CHAPITRE IV
Méthode utilisée pour la recherche des *N* meilleures solutions

1 Introduction

Nous avons décrit précédemment un ensemble de méthodes optimales ou sous-optimales permettant de rechercher les *N* meilleures solutions. Les méthodes optimales sont coûteuses en calcul si le nombre de solutions développées est très élevé. Dans le cadre de cette thèse, des tests préliminaires ont été réalisés sur le corpus des chiffres isolés (0 à 9), pour la recherche des *N* meilleures solutions (10 solutions possibles au maximum) en utilisant la méthode des *N* meilleurs modèles (voir chapitre précèdent). Nous avons constaté qu'à l'aide d'un post-traitement optimal on avait de fortes chances de réduire le taux d'erreur résiduel pour un nombre *N* de solutions petit ($N<5$). De ce fait, on a choisi de rechercher les *N* meilleures solutions par une méthode optimale. Entre la méthode de Schwartz et la méthode de Soong, on a préféré cette dernière. En effet, la méthode de Schwartz réalise une recherche en largeur utilisant l'algorithme de Viterbi. Ce type de recherche ne permet pas d'avoir les *N* meilleures solutions dans l'ordre de leurs scores réels. Dans la méthode de [Soong, 91], à cause de l'exactitude de la valeur de la fonction *h*, les *N* meilleures solutions sont déterminées dans l'ordre de leurs scores réels.

Cependant, en vue d'introduire le post-traitement segmental, pour retrouver la solution correcte dans la liste proposée, nous étions amenés à mémoriser dans une pile, le score, l'alignement et l'étiquette de chaque solution développée. Pour une recherche efficace et rapide des *N* meilleures solutions la gestion de la pile devenait un point critique.

Ce chapitre va décrire la manière dont a été adaptée la méthode de [Soong, 91], pour rechercher les *N* meilleures solutions. Par la suite nous allons présenter, sans toutefois rentrer dans les détails, comment gérer la pile pendant la phase de recherche des *N* meilleures solutions.

2 Description de la méthode de recherche

La méthode de recherche des N meilleures solutions se déroule en deux phases.

2.1 Phase aller

Elle est réalisée par l'algorithme de Viterbi (chapitre I paragraphe 4.4). Cet algorithme se charge de calculer et de mémoriser, pour chaque instant τ (τ varie entre 1 et T) et pour chaque état q_j du réseau (j varie de 1 à NQ), la probabilité maximale d'observation des τ premières trames le long du meilleur des chemins atteignant l'état q_j à l'instant τ. Le coût du noeud n_j (i.e. τ, q_j) est déterminé par :

$$h(\tau, q_j) = \underset{q_i}{Min}\left\{h(\tau-1, q_i) - \log\left[a_{i,j}.B_{i,j}(X[\tau])\right]\right\} \qquad \text{(IV.1)}$$

où $h(\tau, q_j)$ est le coût minimum pour passer de l'état q_i (i.e. τ, q_i) vers l'état q_j à travers la transition $t_{i,j}$ en émettant la trame acoustique $X[\tau]$ par la densité de probabilité $B_{i,j}$ associée à cette transition. Le calcul de h nécessite la prise en compte du coût du chemin déjà emprunté pour passer de l'état initial q_1 à l'état q_i.

2.2 Phase retour

L'objectif de cette phase retour est de retrouver à partir du noeud final n_F du réseau acoustique le noeud de départ en développant les N meilleurs chemins syntaxiquement différents. Cette phase est réalisée par l'algorithme A*. Le coût global de la séquence de mots correspondant à un chemin s en cours de développement, à un instant donné τ et pour un état intermédiaire du réseau acoustique q_j, est défini par :

$$f(s, \tau, q_j) = g(s, \tau, q_j) + h(\tau, q_j) \qquad \text{(IV.2)}$$

Ces notations montrent qu'on tient compte du chemin emprunté s, qui fournit une information importante : la séquence de mots partiellement reconnue.

