Optimisation de la construction des réseaux de confusion

Sommaire

Comme nous l'avons décrit dans le chapitre 3, dans un système de dialogue en langage naturel, le processus de reconnaissance vocale ainsi que l'analyse en concepts de l'hypothèse de reconnaissance sont transparents pour l'utilisateur. Les éventuelles erreurs à ces niveaux ne sont pas perçues directement par celui-ci. L'interprétation est ce qui guide la réponse du gestionnaire de dialogue et ce sont les erreurs au niveau interprétation qui reflètent le mieux les performances du système perçues par l'utilisateur.
Dans sa formulation théorique, l'algorithme de génération des CNs correspond à un critère de minimisation du taux d'erreur mot, alors que dans une application de dialogue on cherche à minimiser l'erreur au niveau interprétation. Nous proposons d'adapter l'algorithme du "**pivot**" dans le but de construire des CNs qui optimisent plus généralement les performances du processus complet d'interprétation, et ceci afin d'améliorer les performances du système perçues par l'utilisateur. Ces modifications introduisent un traitement différencié des mots du graphe et privilégient le traitement des mots porteurs de sens pour l'interprétation, favorisant ainsi une minimisation du taux d'erreur interprétation.
Le chapitre est organisé en six sections. Une analyse du comportement de l'algorithme

du "**pivot**" ainsi que des résultats obtenus sur le corpus **Test_II** sont présenté au 6.1. Dans la section 6.2 nous proposons une nouvelle approche heuristique pour le calcul des relations d'ordre entre les transitions. Les modifications apportées à l'algorithme du "**pivot**" sont présentées au 6.3. Dans la section 6.4 nous présentons l'influence des mesures de confiance sur les performances des CNs. Une étape d'élagage des classes finales des CNs est décrite au 6.5. Dans la dernière section 6.6 nous présentons une étape de post-traitement des CNs qui consiste en un algorithme de *parsing* basé sur les règles d'allumage des concepts.

6.1 Analyse de l'algorithme du "pivot"

Afin de pouvoir proposer des modifications de l'algorithme du "**pivot**" nous devons tout d'abord analyser le comportement de l'algorithme et la structure des CNs sur un corpus de données réelles. Pour réaliser cette analyse et pour présenter les performances des différentes modifications que nous détaillerons dans les sections suivantes, nous utilisons le corpus de test **Test_II**, composé de 6501 énoncés réels collectés à partir de l'application de dialogue en langage naturel 3000 (voir 4.3 pour une description plus détaillée). La **Methode 3** [1] est utilisée pour la normalisation lors de l'évaluation du *WER*, du *WER* oracle ou des métriques comme la précision et le rappel.

	WER Oracle	C	I	S	O
Graphes de mots	18.9%	84.8%	3.6%	10.6%	4.6%
CNs	6.2%	93.8%	0.1%	0.8%	5.4%

TABLE *6.1* – *Performances de l'algorithme du "**pivot**" en termes de* WER *oracle*

Le tableau 6.1 détaille les performances en terme de *WER* oracle de l'algorithme du "**pivot**". Les quatre dernières colones donnent le détail des erreurs pour les deux métriques (C - Mots corrects, I - Insertions, S - Substitutions, O - Omissions). On observe une nette amélioration des performances avec les CNs qui donnent un *WER* oracle trois fois inférieur à celui obtenu sur les graphes de mots. Ces performances sont dues au fait que la structure des CNs permet la création de nouveaux chemins qui n'existent pas dans le graphe. Le nombre très réduit d'insertions et de substitutions pour la solution oracle des CNs est une conséquence de la structure des réseaux de confusion. Lorsqu'on recherche la séquence la plus proche de la référence, les transitions portant l'omission, présentes dans un grand nombre de classes, permettent la construction d'un chemin entre deux mots de la solution oracle se trouvant dans deux classes non-adjacentes sans passer par d'autres mots. En effet, à la différence d'un graphe où un chemin entre deux mots non-adjacents passe obligatoirement par d'autres mots, dans un CN ce même chemin peut ne pas passer par d'autres mots si les classes parcourues contiennent toutes une transition portant l'omission. Le tableau 6.2 donne le détail du *WER* pour la *1-best*

1. Cette méthode est définie dans la section 4.4.1. Les énoncés non-parole ainsi que les énoncés ne contenant que des *OOV* et *SPR* sont considérés comme des énoncés à rejeter et sont annotés comme tel dans la référence et dans l'hypothèse de reconnaissance.

		WER [2]	C	I	S	O
Énonces non-parole	1-best	7.0%	6.5%	4.0%	3.1%	0.0%
1333 én.	*consensus hypothesis*	11.1%	3.7%	5.3%	5.8%	0.0%
Énonces parole	1-best	35.0%	67.4%	11.9%	17.5%	5.5%
5168 én.	*consensus hypothesis*	37.9%	65.6%	13.0%	18.6%	6.3%
Total	1-best	42.0%	73.9%	15.9%	20.6%	5.5%
6501 én.	*consensus hypothesis*	49.0%	69.4%	18.3%	24.4%	6.3%

TABLE 6.2 – WER *de la* 1-best *et de la* consensus hypothesis *sur les énoncés non-parole et les énoncés parole*

de la première passe de reconnaissance et la *consensus hypothesis*. Nous avons divisé le corpus en deux catégories : les énoncés non-parole (ne contenant que du bruit) et les énonces parole. Le *WER* sur chaque catégorie est calculé comme une contribution sur le *WER* de l'ensemble du corpus. Ainsi, le *WER* total de 42% = *WER* énoncés non-parole 7% + *WER* énoncés parole 35%.
On observe que, sur l'ensemble du corpus de test, les performances de la *consensus hypothesis* sont inférieures par rapport à la *1-best* avec une augmentation sur les trois types d'erreurs. On retrouve la même tendance sur chacune des deux catégories.
Les énoncés non-parole représentent 20% du total du corpus. 16% des erreurs sur la *1-best* sont produites par les énoncés de cette catégorie, alors que ce pourcentage est de 22% pour la *consensus hypothesis*. En effet, 68% des énoncés non-parole sont correctement détectés par la *1-best* alors que ce chiffre est de seulement 40% pour la *consensus hypothesis*. Ceci explique l'augmentation du nombre d'insertions et de substitutions sur cette catégorie pour la *consensus hypothesis*.

Distribution du même mot sur plusieurs classes adjacentes. Parcours topologique du graphe

Les systèmes de reconnaissance vocale rencontrent plus de difficultés à reconnaître les mots courts par rapport aux mots longs. Dans un graphe de mots, le système de reconnaissance vocale a tendance à générer un nombre élevé d'hypothèses pour le même mot mais avec des longueurs, instants de début et scores acoustiques différents. Il arrive fréquemment que, dans un CN, des hypothèses de mots différents ayant des longueurs différentes soient groupées dans la même classe. Des mots courts deviennent de surcroît des alternatives possibles pour des mots beaucoup plus longs. De ce fait, il arrive fréquemment que des hypothèses de mot représentant une meilleure alternative pour les mots longs de la classe ne puissent plus être insérées dans cette classe. Ceci se produit lorsque les mots courts, insérés lors des itérations antérieures, précèdent ces hypothèses de mots sur un chemin dans le graphe (on a une relation d'ordre entre l'hypothèse et le mot court de la classe).
La figure 6.1 montre un exemple réel, extrait d'un CN généré pour un enregistrement

2. Le *WER* calculé pour chaque catégorie est une contribution du *WER* total. On divise le nombre d'erreurs de chaque catégorie par le nombre total de mots de la référence sur l'ensemble du corpus.Pour les énoncés non-parole la référence est constituée d'une étiquette représantant un rejet.

du corpus de test. La référence de l'enregistrement contient le mot "**information**" qui est omis dans la *consensus hypothesis*. On observe que le mot "**information**" est présent dans deux classes adjacentes. Ceci est dû au fait que, dans le graphe, il existe des chemins qui contiennent la séquence de mots "**d' information**". Comme le mot "**d'**" est déjà inséré dans la première classe (du au parcours topologique du graphe), les transitions portant le mot "**information**" et qui sont précédées par une transition portant le mot "**d'**", insérée auparavant dans la classe, ne peuvent pas être insérées dans la même classe en raison d'une relation d'ordre entre les transitions.

