Réseaux de confusion

Sommaire

En reconnaissance de la parole, l'approche standard (Bahl et al., 1983) utilisée pour le calcul de la meilleure solution est basée sur le critère de maximum *a posteriori* (MAP - *maximum a posteriori probability*). Ainsi, un SRAP produit comme résultat une séquence d'hypothèses de mots correspondant au chemin ayant la meilleure probabilité *a posteriori* étant donné les modèles acoustique et de langage. La théorie de Bayes (Duda et Hart, 1973) dit que la maximisation de la probabilité *a posteriori* de l'énoncé minimise le taux d'erreur phrase (SER - *Sentence Error Rate*, la probabilité d'avoir au moins une erreur dans la phrase). Or, la métrique employée généralement dans les évaluations d'un SRAP est le taux d'erreur mot (*WER*). On peut supposer de manière empirique que le

SER et le *WER* sont des métriques corrélées, et donc la minimisation du *SER* devrait conduire à la minimisation du *WER*. Des expériences (Stolcke et al., 1997) ont montré que la minimisation du *SER* ne garantit pas une minimisation du *WER*. Intuitivement on peut néanmoins envisager qu'en maximisant la probabilité *a posteriori* des mots, au lieu de la probabilité *a posteriori* des phrases, on devrait minimiser le *WER*.
La solution proposée dans (Mangu et al., 2000) est une approche de minimisation du *WER* basée sur l'utilisation des graphes de mots. Cette approche définit une nouvelle distance d'édition entre des hypothèses multiples basée sur la création d'un nouvel alignement pour toutes les hypothèses du graphe de mots, appelé réseau de confusion. Même si cette approche a tendance à surestimer le taux d'erreur mot dans certains cas, il a été montré (Mangu et al., 2000) que c'est une bonne approximation de *WER* standard.
Dans la section 5.1 nous présentons deux approches de minimisation du *WER* basées sur une liste de *N best* et sur des graphes de mots. Les différents concepts et notions théoriques utilisés par l'approche proposée dans (Mangu et al., 2000) sont présentés dans la section 5.2. Les différents algorithmes de génération des réseaux de confusion sont présentés dans la section 5.3. La dernière section 5.4 présente une comparaison de deux algorithmes de génération que nous avons implémentés. Une discussion des performances ainsi que du choix de l'algorithme utilisé dans la suite des travaux est aussi proposée dans cette section.

5.1 Minimisation du taux d'erreur mot

Pour obtenir le *WER* d'une hypothèse de reconnaissance W on doit calculer la distance d'édition entre W et une séquence de mots de référence R. Ceci est possible seulement si on connaît la séquence de mots de référence R. Si on ne connaît pas cette séquence de référence, ce qui est le cas en pratique pour un SRAP, on ne peut qu'estimer le *WER* sur un espace d'hypothèses. Pour cela on doit faire des hypothèses sur la séquence de mots de référence en choisissant, sur un espace donné, une séquence de mots comme étant correcte étant donné une probabilité associée. Si on considère $\mathcal{H}$ comme l'espace des hypothèses [1], l'espérance du taux d'erreur mot de l'hypothèse $W \in \mathcal{H}$ est donnée par la formule :

$$E[WER(W)|X] = \sum_{\tilde{R}} P(\tilde{R}|X) WER(W|\tilde{R}) \tag{5.1}$$

où $P(\tilde{R}|X)$ est la probabilité *a posteriori* associée à l'hypothèse $\tilde{R}$ et $WER(W|\tilde{R})$ est le taux d'erreur mot de l'hypothèse W connaissant la référence $\tilde{R}$. On obtient alors un nouveau problème d'optimisation par rapport à la séquence de mots W :

$$\hat{W} = \underset{W}{argmin} \sum_{\tilde{R}} P(\tilde{R}|X) WER(W|\tilde{R}) \tag{5.2}$$

L'équation 5.2 pose les bases de la minimisation du *WER* à partir des probabilités *a posteriori* des séquences de mots. Une implémentation algorithmique de cette équation

1. Dans ce cas, une hypothèse est une séquence de mots

implique deux étapes : une somme effectuée sur toutes les références potentielles $\tilde{R}$ suivie d'une minimisation sur l'hypothèse W. La première étape équivaut à calculer le *WER* pour toutes les paires $(W, \tilde{R})$ de l'espace d'hypothèses $\mathcal{H}$. Ceci implique un volume de calcul proportionnel au carré du nombre d'hypothèses dans $\mathcal{H}$. De plus, le calcul du *WER* implique un alignement du W et $\tilde{R}$, réalisé à l'aide de la programmation dynamique. Le temps de calcul du *WER* pour chaque paire d'hypothèses est donc proportionnel au carré de la longueur des hypothèses. La deuxième étape consiste à choisir l'hypothèse W avec la plus petite somme. Le volume de calcul élevé soulève la question de la faisabilité de la minimisation du WER dans un contexte de reconnaissance de la parole continue à très grand vocabulaire (LVCSR - *Large Vocabulary Continuous Speech Recognition*).

5.1.1 Approche par listes de *N best*

Des travaux (Stolcke et al., 1997; Goel et al., 1998) ont montré qu'on peut implémenter un algorithme de minimisation du *WER* en limitant l'espace des hypothèses à une liste de *N best*.

$$\hat{W} = \underset{W}{argmin} \sum_{i=1}^{N} P(W_i|X) WER(W|W_i) \tag{5.3}$$

où W et W_i sont des hypothèses de la liste. Les probabilités *a posteriori* des hypothèses sont estimées utilisant la liste de *N best*.
Pour chaque hypothèse dans la liste on calcule la somme des distances d'édition par rapport aux autres hypothèses de la liste pondérées par leur probabilité *a posteriori*. L'hypothèse choisie est celle à laquelle correspond la plus petite somme.
Pour un N très grand, l'algorithme nécessite N^2 alignements pour le calcul de la distance d'édition ce qui peut devenir coûteux pour une liste de plus de 1000 hypothèses. De plus, l'algorithme choisi systématiquement une hypothèse se trouvant dans les 10 meilleures de la liste *N best* initiale (observation empirique dans (Stolcke et al., 1997)). Pour réduire la complexité de l'algorithme, la minimisation du *WER* peut être réalisée sur une liste de K hypothèses plus petite ($K \ll N$ et une complexité de $O(KN)$). Les tests effectués (Stolcke et al., 1997) sur des corpus de type *Switchboard* et *Spanish Call-Home* montrent des performances faibles en termes de *WER*. Des tests sur un corpus de type *North American Business (NAB) Newstask*, avec un *WER* compris entre 10% et 30%, montrent que l'algorithme proposé choisit invariablement l'hypothèse ayant la meilleure probabilité *a posteriori*. Ceci s'explique par le fait que, dans ce type de corpus, une valeur petite du *WER* signifie que le nombre d'erreurs de mots dans une phrase est très petit (une ou deux erreurs). Ainsi, dans beaucoup de cas, une erreur mot correspond à une erreur phrase et inversement.
Il est donc possible de construire un algorithme de minimisation du *WER* basé sur des listes de *N best* et ayant une complexité de calcul relativement réduite. Il est aussi plus judicieux d'utiliser un algorithme de ce type sur des corpus ayant un *WER* élevé pour maximiser ses performances. Toutefois, cette approche n'est pas optimale car le nombre d'hypothèses est très restreint par rapport à l'espace de recherche du SRAP. Ceci a une influence importante sur l'estimation des probabilités *a posteriori* des énoncés de la liste,

mais aussi sur la minimisation du *WER* dans l'équation 5.3 du fait d'un nombre restreint de références.