$g(s, \tau, q_j)$ représente le coût du chemin s entre le noeud final n_F et le noeud n_j ; $h(\tau, q_j)$ est une estimation du coût du chemin restant à parcourir entre le noeud n_j et le noeud n_I.

Dans l'approche utilisée en deux passes, l'estimation de la portion du signal non encore traitée (i.e. $h(\tau, q_j)$) provient des valeurs mémorisées pendant la phase aller. Cette estimation (au sens de l'algorithme A*) est optimale et conduit alors à une recherche très efficace des N meilleures solutions.

La figure suivante illustre comment obtenir la fonction coût f à partir des deux fonctions coût g et h.

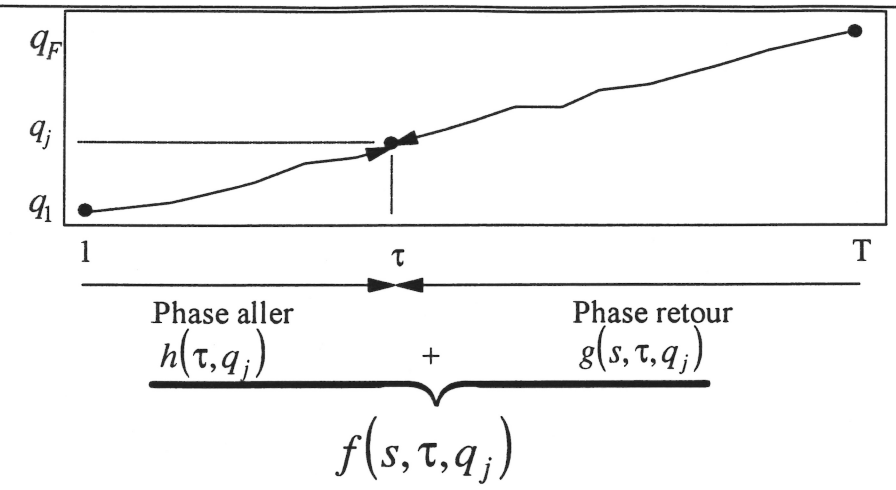

Figure IV.1 Calcul de la fonction coût f

calcul de g

Pour chaque noeud du réseau acoustique n_k, on recherche ses noeuds successeurs n_j. Pour chacun d'eux g est déterminée comme suit :

$$g(s,\tau,q_j) = g(s,\tau+1,q_k) - \log\left[a_{j,k}.B_{j,k}(X[\tau+1])\right] \qquad \text{(IV.3)}$$

$a_{j,k}$ est la probabilité de la transition de passage entre l'état q_j et l'état q_k. $B_{j,k}(X[\tau+1])$ est la probabilité d'émettre la trame acoustique $X[\tau+1]$ par la fonction de densité de probabilité $B_{j,k}$.

Pour mieux comprendre le déroulement de la phase retour, nous présentons ci-dessous une version possible de l'algorithme pour la recherche des N meilleures solutions. (Afin de simplifier la présentation, nous n'avons pas mentionné la gestion de la pile, et en particulier l'élimination des chemins non pertinents.)

Initialisation

$n_k = (T, q_{NQ})$; $g(Null, T, q_{NQ}) = 0$; $f(Null, T, q_{NQ}) = h(T, q_{NQ})$

N_développées = 0 /* Nombre de solutions développées */

N_demandées = N /* Nombre de solutions demandées */

Itérations

Tant que

[(N_développées ⟨ N_demandées) **ET** (il existe n_k ∈ à la pile tq n_k est à développer)]

Faire

Test (n_k ne correspond pas au noeud initial n_0 ?)

vrai

rechercher l'ensemble des successeurs de n_k ;

$E_{n_k} = \{n_j : j = 1, \ldots, K_{n_k}\}$ avec K_{n_k} le nombre de successeurs de n_k

Boucle sur n_j /* Développement de E_{n_k} */

calculer g

calculer f

insérer n_j dans la pile (position fonction du score)

Fin Boucle

faux

/ On est arrivé au noeud initial, donc on dispose d'une solution */

Test (Solution trouvée est-elle différente de celles déjà obtenues?)

vrai

incrémenter N_développées

faux

solution trouvée est rejetée

Fin Test

Fin Test

Rechercher un nouveau n_k dans la pile

Fin Tant que

Résultat

Récupérer de la pile le nom, le score et l'alignement des N solutions développées.