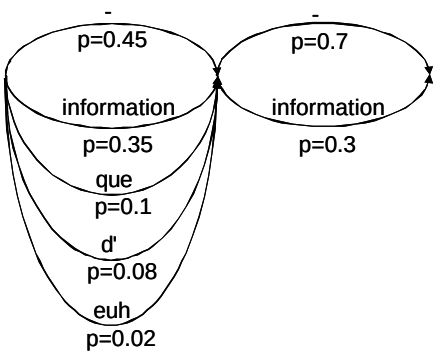

FIGURE *6.1* – Exemple de création forcée d'une classe ("-" représente l'omission)

La situation présentée dans la figure 6.1 est aussi due au fait que les transitions portant des mots courts dans le graphe ont des instants de début très différents. Une partie des transitions portant le mot "**information**", qui ont été insérées dans la première classe, sont aussi précédées des transitions portant le mot "**d'**". Mais en raison des instants de début très différents des transitions portant ce mot, certaines transitions sont regroupées dans la même classe avec le mot "**information**".
La conséquence de cette distribution d'une hypothèse de mot sur plusieurs classes adjacentes est l'omission du mot dans la *consensus hypothesis*. En effet, la masse de probabilité *a posteriori* est répartie sur deux classes au lieu d'être cumulée et dans chacune des deux classes la probabilité de "**information**" n'est pas assez élevée pour devancer celle de l'omission. Une analyse détaillée de ce phénomène nous a permis d'en déterminer les raisons. La figure 6.1 est très représentative dans ce sens, car elle montre clairement que si les transitions portant le mot "**information**" étaient traitées en premier elles auraient été insérées dans la "bonne" classe et aucune omission ne ce serait produite. Ce qui empêche cette insertion est l'ordre de traitement des transitions du graphe qui est donné par le parcours topologique du graphe de mots. On retrouve même des situations où des mots longs du *pivot* sont distribués sur plusieurs classes adjacentes à cause du traitement des transitions dans un ordre topologique.
Une conséquence de ce phénomène est le nombre inhabituellement élevé de classes par mot de la référence dans le réseau de confusion final. Le tableau 6.3 donne la largeur moyenne des CNs par rapport à la référence exprimée par le nombre moyen de classes par mot de la référence et la profondeur moyenne des CNs exprimée en nombre moyen de mots par classe. On observe que, pour un CN, l'algorithme génère en moyenne 34.8 classes par mot de la référence. On obtient ainsi des CNs avec une largeur moyenne de

	# Transitions / Graphe	# Transitions / Mot de la référence
Graphes de mots	16800	12000
	# Classes / Mot de la référence	# Mots / Classe
CNs	34.8	5.9

TABLE 6.3 – *Taille des CNs par rapport à la taille des graphes de mots*

73.1 classes (la longueur moyenne de la référence est de 2.1 mots). Au total cella représente une réduction de 98% du nombre de transitions par rapport aux graphes de mots initiaux. Cependant, la largeur reste importante et, en théorie, on pourrait considérer que pour chaque mot de la référence on a potentiellement 33.8 insertions de mots. Ce n'est pas le cas en pratique mais ce chiffre reste néanmoins élevé.
Une première modification de l'algorithme de génération des CNs que nous proposons est de ne plus parcourir le graphe dans un ordre topologique mais de privilégier d'abord certains ensembles de transitions suivant des critères que nous définissons par la suite. Ainsi, nous proposons de renforcer les mots du *pivot* afin d'éviter leur distribution sur plusieurs classes adjacentes. Les transitions portant des mots du *pivot* forment un groupe qui est traité en premier. A l'intérieur de ce groupe, les transitions sont traitées dans un ordre topologique. Dans la suite de la présentation nous appelons *pivot topologique*, l'algorithme initial du "**pivot**" tel qu'il est présenté au 5.3.2.

Prise en compte des connaissances sémantiques dans la construction des CNs

Un des objectifs principaux de ces travaux est de réaliser des CNs qui minimisent non seulement le *WER* mais, plus généralement, optimisent les performances du processus complet d'interprétation, et ceci afin d'améliorer les performances du système perçues par l'utilisateur. Pour ce faire nous devons tenir compte, dans le processus de génération des CNs, des spécificités du module de compréhension de la parole, tel que l'utilisation des concepts qui attribuent une plus grande importance aux mots non-vides (voir 3.3).

	IER	Correct	FR	Substitutions	FA
1-best ASR	22.7%	92.7%	1.6%	5.7%	15.4%
Consensus Hypothesis	21.8%	89.9%	4.1%	6.0%	11.7%

TABLE 6.4 – *Performances de l'algorithme du "**pivot**" en terme d'*IER.

Le tableau 6.4 présente les performances de la *1-best* ASR et de la *consensus hypothesis* en terme d'*IER*. Un détail des erreurs est aussi présenté (Substitutions, FR - Faux Rejets, FA - Fausses Alarmes). Contrairement aux performances en terme de *WER*, la *consensus hypothesis* améliore légèrement l'*IER*. Le point de fonctionnement en revanche est modifié.
Le tableau 6.5 présente cinq exemples d'énoncés, extraits du corpus de test, qui sont très représentatifs pour expliquer les différences entre les taux de FR et FA de la *1-best* et de la *consensus hypothesis*. Les trois premiers exemples correspondent à des FA et les deux suivants à des FR. Les deux premiers exemples correspondent à une FA présente dans

la *1-best* due au fait que l'hypothèse de reconnaissance est un mot non-vide qui produit une interprétation. La *consensus hypothesis* ne produit aucune interprétation. Le SRAP a plus tendance à produire une hypothèse de reconnaissance qui produit une interprétation pour ces types d'énoncés. Pour le troisième exemple, il s'agit d'une FA présente dans la *consensus hypothesis* sur un énoncé non-parole. En effet, une grande partie des erreurs de FA sur la *consensus hypothesis* provient de ce type d'énoncé. Pour la *1-best*, une partie des FA provient également des énoncés non-parole, mais une autre partie, assez importante, provient des énoncés comme celui présenté dans le deuxième exemple, ce qui n'est pas le cas pour la *consensus hypothesis*. L'accumulation des deux types d'énoncés font que le taux de FA est plus élevé pour la *1-best*. Le quatrième exemple est représentatif de la majorité des FR produits par la *consensus hypothesis*. En effet, sur cet exemple nous avons observé que le mot "**facture**" (qui est un mot non-vide) était distribué sur plusieurs classes adjacentes ce qui à conduit à son omission dans la *consensus hypothesis*. Le cinquième cas est moins fréquent et au total la *consensus hypothesis* génère plus de FR que la *1-best*.

	Mots		Interprétation	
	Référence	Hypothèse	Référence	Hypothèse
1-best	*bruit*	le l' aide	Rejet	Aide
consensus hypothesis		je		Rejet
1-best	*SPR*	annule	Rejet	Annuler
consensus hypothesis		euh ne s'		Rejet
1-best	*bruit*	<*REJET*>	Rejet	Rejet
consensus hypothesis		payer		PayerFacture
1-best	facture	facture	AmbiFacture	AmbiFacture
consensus hypothesis		*vide*		Rejet
1-best	paiement	je demande	PayerFacture	Rejet
consensus hypothesis		paiement		PayerFacture

TABLE 6.5 – *Exemples d'erreurs au niveau interprétation*

L'algorithme *pivot topologique* accorde une importance égale à tous les mots dans le graphe de mots. Ce n'est pas le cas du module d'interprétation pour lequel les mots non-vides ont une importance plus grande car ce sont eux qui allument les concepts qui, à leur tour, allument une interprétation. En plus de renforcer les mots du *pivot* en traitant en premier les transitions du graphe de mots contenant ces mots, nous proposons une deuxième modification importante de l'algorithme de génération des CN qui donne plus de poids aux mots non-vides par rapport aux mots vides lors du traitement des transitions restantes du graphe de mots. Pour ce faire, les transitions portant des mots non-vides seront traitées avant celle portant des mots vides. Les modifications proposées sont détaillées dans la section 6.3.

6.2 Nouvelle approche heuristique de la relation d'ordre entre transitions

La relation d'ordre joue un rôle très important dans la génération des réseaux de confusion. Elle doit être calculée à chaque fois qu'une classe d'équivalence, constituée d'une seule transition, est insérée dans le réseau (entre les états $\hat{S}_s$ et $\hat{S}_f$). Le calcul se fait pour savoir si la transition à insérer est en relation avec une des transitions de la classe d'équivalence (délimitée par ces états) dans laquelle elle doit être insérée. Le calcul ne se fait donc pas seulement pour une paire de transitions mais pour toutes les paires formées de la transition à insérer et d'une des transitions de la classe d'équivalence. En pratique on s'arrête dès qu'une paire de transitions se trouve en relation car cela signifie que toute la classe d'équivalence est en relation avec la classe d'équivalence correspondant à la transition à insérer. Compte tenu de la taille des graphes de mots du corpus de test **Test_II**, le nombre d'insertions de transitions est très grand ce qui entraîne un grand nombre de calculs de relations d'ordre.
Compte tenu de la définition de la relation d'ordre entre deux transitions e_1 et e_2 (donnée dans la section 5.2.3), les deux premières conditions, $e_1 = e_2$ et $Fstate(e_1) = Sstate(e_2)$ sont faciles et rapides à vérifier. La dernière condition doit déterminer s'il existe un chemin dans le graphe entre les deux transitions. Pour ce faire, elle doit soit vérifier l'existence d'un chemin dans le graphe à chaque fois, soit avoir une base de données qui répertorie l'existence d'un chemin entre toutes les paires de transitions du graphe de mots. Les trois conditions sont calculées successivement et on s'arrête dès qu'une condition est remplie. Si le nombre de fois où la troisième condition doit être testée est élevé, vérifier l'existence du chemin dans le graphe à chaque fois est coûteux en temps de calcul alors que la deuxième méthode peut s'avérer gourmande en termes de mémoire de stockage. Dans notre implémentation, c'est cette seconde solution qui a été retenue. Malgré une implémentation optimisée le parcours de la structure est coûteux en temps de calcul.
Nous avons décidé de compter le nombre de fois que chaque condition est utilisée lorsqu'une relation d'ordre entre deux transitions est déterminée. On observe que dans 84% des cas la relation d'ordre entre deux transitions est établie par une des deux premières conditions. Dans seulement 16% des cas une relation d'ordre est établie par la troisième condition. Ces pourcentages sont calculés par rapport au nombre de transitions pour lesquelles on a établi une relation d'ordre lors de leur insertion dans le CN. Pour les autres transitions du graphe, l'algorithme vérifie à chaque fois les trois conditions afin de conclure qu'aucune relation d'ordre avec les transitions des classes d'équivalences n'existe. Le nombre de transitions pour lesquelles une relation d'ordre est établie lors de leur insertion représente moins de 0.1% du nombre total de transitions dans un graphe. Le volume de calculs engendré par la vérification de la troisième condition est très important par rapport au nombre de fois ou celle-ci établi une relation d'ordre. Nous proposons une approche heuristique pour la mise en œuvre de la relation d'ordre entre deux transitions. 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'une transition)