5.1.2 Approche par graphes de mots

Un moyen de contourner le problème d'un espace d'hypothèses trop petit pour les listes de *N best* est l'utilisation de graphes de mots. Le passage aux graphes de mots implique néanmoins certains problèmes de calcul. Le nombre d'hypothèses dans un graphe de mots est très grand par rapport aux N hypothèses d'une liste *N best* et donc le calcul dans l'équation 5.3 pour choisir la meilleure hypothèse devient impossible. Un deuxième problème est posé par le calcul de la distance d'édition entre une hypothèse W et les autres hypothèses du graphe de mots. Ce calcul implique un alignement des mots de l'hypothèse W avec les mots de toutes les autres hypothèses. Cet alignement ne tient pas compte des supports temporels des mots. Or, dans un graphe de mots, l'alignement des chemins est basé sur leur support temporel. Une différence d'un seul mot dans un chemin du graphe peut alors avoir des conséquences globales sur l'alignement de ce chemin avec les autres chemins dans le graphe.
Afin de pouvoir construire un algorithme de minimisation du *WER* basé sur des graphes de mots, une nouvelle structure est proposée dans (Mangu et al., 2000) qui permet de réaliser un alignement des hypothèses multiples du graphe. Cette approche permet d'associer toutes les hypothèses du graphe de mots à un seul alignement et les erreurs entre n'importe qu'elle paire d'hypothèses sont calculées en fonction de cet alignement. Le *WER* se calcule alors en comptant localement le nombre d'erreurs (insertion, substitution, omission) entre deux hypothèses. L'estimation de l'espérance du *WER* de la séquence W ($E[WER(W)|X]$) peut alors être approximée par une somme sur les optimums locaux. La séquence $\hat{W}$ est choisie comme étant l'hypothèse qui minimise localement cette espérance.

5.2 Alignement des hypothèses multiples : concepts et définitions

La difficulté principale de l'approche de minimisation du *WER* à partir des graphes de mots est de trouver un alignement pour les hypothèses du graphe. De manière formelle, un alignement est une relation d'équivalence définie sur les hypothèses de mots du graphe (les transitions d'un graphe de mots) et un ordonnancement complet des classes d'équivalence qui soit cohérent avec celui du graphe de mots d'origine. Dans cette partie nous définissons ces notions et nous présentons le principe de construction de l'alignement.

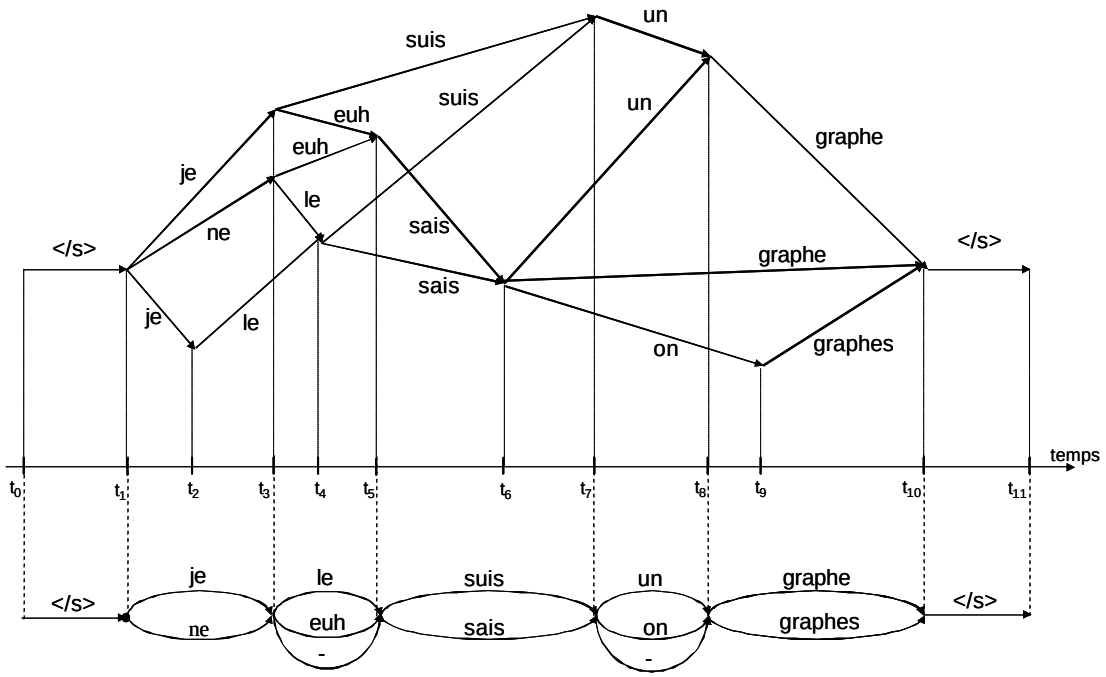

FIGURE *5.1* – Exemple de graphe de mots et le réseau de confusion correspondant. Le symbole "-" marque une omission

5.2.1 Classes d'équivalence

Une classe d'équivalence est constituée d'une ou plusieurs hypothèses de mots (transitions) du graphe ; les transitions ne doivent pas être en relation d'ordre. Par exemple, dans la figure 5.1, chaque transition peut constituer une classe d'équivalence, de même que les transitions portant les mots "**je**" et "**ne**" peuvent former une classe d'équivalence. En revanche, les transitions portant les mots "**un**" et "**graphe**" ne peuvent pas former une classe car il existe un chemin dans le graphe qui passe par les deux transitions.

5.2.2 Recouvrement temporel

Une notion importante utilisée pour regrouper des transitions dans une classe d'équivalence (ou des classes d'équivalence) est le **recouvrement temporel**. Il convient d'introduire cette notion afin de pouvoir quantifier la superposition entre les supports temporels de deux transitions dans un graphe de mots. La valeur du recouvrement temporel peut être calculée de différentes façons et nous précisons pour chaque algorithme décrit dans la section 5.3 la méthode de calcul utilisée.

