[VERN 89] G.J. Vernooij, G. Bloothooft, Y Van Holsteijn, ‘‘A Simulation Study on the Usefulness of Broad Phonetic Classification in Automatic Speech Recognition”, *ICASSP 89*, pp. 85-88, 1989.

[VITE 67] A.J. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm”, *IEEE Transactions on Information Theory*, Vol. 13, No. 2, Avril 1967.

[WEBS 90]] *Webster’s Ninth New Collegiate Dictionary and Webster’s Collegiate Thesaurus*, First Digital Edition, Release 1.0 (v31.0.1), NeXT Computer, Inc. and Merriam Webster Inc., 1990.

[WOOD 70] W.A. Woods, “Transition Network Grammars for Natural Language Analysis”, *Communication of the ACM*, Vol. 13, pp. 591-602, 1970.

[WOOD 95] P.C. Woodland, C.J. Leggetter, J.J. Odell, V. Valtchev, S.J. Young, “The Development of the 1994 HTK Large Vocabulary Speech Recognition System”, *95 ARPA SLT Workshop*, pp. 104-109, 1995.

[YOUN 94] S.J. Young, J.J. Odell, P.C. Woodland, “Tree-Based State Tying for High Accuracy Acoustic Modeling”, *94 ARPA HLT Workshop*, pp. 307-312, 1994.