Nous n'examinons plus la troisième condition qui ne concernait que 16% des cas. Ceci engendre un gain de temps de calcul car on ne recherche plus un chemin entre deux transitions à chaque fois qu'une transition est insérée dans le CN. Du point de vue théorique, cette approche ne définit pas une relation d'ordre car elle ne vérifie plus la propriété de transitivité. Mais pour plus de simplicité nous continuons à l'appeler relation d'ordre dans la suite de ce document.
Cette nouvelle approche de la relation d'ordre soulève quelques questions. Elle permet désormais qu'une transition soit insérée dans une classe alors qu'il existe un chemin entre cette transition et une transition de la classe, les deux transitions n'étant pas adjacentes sur ce chemin. Le problème dans ce cas concerne la somme des probabilités *a posteriori* des mots de la classe qui doit être plus petite ou égale à 1. En effet, regrouper dans une classe deux transitions qui se trouvent sur un même chemin dans le graphe peut faire que la somme des probabilités *a posteriori* des mots de la classe dépasse 1.
Nous avons réalisé des tests sur le corpus de **Test_I** afin de mesurer la fréquence et l'importance de cet événement. Les CNs ont été générés avec l'algorithme du *pivot topologique* qui, pour le calcul de la relation d'ordre, utilise d'abord la définition décrite au 5.2.3, et ensuite l'approche que nous proposons. Nous avons observé, qu'en moyenne, la somme des probabilités *a posteriori* des transitions est plus grande que 1 pour moins de 0.01% des classes par enregistrement (cette somme est toujours plus petite que 1.1).

	# Classes / Mot de la référence	# Mots / Classe
définition de base	24.4	8.7
nouvelle approche heuristique	17.9	9.8

TABLE 6.6 – *Taille des CNs générés sur corpus **Test_I** avec la définition de base de la relation d'ordre et avec l'approche heuristique.*

Les tests effectués sur le corpus **Test_I** ont montré que les performances des CNs obtenus avec les deux définitions sont identiques en termes de *WER* et d'*IER*. Le *WER* oracle reste aussi constant. En ce qui concerne la taille des CNs, on observe dans le tableau 6.6 une diminution relative de 26% de la largeur des CNs due à l'utilisation de la nouvelle approche. L'utilisation de l'approche proposée permet également l'obtention d'un temps d'exécution d'environ 3 fois inférieur à celui obtenu avec la définition de base.
L'approche proposée pour le calcul de la relation d'ordre est utilisée dans les variantes de l'algorithme du "**pivot**" que nous proposons. Dans la suite du document, seul l'algorithme du *pivot topologique* utilise la définition de la relation d'ordre tel qu'elle est décrite au 5.2.3.

6.3 Regroupement des transitions guidé par le contexte applicatif

Comme nous l'avons décrit dans la section 6.1, suite à l'analyse approfondie du comportement de l'algorithme du *pivot topologique* sur un corpus de données réelles, nous proposons d'apporter des modifications (Minescu et Damnati, 2007) à l'algorithme

du *pivot topologique* dans le but de construire des CNs qui optimisent plus généralement les performances du processus complet d'interprétation.

6.3.1 Algorithme de génération multi-niveaux

Le principe de ce nouvel algorithme est d'utiliser des groupes de mots et d'insérer dans le CN les transitions correspondant à ces groupes en plusieurs étapes successives selon l'importance donnée aux groupes. Dans chaque étape les transitions correspondants à un groupe sont insérées dans un ordre topologique, mais sur l'ensemble du processus de génération ce n'est plus le cas. Il s'agit donc d'un algorithme multi-niveaux qui tient compte des mots portés par les transitions pour établir l'ordre d'insertion dans le CN lors de sa génération. L'objectif de ce nouvel algorithme est de rendre plus fiable la construction dans le CN des hypothèses portant sur des mots significatifs pour l'application (les mots non-vides).
L'insertion des transitions dans le nouvel algorithme se fait en quatre étapes. Comme nous l'avons expliqué, un traitement particulier est réservé d'abord aux mots du *pivot* qui sont traités en premier. Ensuite, on traite les mots non-vides ; leur insertion est guidée dans un premier temps par l'analyse en concepts de la séquence du *pivot*. Le calcul du *pivot* reste identique à l'algorithme *pivot topologique* et constitue le premier pas de l'algorithme suivi de la phase d'insertion des transitions. Nous décrivons ci-dessous les quatre étapes de la phase d'insertion des transitions dans le CN.

1. Pour toutes les transitions portant un mot du *pivot* :
   (1) *[Recherche de l'emplacement optimal]*
   (2) *[Insertion transition]*
2. Pour toutes les transitions portant un mot non-vide qui allume le même concept qu'un mot du *pivot* :
   (1) *[Recherche de l'emplacement optimal]*
   (2) *[Insertion transition]*
3. Pour toutes les autres transitions portant un mot non-vide :
   (1) *[Recherche de l'emplacement optimal]*
   (2) *[Insertion transition]* ou *[Création nouvel état et insertion transition]*
4. Pour toutes les transitions restantes qui portent un mot vide :
   (1) *[Recherche de l'emplacement optimal]*
   (2) *[Insertion transition]* ou *[Création nouvel état et insertion transition]*

Les trois processus *[Recherche de l'emplacement optimal], [Insertion transition]* et *[Création nouvel état et insertion transition]* ont été définis dans la description de l'algorithme du *pivot topologique* (voir la section 5.3.2). Les deux derniers sont des processus qui définissent l'insertion d'une transition dans le CN et le choix entre les deux se fait en fonction du calcul de la relation d'ordre entre la transition à insérer et les transitions de la classe d'équivalence correspondant à l'*emplacement optimal*. Nous utilisons l'approche proposée dans la section 6.2 pour le calcul de la relation d'ordre.

Une différence importante entre les deux premières étapes et les deux dernières réside dans le processus d'insertion d'une transition. Ainsi, dans les deux premières étapes on utilise seulement le processus *[Insertion transition]* qui implique que la transition à insérer ne doit pas être en relation d'ordre avec les transitions de la classe d'équivalence correspondant à l'*emplacement optimal*. Si toutefois cette relation d'ordre existe, la transition à insérer est traitée par une des deux dernières étapes (en fonction du mot porté). En procédant ainsi l'algorithme ne créé pas de nouvel état [3] dans le CN pendant les deux premières étapes. Le but est de générer des CNs plus compacts tout en évitant la dispersion des mots sur plusieurs classes adjacentes. Des précisions sur le fonctionnement de chaque étape sont données ci-dessous :

- **Etape 1.** Cette étape permet d'éviter la dispersion des hypothèses de mot du *pivot* sur plusieurs classes adjacentes, et donc de rendre plus représentative leur probabilité *a posteriori*.
- **Etape 2.** Un concept peut être allumé par plusieurs mots différents. Dans la majorités des cas, les mots ont une similarité phonétique [4] importante ce qui signifie que la probabilité de se retrouver en concurrence est assez forte. L'étape 2 permet ainsi de favoriser le regroupement des mots portant le même concept et de fiabiliser ainsi les hypothèses conceptuelles. Le cas des règles faisant correspondre une séquence de plusieurs mots à un concept n'est pas traité et les mots se trouvant dans ce cas sont traités à l'étape suivante.
- **Etape 3.** Cette étape permet l'insertion dans le CN des transitions portant des mots non-vides qui n'ont pas été traitées par les étapes précédentes. A la différence des deux premières étapes, la création d'un nouvel état lors de l'insertion des transitions est autorisée. Cette étape favorise le regroupement des mots non-vides et permet d'éviter une possible dispersion due aux mots vides.

La séparation des étapes 3 et 4 permet d'interrompre la construction des CNs dès l'étape 3. En effet, l'ajout des mots vides supplémentaires (seul ceux étant dans le *pivot* sont conservés) augmente la taille des réseaux alors que ces mots sont sans effet sur les traitements applicatifs en aval (l'allumage des concepts et l'obtention d'une interprétation). En effet, on observe une sur-génération des mots vides dans les graphes de mots (55% des occurrences) par rapport à seulement 35% dans le corpus de test.
Le fait d'arrêter l'algorithme à la fin de l'étape 3 donne la possibilité d'avoir deux variantes du même algorithme dont le choix réside dans les besoins de l'application choisie. Dans la suite de la présentation, nos appelons algorithme *multi-niveaux*, la variante qui comporte toutes les étapes et algorithme *multi-niveaux sans vides* la variante qui s'arrête après l'étape 3.

6.3.2 Performances du nouvel algorithme

Le tableau 6.7 montre les performances en terme de *WER* de la *consensus hypothesis* obtenue avec les algorithmes *multi-niveaux* et *multi-niveaux sans vides* et donne aussi le

3. La création d'un état est détaillé dans la section 5.3.2. Le processus *[Création nouvel état et insertion transition]* est le seul à pouvoir créer un nouvel état.