5.2.3 Relation d'ordre

Intuitivement, sur un graphe de mot on peut établir un ordre entre les transitions se trouvant sur un même chemin. Par exemple, dans le graphe de mots représenté dans la figure 5.1, la transition portant le mot "**je**" et "avant" la transition portant le mot "**suis**", mais aussi "avant" la transition portant le mot "**graphe**". On peut affirmer ceci car dans le graphe de mots on retrouve le chemin "**je suis un graphe**" qui passe par les trois transitions.
Le graphe de mots définit une **ordre partiel** $\leq$ entre les transitions de la façon suivante : si E représente l'ensemble des transitions du graphe de mots, pour $e_1, e_2 \in E$, $e_1 \leq e_2$ si :

- $e_1 = e_2$ ou
- $Fstate(e_1) = Sstate(e_2)$ (où $Sstate$ et $Fstate$ sont les états de début et de fin d'un transition) ou
- $\exists f \in E$ tel que $e_1 \leq f$ et $f \leq e_2$.

D'une manière informelle on dit qu'il existe un chemin dans le graphe de mots qui passe par les deux transitions et que e_1 est "avant" e_2.
On peut étendre cette notion d'ordre aux classes d'équivalence. Si $\xi \subset 2^E$ est un ensemble de classes d'équivalence sur E, on peut définir l'ordre partiel $\preceq$ sur ξ de la manière suivante :

> Soient $C_1, C_2 \in \xi$:
> $C_1 \preceq C_2 \Longleftrightarrow \exists e_1 \in C_1$ et $e_2 \in C_2$ tel que $e_1 \leq e_2$

La relation d'ordre sur les classes d'équivalence $\preceq$ définie ainsi est consistante avec $\leq$ car elle préserve l'ordre temporel sur les mots des transitions dans le graphe. C'est une relation binaire sur ξ qui vérifie les trois propriétés suivantes :

- **réflexive** : elle met tout élément en relation avec lui même.
  $\forall C \in \xi, C \preceq C$
- **antisymétrique** : les éléments distincts ne sont jamais en relation mutuelle.
  $\forall C_1, C_2 \in \xi$ si $C_1 \preceq C_2 \bigwedge C_2 \preceq C_1$ alors $C_1 = C_2$
- **transitive** : deux éléments sont mis en relation dès qu'on peut transiter par un troisième.
  $\forall C_1, C_2, C_3 \in \xi$ si $C_1 \preceq C_3 \bigwedge C_3 \preceq C_2$ alors $C_1 \preceq C_2$

Dans la suite du document, on dit que deux transitions "sont en relation d'ordre" si $e_1 \leq e_2$ ou $e_2 \leq e_1$. On utilise la même expression pour les classes si $C_1 \preceq C_2$ ou $C_2 \preceq C_1$.

5.2.4 Principe de construction de l'alignement

Afin de générer l'alignement, on procède à un regroupement des classes d'équivalence suivant un certain nombre de critères et ceci jusqu'à l'obtention d'un ordre (linéaire) complet. Celui-ci est obtenu lorsque la condition suivante est satisfaite :

> $\forall e_1, e_2 \in E$ avec $e_1 \in C_1$ et $e_2 \in C_2$ on a :
> $C_1 \preceq C_2$ ou $C_2 \preceq C_1$

L'obtention d'un ordre complet en un nombre fini d'itérations est garantie par la définition de la relation d'ordre sur les classes d'équivalence. Étant donné cette relation, deux classes C_1 et C_2, qui ne sont pas en relation ($C_1 \not\preceq C_2$ et $C_2 \not\preceq C_1$), peuvent être regroupées afin de former une nouvelle classe d'équivalence dans le même espace ξ. Cet espace contiendra moins de classes qui ne sont pas en relation et on est assuré d'obtenir un ordre complet après un nombre fini d'itérations. Une preuve formelle est donnée dans (Mangu et al., 2000).
Avant de procéder au regroupement une étape d'initialisation des classes d'équivalence est nécessaire. Une méthode simple est, par exemple, de considérer chaque transition comme constituant une classe d'équivalence différente. D'autres manières d'initialiser les classes d'équivalence sont définies par les algorithmes décrit au 5.3.

5.2.5 Réseaux de confusion

Comme nous l'avons décrit ci-dessus, une fois l'ordre complet obtenu, l'alignement est une succession de classes d'équivalence. L'alignement ainsi obtenu est un réseau de confusion comme dans la figure 5.1. Les classes d'équivalence constituent donc les classes de mots du CN.
Chaque classe d'équivalence regroupe des transitions portant plusieurs hypothèses de mot ; plusieurs transitions pouvant porter la même hypothèse de mot. Dans une classe du CN, une hypothèse de mot est portée par une seule transition. Cette transition est le résultat du regroupement des transitions portant cette hypothèse de mot dans la classe d'équivalence. On calcule ainsi la probabilité *a posteriori* de l'hypothèse de mot du CN en sommant les probabilités *a posteriori* des transitions de la classe d'équivalence. D'une manière générale, la probabilité *a posteriori* d'une hypothèse de mot dans une classe du CN est égale à la somme de probabilités *a posteriori* des chemins du graphe qui contiennent l'hypothèse de mot dans une position donnée normalisée par la somme des probabilités *a posteriori* de tous le chemins du graphe.
On observe dans la figure 5.1 que tous les chemins du graphe passent par un des mots de la première classe du CN. La somme des probabilités *a posteriori* des mots dans cette classe ne peut alors dépasser 1. De manière générale, la somme des probabilités *a posteriori* des mots dans une classe est égale ou inférieure à 1. Mais cette somme peut être strictement inférieur à 1. Par exemple, pour la deuxième classe du CN, le chemin du graphe "**je suis un graphe**" ne passe par aucun des mots de la classe. La différence de probabilité pour atteindre une probabilité totale de 1 dans la classe représente la probabilité d'une omission de mot. Cette omission est alors introduite dans la classe sous la forme d'une transition portant le symbole "-" dont la probabilité *a posteriori* est égale à la probabilité nécessaire pour que la probabilité totale de la classe soit égale à 1. De manière générale, la probabilité de l'omission dans une classe est égale à la somme des probabilités *a posteriori* de tous les chemins du graphe qui ne contiennent aucun mot dans cette position.

Consensus hypothesis

Afin d'extraire la meilleure solution, on doit trouver la séquence de mots W qui minimise l'espérance du *WER*. Étant donné que les décisions prises dans chaque classe sont indépendantes, la minimisation de l'espérance du *WER* de la séquence W peut être approximée par une minimisation réalisée localement sur chaque classe en choisissant le mot correspondant. La meilleure solution des réseaux de confusion, appelée aussi *consensus hypothesis (CH)*, est obtenue en choisissant dans chaque classe la transition ayant la meilleure probabilité *a posteriori*. Cette transition peut porter un mot ou une omission, ce qui signifie qu'aucun mot ne sera présent dans la *consensus hypothesis* dans la position correspondante.