3 Gestion de la pile

La recherche des N meilleures solutions nécessite une gestion rigoureuse du temps de recherche et de l'espace mémoire requis par la pile. La première contrainte intervient essentiellement dans la manière de trouver le prochain noeud à développer. Quant à la seconde, elle est liée à la façon d'éliminer les chemins identiques de la pile. Dans ce qui suit nous allons développer plus en détails ces deux contraintes et proposer les solutions appropriées.

Nous rappelons les deux contraintes :

1/Comment retrouver le prochain noeud à développer?

2/Comment éliminer les chemins identiques pour gagner de la place mémoire dans la pile?

1/ Comment retrouver le prochain noeud à développer?

Dans la recherche des N meilleures solutions, ce qui nous importe le plus c'est d'avoir en notre possession d'une manière très rapide le noeud à développer ; c'est à dire le noeud de coût minimal. Ce mécanisme est géré par l'algorithme Heapsort [Knuth, 73][Press, 90] : après chaque insertion des nouveaux noeuds développés à partir de l'ensemble E_{n_k}, Heapsort arrange la pile de manière à pouvoir fournir le noeud de coût minimal. C'est un des algorithmes les plus rapides pour ce type d'opérations.

2/Comment éliminer les chemins identiques pour gagner de la place dans la pile ?

La recherche des N meilleures solutions est réalisée au niveau acoustique, de ce fait on peut trouver dans la pile à un instant donné des noeuds ayant : un même état acoustique q_j, un même instant temporel τ et une même séquence partielle de mots reconnus s mais des chemins acoustiques s_{a_m} différents. La figure IV.2 illustre ce cas de figure.

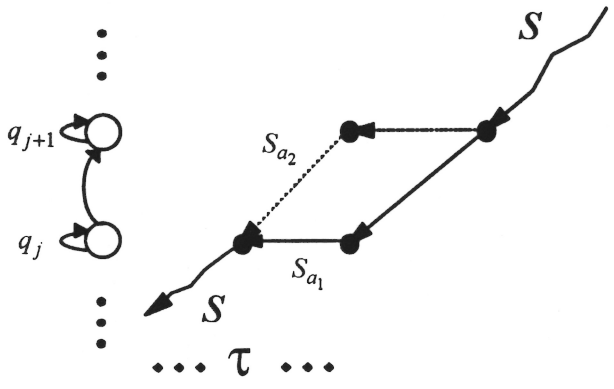

Figure IV.2 Développement d'un noeud au niveau acoustique

Sans ce préoccuper des ces noeuds identiques, nous avons constaté à la fin de la passe retour, que les noeuds de la pile correspondent, dans la plus part des cas, à des développements différents de la meilleure solution. Nous avons conclu que pour avoir les N meilleures solutions, on a le choix entre augmenter la taille de la pile, or il y a toujours une limite ; ou bien d'éliminer ces noeuds identiques. Nous avons décidé de borner la taille de la pile par une valeur maximale à ne jamais dépasser. Cette valeur est surtout liée à la configuration matérielle du système informatique utilisé. Puis de procéder de temps à autre à l'élimination des noeuds identiques de la pile. Or la phase retour est menée d'une manière asynchrone. De ce fait a un instant donné, on recherche dans la pile les noeuds contenant les même informations (chemin, instant et état). Par la suite, on va garder uniquement le noeud qui permet d'obtenir le chemin s avec un coût minimum, i.e. :

$$g(s,\tau,q_j) = \underset{q_k}{Min}\left\{g(s,\tau+1,q_k) - \log\left[a_{j,k}.B_{j,k}(X[\tau+1])\right]\right\} \qquad \text{(IV.4)}$$