4. En général, lorsque plusieurs mots allument le même concept, ces mots sont les formes de singulier, pluriel, masculin, féminin du même mot

	WER	C	I	S	O
1-best ASR	42.0%	73.9%	15.9%	20.6%	5.5%
pivot topologique	49.0%	69.4%	18.3%	24.4%	6.3%
multi-niveaux	46.9%	70.3%	17.2%	23.1%	6.6%
multi-niveaux sans vides	43.1%	70.4%	13.5%	21.3%	8.3%

TABLE 6.7 – *Performances des variantes de l'algorithme* multi-niveaux *en termes de* WER.

détail des erreurs. Si le *WER* de la *1-best* reste légèrement meilleur que les performances des deux algorithmes, l'amélioration du celui-ci par rapport au *pivot topologique* est importante. La diminution de 2% du *WER* pour l'algorithme *multi-niveaux* s'explique par une meilleure gestion de l'insertion des transitions dans le CN qui se traduit par une baisse du nombre d'insertions et de substitutions pour un nombre d'omissions qui reste quasiment constant. En effet, comme montré dans le tableau 6.8, sur les mots vides, on obtient une augmentation de 3.7% de la précision par rapport à l'algorithme du *pivot topologique* ce qui se traduit par une réduction du nombre d'erreurs d'insertion et de substitution sur les mots vides. Dans le calcul des valeurs de précision et rappel dans le tableau 6.8, les énoncés non-parole ainsi que les énoncés vides sont assimilés à des mots vides. Pour les mots non-vides, l'augmentation de 1% du rappel pour l'algorithme *multi-niveaux* montre une amélioration de la reconnaissance de ces mots avec une augmentation du nombre de mots non-vides bien reconnus.

	Ensemble des mots		Mots non-vides		Mots vides	
	Précision	Rappel	Précision	Rappel	Précision	Rappel
1-best ASR	66.9%	73.9%	70.7%	86.7%	57.5%	50.7%
pivot topologique	61.9%	69.4%	68.4%	84.2%	46.2%	42.7%
multi-niveaux	63.6%	70.3%	68.9%	85.2%	49.9%	43.5%
multi-niveaux sans vides	67.0%	70.4%	69.6%	85.2%	59.3%	44.0%

TABLE 6.8 – *Performances des algorithmes en termes de précision et rappel*

En ne conservant que les mots non-vides présents dans le *pivot*, l'algorithme *multi-niveaux sans vides* permet d'obtenir une diminution de près de 4% du *WER* par rapport à l'algorithme *multi-niveaux* en évitant d'augmenter de façon importante le nombre d'insertions de mots vides. Pour les mots non-vides on observe une légère augmentation de la précision pour un rappel constant. Elle est due à une diminution du nombre d'insertions des mots non-vides. En effet, le fait de ne pas insérer dans le CN une grande partie des mots vides a pour résultat une augmentation de la probabilité *a posteriori* de l'omission dans certaines classes ce qui entraîne une diminution des insertions des mots non-vides provenant de ces classes. L'augmentation importante de la précision sur les mots vides est due à la diminution du nombre d'insertions sur ce type de mot. Cette diminution du nombre d'insertions a pour effet d'obtenir un certain nombre de *consensus hypothesis* vides. Comme pour ce calcul nous avons assimilé ces énoncés, ainsi que les énoncés non-parole, à des mots vides, l'obtention d'une *consensus hypothesis* vide pour des énoncés non-parole explique la légère augmentation du rappel sur les mots vides.
Le tableau 6.9 montre les performances en termes d'*IER* pour les deux algorithmes proposés ainsi que le détail des erreurs. Un des objectifs principaux de ces deux al-

	IER	C	FR	Sub	FA
1-best ASR	22.7%	92.7%	1.6%	5.7%	15.4%
pivot topologique	21.8%	89.9%	4.1%	6.0%	11.7%
multi-niveaux	20.2%	91.4%	3.4%	5.2%	11.6%
multi-niveaux sans vides	19.8%	91.5%	3.4%	5.1%	11.3%

TABLE *6.9* – *Performances des algorithmes en terme d'*IER

gorithmes était l'amélioration de la performance du système perçue par l'utilisateur qui passe par une amélioration du taux d'erreur interprétation. On observe ainsi une amélioration de 1.6% en absolu pour l'algorithme *multi-niveaux* par rapport au *pivot topologique*. Celle-ci est due principalement à une diminution du nombre de FR et de substitutions. Une légère amélioration de l'*IER* est obtenue avec l'algorithme *multi-niveaux sans vides* due principalement à une légère baisse du nombre de FA. Les performances équivalentes des deux algorithmes proposés s'expliquent, d'un côté, par le fait que le module de compréhension n'utilise pas les mots vides et de l'autre par le fait que le module de compréhension (plus précisément l'allumage des règles d'interprétation) n'est pas sensible aux insertions au niveau mots, qui contribuent principalement à la différence du *WER* entre les deux algorithmes. La diminution légère du nombre de FA est due à l'augmentation du nombre d'énoncés à rejeter pour lesquels la *consensus hypothesis* est vide ce qui produit également un rejet au niveau interprétation.

	# Classes / Mot de la référence	# Mots / Classe
pivot topologique	34.8	5.9
multi-niveaux	28.2	5.9
multi-niveaux sans vides	16.6	4.5

TABLE *6.10* – *Tailles des réseaux générés avec les différents algorithmes*

La réduction de la taille des CNs générés avec les algorithmes proposées est détaillée dans le tableau 6.10. On observe une réduction relative de 19% de la largeur des CNs générés avec l'algorithme *multi-niveaux* avec une profondeur qui reste constante. Ceci est dû principalement à l'ordre d'insertion des transitions dans le CN qui évite la création intempestive des classes. Le fait de ne pas insérer dans les CNs les transitions portant des mots vides (sauf ceux du *pivot*) permet d'obtenir des CNs encore plus compacts à l'aide de l'algorithme *multi-niveaux sans vides*. Ainsi, la réduction de la largeur des réseaux est de plus de 50% en valeur relative par rapport à l'algorithme *pivot topologique* et de 40% par rapport à l'algorithme *multi-niveaux*. La diminution de la profondeur des CNs de l'algorithme *multi-niveaux sans vides* est une conséquence logique de l'élimination des transitions portant un mot vide du processus de génération.
Le tableau 6.11 montre le détail des erreurs de l'oracle pour les algorithmes proposés. On observe, pour l'algorithme *multi-niveaux*, une légère dégradation des performances qui provient principalement d'une augmentation du nombre d'omissions. La légère augmentation du nombre de substitutions (de 0.8% à 1.4% pour l'algorithme *multi-niveaux* et 1.6% pour l'algorithme *multi-niveaux sans vides*) est due à la méthode d'évaluation du *WER* utilisée. En effet, si la *consensus hypothesis* ou la référence sont vides, la méthode **Methode 3** considère qu'il s'agit d'un rejet et les remplace par l'étiquette

	WER Oracle	C	I	S	O
Graphes de mots	18.9%	84.8%	3.6%	10.7%	4.6%
pivot topologique	6.2%	93.8%	0.1%	0.8%	5.3%
multi-niveaux	7.7%	92.5%	0.2%	1.4%	6.1%
multi-niveaux sans vides	14.4%	85.8%	0.2%	1.6%	12.6%

TABLE *6.11* – WER *oracle des algorithmes de génération des CNs*

<*REJET*>. Ainsi, l'augmentation des substitutions est due à une substitution d'un mot par cette étiquette ou inversement. En réalité, ces substitutions sont en fait des omissions ou des insertions. Dans le cas de l'algorithme *multi-niveaux sans vides* on observe une forte dégradation du *WER* oracle. Ceci est dû principalement à l'augmentation du nombre d'omissions qui est la conséquence directe de l'omission des transitions portant des mots vides (sauf ceux du *pivot*) dans la construction des réseaux.

Les mots vides sont filtrés	WER Oracle
Graphes de mots	14.4%
pivot topologique	2.8%
multi-niveaux	4.4%
multi-niveaux sans vides	4.0%

TABLE *6.12* – WER *oracle des algorithmes de génération des CNs en filtrant les mots vides*

Dans le tableau 6.12 nous avons réévalué le *WER* de l'oracle en ne gardant que les mots non-vides dans la référence et dans la solution oracle afin de faire une évaluation plus cohérente de l'algorithme *multi-niveaux sans vides*. Si la différence entre les *WER* oracle des algorithmes du *pivot topologique* et *multi-niveaux* est due principalement à une augmentation du nombre d'omissions, l'algorithme *multi-niveaux sans vides* améliore légèrement le *WER* oracle par rapport à l'algorithme *multi-niveaux*. Ceci montre bien que la forte dégradation du *WER* oracle observée dans le tableau 6.11 est due à l'absence de certain mots vides dans les CNs générés avec l'algorithme *multi-niveaux sans vides*.