5.3 Description des principaux algorithmes

Comme nous l'avons décrit, la minimisation du taux d'erreur mot à partir d'un graphe de mots passe par la construction de structures plus compactes et plus adaptées à cette tache, les réseaux de confusion. Du fait de la structure des graphes, de leur grande taille et de leur redondance en ce qui concerne les hypothèses de mots (un nombre assez élevé de transitions portant la même hypothèse de mot mais avec des petites différences de support temporel), les post-traitements (comme la génération des réseaux de confusion) sur les graphes de mots peuvent s'avérer très complexes et très coûteux en termes de temps de calcul. Nous avons recensé dans la littérature trois algorithmes principaux pour la construction des CNs. Le premier est apparu en 1999 et a été présenté dans (Mangu et al., 1999) et décrit plus en détail par les mêmes auteurs dans (Mangu et al., 2000). Nous allons l'appeler l'algorithme de génération par regroupement des classes de transitions et il est détaillé dans la section 5.3.1. Un deuxième algorithme de génération de CNs a été décrit en 2003 dans (Hakkani-Tur et Riccardi, 2003). Une description plus détaillé de l'algorithme ainsi que différentes applications des CNs ont été présentés par les même auteurs dans (Hakkani-Tur et al., 2006). Nous allons l'appeler l'algorithme du *pivot* et une description détaillée est fournie dans la section 5.3.2. En 2006, un troisième algorithme de génération des CNs est présenté dans (Xue et Zhao, 2006) . Appelé algorithme de génération par regroupement en groupes d'états il est brièvement décrit dans la section 5.3.3. Pendant les travaux de cette thèse nous avons implémenté et testé les deux premiers algorithmes et nous les décrivons plus en détail. Nous présentons le dernier algorithme afin de donner une vision la plus complète possible de l'état de l'art de la génération des CNs, mais celui-ci n'a pas fait l'objet des travaux dans cette thèse. Nous faisons également une comparaison des performances dans la section 5.4 des deux premiers algorithmes et nous motivons par la même occasion le choix de l'algorithme utilisé dans la suite nos travaux.

5.3.1 Algorithme de génération par regroupement des classes de transitions

Ce premier algorithme présenté dans (Mangu et al., 1999) est un algorithme itératif constitué de deux étapes : le regroupement "*intra-mots*" et le regroupement "*inter-mots*". Dans la première étape, l'algorithme regroupe seulement les classes d'équivalence correspondant à la même hypothèse de mot et dans la deuxième étape, le regroupement se fait sur l'ensemble des classes d'équivalence issues de la première étape. Après cette première étape il est possible de calculer des probabilités *a posteriori* pour les mots mais c'est seulement au bout de la deuxième étape que les hypothèses de mots se trouvant en concurrence peuvent être identifiées.
Les classes d'équivalence sont initialisées de manière à ce qu'elles contiennent toutes les transitions portant le même mot et ayant exactement le même support temporel (trame de début et de fin).

Regroupement "*intra-mots*"

Le but de cette première étape est de regrouper toutes les transitions portant la même hypothèse de mot. Ceci est réalisé en regroupant des classes d'équivalence constituées des transitions portant la même hypothèse de mot qui ne sont pas en relation d'ordre (voir 5.2.3). La fonction de coût utilisée pour cette étape est une fonction de similarité entre deux classes d'équivalence :

$$SIM(C_1, C_2) = \max_{\substack{e_1 \in C_1 \\ e_2 \in C_2}} overlap(e_1, e_2) \cdot P(e_1) \cdot P(e_2) \tag{5.4}$$

où $overlap(e_1, e_2)$ est le recouvrement temporel entre les deux transitions. La valeur du recouvrement temporel est calculée comme étant le nombre de trames communes des support temporels des deux transitions divisé par l'union de leur longueur. Si on considère l'exemple dans la figure 5.2, la formule de calcul du recouvrement temporel est la suivante :

$$overlap(e_1, e_2) = \frac{t(3) - t(2)}{t(4) - t(1)} \tag{5.5}$$

où $t(i)$ est la trame de création de l'état i.

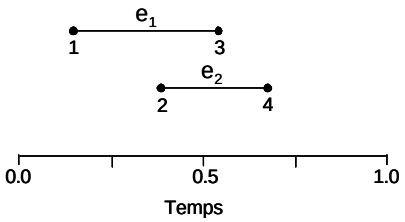

FIGURE 5.2 – Calcul du recouvrement temporel entre deux transitions.

Le recouvrement temporel est pondéré par les probabilités *a posteriori* des transitions

afin de rendre la mesure de similarité moins sensible aux hypothèses de mots peu probables (transitions ayant une faible probabilité). La valeur de la fonction de similarité entre deux classes d'équivalence est calculée en prenant tous les couples de transitions des deux classes et en calculant le produit entre leur recouvrement temporel et leur probabilité *a posteriori*, la meilleure valeur étant gardée.
A chaque itération, l'algorithme calcule la fonction de similarité entre toutes les paires de classes d'équivalence candidates (les classes qui ne sont pas en relation d'ordre et qui correspondent au même mot). Il regroupe ensuite pour chaque mot les 2 classes les plus similaires. A la fin de ce processus itératif on obtient un ensemble de classes d'équivalence, chaque classe étant constituée de transitions portant la même hypothèse de mot et ayant un recouvrement temporel non nul. Toutes les transitions portant la même hypothèse de mot ne se retrouvent pas forcément dans la même classe à cause des contraintes imposées par la relation d'ordre. Pour chaque classe on peut ainsi calculer la probabilité *a posteriori* du mot en sommant les probabilités *a posteriori* des transitions qui composent la classe.

Regroupement "*inter-mots*"

Dans cette étape, l'algorithme regroupe les classes d'équivalence portant des hypothèses de mots différentes. Les candidats à ce regroupement sont toutes les classes qui ne sont pas en relation d'ordre. L'algorithme s'arrête lorsqu'il n'y a plus de candidats (l'ordre total a été atteint).
La fonction de coût pour cette étape est une fonction de similarité basée sur la similarité phonétique entre deux mots :

$$SIM(C_1, C_2) = \underset{\substack{w_1 \in Mots(C_1) \\ w_2 \in Mots(C_2)}}{moy} sim(w_1, w_2) \cdot P_C(w_1) \cdot P_C(w_2) \tag{5.6}$$

où $P_C(w_1) = P(e \in C_1 : Mot(e) = w)$ et $Mot(e)$ est le mot porté par la transition e. La fonction de similarité phonétique $sim(w_1, w_2)$ se calcule sur une transcription phonétique[2] des mots. Sa valeur est égale à 1 moins la distance d'édition entre les transcription phonétiques de mots, sachant que la distance d'édition est normalisée par rapport à la somme des longueurs des transcriptions phonétiques. D'autres fonctions de similarité phonétique plus complexes peuvent être utilisées ; par exemple la fonction de similarité peut prendre en compte différents paramètres des phonèmes comme sa nature (voyelle/consonne) ou le voisement.

5.3.2 Algorithme du "*pivot*"

Un deuxième algorithme à été proposé en 20003 dans (Hakkani-Tur et Riccardi, 2003). Appelé aussi algorithme du "**pivot**", c'est un algorithme itératif qui réalise un regroupement guidé des classes d'équivalence. En effet, l'algorithme utilise une séquence

2. Dans (Mangu et al., 2000) on utilise la transcription phonétique la plus probable

de transitions, appelée *pivot*, comme point de départ et guide pour tout le processus de regroupement. Les classes d'équivalence sont initialisées de manière à ce que chaque classe contienne une seule transition du graphe. Intuitivement, on peut assimiler le graphe de mots à un nuage de points, où chaque point représente une classe d'équivalence. A chaque itération, l'algorithme insère des points du nuage dans le *pivot*, suivant certaines conditions, jusqu'à ce qu'il ne reste plus aucun point dans le nuage.

Le pivot

Le *pivot* est en fait un chemin complet du graphe de mots. Ce chemin peut être un chemin choisi de manière aléatoire. On peut aussi choisir le chemin le plus long, ou le meilleur chemin au sens du MAP. Ce chemin constitue le point de départ de l'algorithme. Les états de ce chemin héritent de l'information temporelle du graphe de mots. Ainsi, pour chaque état on garde l'instant temporel (la trame) du graphe de mots et on peut définir un support temporel entre deux états consécutifs.
Dans l'implémentation de l'algorithme nous utilisons le meilleur au sens MAP, un choix fait aussi dans (Hakkani-Tur et Riccardi, 2003).

Déroulement de l'algorithme

Nous présentons ci-dessous le déroulement de l'algorithme du "**pivot**" et nous détaillons ensuite chaque pas. Nous attribuons un nom à chaque étape (entre crochets) pour faciliter leur présentation dans la suite des travaux (notamment au chapitre 6).

1. Extraire la séquence *pivot*.

2. Pour toute transition T du graphe de mots parcouru dans un ordre topologique :

(1) *[Recherche de l'emplacement optimal]*
Trouver les deux états consécutifs $\hat{S}_s$ et $\hat{S}_f$ qui définissent le support temporel ayant le recouvrement temporel maximal, $overlap(T, \hat{S}_s, \hat{S}_f)$, avec T.

(2) *[Insertion transition]*
S'il n'existe aucune transition entre $\hat{S}_s$ et $\hat{S}_f$ qui soit en relation d'ordre avec T, insérer T entre les deux états.

(3) *[Création nouveau état et insertion transition]*
Sinon :

(a) Créer un nouveau état S_n.

(b) Changer l'état de fin de toutes les transitions qui partent de $\hat{S}_s$ à S_n [3].

(c) Insérer T entre S_n et $\hat{S}_f$.

Dans cette description, les états S_i sont des états du réseau de confusion et non pas des états du graphe de mots.
Le graphe de mots doit tout d'abord être ordonné de manière topologique (les états du

3. La création de l'état S_n est détaillée dans la suite du document. La transition T ne peut que venir "après" les transitions existante entre $\hat{S}_s$ et $\hat{S}_f$ du fait de parcours topologique du graphe

graphe sont ordonnés de telle manière que toute transition allant de l'état i à l'état j satisfait la condition $i < j$). Ensuite, le parcours du graphe de mots se fait lui aussi dans un ordre topologique, partant de l'état le plus petit (l'état du début de graphe) jusqu'au dernier état du graphe.

(1) Pour chaque transition T on doit chercher deux états consécutifs $\hat{S}_s$ et $\hat{S}_f$ qui définissent un support temporel ayant un recouvrement temporel maximal avec T. La valeur du recouvrement temporel est calculée comme étant le nombre de trames communes entre les supports temporels :

$$\begin{aligned}(\hat{S}_s, \hat{S}_f) &= \underset{S_s,S_f}{argmax}\ overlap(T, [S_s, S_f]) \\ &= \underset{S_s,S_f}{argmax}\ (min(t(S_f), t(Fstate(T))) - max(t(S_s), t(Sstate(T))) \end{aligned} \quad (5.7)$$

(2) On vérifie s'il existe une transition entre les deux états qui est en relation d'ordre avec T avec la précision que cette relation d'ordre ne peut être qu'une relation de précédence (la transition se trouvant entre les deux états ne peut que précéder T sur un chemin du graphe) du au parcours topologique du graphe de mots. Si cette transition n'existe pas, l'insertion du T entre les deux états peut se faire de deux manière différentes. S'il existe déjà une transition portant la même hypothèse de mot que T on ne fait que rajouter la probabilité *a posteriori* du T à celle de la transition existante. Sinon, on crée une transition entre les états $\hat{S}_s$ et $\hat{S}_f$ portant la même hypothèse de mot que T et ayant la même probabilité *a posteriori* (le pas 2.2 dans l'algorithme).

(3) Si toutefois il existe une transition entre $\hat{S}_s$ et $\hat{S}_f$ qui précède T sur un chemin du graphe, un nouvel état S_n est inséré dans l'alignement entre ces deux états (le pas 2.2.a). Un instant temporel est attribué à ce nouvel état $(t(S_n) \in (t(\hat{S}_s), t(\hat{S}_f))$. Les transitions qui partent de l'état $\hat{S}_s$ auront désormais comme état de fin le nouvel état créé S_n (le pas 2.2.b). On insère ensuite T entre l'état S_n et $\hat{S}_f$ (le pas 3.3.c). Dans (Hakkani-Tur et al., 2006) l'instant $t(S_n)$ est calculé comme étant la moyenne entre les instant temporels des états $\hat{S}_s$ et $\hat{S}_f$.

5.3.3 Algorithme de génération par regroupement en groupes d'états

L'algorithme de génération des réseaux de confusion présenté en 2006 dans (Xue et Zhao, 2006) part du principe que chaque état du CN est en fait un ensemble d'états du graphe de mots initial. Ainsi, l'algorithme se base sur le regroupement des états du graphe de mots en groupes d'états dans la première phase suivi de l'insertion des transitions entre les groupes d'états.
La construction des groupes d'états est très importante. Dans (Xue et Zhao, 2006) on propose de diviser les états du graphe de mots en un nombre fini de groupes d'états de manière à ce que les états de début et de fin de chaque transition se retrouvent dans deux groupes d'états consécutifs. L'algorithme de génération des CNs par regroupement en groupes d'états se base ensuite sur une série de suppositions. $N = \{n_0, n_1, \ldots\}$

représente l'ensemble des états et $\theta = \{T_0, T_1, \ldots\}$ représentes l'ensemble des transitions du graphe de mots, avec $t(n_i)$ désignant l'instant (trame) attribué à l'état n_i et $T_{u \to v}$ une transition du graphe avec les états de début et de fin qui sont respectivement u et v. $NS = \{N_0, N_1, \ldots\}$ représente l'ensemble des groupes des états dans le CN et $TS_{N_i \to N_j}$ est un ensemble de transitions ayant leurs états de début dans N_i et les états de fin dans N_j. Les propriétés suivantes sont supposées être vraies sur les réseaux de confusion :

a. $\forall n_i \in N_i$ et $n_j \in N_j$, $t(n_i) < t(n_j) \Rightarrow i \leq j$

b. $\forall n_i \in N_i$ et $n_j \in N_j$, $t(n_i) = t(n_j) \Rightarrow i = j$

c. $\forall T_{u \to v} \in \theta$, $u \in N_i$ *et* $v \in N_j$, si $T_{u \to v}$ appartient à l'ensemble de transitions $TS_{N_m \to N_n} \Rightarrow i \leq m < n \leq j$, avec $n = m + 1$. En effet, si les états de la transition ne sont pas dans deux groupes consécutifs, la transition doit être alignée sur deux groupes d'états consécutifs.