6.4 Utilisation des mesures de confiance

Comme nous l'avons expliqué au 2.3, la probabilité *a posteriori* des mots dans les réseaux de confusion constitue une mesure de confiance fiable. Dans cette partie nous évaluons l'influence de cette mesure de confiance sur les performances en termes de *WER* et d'*IER* des CNs générés avec l'algorithme *pivot topologique* et avec les deux variantes de l'algorithme *multi-niveaux* proposées.
Nous utilisons ainsi la régression logistique (voir 2.2.4) pour évaluer la probabilité qu'un mot soit correct étant donné sa probabilité *a posteriori* ($P(COR|P(w|X))$). Un seuil est appliqué sur la probabilité ainsi évaluée des mots de la *consensus hypothesis*. Les mots ayant une probabilité d'être corrects étant donné leur probabilité *a posteriori* en dessous du seuil seront éliminés et nous calculons ensuite le *WER* et l'*IER* pour la nouvelle *consensus hypothesis*. Les paramètres de la régression logistique ont été évalués

sur le corpus de développement **Dev**. Nous utilisons la régression logistique afin de réaliser un calibrage des valeurs de la probabilité *a posteriori* des mots. Du fait d'une dynamique importante des valeurs de la probabilité *a posteriori* il est difficile d'établir une valeur absolue pour le seuil d'élagage. La régression logistique permet de réduire cette dynamique et une valeur pertinente du seuil d'élagage peut être établie.
La figure 6.2 montre les performances des algorithmes en termes de *WER*. Les courbes ont été obtenues en variant le seuil sur la mesure de confiance de 0 à 0.4. Chaque point sur la courbe correspond à une valeur du seuil en partant de la droite vers la gauche. Ainsi, le point le plus à droite sur chaque courbe correspond aux performances des algorithmes avec un seuil de 0, donc pas de filtrage de la *consensus hypothesis*. Le *WER* correspondant à chaque point est calculé en faisant la somme de l'abscisse, qui représente la somme des taux d'insertion et de substitution, et de l'ordonnée, qui représente le taux d'omission.

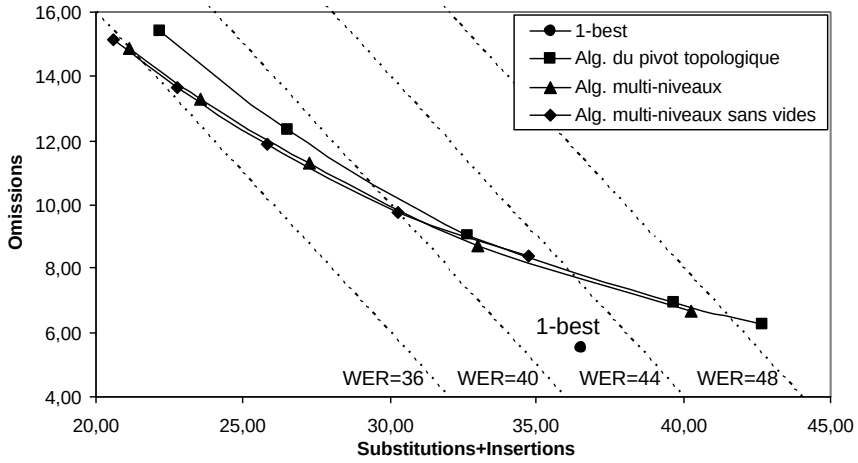

FIGURE 6.2 – *WER* de la *consensus hypothesis* en fonction du seuil sur la mesure de confiance

Comme on peut l'observer sur cette figure, plus on augmente la valeur du seuil plus le *WER* baisse. Ceci est vrai jusqu'à une valeur de 0.4 car pour des valeurs plus grandes le nombre d'omissions est trop important ce qui entraîne une augmentation du *WER*. Pour les trois algorithmes, l'utilisation de la probabilité *a posteriori* en tant que mesure de confiance permet d'améliorer le *WER* par rapport à la *1-best*. En ce qui concerne les performances des deux algorithmes proposés, on observe que leurs courbes sont quasiment superposées. L'utilisation de la mesure de confiance permet d'obtenir des performances équivalentes pour les deux algorithmes.
La figure 6.3 montre les performances en termes d'*IER* en fonction du seuil sur la mesure de confiance : la *consensus hypothesis* est filtrée en appliquant un seuil sur la probabilité $P(COR|P(w|X))$ et l'interprétation est calculée à partir de la séquence de mots filtrée.
L'utilisation des mesures de confiance permet une amélioration de l'*IER* avec un déplacement vers la gauche du point de fonctionnement du système. Contrairement aux performances en termes de *WER*, au delà d'une valeur du seuil de 0.2, l'*IER* augmente

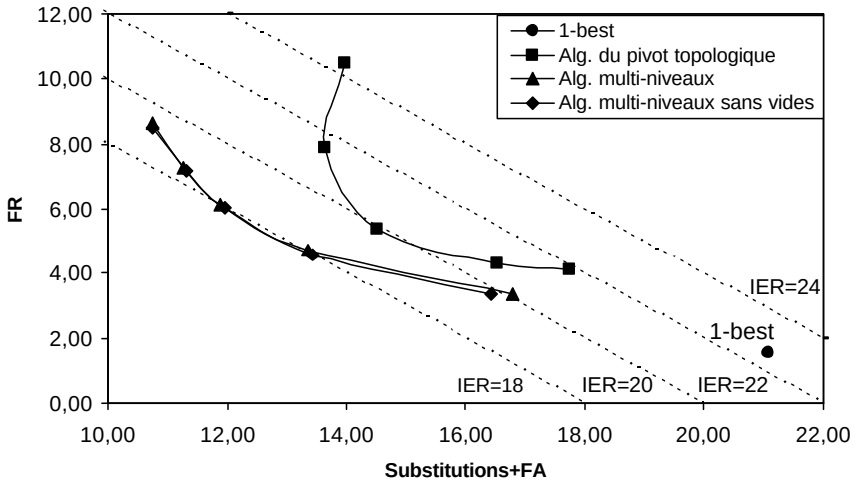

FIGURE *6.3* – Le point de fonctionnement en fonction du seuil sur la mesure de confiance

pour les trois algorithmes. Cette augmentation est très nette pour l'algorithme du *pivot topologique* par rapport aux algorithmes proposés. L'explication réside dans une augmentation plus forte du nombre d'omissions au niveau mot, en particulier pour les mots non-vides, pour l'algorithme du *pivot topologique*.
Afin d'intégrer l'utilisation des mesures de confiance dans le système de dialogue, le choix d'un point de fonctionnement est nécessaire. Ceci équivaut à choisir une valeur pour le seuil sur la mesure de confiance, est lié, d'un côté, à la valeur de l'*IER* et, de l'autre, à sa position sur la courbe. Dans tous les cas, une baisse du nombre de FA et des substitutions se fait au détriment du nombre de FR. Une substitution ou une FA peut entraîner une mauvaise décision du DM qui va orienter le dialogue dans une mauvaise direction. L'utilisateur se retrouve alors aiguillé vers un domaine qu'il n'a pas demandé et d'où il peut s'avérer difficile de revenir en arrière sans recommencer le dialogue de zéro. Un FR, en revanche, aura pour effet une réponse d'incompréhension du système et éventuellement une demande de répétition de la requête. Ceci est moins grave en apparence que le premier cas, mais trop de demandes de répétition peut rapidement faire baisser l'acceptation du système par l'utilisateur.

Diminution relative d'entropie

L'amélioration des performances des algorithmes de génération de CNs, obtenue grâce à l'utilisation de la probabilité *a posteriori* en tant que mesure de confiance, montre l'efficacité de celle-ci. Une autre méthode pour évaluer la fiabilité d'une mesure de confiance est le calcul de la diminution relative de l'entropie croisée ΔH (voir 2.1.3 pour plus de détail). Étant donné que celle-ci mesure l'information additionnelle apportée à l'hypothèse de mot par la mesure de confiance, plus ΔH est élevé plus la mesure de confiance est prédictive.
Le tableau 6.13 montre le détail des performances en termes de diminution relative

	ΔH (%)	H_{init}	H_{MC}
pivot topologique	29.3	0.6784	0.4795
multi-niveaux	30.1	0.6718	0.4694
multi-niveaux sans vides	26.3	0.6470	0.4767

TABLE 6.13 – *Performances de la mesure de confiance en termes de diminution relative d'entropie*

d'entropie. On observe ainsi que la mesure de confiance est plus performante sur l'algorithme *multi-niveaux* que sur les deux autres algorithmes. Effectivement, étant donné les performances équivalentes en termes de *WER* observées dans la figure 6.2 et une entropie croisée initiale plus élevée pour l'algorithme *multi-niveaux*, la réduction d'entropie est plus importante pour cet algorithme que pour l'algorithme *multi-niveaux sans vides*.
La réduction est donnée à titre indicatif, mais la comparaison est difficile dans la mesure où la précision initiale n'est pas la même.

6.5 Élagage *a posteriori* des classes du réseau de confusion

Nous avons montré qu'en utilisant l'algorithme *multi-niveaux sans vides* on peut obtenir une réduction de près de 99% du nombre de transitions dans un CN par rapport aux graphes de mots. Cette réduction est de 36% par rapport aux CNs générés avec l'algorithme *pivot topologique*. De plus ces algorithmes permettent une amélioration importante du *WER* oracle par rapport aux graphes de mots malgré un nombre d'hypothèses de mots très réduit par rapport aux graphes de mots. Toutefois, comme le montre le tableau 6.10, la largeur des CNs reste élevée alors que le nombre moyen de mots par classe est relativement petit. Nos efforts de réduction de la taille des CNs se concentrent donc sur une diminution de la largeur des CNs, tout en tenant compte de la valeur du *WER* oracle.
Une analyse détaillée des CNs a révélé que sur un nombre important de classes, l'omission a la plus grande probabilité *a posteriori* (souvent très proche de 1) de la classe et la différence avec le meilleur mot de la classe est très grande. Étant donné que ces mots ont des probabilités *a posteriori* très faibles, l'élimination de cette classe n'influe pas sur le *WER* oracle [5]. Nous proposons une étape d'élagage des CNs qui permet d'éliminer les classes qui satisfont les deux conditions suivantes :

1. L'omission à la plus grande probabilité *a posteriori* de la classe.
2. $\frac{P(omission)}{P(\text{la meilleure hypothèse de mot})} > Seuil$

Les figures 6.4 et 6.5 montrent les performances en termes de *WER* oracle ainsi que la taille des CNs en fonction du seuil d'élagage. Les figures ont été réalisées en variant le seuil à partir d'une valeur égale à 5 qui correspond au point le plus à gauche sur les courbes. Le point le plus à droite de chaque courbe correspond aux performances des CNs sans élagage.