L'algorithme proposé dans (Xue et Zhao, 2006) est un algorithme itératif qui se déroule en trois étapes :

1. Trier les états du graphe de mots initial en fonction de l'instant $t(n_i)$ associé.
2. Inclure l'état n_0 dans le groupe d'états N_0.
3. Pour chaque état n_i parcouru dans l'ordre topologique ($i = 1, 2, \ldots$) :
   a) Si $n_{i-1} \in N_j$ et il n'existe aucune transition entre les états du groupe d'états N_j et n_i, alors n_i est inséré dans N_j. Sinon, n_i est inséré dans N_{j+1}.
   b) Pour chaque transition $T_{u \to n_i} \in \theta$, avec $u \in N_s$ et $n_i \in N_f$ si $f = s + 1$ alors la transition est insérée directement dans l'ensemble $TS_{N_s \to N_f}$. Sinon, la transition est insérée dans l'ensemble TS_{N_{n-1}, N_n}, avec $s + 1 \leq n \leq f$. L'ensemble de transitions est calculé en fonction de la probabilité *a posteriori* de la transition et du recouvrement temporel entre la transition et les différents ensembles possible : $n = \underset{s+1 \leq k \leq f}{argmax} SIM(TS_{N_{k-1}, N_k}, T)$.

$$SIM(TS_{N_{k-1}, N_k}, T) = \frac{1}{|TS_{N_{k-1}, N_k}|} \times \sum_{l \in TS_{N_{k-1}, N_k}} sim(w(l), w(T)) \cdot overlap(TS_{N_{k-1}, N_k}, T) \quad (5.8)$$

avec la fonction de similarité phonétique $sim(,)$ et la fonction de recouvrement temporel $overlap(,)$ calculées de la même manière que pour l'algorithme de regroupement par classes de transitions présenté au 5.3.1. La longueur $|TS_{N_{k-1}, N_k}|$ du groupe de transitions est égale à $T_{min}(N_k) - T_{max}(N_{k-1})$ où $T_{max}(N_i) = max\{t(n_i) : n_i \in N_i\}$ et $T_{min}(N_i) = min\{t(n_i) : n_i \in N_i\}$.

La complexité de calcul de cet algorithme $O(N)$ est linéaire avec le nombre de transitions dans le graphe de mots.

Lors de la génération d'un réseau de confusion, les transitions qui sont insérées dans les différents groupes peuvent être très longues en comparaison des transitions déjà présentes dans le groupe. Pour cette raison il n'est peut être pas raisonnable d'insérer

la transition dans ce groupe mais de faire en sorte qu'elle puisse être alignée avec deux groupes consécutifs comme dans la figure 5.3. Pour réaliser ceci, à l'étape 4.b, la transition pourra être insérée soit dans le groupe TS_{N_{n-1},N_n} soit dans TS_{N_{n-2},N_n}. Le calcul de la fonction SIM qui permet l'insertion de la transition reste inchangé.

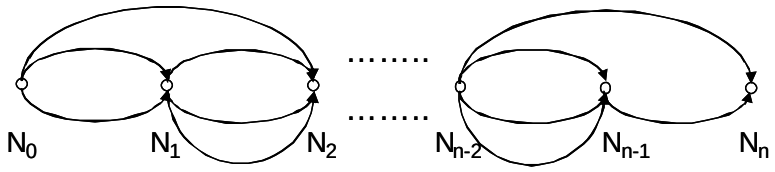

FIGURE 5.3 – Exemple du réseau de confusion modifié

Les tests effectués sur un corpus *Switchboard 2001 HUB-5* montrent des temps d'exécution très inférieurs à l'algorithme de regroupement par classes de transitions malgré un élagage systématique pour cet algorithme (seul 5% des transitions du graphe sont utilisées) et des graphes de mots ayant une taille raisonnable (2-10k transitions). En termes de *WER*, les deux algorithmes de génération de CNs obtiennent de meilleures performances que la meilleure solution du SRAP avec un très léger avantage pour l'algorithme de génération par regroupement en classes d'états.
Comme nous l'avons précisé, cet algorithme n'a pas fait l'objet de travaux dans cette thèse. Nous ne détaillons pas plus son fonctionnement mais nous avons souhaité le présenter néanmoins pour donner une vue complète sur l'état de l'art.

5.4 Discussions

Dans cette partie nous faisons tout d'abord une analyse critique de l'algorithme de regroupement par classes de transitions présenté dans la section 5.4.1. Nous présentons ensuite une comparaison entre les performances de cet algorithme et l'algorithme du "**pivot**" et nous motivons par la même occasion le choix de l'algorithme du "**pivot**" pour la génération des réseaux de confusion.

5.4.1 Algorithme de génération par regroupement des classes de transitions

L'algorithme de génération des réseaux de confusion proposé dans (Mangu et al., 2000) est un algorithme de type "*greedy*" (à chaque itération il prend la décision qui correspond à la meilleure du moment sans prendre en compte les conséquences sur la suite des décisions). Lorsqu'il regroupe deux classes, de nouvelles contraintes sont rajoutées à l'ordre partiel entre les classes restantes. En conséquence, certaines classes qui auraient pu être regroupées avant ne sont plus des candidats valides pour le processus de regroupement. Il est donc important d'utiliser des fonctions de similarité robustes qui soient capables de réaliser les regroupements dans le bon ordre et surtout de privilégier les hypothèses de mots ayant des probabilités *a posteriori* élevées.
L'analyse de cet algorithme porte sur deux aspects : d'un coté, le nombre de transitions

dans les graphes de mots qui influe directement sur la complexité de l'algorithme et qui nécessite l'introduction d'une étape d'élagage et de l'autre coté, l'aspect "*greedy*" de l'algorithme qui fait que les fonctions de similarité et la manière de regrouper les classes d'équivalence sont très importantes.

Élagage des transitions

Une grande majorité des graphes de mots contient des transitions ayant des probabilités *a posteriori* très petites. Ces transitions n'ont pas une influence très grande dans le calcul des probabilités *a posteriori* des hypothèses de mots dans le CN mais elles peuvent avoir une influence négative sur l'alignement et le regroupement des classes d'équivalence. Ceci peut arriver car l'algorithme de génération cherche à respecter à tout moment les relations d'ordre entre les transitions du graphe peu importe leur probabilité. Cette situation est illustrée dans la figure 5.4. Dans cet exemple qui montre un "morceau" d'un graphe de mots, les mots "*je*" et "*ne*" qui sont très proches du point de vue phonétique et ont un recouvrement temporel non nul devraient donc être regroupés ensemble dans la même classe d'équivalence. Du fait de la relation d'ordre, l'existence d'au moins un chemin dans le graphe contenant les deux hypothèses de mots (ce qui est le cas dans notre exemple), peu importe la valeur de la probabilité *a posteriori*, empêchera le regroupement.