5. Des expériences menées dans (Mangu, 2000) ont montré qu'en éliminant, dans une classe, les mots ayant une probabilité *a posteriori* inférieure à 0.1 le *WER* oracle reste inchangé.

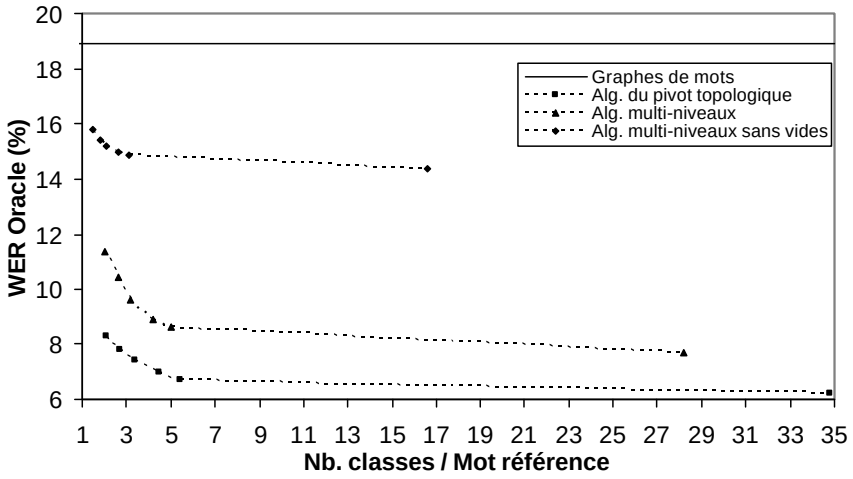

FIGURE *6.4* – *WER* oracle en fonction de la largeur des CNs pour différentes valeurs du seuil d'élagage

La figure 6.4 montre la variation du *WER* oracle et du nombre moyen de classes par mot de la référence en fonction du seuil d'élagage. La courbe correspondant à l'algorithme *multi-niveaux sans vides* montre un *WER* oracle plus élevé par rapport aux deux autres algorithmes pour les raisons que nous avons expliqué au 6.3.2 liés à la non-inclusion d'une grande partie des mots vides dans les CNs.
On observe sur la figure 6.4 que l'évolution de la taille des CNs en fonction du seuil d'élagage n'est pas la même pour les trois algorithmes. Ainsi, pour des valeurs de seuil petites (à gauche sur la courbe) le *WER* oracle pour l'algorithme *multi-niveaux* se dégrade plus rapidement comparé à l'algorithme *pivot topologique*. En ce qui concerne l'algorithme *multi-niveaux sans vides*, on observe une augmentation relative du *WER* oracle moins importante que pour les deux autres algorithmes ce qui montre que l'influence de l'élagage des classes est moins importante sur les performances de l'oracle. De plus, on observe que pour une valeur de seuil égale les CNs générés avec l'algorithme *multi-niveaux sans vides* ont une largeur moyenne 25% à 40% plus petites que pour l'algorithme du *pivot topologique*. La différence entre ce dernier et l'algorithme *multi-niveaux* est moins importante. La figure 6.5 montre l'évolution de la profondeur des CNs par rapport à leur largeur en fonction du seuil d'élagage des classes. On observe ainsi que plus l'élagage est fort (une valeur de seuil petite) plus le nombre moyen de mots par classe augmente fortement. Pour la même valeur du seuil on obtient une augmentation de seulement 60% pour l'algorithme *multi-niveaux* et de 70% pour l'algorithme *multi-niveaux sans vides*. Ceci montre bien l'existence des classes dont l'omission a la plus forte probabilité et qui contiennent un nombre d'hypothèses très faible.
L'effet de l'élagage des classes sur les performances des trois algorithmes est assez inégal d'un algorithme à l'autre. Toutefois, l'algorithme *multi-niveaux sans vides* permet de réduire fortement la taille des CNs avec une augmentation relative de *WER* oracle la moins importante. De plus, à seuil égal, les CNs générés ont la plus petite taille.
L'élagage des classes n'a aucun impact sur le *WER* et sur l'*IER*. Ceci est du au fait qu'on

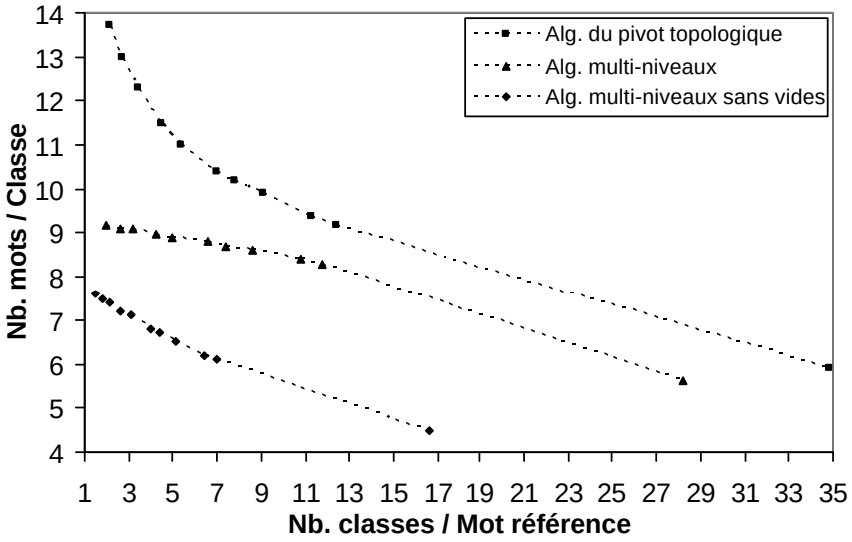

FIGURE *6.5* – Taille des réseaux en fonction des différentes valeurs du seuil d'élagage

élimine seulement des classes pour lesquelles l'omission a la plus grande probabilité *a posteriori*. Étant donné que lors de l'extraction de la *consensus hypothesis* on choisit les mots, ou l'omission, de chaque classe ayant la plus grande probabilité *a posteriori*, pour les classes éliminées le processus d'extraction aurait de toute manière choisi l'omission.

6.6 Parsing des réseaux de confusion

L'étape d'élagage des classes présentée au 6.5 permet de réduire de manière significative la taille des réseaux sans que les performances en terme de *WER* oracle soient dégradées de manière significative. Toutefois, cet élagage est une méthode générale de réduction de la taille des CNs qui ne tient pas compte des caractéristiques des modules applicatifs en aval. Nous proposons un algorithme de post-traitement des CNs qui vise à réduire la taille des CNs tout en privilégiant l'information porteuse de sens pour une application en aval.
Lors de l'analyse en concepts d'une séquence, le module de compréhension favorise l'allumage des concepts qui correspondent à des séquences contenant le plus de mots possibles. Dans un CN, les mots se trouvant dans des classes adjacentes ne forment pas nécessairement des séquences qui allument des concepts. Ceci est d'autant plus vrai pour les CNs générés avec l'algorithme *pivot topologique* ou *multi-niveaux* car les classes contiennent tous les mots vides présents dans les graphes de mots. Pour l'algorithme *multi-niveaux sans vides*, malgré le fait que seule une proportion réduite de mots vides sont présents dans les CNs, les mots non-vides présents dans des classes adjacentes ne forment pas nécessairement des séquences de mots qui correspondent à des concepts. En effet, certains mots non-vides n'allument un concept que s'ils font partie d'une séquence de plusieurs mots, alors que d'autres allument tout seuls un concept. Le pro-

cessus d'interprétation étant plus robuste si on privilégie l'allumage des concepts par des séquences de mots le plus longues possibles, l'algorithme de *parsing* que nous proposons vise à réduire la taille des CNs tout en favorisant la présence de ces séquences dans le CN.
Les graphes de mots, et par conséquence les CNs, contiennent des hésitations ou des erreurs dues aux disfluences. Si ces disfluences apparaissent dans la *consensus hypothesis* au milieu d'une séquence qui allume un concept, celui-ci ne pourra pas être allumé. Étant donné que les règles des concepts ont été construites manuellement elles ne contiennent pas ces disfluences et l'algorithme de *parsing* permet donc de les filtrer.