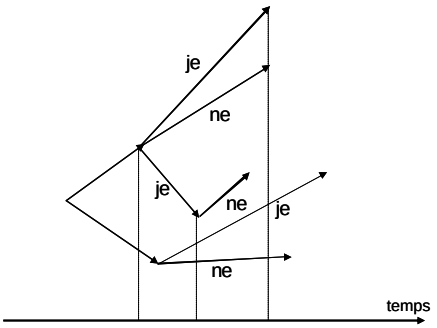

FIGURE 5.4 – Exemple d'un "morceau" d'un graphe de mots

Afin d'éviter ces situations, les auteurs ont introduit une étape d'élagage (*pruning*) des transitions des graphes de mots ayant une probabilité *a posteriori* en dessous d'un seuil établi de façon empirique. Ainsi, l'étape d'initialisation des classes d'équivalence et les deux étapes de regroupement ne traitent que les transitions ayant survécu à l'étape d'élagage.
Les testes menés dans (Mangu et al., 2000) sur un corpus de type *Switchboard* (le nombre de transitions du graphe par rapport au nombre de mots de la référence est de 1350 et le *WER* de la meilleure hypothèse MAP est de 38.5%) ont montré que l'utilisation de seulement 2% des transitions du graphe permet l'obtention d'une *consensus hypothesis* ayant le même *WER* que l'hypothèse du SRAP et que 5% des transitions permettent d'obtenir une amélioration du *WER* de 1.2%. Les tests ont aussi montré que pour une valeur de seuil d'élagage de 10% les performances en termes de *WER* restent constantes.

Toutefois aucune valeur du *WER* pour des CNs construits sans élagage des transitions n'est donnée. On ne sait donc pas si l'utilisation de toutes les transitions du graphe de mots influence de manière négative les performances des CNs.
Les tests montrent aussi que pour un seuil d'élagage de 5% (on utilise seulement 5% des transitions du graphe) le *WER* oracle sur CN [4] est équivalent à celui obtenu sur les graphes de mots [5] (10%). Pour des valeurs plus grandes du seuil le *WER* oracle est nettement amélioré (3.5% de réduction absolue pour un seuil d'élagage de 16%). L'utilisation d'une valeur du seuil d'élagage plus grande peut donc être justifiée si des algorithmes de post-traitement sont appliqués sur les CNs.
Un autre aspect très important lié au seuil d'élagage, et donc au nombre de transitions utilisées pour la génération des CNs, est le temps de calcul. Avec une complexité de calcul de $O(N^3)$ (N est le nombre de transitions dans le graphe), le temps de calcul peut vite devenir prohibitif pour une application temps réel dans laquelle les graphes de mots ont une taille importante. Il est important de pouvoir utiliser un seuil d'élagage qui réalise un bon compromis entre les performances des CNs en terme de *WER* et le temps de calcul nécessaire pour leur génération.
A la différence des graphes de mots utilisés dans (Mangu et al., 2000), qui ont un nombre moyen de transitions par graphe de 10000, les graphes de mots générés sur le corpus de test **Test_I**, que nous utilisons pour l'évaluation de l'algorithme, ont un nombre moyen de transitions par graphe de 26800. On retrouve un facteur 2.7 entre les deux valeurs. Les observations, faites dans (Mangu et al., 2000), en ce qui concerne l'élagage rendent d'autant plus nécessaire cette étape préliminaire d'élagage des transitions du graphe sur **Test_I**.

Fonctions de similarité

Un autre aspect important de cet algorithme est constitué par les fonctions de similarité 5.4 et 5.6 qui guident le regroupement des classes d'équivalence. Si pour l'étape "*intra-mots*" la fonction de similarité ne pose pas vraiment de problème, en revanche, pour l'étape "*inter-mots*", on peut se demander si la fonction de similarité phonétique joue ou non un role important dans le regroupement des classes d'équivalence. Ceci d'autant plus que les mots ont différentes variantes de prononciations qui ne peuvent pas être prises en compte dans le calcul de la distance d'édition entre deux transcriptions phonétiques. Les tests réalisés dans (Mangu et al., 2000) ont montré que le fait d'enlever cette similarité phonétique n'a aucune influence sur la valeur du *WER*. On peut conclure que la topologie du graphe de mots est une contrainte suffisante tant que les classes d'équivalence ayant une probabilité *a posteriori* grande sont regroupées en premier. La nouvelle fonction de coût devient :

$$SIM(C_1, C_2) = \underset{\substack{w_1 \in Mots(C_1) \\ w_2 \in Mots(C_2)}}{moy} P_C(w_1) \cdot P_C(w_2) \tag{5.9}$$

4. Le taux d'erreur oracle qui est défini pour les réseaux de confusion de la même manière que pour les graphes de mots.
5. Le taux d'erreur oracle est calculé avant élagage.

L'importance des probabilités *a posteriori* dans le regroupement des classes d'équivalence est aussi démontrée par un *WER* plus élevé lorsque les deux fonctions de similarité ne sont pas pondérées par ces probabilités. Dans ce cas, c'est l'augmentation de nombre d'omissions qui influe le plus sur le *WER*.
Comme nous l'avons expliqué, le bon ordre de regroupement des classes, particulièrement dans l'étape "*inter-mots*" de l'algorithme, est très important car tout regroupement crée de nouvelles relations d'ordre entre les classes restantes. Il est donc important de privilégier le regroupement des classes ayant la meilleure probabilité *a posteriori* mais aussi de tenir compte de leur alignement afin d'éviter de regrouper des classes correspondant à des intervalles de temps très écartés dans le temps. Le but étant de mettre en concurrence des hypothèses de mots dans un même intervalle temporel, il est logique de privilégier le regroupement des hypothèses de mots ayant le meilleur recouvrement temporel. Pour cela, nous proposons de modifier la fonction de similarité de l'étape "*inter-mots*" afin d'introduire la valeur du recouvrement temporel. La fonction devient :

$$SIM(C_1, C_2) = \underset{\substack{w_1 \in Mots(C_1) \\ w_2 \in Mots(C_2)}}{moy} overlap(ST(w_1), ST(w_2)) \cdot P_C(w_1) \cdot P_C(w_2) \tag{5.10}$$

où $overlap(ST(w_1), ST(w_2))$ est le recouvrement temporel entre les supports temporels des mots w_1 et w_2. Ces mots sont issus de l'étape "*intra-mots*" et chaque mot correspond à une classe d'équivalence. Comme nous l'avons expliqué, après cette étape on peut calculer la probabilité *a posteriori* des mots en sommant les probabilités *a posteriori* des transitions qui forment chaque classe d'équivalence. Afin d'utiliser la nouvelle formule de calcul de la fonction de similarité 5.10 pour l'étape "*inter-mots*" nous devons définir un support temporel $ST(w)$ pour chaque mot issu de la première étape. Les instants de début et de fin qui caractérisent le support temporel sont calculés comme étant la moyenne des instants de début et de fin des transitions du graphe qui composent la classe d'équivalence correspondant au mot w. Nous avons aussi essayé une autre façon de définir le support temporel en le rendant équivalent à celui de la transition ayant la meilleure probabilité *a posteriori* de la classe d'équivalence.
Les expériences [6] réalisées ont montré que l'utilisation de la fonction de similarité proposée dans l'équation 5.10 n'améliore pas les performances en terme de *WER*, qui reste constant, par rapport à l'implémentation de l'algorithme utilisant la fonction de similarité donnée par l'équation 5.9. Les deux définitions du support temporel $ST(w)$ produisent les mêmes performances.