Fonctionnement de l'algorithme de parsing

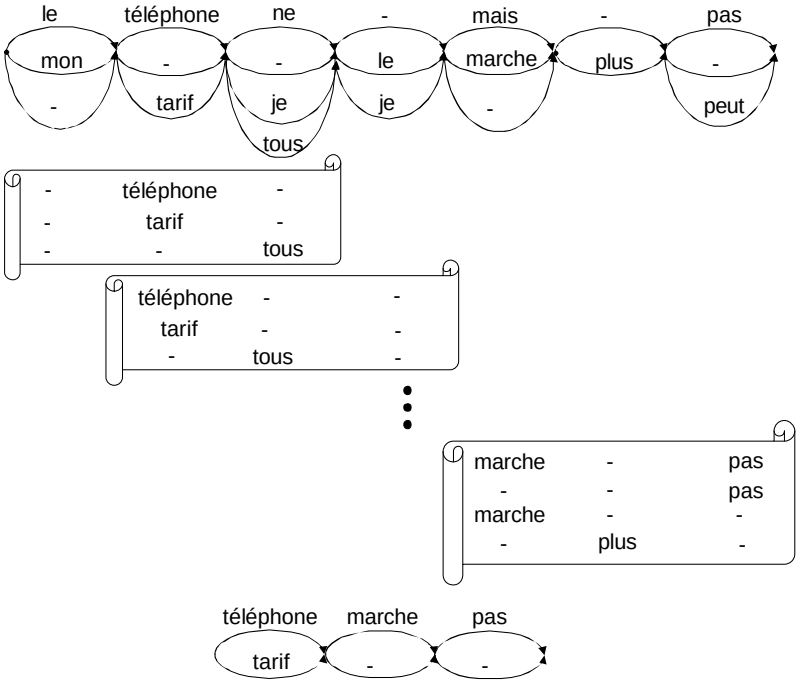

FIGURE 6.6 – Exemple illustrant le fonctionnement de l'algorithme de *parsing* d'un CN.

Nous illustrons le fonctionnement de l'algorithme de *parsing* à travers l'exemple présenté dans la figure 6.6. L'exemple correspond à l'énoncé "**mon téléphone ne marche pas**" et la *consensus hypothesis* extraite du CN est "**le téléphone ne mais pas**". Le tableau 6.14 présente un extrait des règles d'allumage de concepts qui ont dans leur composition un des mots non-vides du CN. Les mots présents dans le CN mais qui ne le sont pas dans les règles sont des mots vides.
Etant donné ces règles, l'analyse en concepts de l'énoncé et de la *consensus hypothesis* est donnée dans le tableau 6.15.
Le principe de l'algorithme de *parsing* est de chercher des séquences de mots qui al-

Séquence de mots	Concept
marche pas	[Probleme]
marche	[Fonction]
ne plus	[Plu]
plus	[Plus]
tarif	[Tarif]
téléphone	[Telephone]
tous	[Tout]

TABLE 6.14 – *Liste des règles d'allumage de concepts*

	Mots	Concept	Interprétation
énonce	mon téléphone ne marche pas	[Telephone] [Probleme]	Serv(DixQuatorze)
consensus hypothesis	le téléphone ne mais pas	[Telephone] [Pas]	Rejet

TABLE 6.15 – *Analyse en concepts de l'énoncé et de la* consensus hypothesis *avant le* parsing.

lument des concepts sur une fenêtre couvrant plusieurs classes adjacentes. L'algorithme de *parsing* est un algorithme itératif, à chaque itération la fenêtre se déplace d'une classe, le but étant de parcourir l'ensemble du CN. Dans notre exemple, la fenêtre couvre 3 classes adjacentes. Ainsi, comme le montre la figure 6.6, la première fenêtre extrait toutes les séquences des mots qui allument un concept. Par exemple, la séquence "**- téléphone -**" qui allume le concept "**[Telephone]**" est formée de deux omissions correspondant à la première et à la troisième classe et du mot "**téléphone**".
Pour chaque séquence de mot extraite ainsi on calcule une probabilité de la manière suivante :

$$P(\text{séquence}) = (\prod_{i=1}^{F} P(w_i))^{1/n} \tag{6.1}$$

où F est la longueur de la fenêtre et n le nombre de mots dans la séquence. Pour notre séquence, $n = 1$ car on a un seul mot dans la séquence, et la probabilité est $P(\text{- téléphone -}) = (P(-) \cdot P(\text{téléphone}) \cdot P(-))^{1/1}$.
Comme nous l'avons expliqué, nous voulons favoriser les séquences contenant plusieurs mots. Dans ce sens, de manière générale, plus il y a des mots dans une séquence plus sa probabilité est grande. En effet, $\prod_{i=1}^{F} P(w_i)) < 1$ du fait qu'on multiplie que des valeurs inférieures ou égales à 1, et la racine n^{ime} est plus grande pour une séquence contenant plus de mots (n plus grand). Dans notre exemple, cette situation est observée dans la dernière fenêtre.
A chaque itération de l'algorithme, la fenêtre se déplace d'une classe vers la droite, on extrait les séquences de mots et on calcule leurs probabilités. L'algorithme s'arrête lorsque toutes les classes du CN on été couvertes. Dans notre cas, la dernière fenêtre pour laquelle on extrait les séquences de mots est celle contenant les trois dernières classes du CN. Dans chaque fenêtre dans l'exemple, les séquences de mots sont ordonnées en fonction de leur probabilité. A la fin de l'algorithme, on construit le CN en

tenant compte des séquences de mots extraites dans chaque fenêtre et de leur probabilité. Ainsi, on peut choisir de prendre en compte que le N meilleures séquences dans chaque fenêtre. Dans l'exemple, on prend $N = 2$ et on reconstruit le CN avec seulement les mots contenus dans les deux premières séquences de chaque classe. Pour le reconstruire, on parcours le CN initial, et on regarde dans chaque classe si les mots (ou l'omission) qu'elle contient font partie des deux première séquences de chaque fenêtre dont la classe fait partie. Par exemple, pour la deuxième classe, les deux mots font partie des deux premières séquence, mais pas l'omission. Seuls les deux mots sont utilisés et forment une classe dans le CN reconstruit. Si aucun mot (ou omission) d'une classe n'est utilisé, le CN reconstruit ne contient pas de classe correspondante, comme c'est le cas dans l'exemple pour la première classe du CN initial.
La probabilité *a posteriori* des mots dans le CN reconstruit, est calculée à partir des probabilités *a posteriori* de mots dans le CN initial qu'on normalise pour que leur somme soit égale à 1 dans le CN reconstruit. La *consensus hypothesis* est calculée en se basant sur les nouvelles probabilités *a posteriori* et elle est ensuite traitée par le module de compréhension pour obtenir une interprétation. Comme le montre le tableau 6.16, suite au *parsing*, la nouvelle *consensus hypothesis* produit la même interprétation que celle de l'énoncé.

	Mots	Concept	Interprétation
énonce	mon téléphone ne marche pas	[Telephone] [Probleme]	Serv(DixQuatorze)
consensus hypothesis après *parsing*	téléphone marche pas	[Telephone] [Probleme]	Serv(DixQuatorze)

TABLE *6.16* – *Analyse en concepts de l'énoncé et de la* consensus hypothesis *après le* parsing.

L'algorithme de *parsing* se base sur deux paramètres qui peuvent varier en fonction des besoins applicatifs :

La largeur F de la fenêtre La largeur de la fenêtre est choisie en fonction de la longueur maximale des séquences de mots. Dans notre cas, les règles de concepts contiennent des séquences d'au maximum quatre mots. Un deuxième facteur qui entre en jeu est le nombre d'omissions entre deux mots d'une séquence. Dans l'exemple présenté, la séquence "**marche - pas**" contient une omissions entre les mots "**marche**" et "**pas**". Il convient d'établir un nombre maximal d'omissions entre les mots d'une séquence afin de ne pas former des séquences qui couvrent un nombre trop grand de classes (comme la séquence "**ne - - plus**" dans l'exemple) et qui ne reflète plus la réalité (un nombre élevé d'omissions allonge le support temporel de la séquence). Nous considérons qu'une seule omission peut exister entre le premier et le dernier mot d'une séquence. La largeur de la fenêtre est alors égale à la somme entre le nombre maximal de mots dans une séquence et le nombre d'omissions. Pour l'application de dialogue utilisée, $F = 5$.

Le nombre N de séquences par fenêtre Le nombre de séquences choisies est un compromis entre les performances des CNs en termes de *IER* et la taille des CNs. Nous avons choisi $N = 30$ pour nos évaluations.

Résultats expérimentaux

Les tests ont été effectués sur le corpus de test **Test_II** pour les algorithmes *multi-niveaux* et *multi-niveaux sans vides*. Nous nous sommes tout d'abord intéressé à l'influence de l'algorithme de *parsing* sur les CNs générés par l'algorithme *multi-niveaux* afin de voir si on pouvait améliorer d'avantage les performances en termes d'*IER*. Nous avons également comparé les performances des CNs sur lesquelles on applique une étape d'élagage de classes et ensuite le *parsing*, aux performances des CNs qui subissent seulement l'étape de *parsing*. Cette comparaison a pour but d'évaluer l'influence de la taille initiale des CNs sur l'algorithme de *parsing*.

	IER	C	FR	Sub	FA
1-best ASR	22.7%	92.7%	1.6%	5.7%	15.4%
multi-niveaux	20.2%	91.4%	3.4%	5.2%	11.6%
multi-niveaux + parsing	23.4%	90.6%	3.2%	6.2%	14.0%
multi-niveaux + élagage classes + parsing	22.7%	91.2%	3.2%	5.6%	13.9%

TABLE 6.17 – *Performances du parsing sur les CNs générés avec l'algorithme* multi-niveaux. *Evaluation au niveau interprétation*

Le tableau 6.17 montre les performances en terme d'*IER* de l'algorithme de *parsing* sur la *consensus hypothesis* des CNs générés avec l'algorithme *multi-niveaux*. Les deux premières lignes sont un rappel des performances de la *1-best* et de l'algorithme *multi-niveaux*. Dans la dernière ligne de tableau, les CNs subissent une étape d'élagage de classes avec un seuil de 10 avant le *parsing*. Ce seuil nous a paru être un bon compromis entre la taille et le *WER* oracle. On observe une dégradation des performances pour l'algorithme de *parsing* appliqué sur les CNs sans étape d'élagage qui est due principalement à l'augmentation du nombre de substitutions et de FA. Cette augmentation s'explique par une augmentation du taux de substitution et d'insertion au niveau concept, comme on observe dans le tableau 6.18. Les erreurs d'insertion au niveau concept proviennent principalement des concepts allumés par un seul mot. Le problème réside dans la taille du CN initial (28.2 classes par mot de référence, cf tableau 6.19) qui rend difficile l'extraction des séquences de plusieurs mots avec une fenêtre de largeur 5. Ceci résulte en un nombre important d'insertions de concepts allumés par un seul mot. Par ailleurs, on observe pour *multi-niveaux+élagage classes+parsing* une diminution du nombre d'insertions au niveau concept du fait d'un nombre de classes par mot de la référence beaucoup plus petit, ce qui engendre une baisse du nombre de substitutions au niveau interprétation. On obtient ainsi des performances équivalentes à celles de la *1-best*.
Le tableau 6.19 montre l'influence de l'algorithme de *parsing* sur la taille des CNs. On observe ainsi une réduction importante de la largeur de CNs sans l'étape d'élagage des classes. L'explication pour cette réduction du nombre de classes est donnée dans l'exemple montré dans la figure 6.6. L'algorithme de *parsing* permet aussi une légère réduction de la taille des CNs même si une étape d'élagage des classes est appliquée préalablement (on passe de 2.6 à 2 classes par mot de la référence).

	CER	C	FR	Sub	FA
multi-niveaux	21.0%	85.1%	6.2%	12.1%	2.7%
multi-niveaux + parsing	25.6%	83.5%	9.1%	13.9%	2.6%
multi-niveaux + élagage classes + parsing	23.3%	83.5%	6.8%	13.9%	2.6%

TABLE *6.18* – *Performances du parsing sur les CNs générés avec l'algorithme* multi-niveaux. *Evaluation au niveau concept*

	# Classes / Mot de la référence	# Mots / Classe
multi-niveaux	28.2	5.9
multi-niveaux + élagage classes	2.6	9.1
multi-niveaux + parsing	18.1	3.8
multi-niveaux + élagage classes + parsing	2.0	5.4

TABLE *6.19* – *Influence du parsing sur la taille des réseaux générés avec l'algorithme* multi-niveaux.

Nous avons aussi appliqué l'algorithme de *parsing* sur les réseaux générés avec l'algorithme *multi-niveaux sans vides* avec un élagage préalable des classes (le seuil d'élagage est égal à 10). Le tableau 6.20 montre les performances en termes d'*IER*. On observe ainsi une dégradation des performances par rapport à l'algorithme *multi-niveaux* due à l'augmentation des FA. Ceci s'explique par une tendance de l'algorithme de *parsing* à faire ressortir les séquences de mots formées d'un seul mot sur les CNs générés par l'algorithme *multi-niveaux sans vides* correspondant à des énoncés à rejeter.

	IER	C	FR	Sub	FA
multi-niveaux sans vides + élagage classes + parsing	24%	91.3%	3.2%	5.5%	15.3%

TABLE *6.20* – *Performances du parsing sur les CNs générés avec l'algorithme* multi-niveaux sans vides.

Discussions

Nous avons présenté dans cette section une étude préliminaire sur un algorithme de *parsing* des CNs. Nous avons observé que l'algorithme de *parsing* est plus efficace si les réseaux ne sont pas trop grands et donc une étape d'élagage des classes peut être nécessaire avant d'appliquer le *parsing*. Les performances moins bonnes en terme d'*IER* des CNs ayant subi un *parsing* ont pour cause principale un nombre trop important d'insertions des concepts allumés par un seul mot. Ceci est dû à un nombre réduit de règles d'allumage de concepts dont les séquences ont plus d'un mot (10% du total des règles). Nous pensons que dans un contexte avec des concepts plus couvrants le *parsing*

devrait être plus efficace.
Une autre piste pour l'amélioration de l'algorithme réside dans le calcul de la probabilité des mots dans le CN obtenu après le *parsing*. La nouvelle probabilité *a posteriori* pourrait être calculée à partir de la probabilité *a posteriori* du mot dans le CN initial mais avec un facteur de pondération donné par le nombre de séquences, extraites dans les fenêtres et qui contiennent le mot. Une autre possibilité serait d'estimer la nouvelle probabilité comme étant égale à la fréquence d'apparition d'un mot dans les séquences de mots extraites dans les fenêtres.

6.7 Généralisation de l'algorithme

Comme nous l'avons expliqué dans l'introduction de ce chapitre, le but des modifications présentées n'était pas de construire un algorithme dédié à un système de dialogue homme-machine en langage naturel comme celui utilisé par le service 3000. Les modifications présentées peuvent facilement être généralisées afin qu'elles puissent être utilisées dans d'autres domaines d'application.
L'objectif de ces modifications est de rendre plus fiable la construction des CNs en donnant une importance plus grande aux mots significatifs de l'application choisie. Le principe est de définir des groupes de mots auxquels on attribue des degrés différents d'importance qui définiront l'ordre d'insertion des transitions dans le CN et éventuellement les transitions qui ne sont pas insérées. L'algorithme généralisé comprend alors deux étapes. La première reste celle du calcul de la séquence *pivot*. La deuxième étape est une étape itérative qui traite les groupes de mots en fonction de leur degré d'importance. Ainsi, à chaque itération, l'algorithme insère dans le CN les transitions portant des mots d'un ou plusieurs groupes ayant le même degré d'importance. Un paramètre supplémentaire qui peut être changé à chaque itération est l'interdiction ou non de la création de nouvelles classes dans le CN lors de l'insertion d'une transition. Les groupes de mots qu'on ne souhaite pas introduire dans le CN reçoivent le degré d'importance le plus petit. De cette façon, on peut arrêter les itérations à partir d'un degré d'importance trop petit de manière à ce que les transitions portant ces mots ne soient pas traités par l'algorithme.
Nous avons appliqué cet algorithme pour la génération des CNs pour une application de traduction automatique de la parole continue. Les groupes de mots ont été construits avec les mots appartenant aux classes de mots utilisées par le modèle de traduction. Dans cette application de traduction nous avons également utilisé l'algorithme de *parsing* afin de réduire la taille des CNs tout en tenant compte des syntagmes de la table de traduction utilisée par l'application. Une explication plus détaillée ainsi que les résultats obtenus sont présentés dans les annexes B et C.

6.8 Conclusions

Ce chapitre a présenté dans un premier temps une analyse détaillée du fonctionnement de l'algorithme *pivot topologique*. Nous avons ainsi montré un phénomène de dédoublement de mots sur plusieurs classes du CN dû au parcours topologique des transitions et qui influe de manière négative sur le *WER* de la *consensus hypothesis* mais aussi sur la taille des réseaux. Un autre point abordé a été la prise en compte des caractéristiques du module de compréhension dans la génération des CNs. Les problématiques découvertes lors de cette analyse nous ont ensuite permis de proposer des modifications de l'algorithme du *pivot topologique*.
Tout d'abord, une approche heuristique pour la mise en œuvre de la relation d'ordre entre deux transitions a été proposée afin de rendre son calcul plus rapide et moins gourmand en ressources mémoire. Cette approche permet de générer des CNs plus compacts sans modifier leur performance en termes de *WER* et d'*IER*. Nous avons ensuite proposé deux versions d'un algorithme de génération multi-niveaux, permettant à la fois une amélioration significative des performances en termes de *WER* et d'*IER* et aussi de réduire de manière importante la taille des CNs. L'utilisation des mesures de confiance pour filtrer la *consensus hypothesis* permet d'améliorer davantage les performances. La différence entre les deux variantes de l'algorithme multi-niveaux réside dans le choix des objectifs de l'application choisie. Ainsi, si on a besoin d'un espace des hypothèses plus grand avec de bonnes performances au niveau interprétation, on choisi l'algorithme *multi-niveaux*. En revanche, si on se trouve dans un contexte avec de fortes contraintes de temps de calcul et de tailles des données, l'algorithme *multi-niveaux sans vides* peut constituer un meilleur choix.
Deux étapes de post-traitement des réseaux de confusion ont aussi été proposées. L'étape d'élagage des classes du CN permet de réduire de manière importante la taille des CNs avec une dégradation des performances de l'oracle moindre. L'algorithme de *parsing* permet de réaliser une réduction de la taille des CNs tout en tenant compte des caractéristiques des modules applicatifs en aval. Les résultats obtenus sont encourageants et montrent l'intérêt d'effectuer un élagage des réseaux en privilégiant l'information porteuse de sens pour une application en aval. Des perspectives d'amélioration de l'algorithme ont été discutées. Nous avons également utilisé l'algorithme de *parsing* dans un contexte de traduction automatique de la parole basée sur la traduction des CNs. Cet aspect est présenté dans les annexes B et C.