5.4.2 Algorithme du "pivot"

L'algorithme du "**pivot**" a une complexité de $O(N \times k)$, où N est le nombre de transitions dans le graphe de mots et k est le nombre de classes finales dans le CN, qui en règle générale est beaucoup plus petit que N. Par exemple, si le meilleur chemin est

6. Les tests ont été réalisés sur un corpus de parole continue de 1600 énoncés contenant 6400 mots. Le nombre moyen de transitions par graphe est de 5000.

utilisé comme *pivot*, k est égal à la longueur du meilleur chemin plus le nombre d'états inséré dans l'alignement. Du point du vue de la complexité de calcul, cet algorithme est plus rapide que l'algorithme proposé dans (Mangu et al., 2000) qui a une complexité de $O(N^3)$.
Dans (Hakkani-Tur et al., 2006) on retrouve une comparaison entre les performances de l'algorithme du "**pivot**" et l'algorithme proposé dans (Mangu et al., 2000). Le corpus utilisé a été collecté à partir d'un service téléphonique de dialogue homme-machine *How may I help you ?* de AT&T (Gorin et al., 1997) et la comparaison des performances est réalisée sur une tache de classification binaire dans laquelle le système essaie de rejeter les mots mal reconnus en fonction d'un seuil sur le score de confiance des mots. Le score de confiance utilisé dans ce cas est la probabilité *a posteriori* des mots. Les performances des CNs sont supérieures par rapport aux performances du meilleur chemin du graphe. En revanche, les courbes des fausses alarmes par rapport aux faux rejets obtenues avec l'algorithme du "**pivot**" et avec l'algorithme de regroupement par classes de transitions sont identiques, mais les temps du calcul de l'algorithme du "**pivot**" sont nettement inférieurs (sans avoir eu recours à un élagage a priori des transitions du graphe de mots).

5.4.3 Choix de l'algorithme de génération des CNs

Dans cette partie nous faisons une comparaison entre les performances de la meilleure solution ASR et la meilleure solution des CNs générés à la fois par l'algorithme de regroupement par classes de transitions et par l'algorithme du "**pivot**".
Les tests ont été effectués sur un sous-corpus du corpus de test **Test_I**. Ce sous-corpus est composé de 1999 enregistrements pour un total de 5519 mots. Nous n'avons pas pu réaliser les tests sur l'intégralité du corpus de test **Test_I** pour des raisons liées à l'implémentation de l'algorithme de regroupement de transitions et à son temps d'exécution. Le tableau 5.1 montre les résultats obtenus sur ce sous-corpus en termes de *WER*. Le nombre de mots corrects ("C") et le détail des erreurs ("I" - insertions, "S" - substitutions, "O" - omissions) sont donnés dans ce tableau .

	WER	C	I	S	O
1-best ASR	44.4%	66.9%	11.3%	24.8%	8.3%
CN par regroupement des classes (seuil=5%)	44.2%	66.6%	10.8%	23.3%	10.1%
CN par regroupement des classes (seuil=30%)	43.0%	67.1%	10.1%	22.3%	10.6%
Algorithme du "**pivot**"	44.0%	66.5%	10.5%	23.8%	9.7%

TABLE 5.1 – *Comparaison des performances des algorithmes de génération des CNs*

La ligne "1-best ASR" donne les performances de la meilleure solution ASR. Les deux lignes suivantes correspondent à l'algorithme de regroupement par classes de transitions [7] avec deux valeurs différentes pour le seuil d'élagage des transitions. La dernière ligne de ce tableau correspond à l'algorithme du "**pivot**".

7. Nous avons utilisé la fonction de similarité décrite dans l'équation 5.9 pour l'étape "*inter-mots*"

Comme le montre le tableau 5.1, avec l'algorithme de génération des CNs par regroupement des classes et un seuil de 5% on obtient une légère amélioration du *WER* par rapport à la *1-best*. L'algorithme du "**pivot**" engendre une légère amélioration du *WER* par rapport à la meilleure solution ASR et par rapport à l'algorithme de base de génération par regroupement des classes de transitions. Une différence importante entre les deux algorithmes réside dans le nombre de transitions utilisées pour générer les CNs et dans les temps d'exécution. En effet, dans le cas de l'algorithme de regroupement par classes de transitions on a besoin de réaliser un élagage de transitions avant, en raison de la complexité de l'algorithme. Ceci n'est pas le cas pour l'algorithme du "**pivot**" qui utilise toutes les transitions du graphe. De plus, le temps d'exécution de l'algorithme du "**pivot**" est nettement inférieur.
Pour l'algorithme de regroupement par classes de transitions, l'utilisation d'un seuil d'élagage plus élevé engendre une amélioration du *WER* de 1.2% en absolu par rapport à l'utilisation d'un seuil de 5%. On observe une amélioration du *WER* en utilisant une valeur du seuil plus élevée, contrairement aux observation faites dans (Mangu et al., 2000). Cette amélioration dépasse aussi les performances de l'algorithme du "**pivot**". Toutefois, l'augmentation du seuil engendre une augmentation significative du temps d'exécution de l'algorithme.
Dans une application temps réel, tel que le service 3000, il n'est pas envisageable d'utiliser un algorithme de génération des CNs avec un temps d'exécution très long. Les performances en temps de calcul de l'algorithme de regroupement par classes de transitions le rend inadapté à une utilisation temps réel . En revanche, l'algorithme du "**pivot**" obtient des performances en terme de *WER* comparable et avec un temps d'exécution qui est nettement inférieur. Pour ces raisons nous avons décidé d'utiliser l'algorithme du "**pivot**" comme algorithme pour la génération des CNs dans nos travaux.

5.5 Conclusions

Dans ce chapitre nous avons présenté une approche de minimisation du *WER* basée sur les graphes de mots. Cette approche, proposée dans (Mangu et al., 2000) consiste en la construction d'un seul alignement pour toutes les hypothèses du graphe de mots. Nous avons détaillé les principes de construction de cet alignement, appelé réseau de confusion. Nous avons également présenté trois algorithmes de génération des réseaux de confusion. Une comparaison des performances de deux algorithmes implémentés est également présentée, et nous motivons aussi le choix de l'utilisation de l'algorithme du "**pivot**" dans la suite de nos travaux.